

CZECH UNIVERSITY OF LIFE SCIENCES IN PRAGUE
FACULTY OF ECONOMICS AND MANAGEMENT



**Neural Network Architectures for Object Recognition in
Digital Photographs**

Dissertation

Author: Ing. Adéla Hamplová

Supervisor: doc. Ing. Arnošt Veselý, CSc., Dept. Of Information Engineering

Praha 2024

Neural Network architectures for object recognition in digital photographs

Abstract

This dissertation is a collection of scientific articles discussing the topic of neural networks suitable for image recognition in digital photographs. Since this topic is very broad and general, this dissertation investigates explicitly the use of neural networks suitable for text recognition, namely two historical alphabets - the Palmyrene alphabet, which was used to write texts in the Palmyrene dialect of Aramaic, and the cuneiform script. Using the Palmyrene alphabet as an example, a custom cascade pipeline is presented to build an OCR algorithm that identifies individual characters using a segmentation neural network and evaluates which character is the correct one using a custom optimal classifier or directly using multi-class instance segmentation. The finished solution is presented in a mobile and web application. GAN-type neural networks were used to refine the results of the classifier using training on the augmented dataset, and their features were investigated. On the other hand, cuneiform analysis was performed using an object detection algorithm, where individual strokes in the image were detected and redrawn onto a blank canvas using a custom utility. I believe that the contribution of this dissertation is not only theoretical but also practical.

Keywords: Artificial Intelligence, Computer Vision, Object Detection, Convolutional Neural Networks, Data Augmentation, Text Segmentation, Pattern Recognition

Table of contents

1	Introduction	5
2	Theoretical basis and current state of art	7
2.1	Artificial Neural Networks.....	7
2.1.1	Brief history of ANNs.....	7
2.1.2	Description.....	8
2.2	Convolutional Neural Networks.....	9
2.2.1	Main principles	9
2.2.2	CNN layers.....	10
2.2.2.1	Convolutional layer	10
2.2.2.2	Subsampling layer	13
2.2.2.3	Flatten layer	14
2.2.2.4	Dense layers and output functions.....	14
2.2.3	Training Convolutional Neural Networks	15
2.2.4	Development of CNNs.....	16
2.2.5	Classification on mobile devices	19
2.2.6	Latest research	21
2.3	Computer vision tasks	21
2.4	Classification.....	22
2.4.1	Determining dataset size	22
2.4.2	Classifier success measures	23
2.4.2.1	Confusion matrix	23
2.4.2.2	Measures derived from confusion matrix.....	25
2.5	Object Detection.....	26
2.5.1	Traditional image processing techniques vs Deep Learning	27
2.5.2	Bounding box formats.....	27
2.5.2.1	PASCAL VOC bounding boxes	27
2.5.2.2	COCO Json bounding boxes.....	28
2.5.2.3	Tensorflow Object Detection bounding boxes	28
2.5.2.4	YOLO DarkNet bounding boxes.....	28
2.5.3	Latest Object detection algorithms	28
2.5.3.1	Two-Stage Detection	29
2.5.3.2	One-Stage Detection.....	31
2.5.4	Comparing object detection algorithms	36
2.5.4.1	Evaluation options	36

2.6	Segmentation.....	36
2.6.1	Mathematical methods used for segmentation.....	37
2.6.2	Object detection algorithms used for segmentation.....	38
2.6.2.1	Evolution of semantic segmentation.....	38
2.6.2.2	Early approaches of instance segmentation.....	38
2.6.2.3	Mask R-CNN.....	39
2.6.2.4	YOLO.....	41
2.7	Dataset augmentation.....	41
2.7.1	Classical methods.....	41
2.7.1.1	Keras generator.....	41
2.7.1.2	Online platform Roboflow.....	42
2.7.2	Deep learning methods.....	44
2.7.2.1	Neural Style Transfer.....	44
2.7.2.2	Feature space data augmentation.....	45
2.7.2.3	Variational autoencoders.....	46
2.7.2.4	Generative adversarial networks.....	47
2.8	Optical Character Recognition.....	48
2.9	Research gap.....	50
3	Commentary.....	51
3.1	Palmyrene Aramaic analysis using computer vision algorithms.....	51
3.1.1	Palmyrene alphabet research outcome.....	53
3.2	Cuneiform stroke detection.....	54
3.2.1	Confusion matrix used for object detection algorithms.....	54
3.2.1.1	Suggested method to constructing confusion matrix.....	55
3.3	Cascade-style approach to creating historical OCR systems.....	55
3.3.1	Data acquisition.....	55
3.3.2	Annotation.....	56
3.3.3	Multi-class instance segmentation.....	56
3.3.4	Post-processing.....	56
3.3.5	Text in machine-readable format.....	56
3.4	Next steps.....	56
	Conclusion.....	57
4	References.....	58
5	List of Figures, Tables and Abbreviations.....	66
5.1	List of Figures.....	66
5.2	List of Tables.....	66

5.3 List of Abbreviations.....	66
List of Attachments.....	69
Attachment 1	69
Attachment 2	69
Attachment 3	69
Attachment 4	69
Attachment 5	69
Attachment 6	69
Attachment 7	69

1 Introduction

The presented text of the commented dissertation - a compilation of scientific articles - consists of the theoretical basis and practical applications of Artificial Neural Networks used for object segmentation, classification, and detection, as well as expanding datasets from digital photographs. The main goal was to create a simplified, cascade-style approach, which is meant to develop historical alphabet Optical Character Recognition (OCR) and utilise multiple types of Convolutional Neural Networks. This novel cascade approach builds on previous research in Convolutional Neural Networks, Generative Adversarial Networks, Object Detection, and Optical Character Recognition and groups the knowledge from these areas into one semi-automated way of use, with nuances in use for alphabets and other types of writing. A modified method of constructing a confusion matrix, used for evaluating object detection algorithms when ground truth labels are not available to calculate Intersection over Union IoU, is presented here as well.

Thanks to the latest technological development of powerful GPUs, TPUs and pocket size, well-performing cameras, computer vision and object detection are nowadays ones of the critical applications of Artificial Intelligence, as they allow computers to identify the contents of their environment by saying what is in the picture, where and how it is represented. It has become a standard that has contributed to automatising in a vast number of human and machine activities in recent years. From industry, where it is necessary to consistently overview the processes of manufacturing and quality of products, counting people in shopping malls or detecting risky behaviour in warehouses, through expert systems or decision support systems, automatic number plate recognition barriers, medical diagnosis support including tumour detection, kidney stones detection and many others, followed by self-driving cars, to common applications such as photo editing programs, face tagging or personalised advertisements. The connecting element of all these applications is Deep Learning, specifically of various architectures of Convolutional Neural Networks.

The use of computer vision can make commercial organisations more competitive by automating tasks that otherwise require the use of human resources. In many applications, a machine can be more accurate or faster than a human in some respects. For this reason, computer vision technology continues to evolve and improve rapidly. Nowadays, there is a wide range of ready-made algorithms designed for object recognition in images, from simple

Single Shot Detectors, which mark selected objects in a smaller image in real-time, to complex mask Region-based Convolutional Neural Networks, which, in addition to detection, also offer marking of a cluster of pixels belonging to a given class, i.e., a mask - showing the outline of the object. However, each object detection task cannot be approached in a unified way and must be solved individually.

The utilisation of computer vision for automatic reading counts as one of the critical applications of Artificial Intelligence. As part of my doctoral studies and grant projects included in it, as well as the preparation of this dissertation, I decided to explore and expand the possibilities of reading historical scripts.

In particular, this dissertation focuses on using existing and constructing new architectures of Artificial Neural Networks that are applicable to the task of detecting historical scripts and expanding datasets of letters in alphabets or their elementary subparts (which, in the case of cuneiform fonts are the individual wedges) and presents novel solutions of these problems using a fully convolutional approach.

The conducted research was published in 3 different WOS/Scopus-indexed journals and 4 WOS/Scopus-indexed conferences as an output of grant projects:

- PEF IGA 2021A0004 - “Reading Palmyrene Alphabet Characters Using Artificial Intelligence Tools” [1] [2]
- PEF IGA 2022A0001 - “Research on methods for automatic dataset expansion using machine learning tools” [3]
- UGC project reg. No CZ.02.2.69/0.0/0.0/19_073/0016944 internal no. 31/2021 - “Cuneiform analysis using Convolutional Neural Networks” [4] [5]
- PEF IGA 2023A0004 – “Text segmentation methods of historical alphabets in OCR development” [6] [7]

2 Theoretical basis and current state of art

2.1 Artificial Neural Networks

2.1.1 Brief history of ANNs

Artificial Neural Networks (ANNs) are one of the areas of artificial intelligence, in addition to expert systems, fuzzy systems and genetic algorithms. They appeared in practice in the field of artificial intelligence in the 1960s after years of previous research. [8]

The first simple neuron model dates back to 1943 when Warren McCulloch and Walter Pitts proposed a mathematical model of the central nervous system and declared that “at any instant, a neuron has some threshold, which excitation must exceed to initiate an impulse” and described the propagation of the impulse. [9] In 1949, Donald Hebb's book “The Organization of Behavior” provided guidance on how to apply the learning rule to neuronal synapses. [10] In 1951, Marvin Minsky created the first SNARC neurocomputer. [11] In 1957, Frank Rosenblatt generalised the neuron model to a perceptron calculating real numbers in his report from Cornell Aeronautical Laboratory. [12] In 1958, together with Charles Wightman, he built the "Mark I Perceptron" neurocomputer with 512 parameters at the MIT laboratories, which was able to recognise characters, and in 1960, John C. Hay et al. wrote an operator manual [13]. In 1965, Bernard Widrow and his students created the Adaptive Linear Element (ADALINE) similar to perceptron, but the individual elements performed linear functions compared to perceptron [14], after them and one of his doctoral students, Marcian Edward “Ted” Hoff, who was also the co-inventor of microprocessors, the Widrow-Hoff least mean square training algorithm was named, it was published in Hoff's dissertation. [15] Another pioneer in the field of neurocomputers was Karl Steinbuch, who published a comparison of two adaptive classification networks with Widrow [16] and developed a model of a binary associative network, the principle of which is based on associative memory and the provision of certain information based on its partial knowledge.

For almost twenty years, neurocomputers have only been used for experimental purposes because perceptron has been shown to be unable to perform equivalence $x_1 \Leftrightarrow x_2$ (EQV) and non-equivalence $x_1 \oplus x_2$ (XOR), as these Boolean functions are not linearly separable, and the linear separability of data sets is an essential prerequisite for constructing a

single perceptron. It was not until 1982 that grant projects by John Hopfield (after whom Hopfield Neural Networks are named) excelled, proving the connection of some models with physical models of magnetic materials [17], according to which the Hopfield networks based on the principle of auto-associative memory were named. In 1986, David E. Rumelhart et al. published a practical backpropagation learning algorithm for multilayer networks in the Parallel Distributed Processing (PDP) group [19]. However, it was mentioned previously in 1974 in Paul John Werbos's dissertation [18]. In the 1990s, journals about Artificial Neural Networks began to publish, and the international journal Neural Network World [19] has been available in the Czech Republic since 1991.

It was not until 2012 that the field experienced a breakthrough in ImageNet classification contests [20], because until then other methods were more successful. Since then, artificial intelligence has been developed in a wide range of areas.

2.1.2 Description

Artificial neurons are an abstraction of the mechanism that processes information compared to the way that biological neurons send information to and within the brain and determine how to respond to that information. The training or prediction presents the first neural layer with an input from which we need to obtain an output. The input of the neuron to the next layer is always the output from the previous layer, and only the last layer shows the output. The whole network behaves based on parameters (threshold and weights) that determine the course of networks, so it is an oriented graph. [12]

An output – activation function (formerly, output function and activation functions had different meanings, but now it is interpreted the same way) is a function that converts an aggregated signal into an output signal, for example linear, binary - sigma ($\sigma(h)$), logistical sigmoid, signum, tanh, ReLU (Rectified Linear Unit), leaky ReLU, eLU (Exponential Linear Unit), softmax (generalised sigmoid, counting probability of the input belonging to a single class), or other. [21] These functions define the outputs of the neurons. Relevant activation functions of individual Neural Network layers will be explained later in the text of individual chapters. The simplest neuron – perceptron – is explained in the following Figure 1.

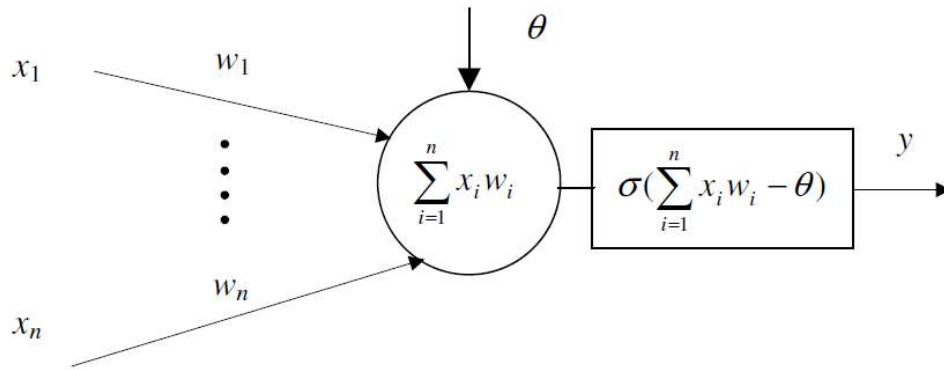


Figure 1 - mathematical model of perceptron with output function $\sigma(h)$ [22]

where:

$\mathbf{w} = (w_1, \dots, w_n)$ is the vector of weights

$h = \sum_{i=1}^n x_i \cdot w_i$ is the postsynaptic potential

and the output function is $\sigma(h) = 1$ for $h \geq 0$ and $\sigma(h) = 0$ for $h < 0$.

Unfortunately, the analysis of the dynamic behaviour of a Deep Neural Network is extremely complicated, since the calculation process is not linear and may include hundreds of thousands of computed parameters, and is unlike classical algorithms, where it is possible (for more complex programs such as debug mode) to follow the program step by step. Therefore, studying these problems is both practically and theoretically a core issue.

2.2 Convolutional Neural Networks

To analyse images, we use special kinds of Neural Networks called Convolutional Neural Networks (CNNs). CNNs are multilayer Neural Networks with thousands of parameters. They are commonly used to recognize objects in an image directly from individual pixels, regardless of their distortion, shift within the image, colour change, or other criteria. The name convolution means filtering performed by a feature map, automatically extracting object features (such as edges, arches and more). The cornerstone of each image analysis is classification, done by CNN.

2.2.1 Main principles

CNNs, unlike densely connected networks, use three main principles – local connectivity, shared weights (or weight replication) and spatial or temporal subsampling. [23] Thus, one layer of a convolutional Neural Network is not entirely connected to the next, but

only to selected parts, called subregions, avoiding an unmanageable number of parameters in hidden layers.

2.2.2 CNN layers

The architecture of CNNs consists of an input convolutional layer (in Deep Learning Python library keras [24] it is called Conv2D), a subsampling layer (MaxPooling2D) and a suitable iteration of these layers, a layer producing a one-dimensional vector (Flatten), a densely connected layer (Dense), and an output layer (Dense) whose number of neurons corresponds to the number of classes to be classified. In the case of localising an object in the image, the output layer will usually, apart from the class index, contain two points - [xmin, ymax] and [xmax, ymin] – which, by joining, will create a frame (bounding box), around the object within the image. There are more options to calculate the bounding box, which will be explained in reference to relevant existing object detection algorithms in future chapters. More layers, like Dropout for regularisations and randomly zeroing out weights, may be used in a CNN. However, the following description deals with obligatory layers. The complete list of keras layers and their description can be found in [25].

2.2.2.1 Convolutional layer

2.2.2.1.1 Image representation

Each input image Z is represented in the form of a 3-dimensional (in case of RGB spectrum) $M \times N \times D$ or 2-dimensional (in case of black and white spectrum) $M \times N$ array, where M is the width, N is the height and D is the depth. For example, the black and white letter “O” in a 9 x 12 px grid with zero-padding $P = 1$ looks as follows:

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	123	255	123	0	0	0	0
0	0	0	123	255	123	255	123	0	0	0
0	0	123	255	123	0	123	255	123	0	0
0	0	255	123	0	0	0	123	255	0	0
0	62	255	0	0	0	0	0	255	62	0
0	123	255	0	0	0	0	0	255	123	0
0	123	255	0	0	0	0	0	255	123	0
0	62	255	0	0	0	0	0	255	62	0
0	0	255	123	0	0	0	123	255	0	0
0	0	123	255	123	0	123	255	123	0	0
0	0	0	123	255	123	255	123	0	0	0
0	0	0	0	123	255	123	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

Table 1 - Convolutional layer input - a black and white letter "O" represented in pixels

2.2.2.1.2 Convolutional filters and feature maps

A small matrix of numbers called **convolutional filter c** (also called convolutional kernel or window) of size width W x height H , where $W, H > 0$, is applied on the image with a specified **stride S** (which is a step size, by which the convolutional filter is shifted), $S > 0$. Within it, individual components of the object, such as corners or edges, are recognized. Usually, multiple filters are applied. When applying the filters, **zero-padding P** around the image, which adds zeroes around the input image, can be used, so that there is no loss of information around the image corners during the computation, $P \geq 0$. The output is called **feature map F** (also called activation map), which, after being transformed by output function, transfers to subsequent layers. The equation is following. The indices of rows and columns of the output feature map are i and j .

$$F[i, j] = (z * c)[i, j] = \sum_m \sum_n c[m, n] \cdot z[i - m, j - n] \quad (1)$$

The dimensions I – the width – and J – the height – of the output feature map can be counted from the sizes of input $M \times N$, padding P , stride S , and window's width and height W, H .

$$I = \frac{M - W + 2P}{S + 1} \quad (2)$$

$$J = \frac{N - H + 2P}{S + 1} \quad (3)$$

The number of parameters in a convolutional layer is

$$params(conv) = (f_{in} \cdot I \cdot J + 1) \cdot f_{out} \quad (4)$$

where:

f_{in} = number of input feature maps

$I \cdot J$ = convolutional window size

f_{out} = number of output feature maps

2.2.2.1.3 Output functions of convolutional layers

There are four main non-linear activation functions used in convolutional layers – logistic sigmoid, tanh, ReLU and leaky ReLU. Their graphs are visible in Figure 1 below.

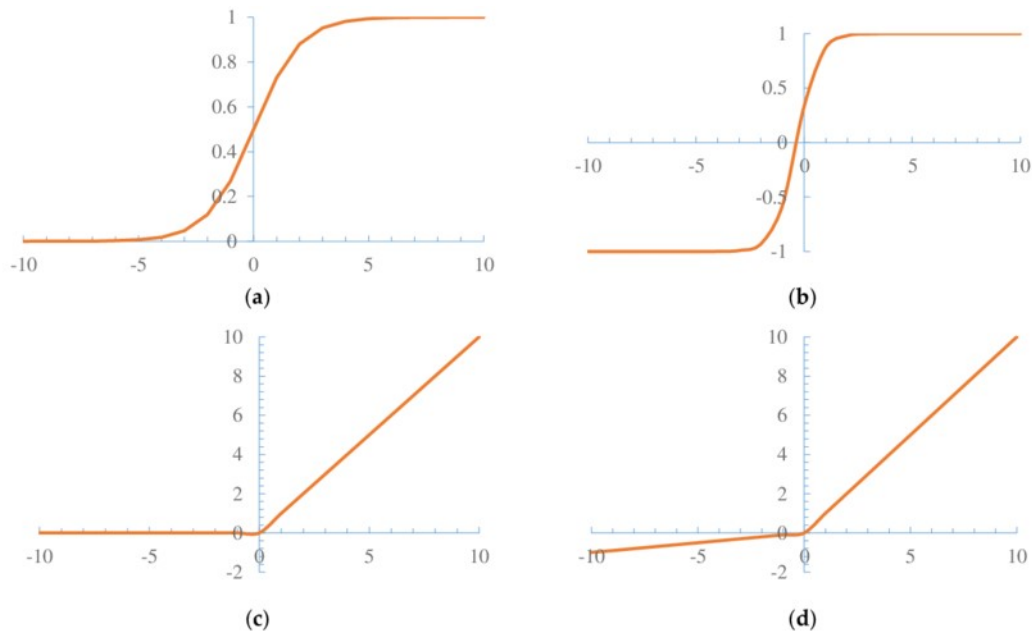


Figure 2 – activation functions of convolutional layers: (a) sigmoid (b) tanh (c) ReLU (d) leaky ReLU [26]

Sigmoid or **logistic sigmoid function** is such function, which is increasing, continuous and smooth and reaches values between 0 and 1. The main disadvantage of a sigmoid function is, that its gradient rapidly converges towards 0 (Vanishing Gradient problem). In keras, sigmoid function is equivalent to two-element softmax (which converts a vector of values to a probability distribution). [27]

$$sigm(h) = \frac{1}{1+e^{-h}} \quad (5)$$

Another commonly used output function is **Hyperbolic Tangent** (\tanh), also belonging to sigmoid functions. Tanh is converting the output feature map values to values between -1 and 1.

$$\tanh(h) = \frac{1 - e^{-2x}}{1 + e^{-2x}} \quad (6)$$

One of the most common output functions in a convolutional layer, transforming output feature maps, is **Rectified Linear Unit** ($ReLU$). In contradiction with sigmoid functions, it prevents gradient vanishing problem, and therefore it is preferred. ReLU is first mentioned in Fukushima's Neocognitron [28] and has been regularly used since the publication of Krizhevsky's AlexNet [29]. By zeroing out negative values from the output feature maps, it prevents negative pixels from passing to following layers.

$$ReLU(h) = \max(0, h) \quad (7)$$

However, even ReLU encounters its problem, called "dying ReLU". A dead ReLU always outputs 0, reached by using a large negative bias, resulting in the training not progressing, because the gradient of 0 is also 0. That is why **Parametric Rectified Linear Unit** ($PReLU$) was introduced, assigning a non-zero slope to the negative input values, [30] simplified into **leaky ReLU** with a constant value (usually 0.01) instead of a parameter α , the mathematical definition of PReLU is following.

$$PReLU(h) = \max(0, h) + \alpha \cdot \min(0, h) \quad (8)$$

2.2.2.2 Subsampling layer

Usually following a convolutional, another CNN layer called the subsampling (also pooling) layer is placed which aggregates neighbouring pixels according to its type, resulting in a reduction of the input size. Pooling is a sample-based discretisation process, aggregating the values of adjacent units. We distinguish max pooling (where we take the highest value of neighbouring pixels and pool them to a downsized matrix, shifting a specified window by a specified stride) and average pooling (where we take the average value). There are 0 parameters in a pooling layer, as it is only reducing dimension and not learning any new information.

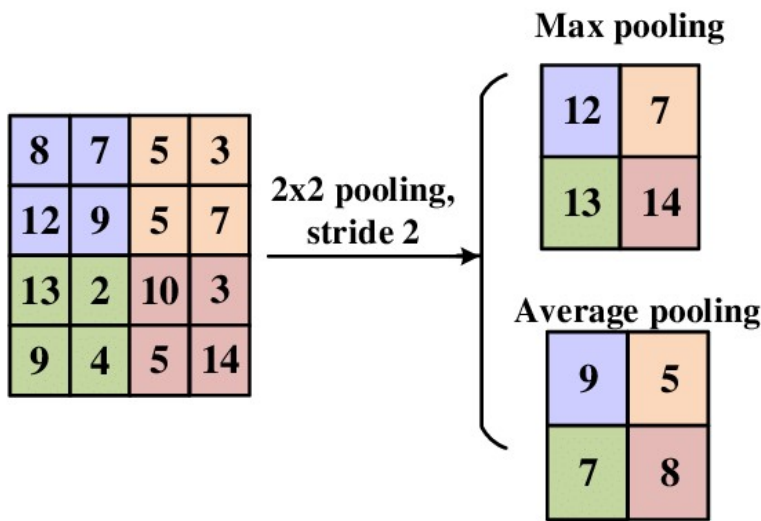


Figure 3 - Demonstration of maxpooling and averagepooling operation [31]

2.2.2.3 Flatten layer

Within a Flatten layer, a one-dimensional vector is created from a multi-dimensional tensor at the input of the given Flatten layer, with no effect on the batch. In keras, the input to Flatten layer is usually in the format (batch, channels, height, width), if ordering of the inputs is “channels_first” or (batch, height, width, channels), if the ordering is “channels_last”. Not affecting the batch, an example input (None, 1, 10, 64) will result in (None, 640) output. A Flatten layer also has 0 parameters, as it only changes the shape but perceives the same information.

2.2.2.4 Dense layers and output functions

A Dense layer is fully connected – all neurons from the Flatten layer are connected to all neurons in the Dense layer.

The number of parameters of the Dense layer is following.

$$param(Dense) = (in + 1) \cdot out \quad (9)$$

where:

in = input

out = number of output neurons

Output function of Dense layers depend on the task, which the CNN solves. The last but one hidden Dense layer is *logistical sigmoid* in case of binary classification and *logistical sigmoid* or *tanh* in case of multi-class classification. In the last Dense layer, in the case of binary classification the output neurons will be *logistical sigmoid*. In case of a multi-class classification, *logistical sigmoid* (in case of non-exclusive classification) or *softmax* (in case of exclusive classification) is chosen for output neurons.

2.2.3 Training Convolutional Neural Networks

Modern Convolutional Neural Networks are trained with error backpropagation algorithm, which is slightly different for each error function. It was originally derived for the use with **Sum of Square Errors** E_{SSE} loss function and then, similarly derived for **Cross-entropy** E_{CE} . E_{SSE} is used for regression, therefore it is not suitable for measuring error of a classification network. For binary classification with one output neuron, we minimise the **Binary Cross-entropy** error (loss) function *binary* E_{CE} and for multi-class classification, we minimise **Categorical Cross-entropy** *categorical* E_{CE} . *Cross-entropy* is counted as follows.

$$\text{binary } E_{CE} = -\sum_{i=1}^m (d_i \cdot \ln y_i + (1 - d_i) \cdot \ln(1 - y_i)) \quad (10)$$

$$\text{categorical } E_{CE} = -\sum_x \sum_{i=1}^m d_i \cdot \ln y_i \quad (11)$$

where:

y_i = output of neuron with respective index i

d_i = i -th component of the respective category (either 0 or 1)

m = number of categories

The *binary* E_{CE} back-propagation is counted as follows. [32] At first, the total error across all neurons is counted as the sum of errors of outputs as in (10). Considering the logistic output function y_i in a layer

$$y_i = \frac{1}{1+e^{-s_i}} \quad (12)$$

where:

$$s_i = \sum_j h_j w_{ji} \quad (13)$$

where:

h_j = the postsynaptic potential of respective neuron

w_{ji} = the weight of respective connection

s_i = weighted sums of the hidden layer activations

we compute the **gradient** (partial derivation) with respect to the weights connecting hidden neurons in the last layer, using chain rule.

$$\text{grad } E_{CE} = \frac{\partial E_{CE}}{\partial w_{ji}} = \frac{\partial E_{CE}}{\partial y_i} \frac{\partial y_i}{\partial s_i} \frac{\partial s_i}{\partial w_{ji}} = \frac{y_i - d_i}{y_i(1-y_i)} \cdot y_i(1-y_i) \cdot h_j = (y_i - d_i) \cdot h_j \quad (14)$$

In hidden layers, we also compute components of gradient and then we recursively propagate the error within the network, from the last layer to the first, and update the weight vector w^n accordingly. [22]

$$w^n = w^{n-1} - \varepsilon \cdot \text{grad } E_{CE} + \mu \cdot w^n \quad (15)$$

where:

$\varepsilon > 0$ = parameter controlling the step size

$\mu > 0$ = parameter controlling the speed and stability of algorithm (momentum)

When the termination condition is met, the training is completed (it can be a selected number of epochs or not getting better results for a specified number of epochs).

2.2.4 Development of CNNs

The first self-organised (unsupervised – learning without labelled data) predecessor of CNN using local connectivity was called Neocognitron. It was published in Biological Cybernetics by Kuniyiko Fukushima in the 1980's. It was named after extending “cognitron”, a model proposed by the author in earlier years, based on Hubel and Wiesel model. Neocognitron is capable of recognising patterns according to the geometrical similarity (Gestalt) independent of their position. The main benefit over previous Neural Network models was the ability not to be affected by the shifting or small distortion of an object within an image. [28]

In 1998, Yann LeCun et al. published a CNN trained with the gradient back-propagation algorithm, called LeNet, and applied it to the task of recognising handwritten characters almost without pre-processing the images; this time, it involved supervised

learning (using labelled data). In the same paper, they presented a new learning paradigm, Graph Transformer Network (GTN) for Optical Character Recognition (OCR) eliminating the deficiencies of using fixed-size vectors for the set of parameters and the state information communicated between the modules. [23] It was a generalisation and an extension of Hidden Markov Models.

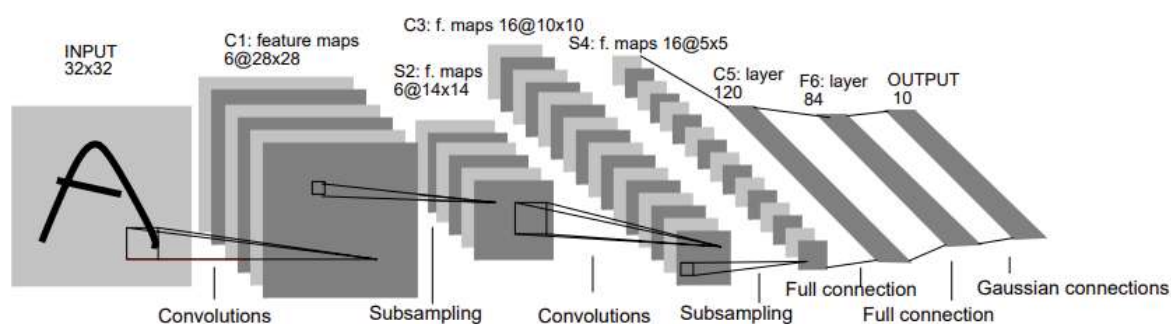
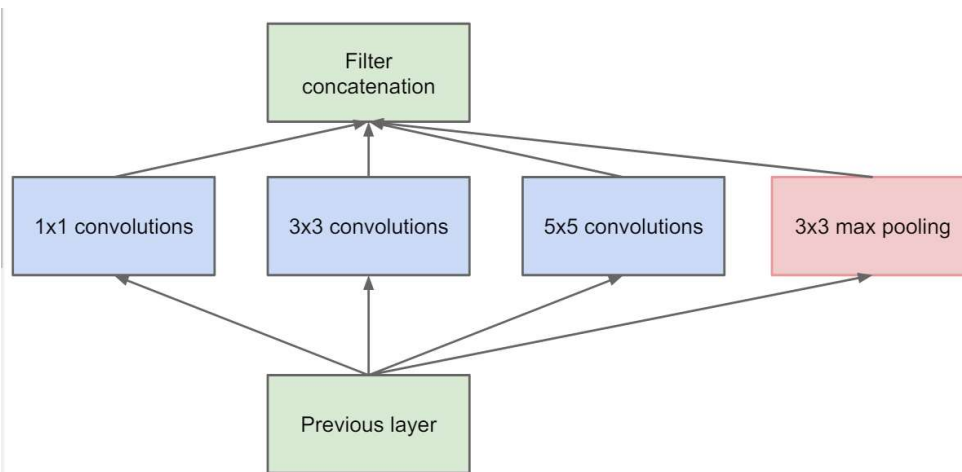


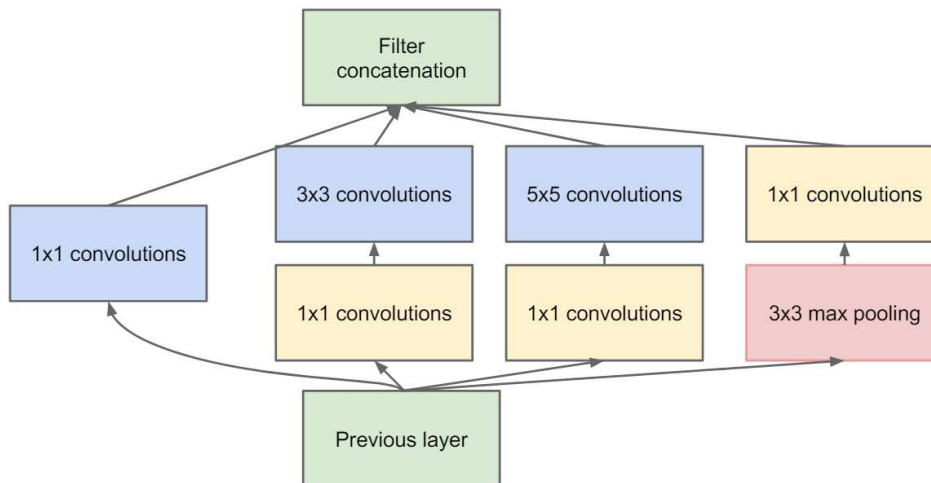
Figure 4- LeNet-5 used for recognition of handwritten character (32 x 32 pixels), Source: [23]

In 2009, a milestone was reached, when a database of about 3.2 million hand-annotated images called ImageNet was published in IEEE, meant for the use of image classification, object recognition and automatic object clustering, and since 2012, there were ImageNet classification contests. [20] In 2012, AlexNet was presented as a winner of the ImageNet challenge, with 17% error rate. [29] It has a quite simple architecture, consisting of 5 convolutional, 3 pooling and 3 Dense layers, utilising ReLU output function in convolutional layers. The same architecture was published under the name ZFNet in 2013, pointing out the high impact of tuning hyperparameters. It won the contest ImageNet Large Scale Visual Recognition Challenge 2013, dropping the error to 11 %. [33]

One of the 2014's ImageNet contestants was a much deeper (11, 13, 16 and even 19 layers) Visual Geometry Group Network (VGGNet), presented in ICLR 2015 conference. [34] Another contestant and the winner was GoogLeNet – an Inception network. Google came up with its own complex Neural Network design, introducing a combination of repeating Inception modules (Figure 5). There were two versions of inception modules, a naïve version and with dimension reduction. Another difference from other networks was also the omission of densely connected layers. [35]



(a) Inception module, naïve version



(b) Inception module with dimension reductions

Figure 5 - Inception modules presented in GoogLeNet [35]

Very deep sequential neural networks encounter the Vanishing Gradient problem, because, when training using gradient-based learning algorithms, the weights update according to the partial derivate of error function, and if the derivate is too small, the weights practically don't update and there's hardly any training going on (the weights stay on similar level to the zero or random initialisation). [36] One of the first networks to try solving this problem were Residual Networks (ResNet), winning the ImageNet challenge in 2015, published in 2016 IEEE Conference on Computer Vision and Pattern Recognition. [37] Residual Networks can be called an updated version of VGGNet [34], combining residual blocks. In residual blocks, there is a skip connection – called identity connections – to the end of the block instead of just connecting to the subsequent layer. This helps the model leave out the layers, which cause a Vanishing Gradient problem.

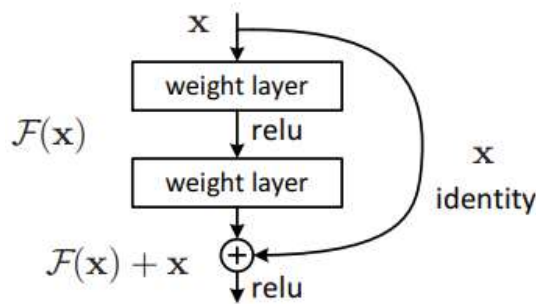


Figure 6 - Residual block [37]

In 2016, an upgraded version of GoogLeNet [35] was proposed by Francois Chollet (author of Keras) and published in IEEE in 2017, it was called Xception [38], merging the ideas of GoogLeNet and ResNet, replacing the inception blocks with depth-wise separable convolutional layers. The difference between standard convolutional layers and depth-wise separable convolutional layers is, that standard layers use convolutional filters to capture both spatial (such as line, edge, oval) and cross-channel patterns (combinations like ear, nose, mouth, creating a face) at once, while separable layers model each pattern category separately.

A year later, in 2018, SENet was created by Jie Hu et al. [39]. It consists of Squeeze-and-Excitation (SE) blocks, Inception and Residual units. SENets, at the cost of computational complexity, increase the state of art performance of CNN on different tasks and datasets. Each SE block is, in fact, a small CNN, which analyses the output of the unit to which it is attached, not looking for spatial patterns, but focusing on the depth dimension (each block has 3 layers – Global average pooling layer, Dense layer with ReLU activation and Dense layer with Sigmoid activation). Then, it recalibrates the feature maps (reduces the irrelevant ones and boosts the relevant ones) in accordance with a recalibration vector.

In 2019, EfficientNet was published by Tan et al. [40]. They presented the concept of scaling up MobileNets and ResNets. The compound scaling method further improved the accuracy on ImageNet by up to 2.5 % with fewer parameters.

2.2.5 Classification on mobile devices

With the development of smart phones, tablets and other handheld computers used in the industry such as in autonomous driving cars or autopilots, which have limited

computational capacity, arose the need to create networks with a lower number of parameters while keeping the target accuracy. These networks are called lightweight [41] networks.

SqueezeNet, limiting the size of the AlexNet model from about 240 MB to less than 7 MB was presented in 2016 by Iandola et al. [42] To achieve a small number of parameters in the CNN, they replaced some commonly used 3x3 convolution filters with 1x1 filters, limiting the parameters 9 times, while keeping the same number of filters. They also decreased the number of input channels to 3x3 using the Squeeze layers and used delayed subsampling (pooling). By delaying the subsampling to later layers while limiting the parameters with decreasing the filter size, they kept the accuracy high. In the same paper, they presented Fire modules, utilising Squeeze convolutional layers with 1x1 filters followed by an Expand layer with 1x1 and 3x3 convolutional filters. They presented three types of architectures of the final SqueezeNet, the simplest architecture consists of 1 Convolutional layer at the start, 8 Fire modules, 1 Convolutional layer at the end followed by AveragePooling and a Dense layer with softmax activation.

The MobileNet family was presented in 2017 by Howard et al. [43]. It was developed, as the name suggest, for the use in mobile phones as well as in embedded vision applications, aiming to reach a high speed while keeping the model size small. Howard et al. suggests using two new hyperparameters (width multiplier and resolution multiplier), that trade-off between accuracy and latency of the models. The architecture consists of Depthwise Separable Convolutional layers, which are a form of factorized standard convolutions, Batch normalisation layers, ReLU and a 1x1 pointwise Convolutional layers. MobileNet has slightly lower accuracy on standard ImageNet dataset than VGG16 [34] or GoogLeNet (Inception) [35], but is very fast.

In 2018, ShuffleNet was published by Zhang et al [44]. They grouped feature maps and performed the convolutions on groups, reducing the computational cost. It is called ShuffleNet as it shuffles the channels for each feature map group, to distribute information across channels, consists of a Convolutional layer, 3 Shuffle blocks, global pooling and Dense layer. The channel shuffle is differentiable, which means it can be used in end-to-end training. The network was 18 times faster than SqueezeNet.

More techniques to reduce the number of parameters were presented in a network family called ShiftNet by Wu et al., built from shift-based modules [45]. The main goal was

to limit the number of floating-point operations (FLOPs). They came up with a FLOP-free shift operation with zero parameters as an alternative to depth-wise 3x3 convolutions. While keeping the same accuracy as SqueezeNet, they limited the model size to about two thirds.

FE-Net introduced by Chen et al. [46] limits the shift operations presented in ShiftNet to only a few feature maps, in Sparse Shift Layers (SSL). The team claimed, that ShuffleNet and MobileNet are inefficient in practice because they occupy about 80 % of GPU runtime, which mismatches the theoretical FLOPs, and suggested a solution to it by using blocks of 1x1 Convolutional layers with a limited number of shifts, reached by penalising the shift operation during optimisation with the use of quantisation-aware shift learning method. They claim and prove that not all shift operations are necessary and slow down the computation a lot. While leaving out some of the shifts, the accuracy drops a little, but the speed is increased significantly.

In 2020, and re-printed in Springer in 2022, EfficientNet-eLite was presented by Wang et al. [47], based on Tan's EfficientNet [40]. They introduced Network Candidate Search (NCS), which measures the different models' resource usage and performance and suggests downscaling the EfficientNet. They reached an even lower number of parameters and a slightly higher accuracy in their new network.

2.2.6 Latest research

The latest research on CNNs is, on top of building architectures (still utilising suggested blocks from previous research), increasing speed, limiting model size and tuning hyperparameters, focused mainly on different ways to find and classify objects within an image, to identify relevant regions of interests and to find the best approach to creating the most fitting bounding boxes and segmentation, which opens a whole another chapter of computer vision tasks.

2.3 Computer vision tasks

Computer vision utilises CNNs and aims to make computer systems perceivable to the visual world by recognising the meaning of pixels (i.e., objects) in pictures. [41] There are three main tasks of computer vision, which can be represented by questions about the objects in a visual scene:

- Classification (what)
- Object detection (where)
- Scene understanding – segmentation (how)

In the early days of computer vision, manual feature extraction was combined with classic machine learning techniques. However, this has prevented computers from recognising more complex objects that have many shapes and colours (cats, dogs, ...). Nowadays, in some tasks, AI techniques are better performing than humans, as automatic feature extraction is incorporated in convolutional layers of CNNs, and manual feature extraction only serves to reduce the dimensionality of input data or is used in different tasks, which belong to unsupervised learning, such as clustering, autoencoders or bag-of-words technique in Natural Language Processing (NLP).

2.4 Classification

Classification is a process of computing a probability vector of each class, already described in [Chapter 2.2.3](#). The training datasets consist of tuples (*image, class*). In classification networks, the number of neurons in the last Dense layer corresponds with the number of object classes, and we use the softmax function as activation and the error function cross-entropy. (10) (11) In the 1990s, with the rise of Artificial Neural Networks, kernel methods began to emerge. [48] In 1995, Vapnik and Cortes published the Support Vector Machine (SVM) classification method. [49] Revolution in approach to classification has been reached since ImageNet contests, launching the rapid development of different neural network architectures, tuning hyperparameters and creating special networks for individual tasks. Nowadays, the ImageNet database contains over 14 million images.

2.4.1 Determining dataset size

The fundamental part of each practical research is data in suitable quality and quantity. As stated in Cho Junghwan's research about classification accuracy of Computer Tomography (CT) scans of different body parts [50], to get an approximately 97.25 % classification accuracy, we need about 1000 images per class, or to get 99.5 % accuracy, it is more than 4000, as visible in Figure 7. The dataset size of course also depends on the complexity of classified data – the more complex data, the more we need.

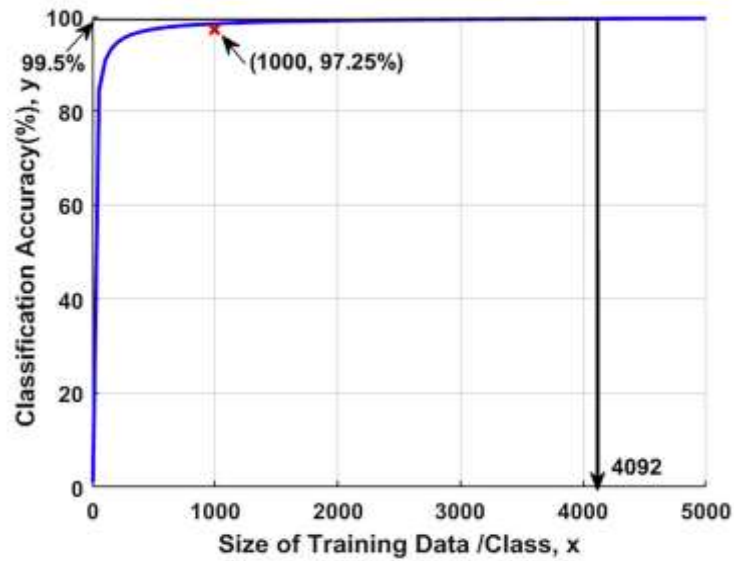


Figure 7 - Number of data needed per class for a high classification accuracy [50]

2.4.2 Classifier success measures

The generalisation capability, i.e., the ability to correctly classify new inputs that did not appear in the training set, can be measured by standard metrics for each classifier, both binary and multi-class.

2.4.2.1 Confusion matrix

These measures are calculated from four basic values, written in a *confusion matrix*. A binary confusion matrix of a binary classifier, deciding, whether the image contains the object of interest or not, consists of:

- true positives TP , which are values classified as *true* and are *true* in real
- true negatives TN , which are values classified as *false* and are *false* in real
- false positives FP , which are values classified as *true* and are *false* in real
- false negatives FN , which are values classified as *false* and are *true* in real

	Predicted Class		
Real class		True	False
	True	TP	FN
	False	FP	TN

Table 2 - Binary confusion matrix

A multi-class confusion matrix for 4 disjunctive classes is following:

actual / predicted	class 1	class 2	class 3	class 4
class 1	<i>TP</i> for class 1	<i>FN</i> for class 1, <i>FP</i> for class 2	<i>FN</i> for class 1, <i>FP</i> for class 3	<i>FN</i> for class 1, <i>FP</i> for class 4
class 2	<i>FN</i> for class 2, <i>FP</i> for class 1	<i>TP</i> for class 2	<i>FN</i> for class 2, <i>FP</i> for class 3	<i>FN</i> for class 2, <i>FP</i> for class 4
class 3	<i>FN</i> for class 3, <i>FP</i> for class 1	<i>FN</i> for class 3, <i>FP</i> for class 2	<i>TP</i> for class 3	<i>FN</i> for class 3, <i>FP</i> for class 4
class 4	<i>FN</i> for class 4, <i>FP</i> for class 1	<i>FN</i> for class 4, <i>FP</i> for class 2	<i>FN</i> for class 4, <i>FP</i> for class 3	<i>TP</i> for class 4

Table 3 - 4-class confusion matrix

The values of TP_i , FN_i and FP_i can be obtained from the multi-class confusion matrix.

FN_i is obtained as the sum of the FN_i values for the corresponding class in all columns except the element on the diagonal of the confusion matrix. Thus, for class 1, it is the sum of the values of the 2nd, 3rd, and 4th columns.

FP_i is obtained as the sum of the columns of the corresponding class, again excluding the element on the diagonal.

TP_i is then the corresponding element on the diagonal.

TN_i is mathematically expressible as

$$TN_i = N_i - (TP_i + FP_i + FN_i) \quad (16)$$

where N_i = the number of elements of the i -th category

2.4.2.2 Measures derived from confusion matrix

There are multiple measures derived from the confusion matrix, namely correctness (accuracy) c , error e , precision p , recall (sensitivity) r , F-measure F . In case of binary classification, they are calculated as follows.

$$c = \frac{TP+}{N} \quad (17)$$

$$e = 1 - c \quad (18)$$

$$p = \frac{TP}{TP+FP} \quad (19)$$

$$r = \frac{TP}{TP+FN} \quad (20)$$

$$F = \frac{2 \cdot (r \cdot p)}{r+p} \quad (21)$$

where:

$$N = TP + FN + FP + TN \quad (22)$$

If $s = p$, then $F = s = p$

In case of disjunctive multi-class classification, for each category $i = 1, \dots, C$, where C = number of categories, a binary decision is made, whether the object belongs to the category i or it belongs to any other category $j \neq i$. Then, the evaluating measures are counted as follows and the overall parameters of precision p , recall r and F -score F are their arithmetic means.

$$c = \frac{\sum_{i=1}^C TP_i + \sum_{i=1}^C TN_i}{N} \quad (23)$$

$$p_i = \frac{TP_i}{TP_i + FP_i} \quad (24)$$

$$r_i = \frac{TP_i}{TP_i + F_i} \quad (25)$$

$$F_i = \frac{2 \cdot (r_i \cdot p_i)}{r_i + p_i} \quad (26)$$

2.5 Object Detection

Object detection [51, pp. 483-488] is a combination of two tasks – *classification* and *localisation*. Localisation can be described as a regression task, predicting bounding boxes around the desired object in different formats. The dataset then consists of tuples in form of *(image, (class, bounding_box))*. In deep learning projects using object detection, the hardest and most time and resource-costly problem is getting the labels, as it has to be hand-made.

At the start, object detection was done by a sliding window detection. The original approach was to train a binary classifier, deciding, whether the object of interest is in the given window of cells in the grid, to which the image was divided. Just like in the calculation of convolutions, the window shifted from left to right, top to bottom. For each combination of cells in a given grid, it calculated whether there was the object of interest in that window, along with its probability. When this window shifted by one step, the classifier detected some of the cells more than once. When the probabilities were counted for a small window, the window size increased and slid again across all regions. Due to running through the CNN many times, this approach is very slow. It also needs post-processing, as the same objects are detected multiple times. A common post-processing technique, deleting bounding boxes, that are overlapping each other, keeping only the one with the highest presence of the object – “objectness”, is called *non-max suppression*.

In 2014, Fully Convolutional Networks (FCN) were presented by Long et al. [52]. They transferred at that time contemporary architectures – AlexNet [29], VGG Net [34] and GoogLeNet [35] to classification and pixel-to-pixel classification (i.e., segmentation) and also presented a new architecture. By replacing dense layers with convolutional layers, the image only has to be processed once, significantly speeding up the object detection. One of the generally used architectures utilising this principle is YOLO (You Only Look Once) [53], which is described in more detail in the chapter [One stage detection algorithms](#).

2.5.1 Traditional image processing techniques vs Deep Learning

A study was published by HCL Tech [54] about the comparison of Deep Learning and traditional image processing (TIP) techniques. TIP like scale-invariant feature transform (SIFT) [55], Histograms of oriented gradients (HOG) [56] and other algorithms usually reach lower accuracy in all computer vision areas – classification, object detection and segmentation, and require more fine-tuning and expert analysis. These methods are based on block-wise orientation histograms. The traditional techniques are more domain-specific and less flexible, however, they are still relevant in some cases. They do not need large datasets, a high computing power, the annotation time shortens significantly, they have a high domain expertise and algorithm transparency. However, feature engineering is required. For some applications like 3D modelling, noise reduction, image registration and data compression, traditional models are still suitable. According to Boesch [57], traditional image processing in OpenCV don't require annotated images. On the contrary, deep learning caused, that in some cases, machine perform better than humans. In comparison with Deep Learning methods, TIP have been recently used sparsely in contemporary image processing research.

2.5.2 Bounding box formats

There are many different object detection algorithms, which accept various types of bounding boxes. The most commonly used are PASCAL VOC (.xml), COCO (.json), Tensorflow Object Detection (.csv) and YOLO DarkNet (.txt). [58] Each of these types describe the boxes in another way.

2.5.2.1 PASCAL VOC bounding boxes

The name comes from an abbreviation of “Pattern Analysis, Statistical Modelling and Computational Learning Visual Object Challenge”. PASCAL VOC bounding boxes are saved as a .xml file and are presented as a set of absolute coordinates x_{\min} , x_{\max} , y_{\min} , y_{\max} closed in <bndbox> tags. Each coordinate set is related to one object with a class name and there is one .xml file for each image in the dataset, which contains all bounding boxes in the given image.

2.5.2.2 COCO Json bounding boxes

COCO Json is named after the commonly used dataset COCO - Common Objects in Context, which contains approximately 328000 images in 91 classes, with about 2.5 million labelled objects. There is one .json file for the whole dataset, which includes the list of categories, image IDs, object categories and the bounding boxes are saved as an array of four absolute numerical values in “bbox”: [x -coordinate of the upper left corner, y -coordinate of the upper left corner, *width* of object, *height* of object].

2.5.2.3 Tensorflow Object Detection bounding boxes

When using Tensorflow Object Detection API, a TF Record file is needed to train the detector network. It has a binary, human non-readable format, therefore it is generated using a script from a .csv file. Each .csv file contains a table describing the whole dataset – one row per image – with the following columns: filename, width, height, class name, x_{min} , x_{max} , y_{min} , y_{max} . The coordinates are also absolute values.

2.5.2.4 YOLO DarkNet bounding boxes

YOLO Darknet bounding boxes work with relative values of coordinates (values of the centre of the object and dimensions of the object are normalised between 0 and 1) and are saved in a .txt file. For each image in a dataset, there is one .txt file, containing one bounding box on each line. The values are separated with blank spaces and come in this order: *class_index* x_{centre} y_{centre} *width* *height*. As images are resized to a square of fixed size, working with relative coordinates has many advantages. When detecting an object and saving labels in YOLO format, it is possible to retrieve information from the original, non-resized image.

2.5.3 Latest Object detection algorithms

Algorithms implementing object detection are based on two approaches. [59] Two-stage detection and one-stage detection algorithms. Generally, it can be said, that one-stage detectors are faster and structurally simpler, and two-stage detectors have a higher recognition and localisation accuracy but are harder to implement. Therefore, each of them is suitable for different practical applications.

2.5.3.1 Two-Stage Detection

These object detection algorithms are divided in two stages. The first stage is *predicting candidate bounding boxes*, using traditional Computer Vision methods or Deep Learning, while the second one is *classification with bounding box regression*. It means, that two-stage detectors first find a Region of Interest (RoI), then crop the image and classify the cropped image. Because the cropping operation is non-differentiable, such detectors are usually not end-to-end trainable (all parameters of the model cannot be simultaneously trained for one loss function). There are various algorithms using different approach to this two-stage detecting, the most popular are Regions with CNN features (R-CNN), Fast R-CNN, Faster R-CNN, Spatial Pyramid Pooling Network (SPPNet), Feature Pyramid Networks (FPN), Detecto, combining previous approaches, and Gated Recurrent Convolutional Neural Network (G-RCNN).

2.5.3.1.1 R-CNN, Fast R-CNN, Faster R-CNN

One of the prominent two-stage detectors is the R-CNN Family. All versions of R-CNN use PASCAL VOC bounding boxes

In 2014, the first version of R-CNN was presented by Girshick et al. [60]. Due to large receptive fields and strides, Girshick decided not to use sliding-window technique for object localisation and introduced recognition using regions instead. R-CNN consists of three steps – region proposals, feature extraction and classification. Approximately 2000 regions, which are category independent, are proposed during prediction, and by using CNN, a fixed-length feature vector from each proposal is extracted and after that, each region is classified with a SVM, which is category specific.

As the training of R-CNN is a multi-stage pipeline, which is expensive in memory and time and the detection is quite slow, a year later, Girshick published an improved version, 213 times faster at testing and 9 times faster at training, called Fast R-CNN [61]. He introduced multi-task loss and a single-stage multi-task training, which saved time and space, as it left out the need of feature caching. He also replaced a SVM classifier with a softmax CNN classifier, which also increased the speed and performance.

In 2017, Ren et al. followed Girshick's research and introduced Faster R-CNN [62]. He focused on speeding up the region proposal step, as in Fast R-CNN, it consumed same or even more time than the detection network. They proposed an end-to-end trainable Region Proposal Network (RPN), which shared layers with the object detection networks and significantly speeded up the testing time, getting close to real time with 10 ms per image. They also presented, at that time novel, anchor boxes, they served as a reference when using multiple aspect ratios and scales of images.

2.5.3.1.2 Spatial Pyramid Pooling Network (SPPNet)

Spatial Pyramid Pooling Network – the SPPNet – comes from the very author of Detectron, Res-Net and Mask R-CNN – Kaiming He – et al. [63]. The Spatial Pyramid Pooling method does not require a fixed-size input that is normally reached by resizing or changing the original image – cropping or warping (a combination of cropping and stretching into a square). Making such manipulations unnecessary is a convenience when using different sizes, scales and aspect ratios of images in datasets. The reason for using fixed-size images is that the fully connected (Dense) layers require a fixed-length input by definition.

As convolutional and pooling layers work on a sliding-window basis and can work with any size and shape of input, the authors put a newly introduced SPP layer, which creates a fixed-size output from a variable-size input, on top of the last convolutional layer in the network before the classifiers (SVM or softmax - Dense layers).

Getting a fixed-size vector from a variable-size input can be reached by using two approaches – the first one is a vector space model Bag-of-Words (BoW) [64], which maps the visual features and their number occurrences, making it a fixed-size vector (i.e.: {eyes: 8, legs: 8, torso: 1, ears: 0, nose: 0, claws:2, tail: 0} – a simplified example of animal features, in this case, representing a spider; in real use, the features are extracted automatically by the CNN). The other option is using a BoW improvement – Spatial Pyramid Pooling, which maintains spatial information by pooling in local spatial bins, regardless of the image size.

SPPNet had good results on ImageNet's contest ILSVRC 2014 and was ranked #2 in object detection and #3 in classification.

2.5.3.1.3 Feature Pyramid Network (FPN)

Feature Pyramid Networks (FPN) were published in 2016 [65]. They follow up on the SPPNet, with a significant improvement in generic feature extraction. In combination with R-CNN, it surpassed the object detection results of COCO dataset. This is also one of the flagship networks used for generating segmentation proposals, following the DeepMask [66] framework.

2.5.3.1.4 Detecto – ResNet, R-CNN and FPN

Detecto [67] is Python framework released in 2019, which combines Faster R-CNN, ResNet-50 and FPN. As a callable Python package, it is easy to implement on custom data, with a very high precision in comparison to other object detection algorithms. It requires bounding boxes in PASCAL VOC format and uses PyTorch instead of the usual TensorFlow.

2.5.3.1.5 Gated Recurrent Convolutional Neural Network G-RCNN

Wang et al. released G-RCNN in 2022 [68]. The principle stands in introducing gates controlling the amount of information on input of a recurrent CNN. They follow-up on the research of CNN with Adaptive Receptive Field, which works with deformable convolutions, and combine it with skip connections introduced in ResNet [37] and recurrent connections between neurons in the same layer. These networks are used for object recognition (classification), but also scene text recognition (OCR) and object detection.

2.5.3.2 One-Stage Detection

One-stage detection algorithms only predict bounding boxes and leave out predicting RoI. Because of that, it leverages anchors and a grid box to localise the object and constraint its shape. The three most popular one-stage detectors are YOLO, SSD and RetinaNet. The disadvantage of such detectors is a fixed-size input.

2.5.3.2.1 YOLO

YOLO is a famous family of networks with a controversial history, which belongs to convolutional networks designed to classify and detect objects in images. The first version of YOLO was proposed by Redmond et al. in 2015 [53] as YOLOv1, improved in 2016 [69] as YOLOv2 and subsequently in 2018 [70] as YOLOv3, then he left the research due to potential misuse and various teams came up with further versions. At the time (as in the

beginning of 2023), 9 versions are available. Older versions of YOLO use TFRecords to load datasets, and later versions (YOLOv5+) require YOLO bounding boxes

At its release time, YOLOv1 [53] was revolutionary, because it was so fast it could run in real time (over 20 FPS) in a video; it exceeded the speed of Fast R-CNN with a 57.2% mAP (mean Average Precision) on VOC 2007 dataset, however, when comparing the prediction on one image, there was a significant number of localisation errors. Redmond and his team's approach differed from other detectors, as they framed object detection as a regression problem with regards to a grid of cells instead of using a sliding window classifier (which was a common approach before 2014). However, this approach also comes with its downsides – all images are resized to 448 x 448 pixels, then a single convolutional network is run to detect object with regards to the grid, and in order to avoid multiple detections of one object and remove overlapping bounding boxes, non-max suppression is applied. However, each grid cell can only detect two object and only one class (making it 98 objects in total), therefore, YOLOv1 struggles with images containing larger groups of small objects (i.e., flocks of birds). YOLOv1's architecture consists of 24 Convolutional layers, 4 MaxPooling layers and 2 Dense layers. The whole prediction is encoded as a tensor $S \times S \times (B * 5 + C)$, where S = height and width of the image, B = number of bounding boxes, C = number of classes.

YOLOv2 [69], or YOLO9000, named after the capability of detecting over 9000 classes, reached 76.8 % mAP on VOC 2007 (containing 20 classes) in 67 FPS. With 156 classes, the mAP drops to 16, dropping even more when more classes are used. In comparison with YOLOv1, several changes were made. The input image size shrunk from 448 x 448 to 416 x 416 and there was a change in the architecture – the base of YOLOv2 is a newly proposed Darknet-19, which has 19 Convolutional layers and 5 MaxPooling layers, using 3x3 convolutions as well as 1x1 in order to compress the feature representations. By removing the Dense layers, YOLOv2 had to introduce dimension clusters as anchor boxes to predict bounding boxes instead of a static cell grid. For each anchor box, the class and objectness is predicted. Although using anchor boxes allows more objects to be detected (over a thousand within one image), the prediction accuracy lowered a bit. An average recall of public datasets (like VOC, COCO or ImageNet) reached about 81 %, which is enough for analysing a video, for example a surveillance camera, but not enough for precise detection

(like in OCR). In the same paper, the system called WordTree is introduced, which is used to unite the labels from different sources (COCO and ImageNet at once).

In an unusually informal technical report presenting version 3 [70], Joseph Redmond and his teacher Ali Farhadi presented more improvements in YOLO. Darknet-19 was replaced by Darknet-53 and detecting small objects improved, however, detecting large objects deteriorated. The author (who himself was funded by Google and the Office of Naval Research) was questioning the use of fast and precise object detectors – as most such research was funded by the military or big corporations like Google and Facebook – Are the detectors going to be used to harvest personal data and sell it to other subjects? Are they going to be used by the army to train automatic targeting systems in order to kill a lot of people? Redmond then stopped developing YOLO (and computer vision research altogether) in fear of the potential misuse, however, other teams took over.

After 2 years of inactivity on YOLO, much to Redmond's dismay, it was taken over by Alexey Bochkovskiy et al. in 2020 and YOLOv4 was published [71]. Some of the changes are the input image size, which has grown to 512 x 512, anchor optimisation, mosaic data augmentation (proposed by Glen Jocher), cross mini-batch normalisation, dynamic mini-batch size, IoU threshold, class label smoothing, and training tuning – genetic algorithms were used in order to select the optimal hyperparameters during learning and different loss algorithms were used for bounding box regression. These changes caused an improvement both in speed and in accuracy.

Also in 2020, Glen Jocher released YOLOv5 [72] with some differences from YOLOv4, like automatic learning of bounding box anchors and an almost 10 times lower model size, making it a lightweight model suitable for mobile real-time applications, however, only as a code (which has been continually improved ever since), and by now (as of 2023), the paper was not presented. Many people had a problem with Jocher naming his algorithm YOLOv5, as it was supposedly not novel enough and he is not the original author of YOLO. He has made alliance with Joseph Nelso, the CEO of Roboflow [58], which is an online platform for annotation, augmentation and also training different deep learning projects, and since 2022. YOLOv5 [72] can be also used for segmentation [73], if trained on images with polygon annotations instead of standard rectangular bounding boxes.

YOLOR (You Only Lear One Representation) followed the YOLO line in 2021 [74]. Published by Wang et al., they proposed a network, which integrates both types of knowledge - implicit (which has nothing to do with the observation – subconscious learning) and explicit (directly corresponding with the observation – conscious learning). Such network is capable of general representation as well as sub-representations for various tasks, as it is a multiple-output network with a single input. Some of the tasks are questions like what the object is, where it is, what colour it has etc.

Along with YOLOR, YOLOX surpassing all previous YOLO versions was published in the same year by Ge et al. from Megvii Technology. [75] In YOLOX, there are no anchors, as in order to get optimal anchors, cluster analysis has to be performed before training, and the objective was to speed up the process, the predictions look for a grid top, object width and height. YOLOX has several versions with different sizes. One version is based on Darknet53, then there is a L-version, Tiny and Nano with only 0.91 M parameters. Also an advanced assignment of labels called SimOTA is presented in this paper; it calculates pair-wise matching degree, which is represented by quality or cost for each “prediction – grid top” – pair, then it selects best predictions and assigns the grids of positive predictions as positive and the rest as negatives.

In 2022, YOLOv7 was published by YOLOR’s authors Wang et al. [76]. It does not have a single specific architecture, but rather, it is more of a structure as it uses a novel model scaling method that scales concatenation-based ELAN and residual-based CSPDarknet models. In order to assign labels dynamically, they used a set of methods that were bag-of-freebies to improve the model's accuracy and speed.

YOLOv6 was released a few months after YOLOv7 by Li et al. [77], even more improving the accuracy and speed by using four main techniques – presenting different scales of models, a self-distillation strategy on classification and regression, a broad verification of advanced detection techniques and reforming the quantisation scheme using a RepOptimizer and a channel-wise distillation. YOLOv8 [78] was also released on GitHub by Ultralytics at the beginning of 2023 (without the paper yet). YOLOv8 is able to work with bigger square images in real-time, by default, 640 x 640 pixels with almost 54 mAP. YOLOv9, developed by Chien-Yao Wang et al., followed in February 2024. [79] They built a new YOLO on a proposed Generalized Efficient Layer Aggregation Network (GELAN) architecture.

2.5.3.2.2 SSD – Single Shot Detector

Presented by Liu et al. in 2016 [80], the Single Shot MultiBox Detector (SSD) detects objects using only one deep Convolutional Neural Network. It creates default bounding boxes and generates scores for the presence of each category of objects in each default bounding box. It also adapts the box to match the shape of the given object better. Although the SSD is a relatively simple algorithm, it reaches a sufficient accuracy on small images and outperforms Faster R-CNN in speed (as most one-shot detectors do with two-stage detectors). As Liu criticised the two-stage detectors for being too slow for real-time applications, he suggested SSD with a feed-forward CNN, VGG-16 base, which makes it even faster than YOLO. The loss of the model is counted as a weighted sum between the localisation losses.

2.5.3.2.3 RetinaNet

In 2017, Lin et al. released another one-stage detector, called RetinaNet [81]. They came up with a novel loss, addressing the problem of one-stage detectors with class-imbalanced datasets. It is called the *focal loss*, which extends the cross-entropy loss with a modulating factor $(1 - p_t)^\gamma$ utilising a tuneable focusing parameter $\gamma \geq 0$, with an experimentally reached optimal value of 2, focusing on learning on hard negative examples. The modulating factor extends the range in which a prediction receives a low loss. The architecture of RetinaNet consists of an FPN backbone and a feedforward ResNet, minimising focal loss during training.

2.5.3.2.4 SqueezeDet

SqueezeDet [82] is a Fully Convolutional Network (FCN) that was released by the Shift [46] author team with the aim of use in autonomous driving. Inspired by YOLO but with a smaller (and scalable) model size, the SqueezeDet team adopts a single-stage detection pipeline using anchors. However, they use only convolutional layers not just to extract feature maps but also for a novel output layer called ConvDet, which predicts bounding boxes. The coordinates of bounding boxes are counted as a regression with regard to the relative coordinates of anchors.

2.5.4 Comparing object detection algorithms

2.5.4.1 Evaluation options

In object detection algorithms, we usually evaluate automatically on labelled test data. It is not possible to simply count TP , FN , FP , and TN as it is done in classification. [83]

For this reason, the most common metric to measure how well the model predicts the bounding boxes is Intersection over Union (IoU). It is the area of intersection of real and predicted bounding boxes, divided by their union. The higher the IoU, the more precise the detection is. We can find it in `tf.keras.metrics.MeanIoU` class. [84]

From IoU, we can calculate TP , FN , FP , and TN . We establish an IoU threshold, which is considered sufficient.

If $IoU \geq IoU\ threshold$, we mark the box as TP . All boxes without intersections are marked as FP , and boxes with low IoU are marked as FN . Then, we construct the confusion matrix just like with classification.

In some cases where we don't have the ground truth labels available, it is inconvenient to use this method; therefore, in the comments section, a method is suggested.

2.6 Segmentation

Segmentation is the most complex computer vision task of all three. It can be explained as pixel-level classification. By clustering pixels belonging to a selected class, the algorithm shows, where the exact object boundaries are located. There are two types of segmentation: semantic segmentation, which clusters all pixels of objects belonging to the same class, even if they overlap, and *instance segmentation*, which identifies each object instance separately, even if they belong to the same class. Instance segmentation decides the outlines of an instance in accordance with its shape, texture, brightness, and colour. [85]

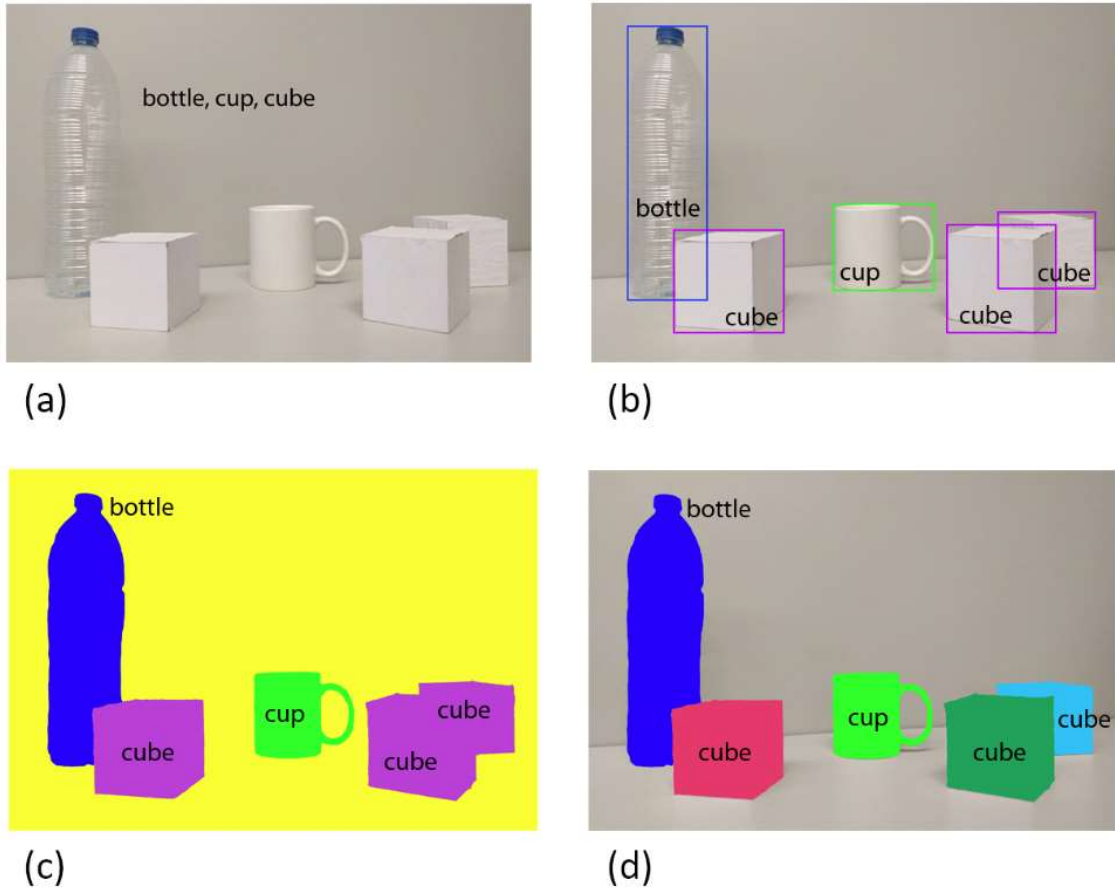


Figure 8- Computer vision tasks: (a) image classification, (b) object detection, (c) semantic segmentation (d) instance segmentation. [86]

2.6.1 Mathematical methods used for segmentation

There are two main strategies in image segmentation:

- similarity – this approach is based on thresholding, comparing the similarity of neighbouring pixels
- discontinuity – this method incorporates algorithms of line, point and edge detection

Various mathematical techniques are available to segment the content of an image; however, due to the focus of this dissertation (neural networks), these methods are not described in detail.

Methods such as graph cuts, pseudo-Boolean programming, and fast optimisation [87] are used to create a Markov Random Field (MRF) model estimation, which are undirected graphs.

Another one is constructing a Bayesian network (multiple variants of Directed Acyclic Graphs – DAGs). Baxter et al. [88] described a Directed Acyclic Graph Max-Flow (DAGMF) image segmentation, capable of segmenting a wide variety of input images from different areas. This approach orders labels into a set of continuous spaces, marking each pixel in the input image with a category.

2.6.2 Object detection algorithms used for segmentation

2.6.2.1 Evolution of semantic segmentation

In 2017, Garcia-Garcia et al. presented a review of semantic segmentation done with deep learning techniques [86]. In semantic segmentation, each pixel in the picture is classified. Different objects of the same class are marked the same. Regardless of whether there is one or more objects of the same class, they are not distinguished.

Instance segmentation is, however, more challenging, as it requires precisely segmenting each instance while correctly detecting all objects. It combines object detection and semantic segmentation within each bounding box separately.

2.6.2.2 Early approaches of instance segmentation

The origin of segmentation was a previously mentioned Fully Convolutional Network by Long et al. [52] Many instance segmentation papers are based on segment proposals. Earlier proposals from around 2013 were based on bottom-up segments [89] [90], DeepMask by Chen et al. from 2016 [66] and research following it by Pinheiro et al. [91] is based on Fast R-CNN and segment candidate proposals, where the segmentation precedes object recognition, making it less accurate, however, at that time, it improved the state of art in the recall by 10 – 20 %. In 2015, Dai et al. [92] suggested a complex model, which consists of a sequence (or, as they call it, cascade) of three convolutional models. The first one differentiates instances, the second one estimates masks, and the last one is a classifier. It is a sequence and is not predicted parallelly. As already described in Chapter 2.5.3.1. [Two-Stage Detection](#), Feature Pyramid Networks (FPN) were one of the flagship object detectors with automatically generated mask proposals.

The most contemporary instance segmentation models are currently based on Mask R-CNN and, since 2022, also on YOLOv5, YOLOv7 and since 2023, on YOLOv8 and YOLOv9 in 2024.

2.6.2.3 Mask R-CNN

Mask R-CNN was proposed in 2017 by He et al. [93]. They are an extension of Fast R-CNN [62], which added an object mask in parallel with the bounding boxes – when detecting objects, Mask R-CNN also generates segmentation masks for each instance. In comparison to Dai’s cascade approach [92], it is much faster. The original code [94] is called Detectron. The authors presented a new type of layer, *RoIAlign*, which preserves the exact spatial locations of detected objects. In comparison to previously published *RoiPool* layers, the mask accuracy increased by 10 to 50 %. In this approach, each class is predicted independently, and a binary mask in the form of a polygonal label is created for each object class within every Region of Interest.



Figure 9 - Masks created with polygonal labels for training a Mask R-CNN network

source: <https://bit.ly/3wTxzK4>

In 2017, Waleed et al. [95] proposed an improved Mask R-CNN based on a Feature Pyramid Network (FPN) and a ResNet101 backbone. In comparison to the original Mask R-CNN, where square inputs were needed, they preserved the aspect ratio of images, generated the bounding boxes in a different way and decreased the learning rate from 0.02, as in combination with small batch sizes, causing exploding weights.

In the state of the art around 2022, Mask R-CNN could be used for segmenting objects like aeroplanes, cars, animals, and people for tracking; however, the precision of masks was not very big, as visible in the two Figures below. It made Mask R-CNN unusable by itself (without pre- or post-processing) for applications like Optical Character Recognition (OCR).

However, in 2023, the precision has risen greatly, which opened the possibility of applying this technique to use instant segmentation for experimental letter recognition.

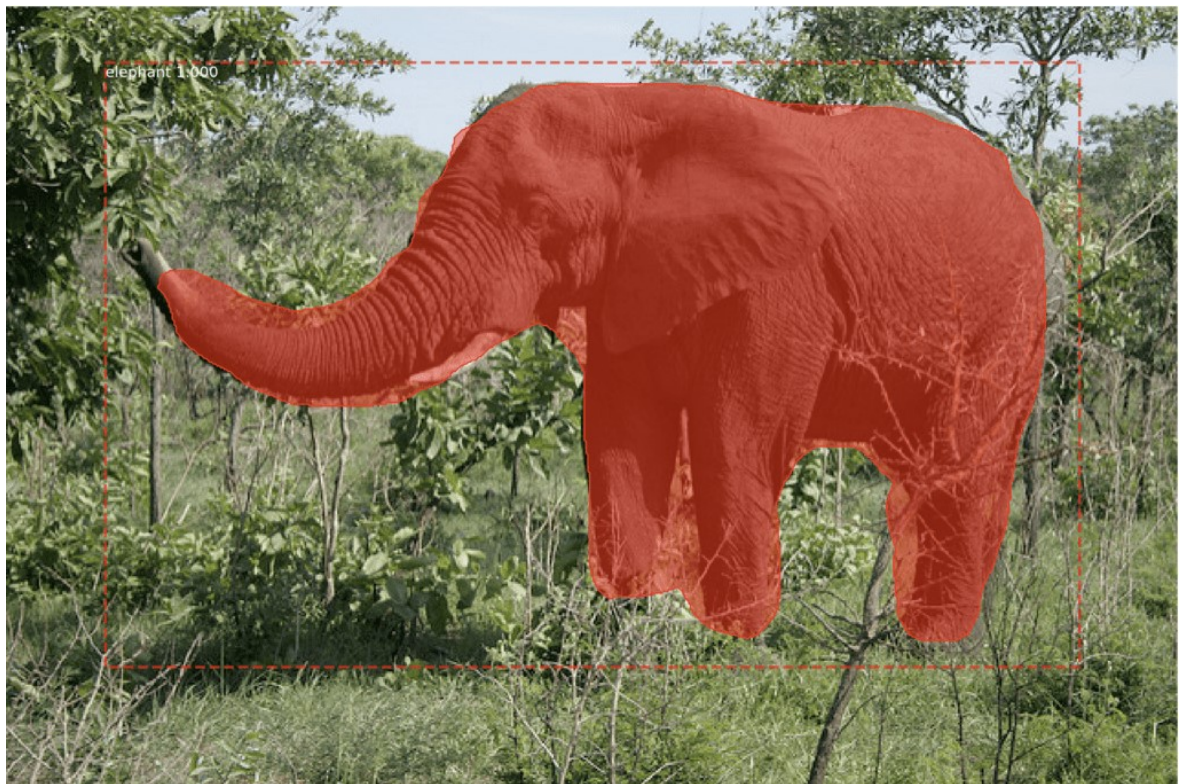


Figure 10 - Elephant predicted with Mask R-CNN [96]



Figure 11 – Houses segmented with Mask R-CNN: Mapping Challenge converting satellite images to maps [95]

2.6.2.4 YOLO

Since 2022, as described on the Roboflow blog [97], YOLOv5 has also been usable for instance segmentation. It uses the same polygonal labels as Mask R-CNN for generating object outlines. In 2023 and 2024, more future YOLO versions followed this trend, further improving the quality of predicted segmentation masks.

2.7 Dataset augmentation

In order to conduct research, and not just image processing research, a sufficient amount of good quality data is needed. Machine learning projects usually require a high amount of training data to be reliable. [50] [54] While working with public datasets, which usually consist of a large number of images, makes the process easier, there are not always the data that we need.

When training deep learning models on self-made, custom-created datasets for our own research, we often face the challenge of obtaining data of sufficient quality and scope. In some cases, only a limited amount of data is available, and the annotation process (or, for classification, the process of cutting images to square inputs containing only the object of interest) is also time-consuming. Under these circumstances, along with optimising network architecture and training parameters and incorporating regularisation techniques (for example, adding dropout layers to the model), it is appropriate to consider including synthetic data that is built upon a foundation of real data and transformed using mathematical or other techniques.

We distinguish between classical image filtering methods, geometric transformations (both of which only manipulate the original images) and machine learning methods that create custom data as close to real data as possible.

2.7.1 Classical methods

2.7.1.1 Keras generator

There is a class called `ImageDataGenerator` (callable by `tf.keras.preprocessing.image.ImageDataGenerator`) in the Keras framework, which allows the researchers to use several filtering methods and geometric transformations to augment the

input image dataset. Filtering methods include changes in saturation, colour depth, colour spectrum, contrast, brightness, focus, blur, and noise inserted into the original images. On the other hand, geometric transformations are rotating, flipping, stretching, cropping, and narrowing. Keras also allows for the replacement of parts of images with blank pixels or noise and the compiling of multiple images in a tile, which consists of random crops.

2.7.1.2 Online platform Roboflow

In January 2020, an online platform called Roboflow was released on the domain roboflow.com. This platform includes many features useful for machine learning projects, such as creating project datasets in clouds and callable from notebooks (Jupyter, Google Colab, Kaggle, etc.). Supported project types are object detection, single-label classification, multi-label classification, instance segmentation, semantic segmentation, keypoint detection and “other”. For object detection and segmentation, an annotation tool is available, supporting both types of labels – bounding boxes and polygonal labels. In the premium version, Roboflow also offers training DL models on “Roboflow train”. When exporting datasets for use in external notebooks, the user can choose the dataset format (from the ones explained in detail in [Chapter 2.5.2 – Bounding box formats](#)) and also pick pre-processing and augmentation methods. The pre-processing options offered are visible in the following Figure:



Figure 12 - Roboflow pre-processing options

source: www.roboflow.com

Augmentation options include image level augmentations and bounding box level augmentations, which are shown in the Figure below. These augmentations, just like the ones in the Keras generator, are classified as filtering and geometric methods.

IMAGE LEVEL AUGMENTATIONS



BOUNDING BOX LEVEL AUGMENTATIONS ?

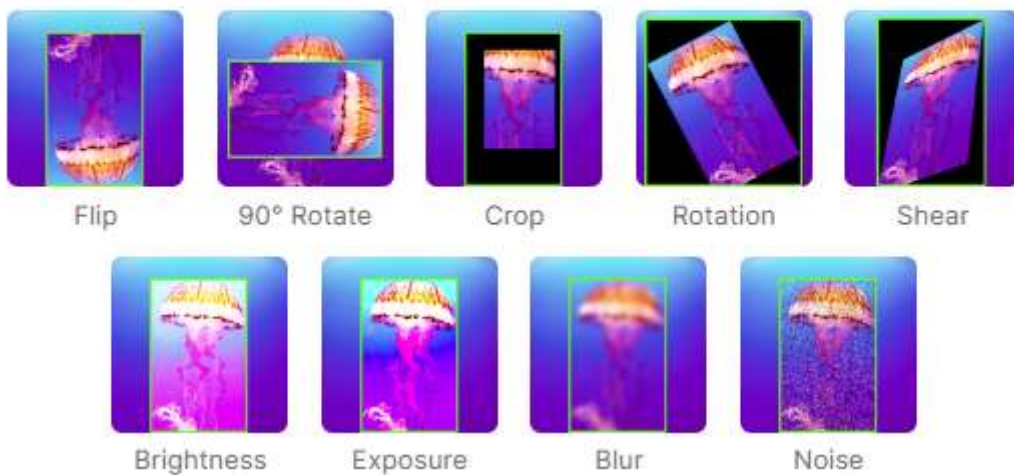


Figure 13 - Roboflow augmentation options

source: www.roboflow.com

2.7.2 Deep learning methods

In addition to classical filtering and geometric methods, there are also augmentation techniques using artificial intelligence to increase the dataset size. Using these methods, new, original data are created that take on the features of the original images. Such methods are Neural style transfer, Feature space data augmentation, such as Variational autoencoders (VAE) and more, Generative adversarial networks (GANs) and others (Deep Dream etc.).

2.7.2.1 Neural Style Transfer

Using the Neural Style Transfer (NST) method, we work with the terms image content and *style*. NST was presented by Leon Gatys et al. in 2015. [98] Image content is defined in higher CNN layers as a high-level structure of the image (mainly rough outlines defining the main features) and the style defined in lower CNN layers consists of colours, textures, and visual patterns. This method is incorrectly, but commonly called “filters” in mobile phone cameras. Simply put, if you take a photo of a face and make it look like it was painted in Leonardo da Vinci style or a landscape that you change to look like a painted canvas, this is NST. This method has been used in the latest research as well. Daru et al. [99] presented a method of designing drapes with a combination of various binary masks and NST. The part of the image of a drape where the used mask was black had a different target style than the part of the image where the mask was white. Wu et al. [100] proposed a Direction-aware NST with a custom direction field loss function, improving the state of art of generating mosaic and canvas-like images. Xinyu et al. [101] extended the use of NST to stereoscopic images in 2018, which was enhanced by Friedrich et al. [102] in 2021 by creating a pipeline based on a high-resolution voxel representation with the goal of creating complete 3D shapes with a transferred neural style.

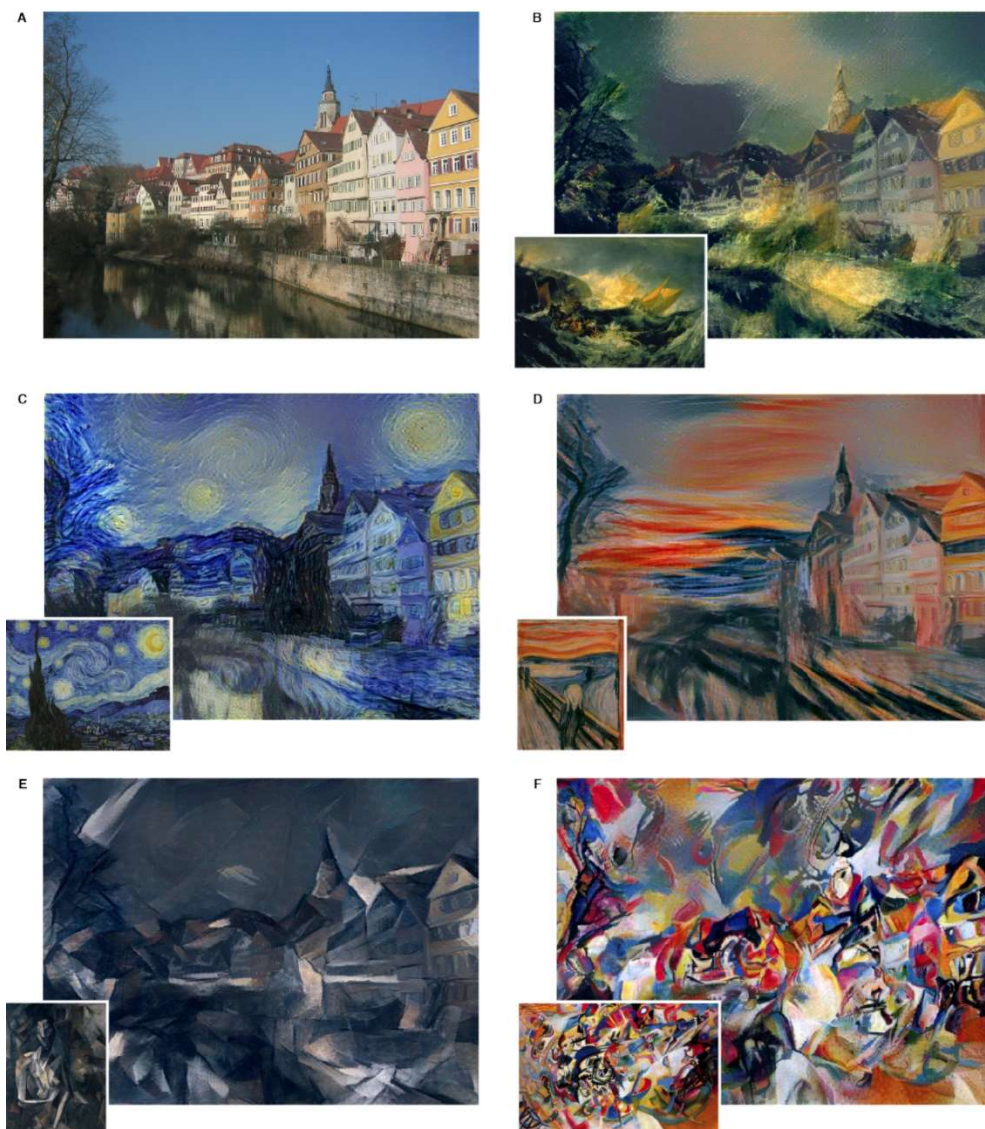


Figure 14 - Neural Style Transfer by Leon Gatys [98]

2.7.2.2 Feature space data augmentation

Feature space data augmentation (FDA) is a group of methods used to improve classifier performance. Kumar et al. [103] described different FDA methods as follows. These techniques first extract the features from the original datasets, and then they generate new data from the latent feature space. After this process, the synthetic data is added to training sets in order to improve the classification accuracy. One of the FDA's methods is upsampling, which creates higher-resolution pictures by applying features learned on smaller images. Random perturbation adds noise from the uniform distribution. Linear delta is a simple method of generating new examples by subtracting the difference between two examples from the same class and combining it with a third example. Extrapolation between

samples in latent feature space creates new samples as well, and delta-encoder, similarly to variational autoencoder, consists of an encoder and decoder; at first, it extracts the differences between two samples within a class (deltas); then, it applies these deltas to create synthetic data from a different class.

2.7.2.3 Variational autoencoders

Variational autoencoders (VAE) were explored by two independent teams and published at about the same time by Kingma et al. [104] and Rezende et al. [105]. Chollet [48, p. 271] describes the key idea to generating pictures as a low-dimensional vector latent space of representations, where any point can be mapped to a realistically looking image.

VAE is a composition of two neural networks. One of them, *the encoder*, accepts real images as input and encodes them into a compressed representation using ANN, usually CNN. This compressed representation consists of two parameters in the latent space – z_mean and z_log_var .

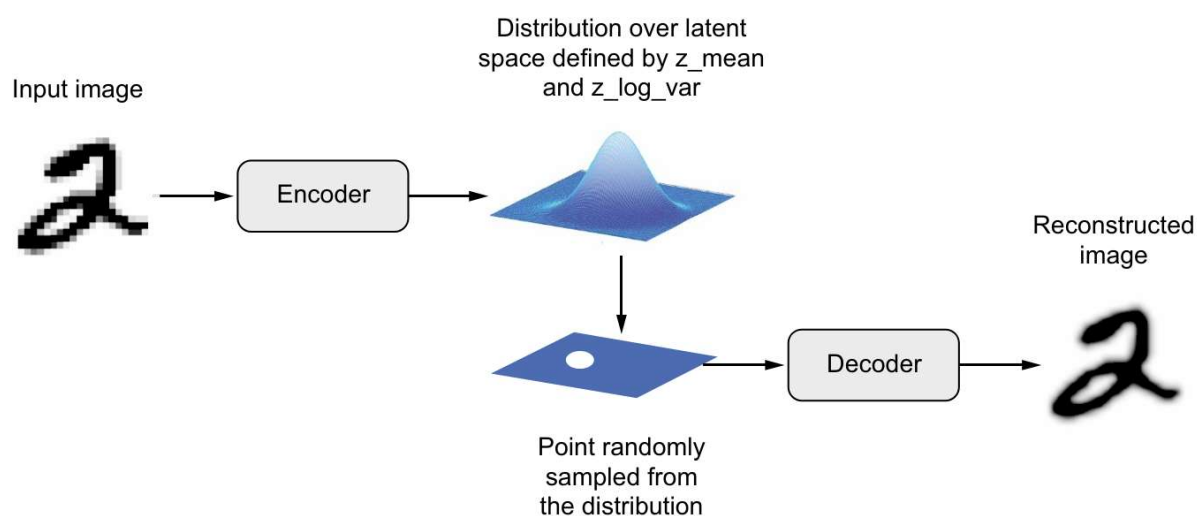


Figure 15 - Principle of VAE [48, p. 274]

The randomly sampled point, as shown in the figure above, is picked using this equation.

$$z = z_{mean} + \exp(z_{log_var}) * \epsilon \quad (27)$$

where ϵ = a random tensor of small values.

The *decoder* network maps this z point back to the original image. The points, which are near each other in the latent space, will generate a very similar output using the decoder.

The representations in the latent space are highly structured and can be used for smooth transitions of images.

VAEs have been used in contemporary research. Chadebec et al. [106] increased accuracy from 80.7 % to 88.6 % by incorporating synthetic data into classification datasets in the OASIS database. Elbattah et al. [107] used VAE to generate eye-tracking scan paths with the goal of reducing dataset imbalance.

2.7.2.4 Generative adversarial networks

Generative Adversarial Networks (GANs), first presented by Ian Goodfellow et al. [108] also consist of two neural networks. One of them is called the *generator*, while the second one is the *discriminator* (real-fake classifier). Both networks are trained alternately, each to an adversarial goal. The generator captures the distribution of data and tries to generate real-like input, and the discriminator estimates the probability of the input to be real (it comes from the training set) or fake (it was generated). At first, the generator generates Gaussian noise, which is easily identified as fake by the discriminator, but as the training goes on, the generated images resemble real images more and more. In an ideal situation, a Nash equilibrium is reached. It is a state in which no player can unilaterally improve his strategy to beat the opponent. In the context of a GAN, this means that the generator generates so well that the discriminator has exactly a 50% chance of identifying the input correctly.

GANs, like the VAEs, consist of two neural networks. However, unlike VAE, they do not create structured, continuous latent spaces. GANs have the potential to produce highly realistic images, whereas the images from the VAE are merged, because of the continuity in the latent space. Géron [51, p. 591] states that although VAEs have been very popular for a long, GANs have already surpassed them with their capability to create more realistic images. It is, therefore, advisable to consider what data we are augmenting before choosing the right method.

Contemporary research has introduced improvements to basic GANs. Deep Convolutional GAN – DCGAN [109] presented the following improvements in the architecture and guidelines for a more stable network:

- Replacing pooling layers with strided convolutions in the discriminator and fractional convolutions in the generator
- Using batch normalisation
- In deeper architectures, removing fully connected hidden layers
- Using ReLu activation in all layers of the generator, except for the last layer with Tanh
- Using LeakyReLu in all layers of the discriminator

Progressively growing GAN by Karras et al. [110] present a novel approach to train low-resolution images at first and progressively increase them to high resolution, usually up to 1024 x 1024 px. In their paper, the discriminator network is called “the critic”. The principle of the upscale lies in the smooth adding of layers. The layers operating in higher resolutions are treated like a residual block, in which layers have small weights linearly increasing from 0 to 1. Ledig et al.’s Super-resolution GAN (SRGAN) [111] is focused on creating highly detailed textures when upscaling the generated images.

Recent use of GANs in contemporary research projects includes augmenting datasets of palmprint [112], improving cancer classification on gene expressions data [113], augmenting X-ray security images for threat detection [114], improving liver lesion classification with synthetic data [115] and many more.

2.8 Optical Character Recognition

Optical Character Recognition (OCR) systems are among the most complex applications of computer vision and image recognition, in addition to the scene understanding required for self-driving vehicles, for instance. It is the process of classifying patterns corresponding with alphanumeric or other characters from a digital photograph or PDF scan. [116] This recognition is done in several, at least three, basic steps - segmentation of text against the background and individual parts of the text, feature extraction and feature classification. [117]

Holistic OCR systems contain 9 stages [118], from the actual acquisition of image data to the conversion of text into machine-readable format:

1. Scanning - digitising physical documents (papers, photographing historical artefacts - reliefs, vases, funerary steles, etc. inscriptions)
2. Local segmentation, which distinguishes the text parts of a photograph from graphics
3. Optional pre-processing, in which noise is reduced, text is rotated to the correct position, the image is normalised, compressed
4. Segmentation, also called binarisation - separation of non-text and text parts, and further segmentation into individual characters or parts of characters
5. Representation - global, statistical, or geometric representation of characters
6. Feature extraction - in current OCR systems, template matching is most commonly used based on the extracted distribution of points identified in an image
7. Training and recognition - creating templates of individual features, fixed or elastic, statistical techniques, cluster analysis, use of fuzzy logic or artificial neural network
8. Post-processing - detected features are composed into words compared with a dictionary, and modified to make sense
9. The final step is the output text itself in a machine-readable format, which can be further manipulated, e.g., using NLP - natural language processing systems (e.g., translating into other languages, creating automatic text summaries, etc.)

Current research in OCR [119] combines multiple systems in a hybrid solution for better recognition, which reduces the error rate when using only one OCR algorithm. Post-processing methods are being improved to account for different types of errors and produce automatic corrections [120], and systems are being developed for languages that were previously unavailable, such as Arabic [121], Hindi [122], Japanese [123], and many more. In a comparative study of current OCR systems (here, regarding Latin OCR) from 2021 [124], the problems encountered by these systems in common situations are mentioned. One of them is the problem of recognising letters on a scanned document, in case the letters are distorted, blurred or parts of the characters are missing completely. Another problem is font differences, which create the need to extract multiple different features for each character class or to use multiple templates and elastic templates for character classification.

At the time, a common way to classify a character in OCR is using elastic templates. Contemporary research uses neural networks for text segmentation, like in case of car license plate reading [125], where Mask R-CNN serves to segment the text area, and it is forwarded to Terrasect OCR for transliteration. Only one pioneer research called “Rosetta” [126] proposes a convolutional approach because it uses neural networks for both text localisation and text recognition. Rosetta is used to analyse text in so-called “memes”, which are uploaded to the social media Facebook every day.

2.9 Research gap

The exploration of neural network architectures used for object recognition in images, image segmentation and augmentation of learning sets, applied to recognising historical alphabets is best done with real-world examples. It is not possible to define one correct approach to analyse historical scripts, as there are many types (alphabets, logographic scripts, ideographic scripts, syllabaries and many more) and each of them is unique and needs to be approached respectively. Based on the knowledge described in the literature review, several research gaps were identified. In this dissertation, an effort is made to obtain a general approach for developing improved neural networks for the purpose of OCR algorithms of historical scripts specific to each type of script.

At the time of writing this dissertation, there were no detectors and classifiers for the characters of the Palmyrene alphabet. Therefore, it was possible to build a custom dataset of the characters used to write Palmyrene Aramaic, explore classifier architectures and optimise them, try different augmentation methods, construct a Generative adversarial network and train it to augment the data, as well as start to work on a complete OCR system, incorporating segmentation, and lastly, connect it with a dictionary. Completion of Palmyrene OCR, namely the dictionary part, will be finished after publishing this dissertation, but it’s appropriate to mention it now, as my team and I are already working on it at the time.

In addition to analysing the Palmyrene alphabet, together with a team from Israel, we decided to explore the possibilities of detecting wedges (otherwise known as strokes) in cuneiform using only object detection algorithms, as this approach was not used before. As cuneiform is a logo-syllabic script, in this case, not the whole vowels, but their parts were

being detected. In the course of this research, I was able to construct different object detection algorithms and test their properties.

During the cuneiform research, we encountered the problem of calculating the confusion matrix for detectors in the absence of ground truth labels. For this reason, I propose a modification of it and present it here in chapter 3.2.1.

3 Commentary

In the Attachment section, five papers are presented about the topic of creating Palmyrene OCR and two papers about detecting strokes in cuneiform script. One article is mentioned but not yet published; therefore, it is not attached as full text.

3.1 Palmyrene Aramaic analysis using computer vision algorithms

In the first, conference article [1] (Attachment 1), the Android application incorporating two classifiers was shown. The first classifier takes an EMNIST-like Palmyrene character, which is drawn on the mobile screen; optionally, the second classifier takes a photo of one Palmyrene character (which needs to be taken by the mobile phone camera). The respective classifier suggests the top three classes with the highest confidence scores; the user picks the character in accordance with the Unicode table character pictured next to the confidence score and saves it to buffer. This way, the user can re-write (or take pictures letter by letter) and classify every letter in the Palmyrene inscription and can then export it as text. The architecture used in this version of the app was `efficient_lite0`. The results of hand-written classification reached 80.2% *F*-score, while the photographic classifier reached only 71.96%. There were three authors in this paper. I created the dataset and the classifier and conducted tests. Mr. Franc created and debugged the mobile application, and Mr. Tyrychtr provided professional guidance while writing the article.

The second, journal, article [2] (Attachment 2) presents a custom CNN architecture, which was obtained by testing different layer combinations and training parameters. An optimal architecture for the classification of hand-written inputs was selected, and the accuracy increased from `efficient_lite0`'s 68.93% to 98,21%. the hand-written Palmyrene character classification task was complete. However, it was not suitable for classifying photos of characters. I created the training scripts in Keras and conducted the tests of architectures

and classification results tests. Mr. Franc updated the mobile application, and Mr. Veselý helped with the methodology and article review.

In the third, conference, article about the topic of Palmyrene OCR [3] (Attachment 3), we present GAN augmentation methods used for expanding photographic datasets. However, as the improvement was not sufficient for practical use, therefore, we explored more options, that are presented in a journal article, for which we have recently received reviews and are editing it in accordance with the reviewer's comments Mr. Franc created the GAN training and generating scripts, I did the data hand-sorting and classifier testing. Mr. Veselý suggested methods and reviewed the article.

In 2022, we extended our research by presenting the Palmyrene OCR web application available at <https://ml-research.pef.czu.cz>, which, at the time of creation offered the same options as the mobile application - to either draw characters by hand or alternately take a picture of one character in Android application, annotating characters in an uploaded photo of Palmyrene inscription. Mr. Svojše developed the web application, I provided extended datasets, classifiers trained on them and conducted tests, and Mr. Franc ensured the operation of the web server. The article related to this topic has not yet been published and, therefore, cannot be referenced, but the web application is already in use.

In the conference article [6] (Attachment 4) published by Springer, the planned capabilities of the Palmyrene OCR web application were described using a diagram, which was, along with a part of a single-class segmentation dataset and methods for further steps, presented in the MOBA workshop of CAISE conference. The dataset was obtained and annotated by me as well as the research plan, and Mr. Pavlíček edited the diagram. Mr. Franc helped with the manuscript preparation. I prepared the methods.

In the end, the methods proposed in the conference articles were supplemented by an additional method, which was multi-class instance segmentation. With the help of new scientific knowledge and experiments, we have found that it is not necessary first to detect individual letters and then categorise them into the correct classes, as was the original plan, but that it is possible to segment and categorise at the same time. In fact, during 2023, the segmentation algorithms were further improved, and YOLOv8 could be trained for multi-class instance segmentation. In a 2024 paper published in CMES [7] (Attachment 5), we performed a thorough analysis that involved a substantial expansion of the dataset, with input

from humanities professor Alexey Lyavdansky of the Russian HSE University, who, for photographs of the Palmyra inscriptions I collected from public and non-public sources, verified which letters were visible in the photographs and provided transcriptions for inscriptions that were not publicly available. We trained four models - for one and multiple classes - on two algorithms and developed a custom algorithm for line processing and letter-in-line comparison directly from the detected segmentation masks in the form of polygons. Mr. Svojše and Mr. Franc also published this module to our web application ml-research. Mr. Novák helped with manuscript formatting and administration, and Mr. Veselý helped with manuscript preparation and methodology description.

Another article, which is also part of the research, is not attached to this dissertation as it is in preparation. In this article, various GAN and SRGAN possibilities and training methodologies are explored, presenting the optimisation of GANs for classification data augmentation and upscaling images on specially downsampled images using the Hierarchical Collaborative Downscaling (HCD) method published in 2023 [127]. The outcome of this study suggests that synthetic data can play a crucial role in enhancing the performance of computer vision algorithms, with potential applications extending beyond the scope of the study to other areas of research and development in computer science and digital humanities. I helped prepare the data and train multiple GAN networks constructed by Mr. Franc. Ondřej Svojše published the relevant content on the ml-research web application, and Mr. Veselý provided methodological guidance and helped with manuscript preparation.

3.1.1 Palmyrene alphabet research outcome

Over the course of the past few years, computer vision algorithms have experienced rapid development, enabling a wide variety of applications to be created. One such application is a novel way of developing OCR, which we have researched for a number of years. Through ongoing experiments and using the most contemporary algorithms in each phase of the research, we have achieved results that allow us to read the characters of the Palmyrene alphabet directly from photos and obtain the transcriptions in either a mobile or web application. A foreign humanities professor with the ability to read and understand the Palmyrene texts has joined our team and is invaluablely helping us with the process. One last step for this research to be complete is to incorporate the datasets in standard OCR algorithms like Google Tesseract and create an NLP algorithm to put the predicted characters in the context of words and sentences.

3.2 Cuneiform stroke detection

Concerning cuneiform stroke detection, two papers are attached to this dissertation. The first one, a conference article [4] (Attachment 6) presents detecting horizontal strokes within images of cuneiform tablets and a journal article [5] (Attachment 7) presents comparing different computer vision algorithms and also some additional tools that we created to detect the strokes successfully. The dataset was processed by Mr. Franc, who also prepared the Detecto algorithm for training in a custom notebook and annotated by partners from two Israeli universities – Shai Gordin from Ariel University and Avital Romach from Tel Aviv University. They also helped us with the texts about cuneiform in both articles. In this research project, I served as a project manager and was responsible for the preparation and training of the YOLOv5 network, programming several utilities, and Mr Čejka participated in training the R-CNN network. Mr. Pavlíček also participated by programming more utilities.

The final outcome of the cuneiform research project was not deployed in any web application and is currently only available on GitHub. The topic was further explored by E Stötzner et al. [128], adding all other types of strokes to the detection, allowing the research to be put into practical use.

According to Google Scholar on the 18th of April 2024 [129], the SPML conference article [5] has been cited 3 times by other author teams (E Stötzner et al., V Yugay et al. and A Bucciero et al.) who followed the development of detecting cuneiform signs directly from 2D photographs.

3.2.1 Confusion matrix used for object detection algorithms

During the evaluation of the cuneiform stroke object detectors, we encountered a problem with the standard algorithm evaluating methodology – counting mean IoU and deriving the confusion matrix from it [83] – as some of the testing data was not labelled. There was a need to establish a method that could be used in detector evaluation cases where few or no ground truth labels are available. Therefore, an altered version of the confusion matrix was suggested. Confusion matrix is traditionally used to evaluate classifiers or used on an object detection algorithm, if it is derived from IoU as described in chapter 2.5.4.1 of this dissertation.

The method of altered confusion matrix construction was applied in the 2024 article in Digital Humanities Quarterly [6].

3.2.1.1 Suggested method to constructing confusion matrix

In case of missing IoU, True Positives TP , False Negatives FN and False Positives FP are counted as follows, and a duplicity or multiplicity D is added:

- The object of interest or its part is bounded by a box.
 - It is denoted as TP .
- The object of interest or its part is not bounded by a box.
 - It is denoted as FN .
- Multiple boxes bound the same object or its part.
 - The number of boxes $b-1$ are denoted as D .
- A box bounds an area, where there is no object of interest or its part.
 - It is denoted as FP .
- TN cannot be counted; therefore, it is set to 0.
 - $TN = 0$.

Standard metrics such as precision p , recall s and F -score can be calculated using the matrix. The number of multiplicities D can be ignored in calculations as bounding boxes with a high overlap can be removed when a threshold for overlap is set. However, it can serve as additional information when evaluating the algorithm.

3.3 Cascade-style approach to creating historical OCR systems

In Chapter 8, a classical 9-step holistic approach to creating OCR algorithms, which consists of scanning, local segmentation, pre-processing, binarisation, representation, feature extraction, training and recognition, post-processing, and finally, getting the text in a machine-readable format, was presented. However, by applying the latest knowledge in the field of computer vision and using previously unavailable or imprecise algorithms, which have achieved much higher success rates in recent years, the number of steps to create an OCR algorithm can be significantly reduced. Although there are different approaches of reading historical scripts, there are many connecting elements that can be summarised in a cascade-style approach. The 9-step process can be simplified to a 5-step process.

3.3.1 Data acquisition

The quality and quantity of the data acquired directly impact the algorithm's ability to generalise and accurately recognise text across different fonts, sizes, and styles which are

present when processing handwriting of historical documents. Different digitising processes can obtain data, but the easiest one is collecting existing photographs of texts or taking new pictures.

3.3.2 Annotation

With the use of manual annotation of whole letters in case of alphabets, or parts of characters used to write historical language, such as cuneiform, we prepare a dataset for training an instance segmentation algorithm that incorporates the previously mentioned steps - local segmentation and pre-processing. An optional step in this step includes using GAN networks to generate more training data if too little is available.

3.3.3 Multi-class instance segmentation

Multi-class instance segmentation sums up the binarisation (segmenting text and non-text areas), representation, feature extraction, training, and recognition from the holistic approach in just one step, as it directly detects non-text and text parts and recognises whole letters or components at once and results in a list of letter.

3.3.4 Post-processing

This step involves techniques such as sorting characters into rows and columns and combining parts of characters into syllables in non-alphabetic scripts.

Advanced post-processing can also be understood as spell-checking, grammar correction, and context analysis to improve the overall accuracy and readability of the extracted text. Additionally, post-processing may also include error correction mechanisms to prevent any misinterpretations or inconsistencies in the recognised text.

3.3.5 Text in machine-readable format

Converting the recognised text into a machine-readable format is the final step in the OCR pipeline. This step requires working with a dictionary in the given language and enables future text processing such as translation or summarisation.

3.4 Next steps

Future research plans are presented. In order to complete the development of the OCR algorithm in Palmyrene Aramaic, advanced post-processing techniques need to be applied to avoid recognition errors, and a dictionary needs to be developed to understand the transcripts

obtained by image processing fully using instance segmentation. Of course, we are also continuously working on annotating more available images with Palmyrene inscriptions to refine the segmentation results as more accurate results will make post-processing easier.

Other research teams have already followed up on the processing of cuneiform script from 2D photographs, yet if our Israeli partners are interested in collaborating on the next steps, we would be happy to participate in the research with them.

Conclusion

The beautiful aspect of science is that there is always something to improve and develop, and new knowledge allows us to educate ourselves and educate others continually. Over the course of my PhD grant projects, I have researched a large number of neural network types and architectures that are used for image processing, such as classification, object detection, dataset augmentation and segmentation. I explored their capabilities and limitations in relation to the detection, augmentation, and classification of historical fonts, both alphabet and logo-syllabic.

Together with several research teams, I have developed a horizontal stroke detector in cuneiform script, a classifier, a segmentation algorithm, and a generator of Palmyrene alphabet characters and proposed a modified method for computing the confusion matrix when no ground truth labels are available to calculate IoU.

I suggested and published a novel cascade approach to OCR creation using Convolutional Neural Networks. It both simplifies and combines the steps of a holistic OCR development approach using state-of-the-art image processing algorithms. This approach is generally applicable to the development of algorithms processing various other scripts, be they historical or contemporary.

I have gained knowledge in computer vision that I will further enhance in other research or commercial projects that I am working on now and will work on in the future. I trust that the results presented in this dissertation will help as a baseline for research by other teams working on the development of processing historical scripts.

4 References

- [1] A. Hamplová, D. Franc and J. Tyrychtr, “Historical Alphabet Transliteration Software Using Computer Vision Classification Approach,” *Lecture Notes in Networks and Systems*, pp. 34-45, 26 04 2022.
- [2] A. Hamplová, D. Franc and A. Veselý, “An improved classifier and transliterator of hand-written Palmyrene letters to Latin,” *Neural Network World*, vol. 32, pp. 181-195, 30 08 2022.
- [3] D. Franc, A. Hamplová and O. Svojše, “Augmenting Historical Alphabet Datasets Using Generative Adversarial Networks,” *Data Science and Algorithms in Systems. CoMeSySo 2022. Lecture Notes in Networks and Systems*, vol. 597, pp. 132-141, 04 01 2023.
- [4] A. Hamplová, D. Franc, J. Pavlíček, A. Romach and S. Gordin, “Cuneiform Reading Using Computer Vision Algorithms,” *SPML 2022: Proceedings of the 2022 5th International Conference on Signal Processing and Machine Learning*, 2022.
- [5] A. Hamplová, A. Romach, J. Pavlíček, A. Veselý, M. Čejka, D. Franc and S. Gordin, “Cuneiform Stroke Recognition and Vectorization in 2D Images,” *Digital Humanities Quarterly*, vol. 18, no. 1, 2024.
- [6] A. Hamplová, D. Franc and J. Pavlíček, “Character Segmentation in the Development of Palmyrene Aramaic OCR,” in *Model-Driven Organizational and Business Agility, Lecture Notes in Business Information Processing (LNBIP, volume 488)*, Zaragoza, 2023.
- [7] A. Hamplová, A. Lyavdansky, T. Novák, O. Svojše, D. Franc and A. Veselý, “Instance Segmentation Of Characters Recognized In Palmyrene Aramaic Inscriptions,” *Computer Modelling in Engineering & Sciences*, 2024.
- [8] N. Muthukrishnan, F. Maleki, K. Ovens, C. Reinhold, B. Forghani and R. Forghani, “Brief History of Artificial Intelligence,” *NEUROIMAGING CLINICS OF NORTH AMERICA* , vol. 4, pp. 393-+, 03 11 2020.
- [9] W. McCulloch and W. Pitts, “A LOGICAL CALCULUS OF THE IDEAS IMMANENT IN NERVOUS ACTIVITY,” *Bulletin of Mathematical Biophysics*, pp. 115-133, 1943.
- [10] D. O. Hebb, *The organization of behavior: A neuropsychological theory*, New York: John Wiley and Sons, Inc., 1949.
- [11] M. Minsky, “A Neural-Analogue Calculator Based upon a Probability Model of Reinforcement,” Harvard University Psychological Laboratories, Cambridge, Massachusetts, 1952.
- [12] F. Rosenblatt, “The perceptron - a perceiving and recognizing automaton,” Cornell Aeronautical Laboratory, 1957.
- [13] J. C. Hay, B. Lynch, D. Russell and B. Smith, “Mark I Perceptron Operators' Manual,” 1960.
- [14] B. Widrow and M. E. Hoff, “Adaptive switching circuits,” in *Proceedings of I.R.E Wescon Convention Record*, 1960.
- [15] B. Widrow, “Thinking About Thinking: The Discovery of the LMS Algorithm,” *IEEE SIGNAL PROCESSING MAGAZINE*, pp. 100-106, 2005.

- [16] B. Widrow and K. Steinbuch, "A critical comparison of two kinds of adaptive classification networks," *IEEE Transactions on Electronic Computers*, p. 737–740, 1965.
- [17] J. Hopfield, "Neural Networks and Physical Systems with Emergent Collective Computational Abilities," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 8, pp. 2554-8, 14 1982.
- [18] P. J. Werbos, "Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Science," *Thesis (Ph. D.)*, 01 1974.
- [19] "Neural Network World," ČVUT, [Online]. Available: <http://www.nnw.cz/>.
- [20] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *IEEE Conference on Computer Vision and Pattern Recognition*, Miami, 2009.
- [21] A. Apicella, F. Donnarumma, F. Isgro and R. Prevete, "A survey on modern trainable activation functions," *Neural Networks*, vol. 138, pp. 14-32, 19 05 2021.
- [22] A. Veselý, *Metody umělé inteligence*, 1 ed., Praha: Česká zemědělská univerzita v Praze, 2012.
- [23] Y. LeCun, L. Bottou, Y. Bengio and P. Haffner, "GradientBased Learning Applied to Document Recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278-2324, 11 1998.
- [24] F. Chollet, A. Gulli, S. Pal, A. Geron, R. Dattaraj, A. Kapoor and J. Moolayil, "Keras: the Python deep learning API," [Online]. Available: <https://keras.io/>. [Accessed 11 10 2022].
- [25] Chollet, Francois, "Keras Layers API," [Online]. Available: <https://keras.io/api/layers/>. [Accessed 12 10 2022].
- [26] Y. Jing and Y. Guanci, "Modified Convolutional Neural Network Based on Dropout and the Stochastic Gradient Descent Optimizer," *Algorithms*, vol. 11, no. 18, 03 2018.
- [27] Chollet, Francois, "Layer activation functions," Keras, [Online]. Available: <https://keras.io/api/layers/activations/>. [Accessed 12 10 2022].
- [28] K. Fukushima, "Neocognitron: A Self-organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position," *Biological Cybernetics*, vol. 36, pp. 193-202, 04 1980.
- [29] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *NIPS'12: Proceedings of the 25th International Conference on Neural Information Processing Systems*, vol. 1, pp. 1097-1105, 03 12 2012.
- [30] K. He, X. Zhang, S. Ren and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification," *2015 IEEE International Conference on Computer Vision (ICCV)*, pp. 1026-1034, 02 2015.
- [31] H. Yingge, I. Ali and K.-Y. Lee, "Deep Neural Networks on Chip - A Survey," *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pp. 589-59, 02 2020.
- [32] P. Sadowski, "Notes on Backpropagation," University of California Irvine, Irvine, CA 92697.

- [33] O. Russakovsky, J. Deng, J. Krause, A. Berg and F.-F. Li, "ImageNet Large Scale Visual Recognition Challenge 2013 (ILSVRC2013)," Stanford University, 2013. [Online]. Available: <https://image-net.org/challenges/LSVRC/2013/>.
- [34] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," *Proceedings of 3rd International Conference on Learning Representations ICLR 2015*, 10 04 2015.
- [35] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich, "Going deeper with convolutions," *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1-9, 2015.
- [36] S. Basodi, C. Ji, H. Zhang and Y. Pan, "Gradient amplification: An efficient way to train deep neural networks," *Big Data Mining and Analytics*, vol. 3, pp. 196-207, 09 2020.
- [37] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770-778, 12 12 2016.
- [38] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1800-1807, 2017.
- [39] J. Hu, L. Shen, S. Albanie, G. Sun and E. Wu, "Squeeze-and-Excitation Networks," *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7132-7141, 2018.
- [40] M. Tan and Q. Le, "EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks," *Proceedings of the 36th International Conference on Machine Learning*, vol. 97, pp. 6105-6114, 2019.
- [41] X. Feng, Y. Jiang, X. Yang, M. Du and X. Li, "Computer vision algorithms and hardware implementations: A survey," in *Integration; Proceedings of 23rd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jeju, South Korea, 2019.
- [42] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size," 24 02 2016.
- [43] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," 17 04 2017.
- [44] X. Zhang, X. Zhou, M. Lin and J. Sun, "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices," *IEEE Xplore*, pp. 6848-6856, 2018.
- [45] B. Wu, A. Wan, X. Yue, P. Jin, S. Zhao, N. Golmant, A. Gholaminejad, J. Gonzales and K. Keutzer, "Shift: A Zero FLOP, Zero Parameter Alternative to Spatial Convolutions," *IEEE Xplore*, pp. 9127-9135, 2018.
- [46] W. Chen, D. Xie, Y. Zhang and S. Pu, "All you need is a few shifts: designing efficient convolutional neural networks for image classification," *Proceedings of Conference on Computer Vision and Pattern Recognition (2019)*, 2019.
- [47] C.-C. Wang, C.-T. Chiu and J.-Y. Chang, "EfficientNet-eLite: Extremely Lightweight and Efficient CNN Models for Edge Devices by Network Candidate Search," *Journal of Signal Processing Systems*, 2022.

- [48] F. Chollet, *Deep Learning with Python*, Second Edition, 2 ed., Manning Publications, 2021, pp. 30-32.
- [49] V. Vapnik and C. Cortes, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273-297, 1995.
- [50] J. Cho, K. Lee, E. Shin, G. Choy and S. Do, "How much data is needed to train a medical image deep learning system to achieve necessary high accuracy?," *Proceedings of ICLR 2016*, 2016.
- [51] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras & TensorFlow*, vol. 7, R. Roumeliotis and N. Tache, Eds., Sebastopol, Canada: O'Reilly Media, Inc., 2020.
- [52] J. Long, E. Shelhamer and T. Darrell, Fully Convolutional Networks for Semantic Segmentation, arXiv, 2014.
- [53] J. Redmond, S. Divvala, R. Girshick and A. Farhadi, You Only Look Once: Unified, Real-Time Object Detection, IEEE, 2015, pp. 779-788.
- [54] HCL Tech, "Traditional Image Processing vs. Deep Learning," *Imaging & Machine Vision Europe*.
- [55] D. G. Lowe, "Distinctive Image Features," *International Journal of Computer Vision*, pp. 91-110, 11 2004.
- [56] "Histograms of oriented gradients for human detection," *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 1, pp. 886-893, 2005.
- [57] G. Boesch, "Object detection in 2022: The definitive guide - visio.ai," visio.ai, 2022. [Online]. Available: <https://visio.ai/deep-learning/object-detection/>.
- [58] Roboflow, "Object Detection - Roboflow," Roboflow, 2022. [Online]. Available: <https://docs.roboflow.com/adding-data/object-detection>.
- [59] A. Lohia, K. D. Kadam, R. R. Joshi and A. M. Bongale, "Bibliometric Analysis of One-stage and Two-stage Object Detection," *Library Philosophy and Practice (e-journal)*, vol. 4910, 02 2021.
- [60] R. Girshick, J. Donahue, T. Darrell and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *2014 IEEE Conference of Computer Vision and Pattern Recognition*, pp. 580-587, 06 2014.
- [61] R. Girshick, "Fast R-CNN," *arXiv*, pp. 1-9, 27 09 2015.
- [62] S. Ren, K. He, R. Girshick and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *arXiv*, 04 01 2015.
- [63] K. He, X. Zhang, S. Ren and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional," *Lecture Notes in Computer Science book series (LNIP, volume 8691)*, p. 346-361, 18 06 2014.
- [64] Sivic and Zisserman, "Video Google: a text retrieval approach to object matching in videos," *Proceedings Ninth IEEE International Conference on Computer Vision*, vol. 2, pp. 1470-1477, 2003.
- [65] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan and S. Belongie, "Feature Pyramid Networks for Object Detection," *arXiv*, 09 12 2016.
- [66] T. Chen, L. Lin, X. Wu, N. Xiao and X. Luo, "Learning to Segment Object Candidates via Recursive Neural Networks," 04 12 2016.

- [67] A. Bi, “Welcome to Detecto's Documentation! --- Detecto 1.0,” 2019. [Online]. Available: <https://detecto.readthedocs.io/>.
- [68] J. Wang and X. Hu, “Convolutional Neural Networks with Gated Recurrent Connections,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 7, pp. 3421-3435, 01 07 2022.
- [69] J. Redmond and A. Farhadi, YOLO9000: Better, Faster, Stronger, IEEE, 2016, pp. 6517-6525.
- [70] J. Redmond and A. Farhadi, YOLOv3: An Incremental Improvement, arXiv, 2018.
- [71] A. Bochkovskiy, C.-Y. Wang and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” *arXiv*, 23 04 2020.
- [72] G. Jocher, A. Chaurasia, A. Stoken and J. Borovec, ultralytics/yolov5: v6.2 - YOLOv5 Classification Models, Apple M1, Reproducibility, ClearML and Deci.ai integrations, zenodo, 2022.
- [73] P. Guerie and T. Lynn, “How to Train YOLOv5 Instance Segmentation on a Custom Dataset,” 21 09 2022. [Online]. Available: <https://blog.roboflow.com/train-yolov5-instance-segmentation-custom-dataset/>.
- [74] C.-Y. Wang, I.-H. Yeh and H.-Y. M. Liao, “You Only Learn One Representation: Unified Network for Multiple Tasks,” *arXiv*, 10 05 2021.
- [75] Ge, Theng; Liu, Songtao; Wang, Feng; Li, Zeming; Sun, Jian; Megvii Technology, “YOLOX: Exceeding YOLO Series in 2021,” *arXiv*, 06 08 2021.
- [76] C.-Y. Wang, A. Bochkovskiy and H.-Y. Liao, “YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors,” 06 07 2022.
- [77] C. Li, L. Li, H. Jiang, K. Weng, Y. Geng, L. Li, Z. Ke, Q. Li, M. Cheng, W. Nie, Y. Li, B. Zhang, Y. Liang, L. Zhou, X. Xu, X. Chu, X. Wei, X. Wei and Meituan, “YOLOv6: A Single-Stage Object Detection Framework for Industrial Applications,” *arXiv*, 07 09 2022.
- [78] G. Jocher, “YOLOv8 by Ultralytics,” 10 01 2023. [Online]. Available: <https://github.com/ultralytics/ultralytics>.
- [79] C.-Y. Wang, I.-H. Yeh and H.-Y. M. Liao, “YOLOv9: Learning What You Want to Learn Using Programmable Gradient Information,” *ArXiv*, 21 02 2024.
- [80] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu and A. C. Berg, “SSD: Single Shot MultiBox Detector,” *Computer Vision – ECCV 2016. ECCV 2016. Lecture Notes in Computer Science*, vol. 9905, pp. 21-37, 2016.
- [81] T.-Y. Lin, P. Goyal, R. Girshick, K. He and P. Dollár, “Focal Loss for Dense Object Detection,” *2017 IEEE International Conference on Computer Vision*, pp. 2999-3007, 10 2017.
- [82] B. Wu, A. Wan, F. Iandola, P. H. Jin and K. Keutzer, “SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving,” *arXiv*, 11 07 2019.
- [83] J. Szczegieliński, “Comparing Object Detection Models - Objectivity,” Objectivity Ltd., [Online]. Available: <https://www.objectivity.co.uk/blog/comparing-object-detection-models/>.
- [84] TensorFlow, “tf.keras.metrics.MeanIoU,” TensorFlow documentation, 18 11 2022. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/metrics/MeanIoU.

- [85] A. M. Hafiz and G. M. Bhat, "A survey on instance segmentation: state of the art," *International Journal of Multimedia Information Retrieval*, vol. 9, pp. 171-189, 12 05 2020.
- [86] A. Garcia-Garcia, S. Escolano-Orts, S. Oprea, V. Villena-Martinez, P. Martinez-Gonzales and J. Garcia-Rodriguez, "A Review on Deep Learning Techniques Applied to Semantic Segmentation," *Applied Soft Computing*, vol. 70, pp. 41-65, 01 09 2018.
- [87] E. Boros and P. L. Hammer, "Pseudo-Boolean optimization," *Discrete Applied Mathematics*, vol. 123, pp. 155-225, 15 11 2022.
- [88] J. S. H. Baxter, M. Rajchl, A. J. McLeod, J. Yuan and T. M. Peters, "Directed Acyclic Graph Continuous Max-Flow Image Segmentation for Unconstrained Label Orderings," *International Journal of Computer Vision*, vol. 123, pp. 415-434, 2017.
- [89] P. Arbeláez, J. Pont-Tuset, J. Barron, F. Marques and J. Malik, "Multiscale Combinatorial Grouping," *2014 IEEE Conference on Computer Vision and Pattern Recognition, 2014*, pp. 328-335, 2014.
- [90] J. Uijlings, K. van de Sande, T. Gevers and A. Smeulders, "Selective Search for Object Recognition," *International Journal of Computer Vision*, vol. 2, pp. 154-171, 2013.
- [91] P. O. Pinheiro, T.-Y. Lin, R. Collobert and P. Dollár, "Learning to Refine Object Segments," *arXiv*, 29 03 2016.
- [92] J. Dai, K. He and J. Sun, "Instance-aware Semantic Segmentation via Multi-task Network Cascades," *arXiv*, 14 12 2015.
- [93] K. He, G. Gkioxari, P. Dollár and R. Girshick, "Mask R-CNN," *017 IEEE International Conference on Computer Vision (ICCV), 2017*, pp. 2980-2988, 2017.
- [94] K. He, G. Gkioxari, P. Dollár and R. Girshick, "facebookresearch/Detectron," Facebook, 2018. [Online]. Available: <https://github.com/facebookresearch/Detectron>.
- [95] Waleed, Abdulla and C. Clauss, "matterport/Mask_RCNN," *Github Repository*, 2017.
- [96] J. Brownlee, "How to Use Mask R-CNN in Keras for Object Detection in Photographs," *Machine Learning Mastery*, 14 05 2019. [Online]. Available: <https://machinelearningmastery.com/how-to-perform-object-detection-in-photographs-with-mask-r-cnn-in-keras/>.
- [97] P. Guerrie and T. Lynn, "How to Train YOLOv5 Instance Segmentation on a Custom Dataset," *Roboflow blog*, 21 09 2022.
- [98] L. A. Gatys, A. S. Ecker and M. Bethge, "A Neural Algorithm of Artistic Style," *Journal of Vision*, vol. 16, pp. 1-16, 2016.
- [99] P. Daru, S. Gada, M. Chheda and P. Raut, "Neural Style Transfer to Design Drapes," *Proceedings of the 8th IEEE International Conference on Computational Intelligence and Computing Research (IEEE ICCIC)*, pp. 626-631, 2017.
- [100] H. Wu, Z. Sun and W. Yuan, "Direction-aware Neural Style Transfer," *PROCEEDINGS OF THE 2018 ACM MULTIMEDIA CONFERENCE (MM'18)*, pp. 1163-1171, 2018.

- [101] X. Gong, H. Huang, L. Ma, F. Shen, W. Liu and T. Zhang, “Neural Stereoscopic Image Style Transfer,” *Lecture Notes in Computer Science, 15th European Conference on Computer Vision (ECCV)*, vol. 11209, pp. 56-71, 2018.
- [102] T. Friedrich, B. Hammer and S. Menzel, “Voxel-Based Three-Dimensional Neural Style Transfer,” *Lecture Notes in Computer Science, 16th International Work-Conference on Artificial Neural Networks (IWANN)*, vol. 12861, pp. 334-346, 2021.
- [103] V. Kumar, H. Glaude, C. de Lichy and W. Campbell, “A Closer Look At Feature Space Data Augmentation For Few-Shot,” *Proceedings of the 2nd Workshop on Deep Learning Approaches for Low-Resource NLP (DeepLo 2019)*, pp. 1-10, 2019.
- [104] D. P. Kingma and M. Welling, “Auto-Encoding Variational Bayes,” *Proceedings of the ICRL 2014 conference; arXiv*, pp. 1-14, 2013.
- [105] D. J. Rezende, S. Mohamed and D. Wierstra, “Stochastic Backpropagation and Approximate Inference in Deep Generative Models,” *Proceedings of the 31st International Conference on Machine Learning*, vol. 32, pp. 1278-1286, 2014.
- [106] C. Chadebec, S. Allasonniere, S. Engelhardt, I. Oksuz, D. Zhu, Y. Yuan, A. Mukhopadhyay, N. Heller, S. Huang, H. Nguyen, R. Sznitman and Y. Xue, “Data Augmentation with Variational Autoencoders and Manifold Sampling,” *Lecture Notes in Computer Science*, vol. 13003, pp. 184-192, 28 10 2021.
- [107] M. Elbattah, C. Loughnane, J.-L. Guerin, R. Caretter, F. Cilia and G. Dequen, “Variational Autoencoder for Image-Based Augmentation of Eye-Tracking Data,” *Journal of Imaging*, vol. 7, no. 83, 11 06 2021.
- [108] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, “Generative Adversarial Networks,” *Association for Computing Machinery*, pp. 139-148, 2014.
- [109] A. Radford, Metz, L. Metz and S. Chintala, “Unsupervised representation learning with deep convolutional generative adversarial networks,” *arXiv*, pp. 1-16, 2016.
- [110] T. Karras, T. Aila, S. Laine and J. Lehtinen, “Progressive Growing of GANs for Improved Quality, Stability, and Variation,” *Proceedings of ICLR 2018; arXiv*, pp. 1-26, 2017.
- [111] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang and W. Shi, “Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network,” *arXiv*, pp. 1-19, 2017.
- [112] G. Wang, W. Kang, Q. Wu, Z. Wang and G. Junbin, “Generative Adversarial Network (GAN) based Data Augmentation for Palmprint Recognition,” *2018 Digital Image Computing: Techniques and Applications (DICTA)*, pp. 156-162, 2018.
- [113] “Data augmentation using MG-GAN for improved cancer classification on gene expression data,” *Soft Computing*, pp. 11381-11391, 27 07 2020.
- [114] J. K. Dumagpi and Y.-J. Jeong, “Evaluating GAN-Based Image Augmentation for Threat Detection in Large-Scale Xray Security Images,” *APPLIED SCIENCES-BASEL*, vol. 11, no. 1, 25 01 2021.
- [115] M. Frid-Adar, E. Klang, M. Amitai, J. Goldberger and H. Greenspan, “SYNTHETIC DATA AUGMENTATION USING GAN FOR IMPROVED LIVER LESION CLASSIFICATION,” *2018 IEEE 15TH INTERNATIONAL SYMPOSIUM ON BIOMEDICAL IMAGING (ISBI 2018)*, pp. 289-293, 2018.

- [116] A. Chaudhuri, K. Mandaviya, P. Badelia and S. K. Ghosh, "Optical Character Recognition Systems for Different Languages with Soft Computing," *Studies in Fuzziness and Soft Computing*, 2017.
- [117] F. T. Yu, "Optical Pattern Recognition," *Optics & Photonics News*, vol. 12, pp. 55-, 2001.
- [118] N. Arica and F. T. Yarman Vural, "An Overview of Character Recognition focused on Offline Handwriting," *IEEE Transactions on Systems, Man and Cybernetics - Part C: Applications and Reviews*, vol. 31, no. 2, pp. 216-233, 05 2001.
- [119] F. Kboubi, A. H. Chaibi and B. M. Armed, "A new strategy of OCR combination," *APPLICATIONS OF CYBERNETICS AND INFORMATICS IN OPTICS, SIGNALS, SCIENCE AND ENGINEERING*, pp. 325-330, 2004.
- [120] T.-T.-H. Nguyen, A. Jatowt, M. Coustaty, N.-V. Nguyen and A. Doucet, "Deep Statistical Analysis of OCR Errors for Effective Post-OCR Processing," *Proceedings of 2019 ACM/IEEE Joint Conference on Digital Libraries (JCDL)*, pp. 29-38, 2019.
- [121] S. Alghyaline, "Arabic Optical Character Recognition: A Review," *CMES-Computer Modeling in Engineering & Sciences*, vol. 135, no. 3, pp. 1825-1861, 2013.
- [122] "The BBN Byblos Hindi OCR system," *Proceedings of SPIE - The International Society for Optical Engineering: Document Recognition and Retrieval XII*, vol. 5676, pp. 10-16, 17 1 2005.
- [123] A. Kokawa, L. S. P. Busagala, W. Ohyama, T. Wakabayashi and F. Kimura, "An Impact of OCR Errors on Automated Classification of OCR Japanese Texts with Parts-of-Speech Analysis," *2011 International Conference on Document Analysis and Recognition*, pp. 543-547, 2011.
- [124] P. Jain, K. Taneja and H. Taneja, "Which OCR toolset is good and why : A comparative study," *Kuwait Journal of Science*, vol. 48, no. 2, pp. 1-12, 2021.
- [125] A. Shanthakumari, R. Kalpana, J. Jayashankari, B. UmaMaheswari and M. Sirija, "Mask RCNN and Tesseract OCR for vehicle plate character recognition," *AIP Conference Proceedings 2393*, vol. 2393, no. 020135, 2022.
- [126] F. Borisjuk, A. Gordo and V. Sivakumar, "Rosetta," pp. 71-79, 19 07 2018.
- [127] B. Xu, Y. Guo, L. Jiang, M. Yu and J. Chen, "Downscaled Representation Matters: Improving Image Rescaling with Collaborative Downscaled Images," in *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, Paris, France, 2023.
- [128] E. Stötzner, T. Homburg and H. Mara, "CNN based Cuneiform Sign Detection Learned from Annotated 3D Renderings and Mapped Photographs with Illumination Augmentation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision.*, Paris, France, 2023.
- [129] D. Franc, A. Hamplová, O. Svojshe and A. Veselý, "Data augmentation methods using Generative Adversarial Networks for improving the accuracy of Computer Vision algorithms," *Journal of Intelligent & Fuzzy Systems*, 2024 (forthcoming).
- [130] Google Scholar, "Hamplová: Cuneiform Reading Using Computer Vision Algorithms - Google Scholar," [Online]. Available: https://scholar.google.co.uk/scholar?cites=13200824122647241410&as_sdt=2005&sciodt=0,5&hl=cs. [Accessed 18 04 2024].

5 List of Figures, Tables and Abbreviations

5.1 List of Figures

Figure 1 - mathematical model of perceptron with output function $\sigma(h)$ [22].....	9
Figure 2 – activation functions of convolutional layers: (a) sigmoid (b) tanh (c) ReLU (d) leaky ReLU [26]	12
Figure 3 - Demostration of maxpooling and averagepooling operation [31]	14
Figure 4- LeNet-5 used for recognition of handwritten character (32 x 32 pixels), Source: [23].....	17
Figure 5 - Inception modules presented in GoogLeNet [35]	18
Figure 6 - Residual block [37]	19
Figure 7 - Number of data needed per class for a high classification accuracy [50].....	23
Figure 8- Computer vision tasks: (a) image classification, (b) object detection, (c) semantic segmentation (d) instance segmentation. [86]	37
Figure 9 - Masks created with polygonal labels for training a Mask R-CNN network.....	39
Figure 10 - Elephant predicted with Mask R-CNN [96].....	40
Figure 11 – Houses segmented with Mask R-CNN: Mapping Challenge converting satellite images to maps [95]	40
Figure 12 - Roboflow pre-processing options	42
Figure 13 - Roboflow augmentation optios	43
Figure 14 - Neural Style Transfer by Leon Gatys [98].....	45
Figure 15 - Principle of VAE [48, p. 274]	46

5.2 List of Tables

Table 1 - Convolutional layer input - a black and white letter "O" represented in pixels	11
Table 2 - Binary confusion matrix	24
Table 3 - 4-class confusion matrix.....	24

5.3 List of Abbreviations

OCR = Optical Character Recognition

GPU = Graphical Processing Unit

TPU = Tensor Processing Unit

CNN = Convolutional Neural Network

ANN = Artificial Neural Network

DNN = Deep Neural Network

AI = Artificial Intelligence

GAN = Generative Adversarial Network
RoI = Region of Interest
SNARC =
EQV = Equivalence
XOR = Nonequivalence
PDP = Paralell Distributed Processing
ReLu = Rectified Linear Unit
eLu = Exponential Linear Unit
ESSE = Error Sum of Square Errors
ECE = Error Cross-Entropy
GTN = Graph Transformer Network
IEEE = Institute of Electrical and Electronics Engineers
VGG Net = Visual Geometry Group Network
ICLR = International Conference on Learning Representations
Res Net = Residual Network
SE Net = Squeeze and Excitation Network
FE Net = Fully Exploited Network
NLP = Natural Language Processing
SVM = Support Vector Machine
CT = Computed Tomography
TP = True Positive
TN = True Negative
FP = False Positive
FN = False Negative
ROC = Receiver Operating Characteristic or Relative Operating Characteristic
FAR = False Alarm Rate
TPR = True Positive Rate
AUC = Area Under Curve
FCN = Fully Convolutional Network
YOLO = You Only Look Once
TIP = Traditional Image Processing
SIFT = Scale Invariant Feature Transform
HOG = Histograms of Oriented Gradients

PASCAL VOC = Pattern Analysis, Statistical Modelling and Computational Learning Visual Object Challenge
COCO = Common Objects in Context
ID = Identification
API = Application Programming Interface
TF = TensorFlow
SPP Net = Spatial Pyramid Pooling Network
BoW = Bag of Words
R-CNN = Regions with CNN features
FPN = Feature Pyramid Network
RPN = Region Proposal Network
G-RCNN = Gated Recurrent Convolutional Neural Network
SSD = Single Shot Detector
IoU = Intersection over Union
DAG = Directed Acyclic Graph
ILSVRC = ImageNet Large Scale Visual Recognition Challenge
mAP = mean Average Precision
FPS = Frames per Second
HCD = Hierarchical Collaborative Downscaling

List of Attachments

Attachment 1: A. Hamplová, D. Franc and J. Tyrychtr, “Historical Alphabet Transliteration Software Using Computer Vision Classification Approach,” Lecture Notes in Networks and Systems, pp. 34-45, 26 04 2022.

Attachment 2: A. Hamplová, D. Franc and A. Veselý, “An improved classifier and transliterator of hand-written Palmyrene letters to Latin,” Neural Network World, vol. 32, pp. 181-195, 30 08 2022.

Attachment 3: D. Franc, A. Hamplová and O. Svojshe, “Augmenting Historical Alphabet Datasets Using Generative Adversarial Networks,” Data Science and Algorithms in Systems. CoMeSySo 2022. Lecture Notes in Networks and Systems, vol. 597, pp. 132-141, 04 01 2023.

Attachment 4: A. Hamplová, D. Franc and J. Pavlíček, “Character segmentation in the development of Palmyrene Aramaic OCR,” CAISE 2023 35th International Conference on Advanced Information Systems Engineering, 2023.

Attachment 5: A. Hamplová, A. Lyavdansky, T. Novák, O. Svojshe, D. Franc and A. Veselý, “Instance Segmentation Of Characters Recognized In Palmyrene Aramaic Inscriptions,” Computer Modelling in Engineering & Sciences, 2024.

Attachment 6: A. Hamplová, D. Franc, J. Pavlíček, A. Romach and S. Gordin, “Cuneiform Reading Using Computer Vision Algorithms,” SPML 2022: Proceedings of the 2022 5th International Conference on Signal Processing and Machine Learning, 2022.

Attachment 7: A. Hamplová, A. Romach, J. Pavlíček, A. Veselý, M. Čejka, D. Franc and S. Gordin, “Cuneiform stroke recognition and vectorization in 2D images,” Digital Humanities Quarterly, 2023.

Attachment 1

A. Hamplová, D. Franc and J. Tyrychtr, “Historical Alphabet Transliteration Software Using Computer Vision Classification Approach,” Lecture Notes in Networks and Systems, pp. 34-45, 26 04 2022.



Historical Alphabet Transliteration Software Using Computer Vision Classification Approach

Adéla Hamplová^(✉) , David Franc , and Jan Tyrychtr 

Czech University of Life Sciences in Prague, Kamýcká 129,
165 00 Praha-Suchbát, Czech Republic
{hamplova, francd, tyrychtr}@pef.czu.cz

Abstract. The article presents the problem of developing mobile software for classification and automatic transliteration of historical alphabets to Latin alphabet using OCR Computer Vision algorithms and is presented on Palmyrene Alphabet. Our suggested solution of semi-automatic transliteration of historical alphabets speeds up and simplifies the process of ancient text analysis and makes reading historical alphabets available to the public. We created a mobile application template for field use and proved the functionality on our own photographic and digitized hand-written datasets of Palmyrene letters, using a MobileNet Artificial Neural Network for character recognition. Such an application helps archaeologists with a faster character transliteration of newly discovered, archived, but untranslated tablets, columns etc., and for checking hand-transliterated texts.

Keywords: Artificial intelligence · Classification · Historical alphabet · Software development

1 Introduction

The technology of Optical Character Recognition, commonly known as OCR, has developed rapidly in the last years, and its applications can be found in various areas of human activities. Applications vary from getting text from scanned documents to automatic registration plate-number recognition. However, not all alphabets have this technology available, and transliteration has to be done manually.

Archaeology, like any other human activity and research area, can profit from such technology. There is a large number of historical documents in different alphabets that archaeologists need to analyze, rewrite by hand to paper and then translate, one of which is Palmyrene.

2 Theoretical Overview

2.1 Image Classification

Artificial Intelligence is an area, which first appeared in the 1950s. It includes a wide area of computer science tasks that don't use rules and data to get answers, but uses data and answers to create rules [1].

The input data points are taken as a training set, and the learning algorithm (also called fitting algorithm) is trying to map these inputs to the expected outputs, by alternating neuron weights and keeping the same activation functions in the network during the process. This process is called learning, and the quality of learning can be measured by computing error – the loss. Loss can be simplified as the difference between predicted and real outputs. In Artificial Intelligence terminology, the trained network is called a model. It consists of network architecture and trained weights [1].

Image classification is using deep learning architectures with Convolutional Neural Networks. CN Networks accept input tensors in the shape of (image_height, image_width, image_depth). Deeper in the network the dimensions are lower and lower, since the Convolution operation is reducing the image to local connections by using feature maps. The last but one layer of CNN needs to be flattened by using the Flatten layer. The output is provided by a Dense layer, which returns an integer – index of class, where the image is predicted to [1].

The classification success rate can be measured by standard metrics. They are called precision, recall, F-measure and accuracy and are explained in Methodology [2]. These metrics are calculated usually on the training set, which is a subset of images, which have not been used during the training.

2.2 OCR

As early as the 19th century, the first attempts to read letters by machine were recorded, until a technology reading exactly one character was developed [3]. In the 19th century, OCR was already widely used and was getting known to the general public [4]. In 2020, OCR was even used for biometric identification with cross-entropy fitness function to get the smallest squares [5]. Among the latest advances in this area is a specially developed Artificial Neural Network called AutoOCR. In comparison to other applications, there is an automated process of training parameter tuning, and the latest research of OCR applies LSTM (Long Short-Term Memory) layers to neural network architectures for processing words – sequences of characters [6].

Although the technology for reading characters already exists, there is still space for experiments with the optimization of a suitable neural network architecture, special for historical alphabets recognition from either hand-written sources or directly from photographic records.

2.3 Historical Alphabet Digitisation Including Palmyrene

There are multiple researches specialized in historical alphabet character recognition, such as Persian [7] or Bangladeshi [8], as well as hand-written transliterations of cuneiform [9]. However, there is still currently no Palmyrene transliteration available.

At present, there are a large number of published texts in Palmyrene Aramaic, written in a special alphabet, that is similar to but slightly different from Aramaic at the same time. Palmyrene was used in years 100–400 A.D. in the Syrian city of Palmyra (otherwise called Tadmur) and in its surroundings. This language was derived from classical Aramaic but uses its own dialect and alphabet. Palmyrene Aramaic was spoken in

western parts of Syria in the beginning of the first century, on the other hand in eastern parts classical Aramaic was spoken.

The study of texts in Palmyrene Aramaic contributes not only to Palmyrene studies, but also Semitic and Biblical studies, the study of ancient history and art [10]. Ones of the most extensive anthologies were published in years 1996 and 2001 [10, 11].

In 2010, a standard coding for Palmyrene letters was proposed and adopted [12], and it is currently a part of an extended Unicode character table. The Palmyrene alphabet contains 32 characters, 10860-1087F Unicode [13]. The script is written horizontally, right to left, letters are not divided by a blank space, apart from “y” there is no vowel, only consonants. Writing numbers is similar to the Roman way, but there are only characters for number 1, 5, 10 (also used for hundreds) and 20. Numbers 2, 3 and 4 are depicted as a multiple of 1.

Currently, there is a research gap in the field of automatic character reading of historical alphabets and transliterating them to Latin alphabet, part of it means there is also no Palmyrene alphabet transliterator available yet.

2.4 Android Software Development

Android is an operating system based on Linux, mainly focused on mobile devices. Its apps are developed in Java mainly using Android Studio as IDE. There are also other multi-platform development tools, which can be used for Android as well.

A set of basic libraries that are included in each version of Android API is upgraded with every Android system update and some methods in these libraries are added and improved.

Image classification is not included in standard Android API libraries. Premade classifiers using CNN are available in Tensorflow Lite. The most common models for image classification use TFLite MobileNet. Current research recommends using efficient_lite0 network [14]. The architecture of this model is visible in Fig. 1. The dropout layer randomly sets some neuron weights to 0 to prevent overfitting.

Layer (type)	Output Shape	Param #
hub_keras_layer_v1v2 (HubKer)	(None, 1280)	3413024
dropout (Dropout)	(None, 1280)	0
dense (Dense)	(None, 28)	35868
Total params: 3,448,892		
Trainable params: 35,868		
Non-trainable params: 3,413,024		

Fig. 1. MobileNet efficientnet_lite0 model summary

3 Objective

OCR technology is still being developed and improved and currently there is no tool that would help researchers or the general public transliterating Palmyrene characters to Latin.

The main objective of this research is therefore to create a mobile application using artificial intelligence methods (OCR) for historical character classification and transliteration and verify the research on the use case of Palmyrene alphabet. For performance tests, we also need to create two dataset – hand-written and photographic and make a tool for it. The required test accuracy is above 70% in order to consider the classifier satisfactory.

The target group of users are archaeologists, who would use the software for faster Palmyrene Aramaic texts transliteration, the secondary focus group are other researchers and people outside the scientific community that could use the transliteration for educational purposes. The Android application can also be modified if another historical alphabet model is trained. Therefore, it can be used for further semi-automatic transliteration of any alphabet.

4 Methodology

4.1 Acquiring Data

For hand-written dataset creation, we created a software tool called “Write Character” for computer hand-writing by modifying AxelThevenot’s Python-Interface-to-Create-Handwrittendataset” tool available at Github [17]. It creates a dataset, which contains the same number of characters in each class.

In order to get a photographic dataset, a sufficient number of images containing the required script, such as the one in Fig. 2, needs to be obtained and pre-processed. If there are not enough characters available from the photos, the remaining count is filled by an imitation of sandstone tablets – writing letters to sand.

For the acquisition of such photographs, our team established a cooperation with two museums – Musée du Louvre in Paris [15] and Virtual Museum Of Palmyra [16]. Both museums provided images of tablets with Palmyrene inscriptions.

The image pre-processing consists of cutting input images to squares, each square containing one letter and sorting them to folders. If images have low contrast, a gradient edge highlighting method should be used.

4.2 Picking Architecture and Training OCR Model

One of the current recommended neural networks used in Android applications is TFLite MobileNet efficient_lite0 [14]. The final activation function needs to be logistic, as this is a multiple category classification. Using optimisation methods like dropout is advisable.

4.3 Model Evaluation

For model evaluation, following measures are used, two counted from the classification results, one is derived and one is dependent on network output [2]:

Precision

$$p = \frac{TP}{TP + FP} \quad (1)$$

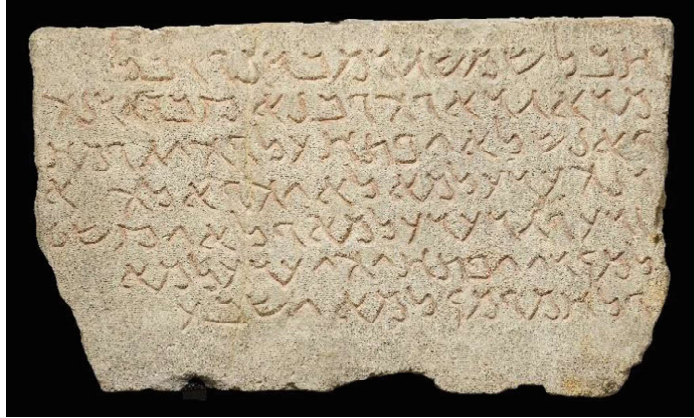


Fig. 2. Example of input image containing Palmyrene script, Antiquités orientales, AO2204, 08-516087, Paris, musée du Louvre, photo (C) RMN-Grand Palais (musée du Louvre) - © Franck Raux, Dalle portant une inscription funéraire palmyrénienne

Recall (sensitivity)

$$s = \frac{TP}{TP + FN} \quad (2)$$

Quality of algorithm (F-measure)

$$F = \frac{2 \cdot (s \cdot p)}{s + p} \quad (3)$$

where

TP = true positives, the number of correctly classified objects as positive.

FP = false positives, the number of incorrectly classified objects as positive.

FN = false negatives, the number of incorrectly classified objects as negative.

If $s = p$, then F -measure = $s = p$.

The values of TP , FP , FN as well as TN (true negatives, the number of correctly classified objects as negative) can be written in a 2-dimensional matrix called “confusion matrix” [2]. If there is just one class, i.e. the image contains the object of interest or not, there is a confusion matrix with 2 columns and rows as visible in Table 1.

Table 1. 2-dimensional confusion matrix for 1 class classification

Actual/predicted	Positive	Negative
Positive	TP	FN
Negative	FP	TN

For multiclass classification, the confusion matrix has as many columns and rows as there are classes, as visible in Table 2, with actual values in the column and predicted

values in the row as well. The TP , FN , FP and TN values are calculated from the matrix as a sum of corresponding values.

Table 2. 2-dimensional confusion matrix for 4 class classification

Actual/predicted	Class 1	Class 2	Class 3	Class 4
Class 1	TP for class 1	FN for class 1, FP for class 2	FN for class 1, FP for class 3	FN for class 1, FP for class 4
Class 2	FN for class 2, FP for class 1	TP for class 2	FN for class 2, FP for class 3	FN for class 2, FP for class 4
Class 3	FN for class 3, FP for class 1	FN for class 3, FP for class 2	TP for class 3	FN for class 3, FP for class 4
Class 4	FN for class 4, FP for class 1	FN for class 4, FP for class 2	FN for class 4, FP for class 3	TP for class 4

Confidence Score

The confidence score indicates the ratio of correct predictions in percentages. It is calculated separately on training, validation and testing dataset using “accuracy” as scoring method during training and predicting. The testing accuracy shows how the model generally performs on images outside of the training set. It is calculated during the training and evaluation in the code.

4.4 Application Development

For the mobile application, we use a wide set of Android libraries, i.e. a native library `android.graphics`, which contains tools that handle standard graphic operations such as picture resizing. It is needed to deliver images from drawing or camera inputs in specific format to image classifier, which accepts explicitly $224 \times 224 \times 3$ inputs to the first convolutional layer in accordance with MobileNet architecture. These parameters, along with normalisation methods, authors information and license, are determined in model metadata.

For image classification, the trained models in `tflite` format are loaded. The input image is transformed, analysed and the output category with confidence score will be returned and displayed in the form of a table with three results with the greatest confidence.

5 Results

5.1 Dataset Creation Results

The “Write Character” tool has been used for the creation of a dataset consisting of 56207 handwritten images belonging to 28 classes. A separate class for number 5 “ γ ”

Table 3. Palmyrene system font “Palmmene” with transcription to Latin

Palmyrene	Latin	Transcription to Latin
Ⲁ	Aleph	ʿ/A
Ⲃ	Beth	B
Ⲅ	Gimmel	G
Ⲇ	Daleth	D
Ⲉ	He	H
Ⲋ	Waw	V
Ⲍ	Zayin	Z
Ⲏ	Heth	KH
Ⲑ	Teth	T
Ⲓ	Yodh	Y
Ⲕ	Kaph	K/C
Ⲗ	Lamedh	L
Ⲙ	Mem	M
Ⲛ	Final Nun	M
Ⲝ	Nun	N
Ⲟ	Samekh	S
Ⲡ	Ayin	ʿ/E
Ⲣ	Pe	P
Ⲥ	Sadhe	TS
ⲧ	Qoph	Q
ⲩ	Resh	R
ⲫ	Shin	SH
ⲭ	Taw	TH
ⲱ	left	←
ⲱ	right	→
ⲱ	1	1
ⲱ	2	2
ⲱ	3	3
ⲱ	4	4
ⲱ	5	5
ⲱ	10, 100	10, 100
ⲱ	20	20

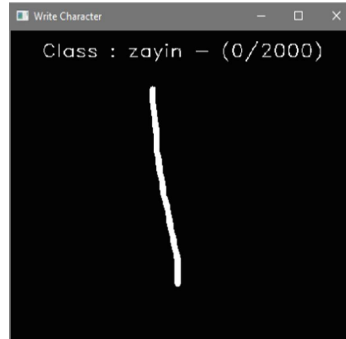


Fig. 3. User interface of the dataset handwriting tool

has been left out, as the character resembles very highly to the character ayin “*y*”, the used system font was “palmme” (Table 3).

We created the photographic dataset by cutting photos provided by museums and by creating sandstone tablet imitations; each class contains roughly 440 photos. The total number of photos was 12351 images (Fig. 4).



Fig. 4. Example of data from photographic dataset: letter Aleph

5.2 Model Training

The training of both models was conducted for 5 epochs using the graphic card Nvidia GeForce GTX 1650. The train/test split was 90% for training and 10% for testing. We

used the library `tflite-model-maker` for the `efficient_lite0` model creation, without any further alteration of the architecture.

Photographic Model

The training accuracy improved from 0.4328 in the first epoch to 0.7523 in the last epoch. The testing accuracy was 0.7712.

Hand-Written Model

The training accuracy was very high even after the first epoch, it reached 0.902, and in the last epoch it reached 0.993. The testing accuracy was 0.977.

Model Evaluation

The testing set for evaluating both hand-written and photographic models consisted of 112 images (4 of each class). We conducted the test directly in the developed Android application, we drew the images in the “draw letter” module and took photos of new sandstone imitation data, manually checked it, and summarised the prediction results in confusion matrices and shortened them to basic confusion matrices, as if it was a true/false classification, and divided the false predictions between *FN* and *FP*. The confusion matrices are visible in Table 4.

Table 4. Shortened confusion matrices of hand-written and photographic models

Actual/predicted	Positive	Negative
Positive	$TP_{HW} = 81$	$FN_{HW} = 20$
	$TP_P = 68$	$FN_{HW} = 27$
Negative	$FP_{HW} = 20$	$TN_{HW} = 0$
	$FP_{HW} = 27$	$TN_{HW} = 0$

The calculated precision, recall and F-values are 80,2% for the hand-written set and 71,96% for the photographic set. The most common errors were false positives in the class “pe”.

5.3 Android Software Template and Palmyrene Transliterator

Our team created a software package – an Android application template, implemented on the use case of Palmyrene letters, using image processing libraries as well as `org.tensorflow.lite`. The module for image analysis uses TFLite MobileNet architecture. There are 2 main usages – drawing a letter and analysing it with the first MobileNet model for hand-written letter classification. The second one starts with taking a picture, and the camera input is fed into the other MobileNet model and returns the same output – three classes with the highest confidence score (Fig. 5).

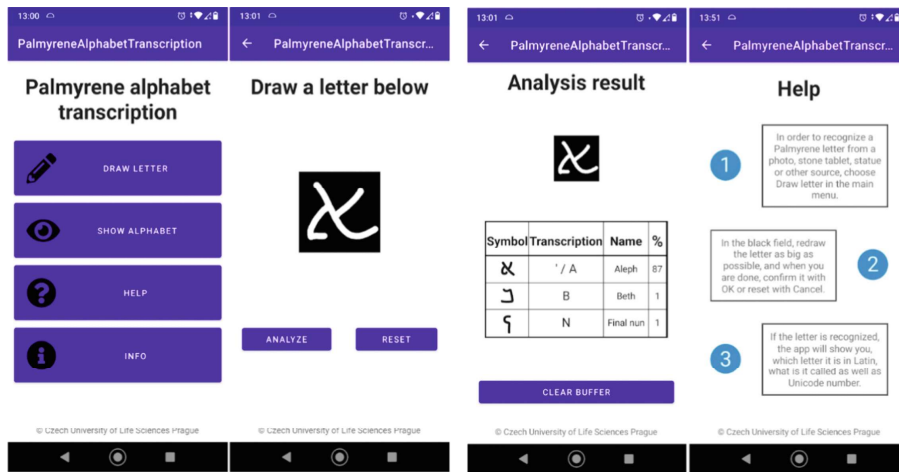


Fig. 5. Mobile application user interface

6 Discussion

In relation to a resemblance of some Palmyrene letter signs, the results of the image classification were satisfactory, as the quality of the algorithm reached 80,2% for hand-written dataset and 71,96% for photographic dataset. The difference is caused by the number of images in the training dataset as well as the difficulty of the input. While hand-written characters are only black and white, the photos vary from black to white in different shades of grey and there is much more noise in the pictures.

In GigaMesh, the analysis of 3D scans of cuneiform tablets has been conducted, and the character extraction success rate has not been published in the article [18].

Also the recognition of handwritten cuneiform vowels published by Yamauchi [9] has not shown any tables with classifiers results.

The character recognition of Ghosh et al.'s model of Bangladeshi signs reached 96.46% testing accuracy on MobileNet [8], and Sazal et al. reached 90.27% accuracy on a 60-class model of hand-written Bangladeshi script [19].

From the artificial intelligence point of view, different authors have used object recognition to achieve other tasks. Cho Junghwan et al. has reached about 97% accuracy on GoogLeNet network architecture (Inception v1) using a dataset of about 4000 very high-quality images of CT body scans, and the decrease rate for lower number of images has been described there accordingly [20]. A smaller success rate from this research has been reached in Great Tits recognition from Smart Nest Boxes using YOLOv3 architecture, where the F-measure reached 83% due to difficult object detection [21].

This fact indicates that the results of Palmyrene letter classifications are satisfactory (reached above 70% of F-measure as specified in the initial criteria) and are comparable with other authors' works, but still have room for improvement, as the Palmyrene alphabet has some letters, which are very similar to each other.

7 Conclusion

We created a software tool, which uses artificial intelligence (OCR) for semi-automation of historical alphabets transliteration and proved it on Palmyrene Aramaic script. The tool consists of several components – dataset drawer, classifier training and an Android mobile application using the trained TFLite MobileNet models for field use. This application can also be used as a template for other historical alphabets, if the models are swapped.

Our tool can greatly speed up the process of analyzing ancient texts in different historical scripts and enable reading such signs for the general public for educational purposes.

Although the results have been satisfactory, further experiments with network layers should be conducted in order to improve the testing accuracy even more, and a photographic dataset of Palmyrene letters can be expanded, as the accuracy was lower than with the hand-written dataset.

Acknowledgment. This article was created with the support of the Internal Grant Agency (IGA) Faculty of Management and Economy, Czech University of Life Sciences in Prague, 2021A0004 – “Reading the characters of Palmyrene alphabet using artificial intelligence tools”.

References

1. Chollet, F., Pecinovský, R.: Deep learning v jazyku Python. pp. 22, 24, 118. Grada Publishing (2019)
2. Géron, A.: Hands-on machine learning with Scikit-Learn and TensorFlow, 2nd ed., pp. 88–92. O’Reilly (2019)
3. Schantz, H.F.: The History of OCR, Optical Character Recognition. Recognition Technologies Users Association, Manchester Center, Vt. (1982)
4. Mori, S., Nishida, H., Yamada, H.: Optical Character Recognition. John Wiley and Sons, New York (1999)
5. Akhtar, Z., Lee, J.W., Attique Khan, M., Sharif, M., Ali Khan, S., Riaz, N.: Optical character recognition (OCR) using partial least square (PLS) based feature reduction: an application to artificial intelligence for biometric identification. *J. Enterprise Inform. Manage.* (2020). <https://doi.org/10.1108/JEIM-02-2020-0076>
6. Zhao, Z., Jiang, M., Guo, S., Wang, Z., Chao, F., Tan, K.C.: Improving deep learning based optical character recognition via neural architecture search. In: IEEE Congress on Evolutionary Computation (CEC), Glasgow, United Kingdom, pp. 1–7 (2020). <https://doi.org/10.1109/CEC48606.2020.9185798>
7. Hajjhashemi, V., Arab Ameri, M.M., Alavi Gharahbagh, A., Bastanfard, A.: A pattern recognition based Holographic Graph Neuron for Persian alphabet recognition. In: International Conference on Machine Vision and Image Processing (MVIP), pp. 1–6 (2020). <https://doi.org/10.1109/MVIP49855.2020.9116913>
8. Ghosh, T., et al.: Bangla handwritten character recognition using MobileNet V1 architecture. *Bull. Electr. Eng. Inform.* **9**(6), 2547–2554 (2020)
9. Yamauchi, K., Yamamoto, H., Mori, W.: Building a handwritten cuneiform character image-set. In: Proceedings of the Eleventh International Conference on Language Resources and Evaluation. [online] LREC 2018 (2018). <http://aclweb.org/anthology/L18-1115>

10. Hillers, D., Cussini, E.: Palmyrene Aramaic Texts. Johns Hopkins Univ. Press, Baltimore (1996)
11. Taylor, D.G.K.: An annotated index of dated Palmyrene Aramaic texts, *J. Semitic Stud.* **XLVI**(2), 203–219 (2001). <https://doi.org/10.1093/jss/XLVI.2.203>
12. Everson, M.: Proposal for Encoding the Palmyrene Script in the SMP of the UCS. Department of Linguistics, UC Berkeley. <https://escholarship.org/uc/item/27b327h7> (2010)
13. Palmyrene, Range: 10860–1087F. <https://www.unicode.org/charts/PDF/U10860.pdf> (2010)
14. Gupta, D.: Mobile application for bird species identification using transfer learning. In: 2021 IEEE International Conference on Artificial Intelligence in Engineering and Technology (ICAIET), pp. 1–6. IEEE (2021)
15. Le Louvre: Louvre Museum Official Website. <https://www.louvre.fr/en>. Last accessed 1 Dec. 2021
16. Palmyra Archaeological Museum: The Archaeological Museum of Palmyra. <https://virtual-museum-syria.org/palmyra/> (2021). Last accessed 1 Dec. 2021
17. Thevenot, A.: GitHub – AxelThevenot/Python-Interface-to-Create-Handwritten-dataset: Python Interface to Create Handwritten Dataset. GitHub (2019)
18. Mara, H., Krömker, S., Jakob, S., Breuckmann, B.: GigaMesh and Gilgamesh – 3D Multiscale Integral Invariant Cuneiform Character Extraction. In: The 11th International Symposium on Virtual Reality, Archaeology and Cultural Heritage. VAST 2010 (2010)
19. Sazal, M.M.R., Biswas, S.K., Amin, M.F., Murase, K.: Bangla handwritten character recognition using deep belief network. In: Int. Conf. Electr. Inf. Commun. Technol. EICT 2013 pp. 1–5 (2013)
20. Junghwan, Ch., et al.: How much data is needed to train a medical image deep learning system to achieve necessary high accuracy? 1511.06348 (2015)
21. Hamplová, A., Pavlíček, J.: Object recognition tool for “smart nest boxes.” In: Proceedings of IAC in Budapest 2020, IAC-ETITAI. Czech Institute of Academic Education, Prague, Czech Republic, pp.105–109 (2020)

Attachment 2

A. Hamplová, D. Franc and A. Veselý, “An improved classifier and transliterator of hand-written Palmyrene letters to Latin,” *Neural Network World*, vol. 32, pp. 181-195, 30 08 2022.



AN IMPROVED CLASSIFIER AND TRANSLITERATOR OF HAND-WRITTEN PALMYRENE LETTERS TO LATIN

A. Hamplová*, D. Franc†, A. Veselý‡

Abstract: This article presents the problem of improving the classifier of hand-written letters from historical alphabets, using letter classification algorithms and transliterating them to Latin. We apply it on Palmyrene alphabet, which is a complex alphabet with letters, some of which are very similar to each other. We created a mobile application for Palmyrene alphabet that is able to transliterate hand-written letters or letters that are given as photograph images. At first, the core of the application was based on MobileNet, but the classification results were not suitable enough. In this article, we suggest an improved, better performing convolutional neural network architecture for hand-written letter classifier used in our mobile application. Our suggested new convolutional neural network architecture shows an improvement in accuracy from 0.6893 to 0.9821 by 142 % for hand-written model in comparison with the original MobileNet. Future plans are to improve the photographic model as well.

Key words: *artificial intelligence, classification, historical alphabets, mobilenet, computer vision*

Received: February 25, 2022

DOI: 10.14311/NNW.2022.32.011

Revised and accepted: August 30, 2022

1. Introduction

1.1 Historical alphabet digitization including Palmyrene

Many researches are focused on character recognition of letters from historical alphabets. These include Persian [5], Bangladeshi [3] and cuneiform, which is transliterated by hand [19] and photos of these transliterations are classified.

Until recently, there was no Palmyrene transliteration available. There is a large number of Palmyrene Aramaic memorabilia, which is written in Palmyrene alphabet. It is similar to classic Aramaic with some differences in the alphabet and dialect. This dialect was used in western parts of Syria, and classic Aramaic was

*Adéla Hamplová; Czech University of Life Sciences in Prague, PEF KII, Kamýcká 129, CZ-165 00, Praha 6 – Suchdol, Czech Republic, E-mail: hamplova@pef.czu.cz

†David Franc – Corresponding author; Czech University of Life Sciences in Prague, PEF KII, Kamýcká 129, CZ-165 00, Praha 6 – Suchdol, Czech Republic, E-mail: francd@pef.czu.cz

‡Arnošt Veselý – Corresponding author; Czech University of Life Sciences in Prague, PEF KII, Kamýcká 129, CZ-165 00, Praha 6 – Suchdol, Czech Republic, E-mail: vesely@pef.czu.cz

spoken in the eastern parts. This script was used in the nearest surroundings and inside the Syrian city of Palmyra (also called city of Tadmur) in the years 100–400 A.D.

Translating Palmyrene texts is contributing to the study of ancient art and history, as well as Palmyrene and Biblical studies. The largest anthologies were published in 1996 by Hillers et al. [7] and in 2001 by Taylor et al. [15].

Palmyrene font became part of Unicode in 2010 [1] when the coding for Palmyrene letters was proposed. The alphabet consists of 32 characters in the range 10860–1087F in Unicode [12] (Palmyrene, 2010). Apart from “y”, there are only consonants in the script. The alphabet is read from the top right to left corner; words are not divided by a blank space. As for numbers, there are only four characters, which mean 1, 5, 10 (or 100) and 20.

1.2 Image classification on mobile devices

Android applications are developed mostly using the IDE Android Studio and in each version of IDE, a set of standard libraries is added and some of the existing ones are updated. Among standard Android API libraries, image classification is not included. The TensorFlow documentation website [17] recommends using convolutional neural network MobileNet for image classification on mobile devices.

Current research [4] recommends using MobileNet version `efficient_lite0`. It was introduced by Tan and Le in 2020 [16] and is also presented in Sonawane’s paper [14] in comparison with other architectures.

1.3 Android software template and Palmyrene transliterator

In our research we suggest a mobile software tool for automatic character reading of historical alphabets and transliterating them into Latin alphabet. This tool called “Palmyrene Alphabet Transcription” has a potential to help to speed up the process of processing archived but not transliterated and untranslated documents or can be used directly in the field by archeologists.

Our tool is an Android application that can serve as a template for other alphabets as well. The two main use cases of our tool are hand-written letter analysis and letter analysis from photo.

In the first one our tool asks the user to draw a letter on the screen. The drawn image is resized and sent on the input of the convolution neural network (CNN). CNN classifies the image and then the three possible transliterations with the highest confidence score are displayed, see Fig. 1.

The second possibility how to use our tool is using it for transliteration of letters given in photos. Instead of writing the letters by hand, the user takes a photograph of the letter directly from sandstone tablet like in Fig. 2 or other document type that is written in Palmyrene.

The user interface is visible in Fig. 3. There is also a help available, as well as info and alphabet table available in the application.

The target group of users of this tool are archaeologists, who would use the software for faster Palmyrene Aramaic texts transliteration, the secondary focus group are other researchers and people outside the scientific community that could

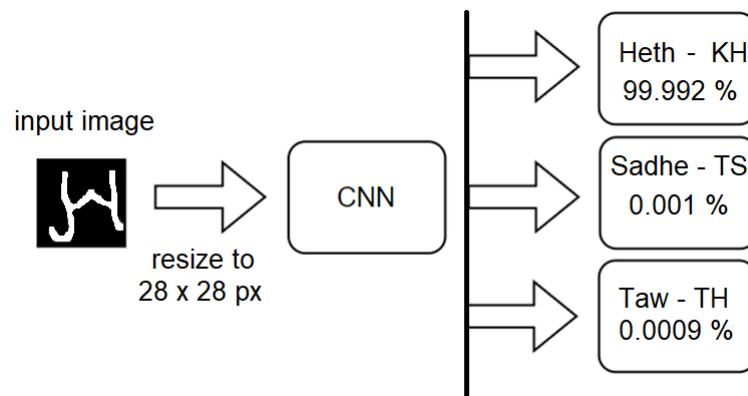


Fig. 1 In-app character classification.



Fig. 2 Example of a sandstone tablet containing Palmyrene script, Inv. 2983/9507, © The Archaeological Museum Of Palmyra.

use the transliteration for educational purposes. The Android application can also be modified if another historical alphabet model is trained. Therefore, it can be used for further semi-automatic transliteration of any alphabet.

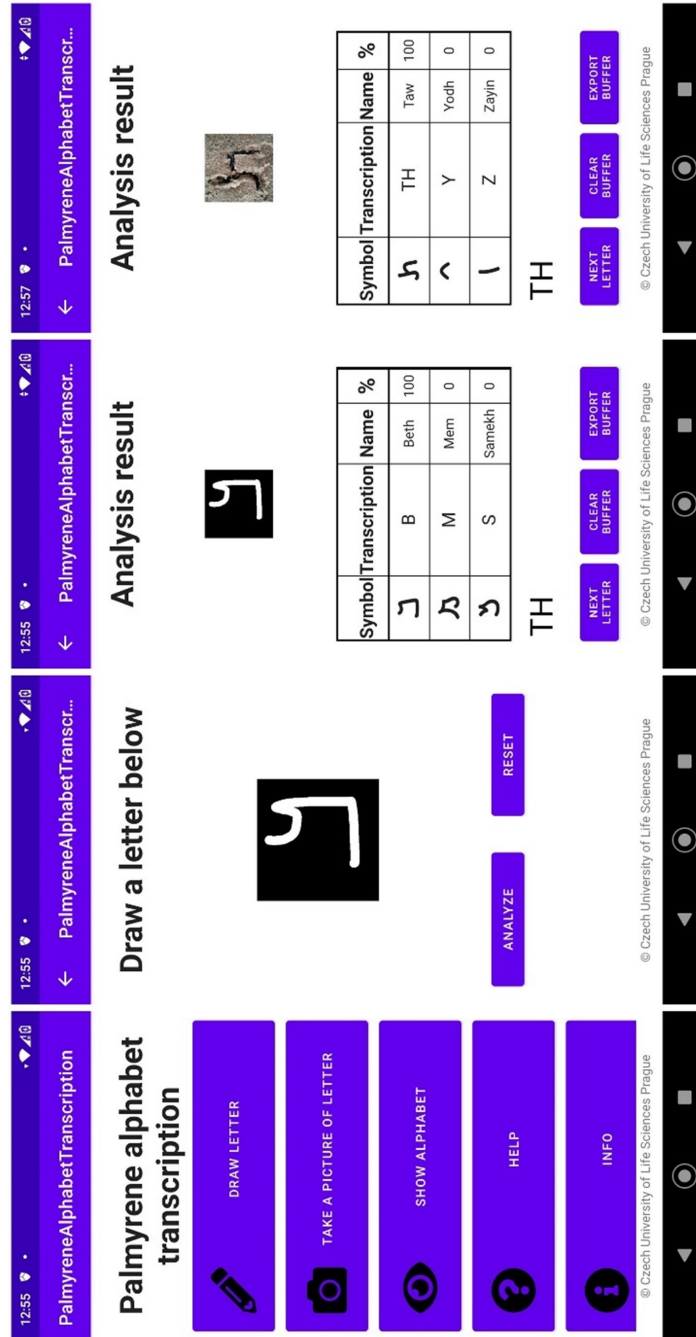


Fig. 3 “Palmyrene Alphabet Transcription” mobile application user interface.

2. Objective

Our goal was to design a suitable classifier for transcribing Palmyrene characters into Latin, with special regard to its use in mobile applications. Current research suggests using MobileNet classifier. Therefore, our first step was to verify the possibility of using CNN with the MobileNet architecture and to train it on the Palmyrene alphabet character set. Because the results obtained were not satisfactory, we designed our own CNN architecture, trained it and then we compared the results with the results obtained using MobileNet.

Our results confirmed that it is possible to design CNN architecture of the classifier that gives better results than MobileNet based classifier. It is likely that this classifier would also give good results if trained for transcription of some another similar alphabet.

3. Building the classifier

3.1 Training and validation set

In order to create a dataset of hand-written characters, we modified Axel Thevenot's "Python-Interface-to-Create-Handwrittendataset" tool available at Github [18]. The letters were transcribed from a large number of photographs containing Palmyrene alphabet, such as in Fig. 2. For acquiring these photographs of tablets with Palmyrene inscriptions, we established a cooperation with two museums – Musée du Louvre in Paris [9] as well as Virtual Museum Of Palmyra. [11].

Palmyra alphabet consists of 32 characters, see transcription table in Fig. 4. In our research we considered only 28 characters. We excluded four characters – the numbers 2, 3, 4 and 5. The numbers 2–4 are sequences of the characters 1, and the number 5 looks exactly the same as the letter "ayin". Using graphic tablet, we wrote 56197 letters in total (exactly 2007 samples per each character).

The used system font for Palmyrene is "palmme". Each character class is saved in a different folder with character name and using keras ImageDataGenerator.flow_from_directory function, the dataset is converted to CNN-readable format. With the generator, the data is split into two subsets – training set contains 80 % data and validation set contains the remaining 20 %. We did not use any data augmentation method.

3.2 MobileNet based architecture

Our first choice of mobile network architecture was picked according to the current recommendations — MobileNet efficient_lite0. This architecture consists of a HubKerasV1V2 layer, Dropout layer to prevent overfitting and an output Dense layer. The final activation function is softmax, as the problem solved is a multiple category classification.

The training of the efficient_lite0 model (both photographic and hand-written) used 80 % images for training and 20 % for validation. For model creation, we used the library "tfite-model-maker" and did not alternate the architecture.

Palmyrene	Latin	Transcription
Ⲁ	Aleph	'/A
Ⲃ	Beth	B
Ⲅ	Gimmel	G
Ⲇ	Daleth	D
Ⲉ	He	H
Ⲋ	Waw	V
Ⲍ	Zayin	Z
Ⲏ	Heth	KH
Ⲑ	Teth	T
Ⲓ	Yodh	Y
Ⲕ	Kaph	K/C
Ⲗ	Lamedh	L
Ⲙ	Mem	M
Ⲛ	Final Nun	M
Ⲝ	Nun	N
Ⲟ	Samekh	S
Ⲡ	Ayin	'/E
Ⲣ	Pe	P
Ⲥ	Sadhe	TS
Ⲧ	Qoph	Q
Ⲩ	Resh	R
Ⲫ	Shin	SH
Ⲭ	Taw	TH
Ⲯ	left	
Ⲱ	right	
Ⲳ	1	
Ⲵ	2	
Ⲷ	3	
Ⲹ	4	
Ⲻ	5	
Ⲽ	10, 100	
Ⲿ	20	

Fig. 4 Palmyrene characters and transcription to Latin.

For creating dataset, we call the `tfLite_model_maker.image_classifier.DataLoader` method. By calling this method, input images are resized to 224×224 pixels and loaded into a data generator.

The network is trained calling the function “model.create”, which runs training for 5 epochs, with batch size 128 images. The core of the network is not trained, as there are only 38430 trainable parameters and 3451454 non-trainable parameters.

Model summary is specified below.

Model:	“sequential”	
Layer (type)	Output Shape	Param #
Hub.keras_layer_v1v2 (HubKerasLayerV1V2)	(None, 1280)	3413024
dropout (Dropout)	(None, 1280)	0
dense (Dense)	(None, 30)	38430
Total params:	3451454	
Trainable params:	38430	
Non-trainable params:	3413024	

Tab. I *MobileNet summary.*

The training accuracy was very high even after the first epoch, it reached 0.902, and in the last epoch it reached 0.993. The validation accuracy was 0.677. Model is then saved in .tflite format.

3.3 Design of the custom CNN architecture

To design the network architecture, we conduct experiments with CNN layers. We compare the influence of the number of convolutions in each Convolutional layer, the influence of leaving out pooling layers, compare the difference in performance of AveragePooling and MaxPooling layers and research the influence of the number of repetition of Convolutional/Pooling blocks. We then pick the architecture, which had the highest validation accuracy, lowest validation error and lowest validation loss, and convert it to a mobile version of the model – “tflite” for testing.

The training is conducted on the graphic card NVIDIA GeForce GTX 970 with memory clock rate 1.1775 GHz, 1664 CUDA cores and memory size 8159 MB.

We created 10 versions of CNN architectures and researched the influence of the combination of layers and numbers of convolutions on a small dataset of handwritten letters (153 per class, image size 28×28). The results are visible in Tab. II, where V is version, Acc_i is accuracy in given epoch i , $Loss_i$ is loss in given epoch i , V_Acc_i is validation accuracy in given epoch i , V_Loss_i is validation loss in given epoch i .

Each architecture version was using alternation of Convolutional layers with specified number of convolutions and Pooling layers, either MaxPooling or AveragePooling. The last three layers were always Flatten and two Dense layers. The versions are described in the following Tab. III, where $Conv_i$ means the number of convolutions in each i -th Convolutional layer.

As visible in Tab. II, the combination of low validation loss and high validation accuracy was present in models using the straight alternation of Convolutional and MaxPooling layers, with 3 or 4 such blocks (mostly versions 1 and 3). Adding another Convolutional / MaxPooling block (version 7) lowered the validation accuracy from 0.548 to 0.4967 in the last epoch and increased the validation loss

<i>V</i>	<i>Acc_1</i>	<i>Loss_1</i>	<i>V_Acc_1</i>	<i>V_Loss_1</i>	<i>Acc_15</i>	<i>Loss_15</i>	<i>V_Acc_15</i>	<i>V_Loss_15</i>
1	0.2992	2.74900	0.3326	2.6634	0.9688	0.0583	0.5301	2.5247
2	0.7730	0.79460	0.4330	3.3265	0.9648	0.0511	0.4196	2.7923
3	0.2054	2.87113	0.2366	2.8368	0.9720	0.0550	0.5480	2.6796
4	0.0649	3.25480	0.1373	3.0333	0.9389	0.1334	0.4085	5.2541
5	2.8557	0.26200	0.2334	3.2072	0.9960	0.0758	0.4542	4.2724
6	3.2109	0.08200	0.1652	3.1675	0.9626	0.0805	0.4743	4.5598
7	3.0733	0.11790	0.1417	3.2524	0.9628	0.0621	0.4967	4.0713
8	1.4428	0.59240	0.3571	2.9419	0.9658	0.0499	0.4877	2.9082
9	3.1120	0.12330	0.1730	3.0210	0.9659	0.6030	0.5580	2.9961
10	3.2842	0.08940	0.1696	3.0293	0.9652	0.0928	0.5022	3.5259

Tab. II *Influence of CNN layers on network performance.*

<i>V</i>	Conv_1	Conv_2	Conv_3	Conv_4	Conv_5	Pool
1	16	32	64			Max
2	32	64	128			Max
3	16	32	64	128		Max
4	16	32	64	128		Avg
5	32	64	128			Avg
6	32	64	128	256		Avg
7	16	32	64	128	256	Avg
8	16	32	64			Max
9	16	32	64	64		Max
10	16	32	32	32		Max

Tab. III *CNN versions description.*

from 2.6796 to 4.0713. Using AveragePooling layers conducted in lower validation accuracy to 0.4085 in version 4 and almost doubled the validation loss to 5.2541 in comparison to using MaxPooling layers. The best performing architecture overall was version 3.

3.4 Training of the new CNN model

We used the training / validation split equal to 0.8 / 0.2 using the library Image-DataGenerator from tensorflow.keras. The images are resized to 28×28 pixels.

We picked the best performing model with 4 Convolutional layers alternated by 4 MaxPooling layers (version 3). The model summary is visible below.

We trained the network for 15 epochs, with batch size 128, selected optimizer was “adam”. The training results are visible in Tab. V.

Model:	“sequential”	
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 28, 28, 16)	448
max_pooling2d (MaxPooling2D)	(None, 14, 14, 16)	0
conv2d_1 (Conv2D)	(None, 14, 14, 32)	4640
max_pooling2d_1 (MaxPooling2)	(None, 7, 7, 32)	0
conv2d_2 (Conv2D)	(None, 7, 7, 64)	18496
max_pooling2d_2 (MaxPooling2)	(None, 3, 3, 64)	0
conv2d_3 (Conv2D)	(None, 3, 3, 128)	73856
max_pooling2d_3 (MaxPooling2)	(None, 1, 1, 128)	0
flatten (Flatten)	(None, 128)	0
dense (Dense)	(None, 512)	66048
dense_1 (Dense)	(None, 28)	14364
Total params:	177852	
Trainable params:	177852	
Non-trainable params:	0	

Tab. IV Custom CNN summary.

Acc_1	Loss_1	V_Acc_1	V_Loss_1	Acc_15	Loss_15	V_Acc_15	V_Loss_15
0.861	0.4718	0.9255	0.3092	1	4.66	0.9626	0.3315

Tab. V Results of final model training.

4. Evaluation on testing set

The testing dataset is not created in advance. Testing is conducted directly in the mobile application, and the characters need to be written by hand in the “Draw Letter” module. We use 10 samples in each class for testing of both CNN architectures (280 images in total).

4.1 Metrics

For classifier evaluation, the metrics accuracy acc , error err , recall r Eq. (1), precision p Eq. (2) and F-score Eq. (3) are used. [2] Accuracy acc is the mean of correctly classified characters, error err is the mean of incorrectly classified characters. Parameters of recall Eq. (2), precision Eq. (1) and F-score Eq. (3) are evaluated as follows. For each category i we consider binary decision whether character belongs to the category i versus it belongs to any other category $j \neq i$ and we calculate precision p_i , recall r_i and F $_i$ -score.

$$r_i = \sum_{i=1}^m \frac{TP_i}{TP_i + FP_i}, \quad (1)$$

$$p_i = \sum_{i=1}^m \frac{TP_i}{TP_i + FN_i}, \quad (2)$$

$$F_i = \frac{2 \cdot r_i \cdot p_i}{r_i + p_i}, \quad (3)$$

where

- TP_i is the number of correctly classified objects from category i
- FP_i is the number of characters from the category $j \neq i$ incorrectly classified as being characters from category i
- FN_i is the number of characters from category i incorrectly classified as being characters from some another category $j \neq i$
- m is the number of categories.

The overall parameters of recall r Eq. (4), precision p Eq. (5) and F score Eq. (6) are then evaluated as arithmetic means.

$$r = \frac{1}{m} \sum_{i=1}^m r_i, \quad (4)$$

$$p = \frac{1}{m} \sum_{i=1}^m p_i, \quad (5)$$

$$F = \frac{1}{m} \sum_{i=1}^m F_i. \quad (6)$$

4.2 Results of MobileNet and custom CNN classifier

The detailed results of MobileNet classifier evaluation on testing set are visible in Tab. VI.

The mean error of this classifier is 0.311, while the mean accuracy reached 0.689.

The least recognized Palmyrene character is “20” and “mem” with only 1 (out of 10) true positive recognition. The character “20” was otherwise classified as “pe” and “waw”, while “mem” was classified as “beth”, “nun” and “pe”. 9 characters – “10”, “aleph”, “beth”, “he”, “nun”, “nun_final”, “pe”, “shin” and “waw” were recognized in each case (10 out of 10).

The classifier was over-oriented for character “pe”, as it had most false positive predictions – 28 other letters were classified as “pe”, as it is visually similar to other characters. The second character with most false positive predictions was “nun” with 18 false classifications and the third one was “nun_final” with 9 false positives.

The detailed results of custom CNN classifier evaluation on testing set are visible in Tab. VII. The mean error of custom CNN classifier is only 0.018, while the accuracy reached 0.982, which is a significant 142% improvement from the MobileNet classifier.

The least recognized character is “pe” with 8 true positives out of 10, one was classified as “nun” and the other as “yodh”. 24 characters were recognized in all 10 cases out of 10, 3 had one false negative prediction – “gimel”, “resh” and “sadhe”.

There were only 5 characters with false positives – “20”, “daleth”, “heth”, “nun” and “yodh”, each of them had 1 false positive prediction.

class	TP_i	FN_i	FP_i	r_i	p_i	F_i	err	acc
1	8	2	2	0.8	0.8	0.8		
10	10	0	0	1	1	1		
20	1	9	0	0.1	1	0.182		
aleph	10	0	9	1	0.526	0.690		
ayin	6	4	0	0.6	1	0.75		
beth	10	0	7	1	0.588	0.741		
daleth	5	5	0	0.5	1	0.667		
gimmel	4	6	0	0.4	1	0.571		
he	10	0	0	1	1	1		
heth	9	1	1	0.9	0.9	0.9		
kaph	8	2	2	0.8	0.8	0.8		
lamedh	7	3	0	0.7	1	0.824		
left	2	8	5	0.2	0.286	0.235		
mem	1	9	0	0.1	1	0.182		
nun	10	0	18	1	0.357	0.526		
nun_final	10	0	9	1	0.526	0.690		
pe	10	0	26	1	0.278	0.435		
qoph	7	3	1	0.7	0.875	0.778		
resh	3	7	0	0.3	1	0.462		
right	5	5	0	0.5	1	0.667		
sadhe	4	6	0	0.4	1	0.571		
samekh	3	7	0	0.3	1	0.462		
shin	10	0	2	1	0.833	0.909		
taw	5	5	1	0.5	0.833	0.625		
teth	8	2	0	0.8	1	0.889		
waw	10	0	4	1	0.714	0.833		
yodh	8	2	2	0.8	0.8	0.8		
zayin	7	3	0	0.7	1	0.824		
mean				0.682	0.826	0.672	0.311	0.689

Tab. VI *MobileNet classifier evaluation.*

5. Discussion

The results of the Palmyrene hand-written characters classification were satisfactory, as the accuracy reached 98.21% instead of 68.93% with MobileNet efficient_lite0.

The reason behind the false predictions when using MobileNet is the visual similarity of Palmyrene symbols. In sample datasets used for object detection with MobileNet image classifiers, there are many distinct features that makes the classification easier. Such objects like dogs, cats etc. can be stretched, rotated and shifted within the image and still be recognized and classified correctly, however, in case of characters of alphabets, the precise position, shape and rotation of letters matter and can not be altered, because it would change the meaning of the letter, as some letters look like others if rotated or stretched. The demonstration of such

class	TP_i	FN_i	FP_i	r_i	p_i	F_i	err	acc
1	10	0	0	1	1	1		
10	10	0	0	1	1	1		
20	10	0	1	1	0.909	0.952		
aleph	10	0	0	1	1	1		
ayin	10	0	0	1	1	1		
beth	10	0	0	1	1	1		
daleth	10	0	1	1	0.909	0.952		
gimmel	9	1	0	0.9	1	0.947		
he	10	0	0	1	1	1		
heth	10	0	1	1	0.909	0.952		
kaph	10	0	0	1	1	1		
lamedh	10	0	0	1	1	1		
left	10	0	0	1	1	1		
mem	10	0	0	1	1	1		
nun	10	0	1	1	0.909	0.952		
nun_final	10	0	0	1	1	1		
pe	8	2	0	0.8	1	0.889		
qoph	10	0	0	1	1	1		
resh	9	1	0	0.9	1	0.947		
right	10	0	0	1	1	1		
sadhe	9	1	0	0.9	1	0.947		
samekh	10	0	0	1	1	1		
shin	10	0	0	1	1	1		
taw	10	0	0	1	1	1		
teth	10	0	0	1	1	1		
waw	10	0	0	1	1	1		
yodh	10	0	1	1	0.909	0.952		
zayin	10	0	0	1	1	1		
mean				0.982	0.984	0.982	0.018	0.982

Tab. VII Results of model with $4 \times \text{Conv}/\text{MaxPooling}$ network architecture.

similarity is visible in Fig. 5. The most similar letter to “pe” is “20” and so it had the highest classification error with MobileNet.

When using custom classifier, the improvement is significant (142% better), as the CNN architecture was tested especially for letter classification and is less prone to error when classifying objects with less distinct features.

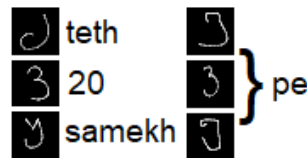


Fig. 5 Visual similarity of character similarity.

Mara H. conducted the analysis of 3-dimensional scans of tablets with cuneiform signs, however the success rate is not presented in the research. [10] Yamauchi researched hand-written cuneiform characters, but also did not publish classifier results. [19]

Ghosh et al.'s model, that recognized Bangladeshi signs, reached 96.46% accuracy on MobileNet, which is 2.4% less than with our upgraded CNN hand-written model. The Bangladeshi script contains 60 letters [3]. Its hand-written were also analysed and reached 90.27% accuracy. [13]

Our classifier is also comparable with other object recognition tasks, for example Cho Junghwan et al. have researched CT body scans. The dataset contained 4000 very high quality images and they reached 97% accuracy on GoogLeNet Inception v1 architecture. They also described, how the results declined if less images were used, accordingly [8].

The F-score of 83% has been reached in the task of great tits and carried food recognition by part of our team. We analysed photos from Smart Nest Boxes. For the model, we used YOLOv3 architecture. The F-measure was lower due to difficult object detection. [6]

Our results of Palmyrene letters recognition were therefore comparable with other author's works. The classifier results were satisfactory, over 70% as initially stated in the success criteria. With a 98.21% classification success, the task of hand-written Palmyrene characters classification can be considered resolved.

6. Conclusion

We have explored the architectures suitable for character recognition for mobile use, which is an ever evolving area. Letters and numbers classification is a special image classification case, as, unlike other objects, the images of alphabet characters can not be manipulated rotation-wise, shape-wise and shift-wise. We conducted experiments with convolutional neural network architectures special for character recognition on Android devices, aiming to improve the classification in comparison with MobileNet, and found out, that the network with 4 Convolutional layers alternated by MaxPooling layers has better classification results than other tested networks and trained this network on our data and improved hand-written classifier results by 142%.

We updated the model in our software tool, which uses artificial intelligence for semi-automation of historical alphabets transliteration and proved its function on Palmyrene Aramaic script. From a general point of view, we can state that if a different model is trained on another alphabet, using the same architecture and mobile application (using different dataset of letters, and with some alternations of in-app texts), this research can serve as a template for other historical script analysis and a foundation of historical optical character recognition (OCR) algorithms.

There is still room for improvement in performance of the photographic model, which is still run on MobileNet and has only 440 images per class in the dataset. We plan to expand the set using keras augmentation and to develop generative adversarial networks. We also plan to create a web application for Palmyrene alphabet recognition, where we will also implement rows recognition and character

segmentation, creating a Palmyrene OCR, aiding researchers with transliterating Palmyrene Aramaic texts in field use and thus contributing to biblical studies. The aim of the next steps of this research is however not just to create a Palmyrene OCR, but to suggest neural network architectures for any historical alphabet character detection, segmentation and classification on mobile and improve the state of art of creating mobile OCR.

Acknowledgement

This article was created with the support of the Internal Grant Agency (IGA) Faculty of Management and Economy, Czech University of Life Sciences in Prague, 2021A0004 – “Reading the characters of Palmyrene alphabet using artificial intelligence tools”.

References

- [1] EVERSON M. *Proposal for encoding the Palmyrene script in the SMP of the UCS* [online]. UC Berkeley: Department of Linguistics, 2010. Available from: <https://escholarship.org/uc/item/27b327h7>.
- [2] GÉRON A. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition*. O’Reilly, 2019, pp. 88–92.
- [3] GHOSH T., ABEDIN M., CHOWDHURY S., TASNIM Z., KARIM T., REZA S., SAIKA S., YOUSUF M. Bangla handwritten character recognition using MobileNet V1 architecture. *Bulletin of Electrical Engineering and Informatics*. 2020, 9(6), pp. 2547–2554, doi: [10.11591/eei.v9i6.2234](https://doi.org/10.11591/eei.v9i6.2234).
- [4] GUPTA D. Mobile Application for Bird Species Identification Using Transfer Learning. In: *2021 IEEE International Conference on Artificial Intelligence in Engineering and Technology (ICAIET)*. 2021, pp. 1–6, doi: [10.1109/IICAIET51634.2021.9573796](https://doi.org/10.1109/IICAIET51634.2021.9573796).
- [5] HAJIHASHEMI V., ARAB AMERI M.M., ALAVI GHARAHBAGH A., BASTANFARD A. A pattern recognition based Holographic Graph Neuron for Persian alphabet recognition. In: *2020 International Conference on Machine Vision and Image Processing (MVIP)*, 2020, pp. 1–6, doi: [10.1109/MVIP49855.2020.9116913](https://doi.org/10.1109/MVIP49855.2020.9116913).
- [6] HAMPLOVÁ A., PAVLÍČEK J. Object Recognition Tool for “Smart Nest Boxes.”. In: *Proceedings of IAC in Budapest 2020. IAC-ETITAI.*, Prague, Czech republic: Czech Institute of Academic Education, 2020, pp. 105–109.
- [7] HILLERS D., CUSSINI E. *Palmyrene Aramaic Texts*. Baltimore: Johns Hopkins Univ. Press, 1996.
- [8] CHO J., LEE K., SHIN E., CHOY G., DO S. *How much data is needed to train a medical image deep learning system to achieve necessary high accuracy?*, 2015. Available from: <https://arxiv.org/pdf/1511.06348.pdf>
- [9] *Le Louvre. Louvre Museum Official Website*. [viewed 2021-12-01]. Available from: <https://www.louvre.fr/en>.
- [10] MARA H., KRÖMKER S., JAKOB S., BREUCKMANN B. GigaMesh and Gilgamesh – 3D Multiscale Integral Invariant Cuneiform Character Extraction. In: *The 11th International Symposium on Virtual Reality, Archaeology and Cultural Heritage. VAST 2010.*, 2010, doi: [10.2312/VAST/VAST10/131-138](https://doi.org/10.2312/VAST/VAST10/131-138).
- [11] *Palmyra Archaeological Museum*. The Archaeological Museum Of Palmyra, 2021 [viewed 2021-12-01]. Available from: <https://virtual-museum-syria.org/palmyra/>.
- [12] *Palmyrene, Range: 10860–1087F* [pdf]. 2010. Available from: <https://www.unicode.org/charts/PDF/U10860.pdf>.




- [13] SAZAL M.M.R., BISWAS S.K., AMIN M.F., MURASE K. Bangla handwritten character recognition using deep belief network. In: *2013 International Conference on Electrical Information and Communication Technology (EICT)*, 2013, pp. 1–5, doi: [10.1109/EICT.2014.6777907](https://doi.org/10.1109/EICT.2014.6777907).
- [14] SONAWANE P., DROLIA S., SHAMSI S., JAIN B. *Self-Supervised Visual Representation Learning Using Lightweight Architectures*, 2021. Available from: <https://arxiv.org/pdf/2110.11160.pdf>.
- [15] TAYLOR D.G.K. An Annotated Index of Dated Palmyrene Aramaic Texts. *Journal of Semitic Studies*. 2001, 16(2), pp. 203–219, doi: [10.1093/jss/XLVI.2.203](https://doi.org/10.1093/jss/XLVI.2.203).
- [16] TAN M., LE Q.V. *Efficientnet: Rethinking model scaling for convolutional neural networks*, 2020. Available from: <https://arxiv.org/pdf/1905.11946.pdf>
- [17] *Image Classification — Tensorflow*. Tensorflow.org, 2021 [viewed 2021-01-14]. Available from: https://www.tensorflow.org/lite/examples/image_classification/overview.
- [18] THEVENOT A. Python Interface to Create Handwritten dataset [software]. Available from: <https://github.com/AxelThevenot/Python-Interface-to-Create-Handwritten-dataset>
- [19] YAMAUCHI K., YAMAMOTO H., MORI W. Building A Handwritten Cuneiform Character Imageset. In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation. LREC 2018.*, Miyazaki, Japan, 2018. Available from: aclweb.org/anthology/L18-1115.

Attachment 3

D. Franc, A. Hamplová and O. Svojshe, “Augmenting Historical Alphabet Datasets Using Generative Adversarial Networks,” Data Science and Algorithms in Systems. CoMeSySo 2022. Lecture Notes in Networks and Systems, vol. 597, pp. 132-141, 04 01 2023.



Augmenting Historical Alphabet Datasets Using Generative Adversarial Networks

David Franc^(✉) , Adéla Hamplová , and Ondřej Svojsě 

Czech University of Life Science in Prague, Kamycka 129, Prague 16500, Czech Republic
{francd, hamplova, svojse}@pef.czu.cz

Abstract. In this paper, we present a method for expanding small classification datasets. Every research project is based on data and methods, including text analysis. When analyzing historical texts in different alphabets, there are not always Optical Character Recognition algorithms available and, in many cases, such texts need to be transliterated and translated manually, or alternatively, an OCR algorithm can be developed. In order to create such an algorithm, a large volume of input data is needed - each alphabet consists of elementary data - either letters, vowels, or in some cases ideograms. The texts need to be segmented into such elements, and then, the elements are classified. In many cases, it is very difficult and time-costly to get a sufficient amount of data, and it is advisable to use augmentation methods. In our research, we propose using Generative Adversarial Network to expand a relatively small dataset of Palmyrene letters and prove that even by adding generated data equal to the third of size of the original dataset, the classification results are improved by 120%.

Keywords: GAN · Palmyrene alphabet · Data science · Data augmentation

1 Introduction

The fundamental part of each object detection and image classification problem is a sufficient amount of visual data. Depending on the object that is classified, they vary from high quality satellite photos when detecting roads [1] or moving objects [2] to small images of letters in the historical alphabet OCR, for example Arabic OCR Dataset [3].

Even a sample dataset like MNIST [4] uses a very large number - more than 60 thousand of small images 28×28 pixels of 10 classes of handwritten Arabic digits 0–9. In general, the recommended number of images per class, for an object to be classifiable, is about 1000 [5].

It is not always possible to get a sufficient number of such images when classifying historical documents. There may not be enough visual data available and getting more data may be very difficult. Therefore, we suggest augmentation methods for historical alphabet datasets and prove it in our research of classifying Palmyrene signs on mobile phones [6].

1.1 Letter Classification

Image classification is the process of sorting such visual files into classes based on features in the images content. The most common Artificial Intelligence algorithms used for image classification are Convolutional Neural Networks, and for analyzing words or sentences, a combination of Convolutional and Recurrent layers is usually used [7].

Classification of letters is usually part of Optical Character Recognition; however it can be done by itself like in the case of Javanese [8], Czech [9] or Palmyrene [6] alphabets. Optical Character Recognition is using segmentation and classification, and for analyzing text sequences, Recurrent Neural Networks with LSTM layers [10].

During classification, images are resized to evenly sized squares, as the input to the classifier requires. Therefore, when training a classifier, it is more convenient to work with square inputs directly, to prevent letter distortion (Fig. 1).



Fig. 1. Photo of Palmyrene letter “He” in square image.

1.2 Getting Data for Letter Classification from Historical Documents

Extracting letters from historical documents can be done by using several methods. When using mathematical methods for letter extraction, a feature extraction like Independent Component Analysis or Principal Component Analysis is used, and then, the letter is classified by a classifier - either SVM or Neural Network [11]. In 2011, Meyer decomposition was used as a feature extraction method [12].

For classification purposes, a dataset of separate letters is necessary, therefore, the most reliable, zero error way, is to cut out the letters manually and sort them out in folders, each folder representing one sign class. Any automatic feature extraction method is prone to error.

Manual method is very time-costly and only a limited number of letters can be processed this way. Therefore, it is best to combine the methods mentioned above.

1.3 Current State of Art of Palmyrene Alphabet Classification

Until March 2022, there was no Palmyrene alphabet transliteration published. In our previous research [6], we have created two classifiers trained on hand-made data, which are run on a mobile phone with Android operating system, the models are saved in.tflite format.

The first one of them is transliterating hand-written signs, it was trained on a dataset of 56207 images (2007 per class) with the size 28×28 pixels and reached 98.21% classification accuracy on testing set. The second one was trained on cut out photos, 440 images per class in mean. The classification accuracy of photos of signs in the testing set reached only 41,13% on the same architecture, which is caused by a higher complexity of photos and a significantly lower number of images in the dataset. It is therefore necessary to increase the number of photos in order to get better results.

2 Methods of Image Augmentation

2.1 Keras Generator

The simplest way to increase the number of images in a dataset is to use preprocessing methods implemented in Keras libraries that allow resizing, random cropping, rotation, flip, translation, zoom, changing height, width, contrast and brightness [13]. However, most of these methods are not desirable, as alphabet signs need to be kept in the same orientation and undistorted.

2.2 Roboflow

Roboflow [14] is a platform used for different machine learning projects, such as object detection, single or multi-label classification, instance or semantic segmentation, key-point detection and others. It allows the researchers to annotate data in-app, generate datasets, and train machine learning models.

For the purposes of this paper, we only mention the augmentation options. They can be divided into desirable and undesirable augmentation for sign detection.

Desirable augmentation options include:

- auto-adjust contrast
- greyscale
- hue
- saturation
- brightness
- exposure
- blur
- noise.

Undesirable augmentation options are:

- auto-orient
- resize
- static crop
- tile
- modify classes

- flip
- 90° rotate
- crop
- rotation
- shear
- cutout
- mosaic.

3 Generative Adversarial Networks

Both Keras generator and Roboflow work with the images that we manually annotate or classify, and only apply filters on them, but the output are no new images. In order to create images that are different from input images, but resemble real inputs, we suggest using Generative Adversarial Networks - GANs.

GAN is one of the architectures that was designed by Ian Goodfellow et al. in 2014. It is a generative network that aims to generate data that resembles real inputs at most. The most well-known example is photo editing, where unwanted objects can be replaced with a believable background. Another example is to generate photos of people who never existed, or to increase the quality and resolution of a small image [15].

Augmentation using GANs has been already used, for example, in research on the recognition of liver lesions from medical images. In that field of research, it is also complicated to gather data [16].

4 Using GAN for Palmyrene Alphabet Augmentation

The Palmyrene alphabet has 28 characters (there are more, but the letter “ayin” looks like number five and numbers 2 and 3 look like digits one next to each other). Therefore, we trained 28 separated GANs - one for each character.

Some characters have more variants, but we decided we would train only one generator per sign and examine the results.

4.1 Network Architecture

Every GAN consists of a discriminating network that is classifying real inputs from generated inputs, and a generating network.

At the start of the training, the generator generates random noise and during training, it adapts features from the discriminator, and generates images, which are more and more similar to real inputs. The discrimination improves in the classification of real from fake images and learns more detailed features, but they are handed over to generator. In the end, it results in the generator generating almost real images.

The architecture of our GAN is visible in Table 1.

4.2 Training Parameters

We have trained these networks on Google Colab with GPU Hardware acceleration.

Training was executed for 200 epochs, 32 images in batch, codings size 100. The training of each GAN took approximately 20–30 min.

Table 1. GAN architecture.

Layer (type)	Output shape	Param #
DISCRIMINATOR		
conv2d (Conv2D)	(None, 50, 50, 64)	1664
dropout (Dropout)	(None, 50, 50, 64)	0
conv2d_1 (Conv2D)	(None, 25, 25, 128)	204928
dropout_1 (Dropout)	(None, 25, 25, 128)	0
flatten (Flatten)	(None, 80000)	0
dense_1 (Dense)	(None, 1)	0
conv2d_3 (Conv2D)	(None, 12, 12, 128)	73856
GENERATOR		
dense (Dense)	(None, 80000)	8080000
reshape (Reshape)	(None, 25, 25, 128)	0
batch_normalization (BatchNormalization)	(None, 25, 25, 128)	512
conv2d_transpose (Conv2DTranspose)	(None, 50, 50, 64)	204864
batch_normalization_1 (BatchNormalization)	(None, 50, 50, 64)	256
conv2d_transpose_1 (Conv2DTranspose)	(None, 100, 100, 1)	1601

Total params: 8,573,826

Trainable params: 8,573,442

Non-trainable params: 384

5 Results

We have trained 28 GANs, one for each Palmyrene sign, and generated new data with them, then trained two classifiers without and with generated data added to the dataset.

The results are presented below.

5.1 GAN Success Rate

In total, there were 12824 images generated, 458 by each class, and correct images were hand-picked. 4089 of generated images looked like the real Palmyrene signs, such as in Fig. 2, and were kept.

There was a high variability of kept images in each class, as visible in Table 2. The generators for signs “teth” and “sadhe” generated less than 10% of the images correctly, which is caused by a high variability of the signs themselves. The generator tends to combine the features of all variants and mix them together, which results in badly generated images.

However, there was a success rate between 73 – 78% for the signs with a low variability, such as “left,” “zayin” and “1” (Fig. 3).



Fig. 2. One of the images from “nun-final” GAN generator.

Table 2. Palmyrene letter generator success rate.

Generated images kept	Palmyrene signs	Percent of generated images kept (%)
< 50	teth, sadhe	5 – 8
50 – 100	lamedh, ayin, daleth, pe, resh, gimel, heth, nun, shin, mem, yodh,	10 – 20
100 – 150	beth, taw, kaph, waw, samekh, he	21 – 28
150 – 200	20, nun_final	40 – 44
200 – 300	aleph, right, 10, qoph	46 – 66
> 300	left, zayin, l	73 – 78

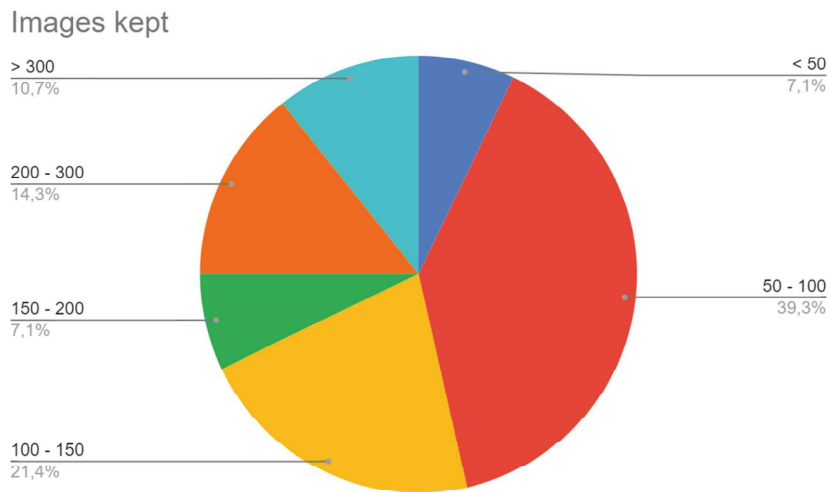


Fig. 3. Percentage of kept generated images.

Table 3. Validation loss during training of both classifiers.

Classifier	Accuracy after epoch 1	Validation accuracy after epoch 1	Accuracy after epoch 20	Validation accuracy after epoch 20
Without generated images	0.5166	0.4831	0.9998	0.6463
With generated images	0.5969	0.6791	0.9937	0.7791

5.2 Classifier Improvement

Both classifiers have been trained for 20 epochs, 128 images with size 100×100 pixels in a batch.

The first classifier was trained on 12101 images without augmentation. For the second one, generated images were added. In total there were 16190 images. Both classifiers were trained on the same following architecture (sequential model).

Table 4. Classifier architecture.

Layer (type)	Output shape	Param #
conv2d (Conv2D)	(None, 100, 100, 16)	160
max_pooling2d (MaxPooling2D)	(None, 50, 50, 16)	0
conv2d_1 (Conv2D)	(None, 50, 50, 32)	4640
max_pooling2d_1 (MaxPooling2D)	(None, 25, 25, 32)	0
conv2d_2 (Conv2D)	(None, 25, 25, 64)	18496
max_pooling2d_2 (MaxPooling2D)	(None, 12, 12, 64)	0
conv2d_3 (Conv2D)	(None, 12, 12, 128)	73856
max_pooling2d_3 (MaxPooling2D)	(None, 6, 6, 128)	0

Total params: 2,471,324

Trainable params: 2,471,324

Non-trainable params: 0

As visible in Table 3, the increase in the volume of dataset had a positive influence on the classification results. While without augmentation, the classifier reached 0.6463 accuracy on validation subset at the last epoch; by adding generated images, this improved by 120,55% to 0.7791. The loss and accuracy graphs are visible in Figs. 4 and 5 (Table 4).

6 Discussion

In 2018 there was research aimed to expand datasets with GAN for liver lesion recognition. Originally, they had only 182 real images of liver lesions and the precision and

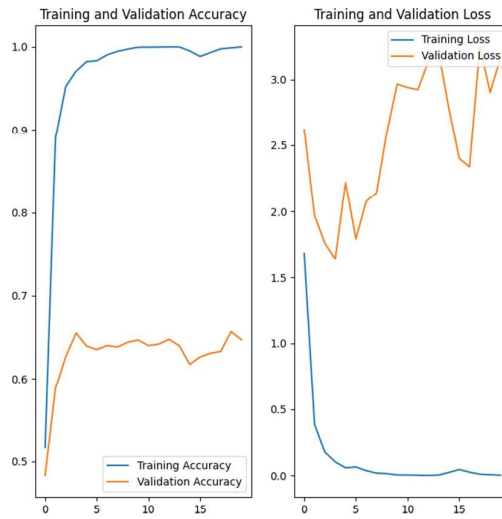


Fig. 4. Loss and accuracy graph of non-augmented classifier.

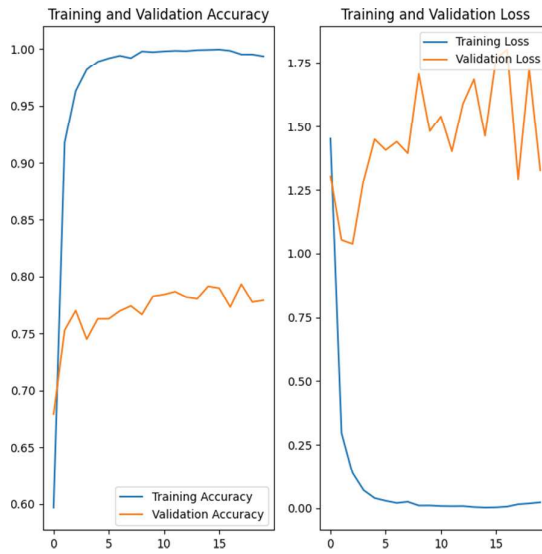


Fig. 5. Loss and accuracy graph of augmented classifier.

recall reached 88.4 and 78.6%. By adding 120 generated images, the classification results were boosted to 92.4 and 85.7% [16]. In the same year, another research used GAN for brain analysis - segmentation of cortical CSF, brain stem CSF and ventricular CSF. They had 80000 real images and 40000 generated ones; the segmentation improved slightly as well [17].

Our classifier accuracy increased from 64 to 77%. When generating Palmyrene signs, the success rate of generated images varied between less than 100 to almost 400 from 458. Some Palmyrene signs have more variants, and the generators tend to combine the features of all variants and in the end the generated images don't look like real signs. However, low variety signs were generated very well.

The solution to this problem is creating multiple separate generators for each variant of one sign.

7 Conclusion

We have analyzed augmentation options for small datasets in order to improve classification results and trained Generative Adversarial Networks for augmenting the photographic dataset of 28 letters and numbers from the Palmyrene alphabet, which is used to train the Convolutional Neural Network - a classifier of photos - in the previously published mobile application "Palmyrene Alphabet Transliteration". The success rate of well generated images was 31.88% due to high variability of signs.

Although the number of kept images was not very large, thanks to adding the generated pictures to the dataset, the classifier results improved by 120.55%. This proves that using GAN for enlarging the datasets for OCR algorithms is desirable, when little original input data is available.

Acknowledgments. This article was created with the support of the Internal Grant Agency (IGA) Faculty of Management and Economy, Czech University of Life Sciences in Prague, 2022A0001 – "Researching methods of automatic augmentation of datasets using Machine Learning tools".

References

1. Jerzy, D.: Particle filter modification using Kalman optimal filtering method as applied to road detection from satellite images. In: Bikonis, A., Nyka, K. (eds.) 20th International Conference on Microwaves, Radar, and Wireless Communication (MIKON). Gdansk (2014)
2. Zhang, X.Y.: Moving object detection in video satellite image based on deep learning. In: Lv, D., Lv, Y., Bao, W. (eds.) Conference on LIDAR Imaging Detection and Target Recognition, vol. 10605. Changchun, PRC (2017)
3. Abu Doush, I.: Yarmouk Arabic OCR dataset. In: International Conference on Computer Science and Information Technology, pp. 150–154. Applied Science Private University, Amman (2018)
4. Deng, L.: The MNIST database of handwritten digit images for machine learning research. *IEEE Signal Process. Mag.* **29**(6), 141–142 (2012)
5. Junghwan, C. et al.: How much data is needed to train a medical image deep learning system to achieve necessary high accuracy? In: International Conference on Learning Representations. San Juan (2015)
6. Hamplová, A., Franc, D.: Historical alphabet transliteration software using computer vision classification approach. In: Proceedings of 11th Computer Science On-line Conference 2022, vol. 2, Artificial Intelligence Trends in Systems. ISBN: 978-3-031-09075-2 (2022)
7. Wang, R.S.: Convolutional recurrent neural networks for text classification. In: IEEE International Joint Conference on Neural Networks (IJCNN). Budapest (2019)

8. Harjoseputro, Y.: A classification Javanese letters model using a convolutional neural network with KERAS framework. *Int. J. Adv. Comput. Sci. Appl.* **11**(10), 106–111 (2020)
9. Krejsa, J., Vechet, S.: Classification of Czech sign language alphabet letters using CNN—preliminary study. In: Fuis, V. (ed.) 26th International Conference on Engineering Mechanics (IM). *Engineering Mechanics*, pp. 310–313. Brno (2020)
10. Naseer, A., Zafar, K.: Meta features-based scale invariant OCR decision making using LSTM-RNN. *Comput. Math. Organ. Theory* **25**(2), 165–183 (2018). <https://doi.org/10.1007/s10588-018-9265-9>
11. García-Manso, A., García-Orellana, C.J., González-Velasco, H.M., Macías-Macías, M., Gallardo-Caballero, R.: Comparing feature extraction techniques and classifiers in the handwritten letters classification problem. In: Diamantaras, K., Duch, W., Iliadis, L.S. (eds.) ICANN 2010. LNCS, vol. 6354, pp. 106–109. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-15825-4_12
12. Coustaty, M., Pareti, R., Vincent, N., Ogier, J.M.: Towards historical document indexing: extraction of drop cap letters. *Int. J. Doc. Anal. Recogn.* **14**(3), 243–254 (2011)
13. Preprocessing layers, keras.io. https://keras.io/api/layers/preprocessing_layers/. Accessed 15 June 2022
14. Dwyer, B., Nelson, J.: Roboflow (version 1.0). Retrieved from <https://roboflow.com>. Computer vision (2022)
15. Géron, A.: Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow: concepts, tools, and techniques to build intelligent systems, 2nd edn. O'Reilly, Beijing. ISBN: 978-149-2032-649 (2019)
16. Frid-adar, M., Diamant, I., Klang, E.: GAN-based synthetic medical image augmentation for increased CNN performance in liver lesion classification. *Neurocomputing* **321**, 321–331 (2018)
17. Bowles, C., Chen, L., Guerrero, R.: GAN augmentation: augmenting training data using generative adversarial networks. In: Arxiv (2018)

Attachment 4

A. Hamplová, D. Franc and J. Pavlíček, “Character segmentation in the development of Palmyrene Aramaic OCR,” CAISE 2023 35th International Conference on Advanced Information Systems Engineering, 2023.



Character Segmentation in the Development of Palmyrene Aramaic OCR

Adéla Hamplová^(✉) , David Franc^(✉) , and Josef Pavlicek 

Czech University of Life Sciences in Prague, Kamýcká 129, 165 00 Praha-Suchdol, Prague,
Czech Republic

{hamplova, francd, pavlicek}@pef.czu.cz

Abstract. In this study, we present the research plan and the segmentation solution in progress for our Palmyrene OCR web and mobile application from sandstone tablet photographs, which will be publicly available on the ml-research.pef.czu.cz web portal in the next steps of the research. In this paper, we compare mathematical segmentation methods with artificial intelligence methods, highlighting the advantages and disadvantages of each solution, and propose a fully automated OCR procedure from photographs using convolutional neural networks exclusively and present a development model of our solution. We also present a partially completed segmentation dataset of the Palmyrene letters to demonstrate the functionality of the proposed procedure. We hope to complete the Palmyrene OCR soon, thus making the writings of ancient Palmyra accessible to the scientific community and the public, signifying progress in the area of Digital Humanities. Since the algorithm is not completely ready yet, we also present its development model here.

Keywords: Artificial Intelligence · Segmentation · Historical Alphabet · Software Development Modelling

1 Introduction

The use of Optical Character Recognition (OCR) has become increasingly important in modern times, enabling the automated processing of scanned documents, handwritten notes, and other images of text. However, to achieve accurate OCR results, it is crucial to first segment the text from the surrounding noise and other elements within the image. In the context of analyzing historical texts, which are not written on paper that can be scanned, developing a reliable OCR algorithm is a challenging task.

In recent years, several historical OCR datasets have been developed (see section: Theoretical overview – Historical OCR in digital humanities), which contain text images and corresponding ground truth segmentations for training and evaluating OCR models.

In this scientific article, we introduce a new segmentation Palmyrene OCR dataset that we have developed and present an analysis of its characteristics and potential applications. Our dataset consists of diverse polygon-labelled images of Palmyrene inscriptions, funerary stelae, and other sandstone tablets with Palmyrene Aramaic texts, provided by

several museums (see section: Methodology - Data). The photographs include varying handwritings of different colors, sizes, and backgrounds. We believe that this dataset will be a valuable resource for researchers and practitioners working on OCR and related applications, enabling them to develop and test robust and accurate OCR models.

As the OCR algorithm is not finished at the time, a model of the development process is presented here as well, allowing easy to understand and easy to follow steps to complete it in the near future.

2 Theoretical Overview

2.1 SW Development Modelling

Process Management is, to quote [1], nowadays “considered as a new way of managing an organization”. According to [2] it is a managerial-economic discipline. The technology of process-oriented management is used here. We can say that process management is actually a set of tools, technologies and methods that are used to design, analyze, validate, approve and manage processes.

The management of an organization, or just in our case the management of a team, is then based on the principles of process mapping. Each process consists of sub-steps or tasks. These have to be executed chronologically or some have to be executed in parallel.

2.2 OCR

Developed first in the mid-20th century, OCR systems are now considered to be one of the most intricate applications of computer vision and image recognition [3]. Essentially, OCR involves identifying patterns that represent alphanumeric or other characters from a digital photograph or PDF scan [4].

The whole process of Optical Character Recognition (OCR) [5] involves the digitization of physical documents such as papers or historical artifacts with inscriptions, through scanning or photographing. The digitized image then undergoes local segmentation to differentiate text from other graphical elements, followed by optional pre-processing to reduce noise and normalize the image. The next step, segmentation or binarization, separates the image into non-text and text parts, and further segments the text into individual characters or parts of characters. The characters are then represented through global, statistical, or geometric representations. Feature extraction is commonly achieved through template matching, based on the distribution of points extracted from the image. OCR systems are trained and recognized through the creation of templates of individual features, using statistical techniques, cluster analysis, or artificial neural networks. Post-processing involves the composition of detected features into words, which are compared with a dictionary and modified for coherence. The final step is the output text in a machine-readable format, which can be further processed through natural language processing systems for tasks such as language translation or text summarization.

2.3 Historical OCR in Digital Humanities

Related Research

The interest in creating historical OCR is constantly rising, as standard historical text analysis needs to be done by experts in the field, who can read and translate the ancient languages. This process is time-consuming and limited to a small number of specialists, therefore automation of this process. A cuneiform NLP algorithm was developed by Gordin et al. from hand-written cuneiform transliterations [6], and a cuneiform horizontal stroke detection was developed thereafter [7]. An evaluation tool for Arabic OCR was published in 2017, presenting custom metrics [8], followed by a new Arabic OCR dataset consisting of almost 9000 images in 2018 [9]. A Bangla classification dataset [10] was published in 2020. A team from Bangladesh and Japan developed BengaliNet [11], a classification network for Bengali hand-written characters with an average of 97.5% accuracy. The Persian alphabet recognition dataset [12] was presented in 2020.

Palmyrene OCR Development

Our team has started working on Palmyrene OCR in 2021, at first by creating a classification dataset of hand-written and photographic data. We have implemented the classifier on efficient_lite0 architecture in an Android application presented in CSOC Springer conference [13], followed by presenting a custom optimized classifier for hand-written Palmyrene characters [14]. As photograph classification was not very successful, we used augmentation methods such as keras augmentation and Generative Adversarial Networks and doubled the accuracy [15]. We also created a web application (for which the paper is under review), available at ml-research.pef.czu.cz. There, the photo classification, and hand-written classification is available as well. At the time, the application accepts only one letter at a time as input, so transliterating the whole sandstone tablets take a lot of time. Therefore, we started working at automatic segmentation.

2.4 Contemporary Segmentation Methods

To improve recognition, current OCR research [16] incorporates different systems into a hybrid solution, which lowers the error rate when only one OCR algorithm is utilized. Systems are being created for languages that were previously unavailable, such as Arabic [17], Hindi [18], Japanese [19], and many more, but not Palmyrene Aramaic.

Post-processing techniques are being improved to account for many forms of errors and make automatic corrections [20]. The issues that these systems run into in typical settings are mentioned in a comparison study of existing OCR systems from 2021 [21], but only for Latin OCR herein. One of them is the challenge of identifying letters on a scanned page when they are deformed, blurry, or when whole characters are missing. The necessity to extract numerous distinct features for each character class or to use multiple templates and elastic templates for character classification is caused by font variations, which is another issue. As the letters in the Palmyrene script are carved in sandstone rather than written on paper, it is logical to conclude that both of these concerns will also be relevant in the OCR development of this script. In order to highlight the

features of characters from background, it will be important to utilize the proper pre-processing techniques. Additionally, neural networks can be used for feature extraction and classification rather than template matching.

Contemporary OCR system approaches involve complex compositions of mathematical and artificial intelligence methods in each phase of the algorithm. We believe that the development of OCR systems can be simplified given the state of artificial neural networks, and we want to investigate this idea here.

3 Objective

The goal of this step of our research is to automate the reading of Palmyrene inscriptions from a photograph of sandstone tablets or funerary stelae. As Palmyrene Aramaic inscriptions are read right to left, top to bottom, we plan to:

- segment each of the characters from the background using computer vision, namely instance segmentation
- program a script, which will redraw the segmentation predictions to polygons in a blank image
- create a SW development model of our solution

4 Methodology

4.1 Data Acquisition

To get high quality photographs of Palmyrene inscriptions, we have contacted several museums and retrieved available images of sandstone tablets, coins, and funerary reliefs:

- Musée du Louvre [22]
- Archaeological Museum of Palmyra [23] (Fig. 1)
- Virtual Museum of Syria [24]
- British Museum [25]
- Ontario Museum [26].

4.2 Pre-processing

Some of the images had poorly visible inscriptions, therefore, preprocessing in the form of adjusting brightness, contrast and color spectrum of these images was necessary, before uploading to the annotation platform.

4.3 Annotating Segmentation Dataset

The annotation of polygonal labels for instance segmentation was conducted in the online tool Roboflow [27]. Roboflow is a computer vision platform that helps researchers and other users build, train, and also deploy computer vision models, such as classification, object detection, instance or semantic segmentation and more. The platform offers a suite of tools and services designed to simplify the process of creating custom machine



Fig. 1. Example photograph of Palmyrene inscriptions used in the dataset, Archaeological Museum of Palmyra, Inv. 1439/8583; text: ḥb' brt l šmšgrm l br bny bzy l'tt b'ly l br 'gylw l ḥbl

learning models for image and video analysis. Roboflow allows users to upload their own datasets of images, then pre-process and augment the data after labelling, in order to improve model accuracy. Roboflow also offers APIs and SDKs that enable more granular control over the machine learning process, or the datasets can be exported and used in custom training.

We have created an instance segmentation dataset and used polygonal label tool to create points, that outline each character. There is just one class – character – and each character is later classified by custom classifier in a custom script Draw-Polygon.

4.4 Draw-Polygon Tool Development

We have developed an algorithm for processing and plotting polygons, which are results of YOLOv5 predictions. The polygons are read from a text file, where each polygon is represented as a set of points. The points are parsed, and their x and y coordinates are extracted into separate lists. Each polygon is plotted as a filled shape using matplotlib. The resulting plot is saved as an image and loaded for further processing with PIL. The image is flipped horizontally and resized with antialiasing. A new blank image is created with size 100x100, and the resized image is pasted in the center. The padded image is saved, and the original image is removed.

The input to the tool is in the format 0 (index class of object – in this case there is just one class - character) followed by a list of (x-coordinate y-coordinate). The output images are visible in Fig. 2.

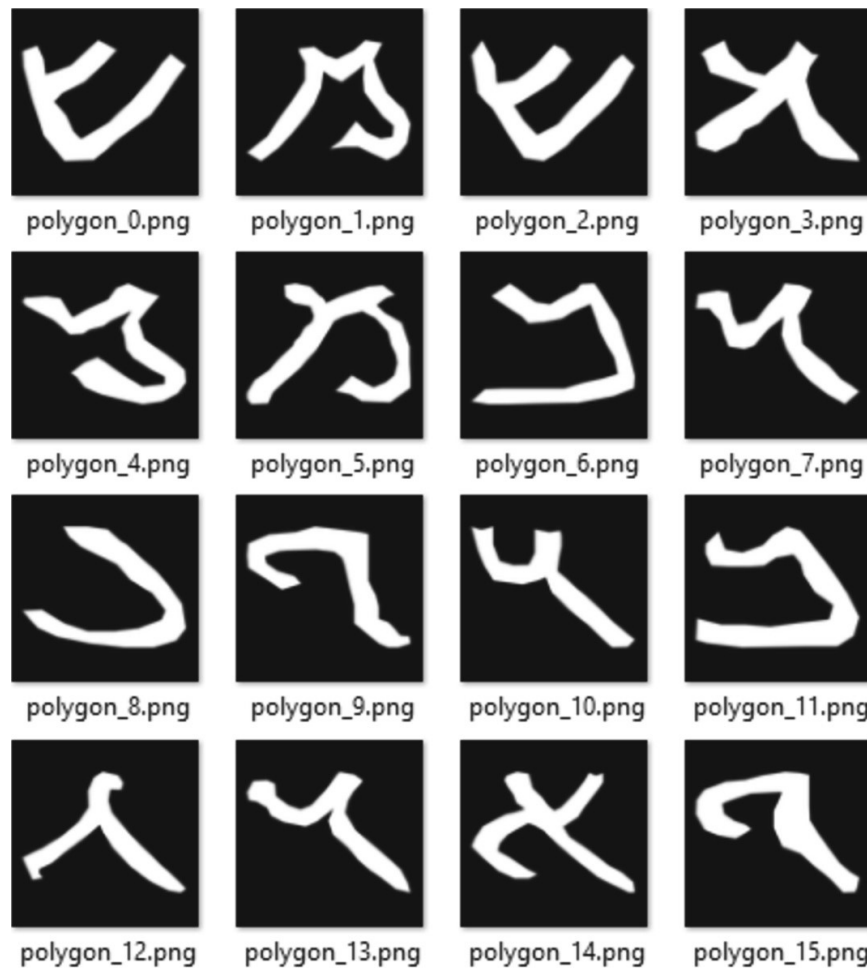


Fig. 2. A sample of output images from Draw-Polygon tool, characters 0: shin, 1: mem, 2: shin, 3: he, 4: samekh, 5: mem, 6: beth, 7: daleth, 8: nun, 9: waw, 10: daleth, 11: beth, 12: gimel, 13: 14: daleth, 15: aleph and 16: waw

4.5 SW Development Modelling

For modeling purposes, the OnLine tool Cardanit BPMN [28] was used. This tool can be used directly in the browser of the computer. It does not need to be installed and a very powerful feature is that it is possible to share the project with another researcher of the team and work on it simultaneously in real time.

BPMN was chosen as the notation for modeling the partial steps of the project. In our case, the chronology is written down as tasks that are carried out in individual years (these are represented by the so-called swimlines). Events that have occurred or will occur (that is, mainly publishing activities) are recorded using textual descriptions. Here we depart from the BPMN notation. We do not use the “event” symbol for them. This is because an “event” is seen as a kind of milestone that performs an action in the process model. In our case, however, it is an external output that does not enter the process model further.

The next model is a block diagram (control chart) of the process of processing the photograph of the Palmyra script. This diagram describes how the image processing

operations, the data preparation procedure for deep neural network learning, and the final transcription (transliteration) of the script into digital form are performed.

4.6 YOLOv5 Instance Segmentation

Instance segmentation models are usually built on object detections models with alterations to predict polygonal labels. There are two types of detectors – one and two stage. The main difference between one-stage and two-stage detectors is the approach used to predict object locations and class labels. One-stage detectors make predictions in a single pass through the network, while two-stage detectors first generate region proposals and then classify objects within those regions. One-stage detectors are faster but less accurate, while two-stage detectors are more accurate but slower. The choice of which detector to use depends on the specific application requirements, including speed and accuracy. As we wanted our solution to work on mobile phones as well, we opted for a single-stage detector edited for instance-segmentation. Therefore, our selected algorithm for instance segmentation was YOLOv5 [29], which was released in 2020 as an object detector and in 2022 extended to instance segmentation.

During training YOLOv5 instance segmentation model, we minimize the three following losses. YOLO also minimizes classification loss, but since we only have one class, we do not track it in this research.

- box loss – it calculates how well the algorithm finds the center of the object and how well the bounding boxes overlap the object.
- seg loss – segmentation loss, it calculates, how well the algorithm creates segmentation masks, how much they fit into the actual shape of the target object.
- obj loss – objectness loss, it is the algorithm’s confidence that an object exists within a given box (apart from polygons, yolo also predicts boxes).

In contrast, we maximize the metric mAP (= mean average precision).

5 Results

5.1 Dataset

We have labelled 28 images of Palmyrene inscriptions so far, with 1343 Palmyrene characters. 24 images were put in training, 1 image to validation and 2 images to testing subsets. We have selected several augmentation methods in Roboflow:

- Grayscale: Apply to 100% of images
- Saturation: Between -82% and $+82\%$
- Brightness: Between -20% and $+20\%$
- Exposure: Between -10% and $+10\%$
- Noise: Up to 5% of pixels

By using this augmentation, the final dataset contained 75 images in total. The dataset is available at [30].

5.2 SW Development Model

In the following three figures, we show the corresponding BPNM and flow diagrams. Figures 4 and 5 include one diagram split in half due to its size (Fig. 3).

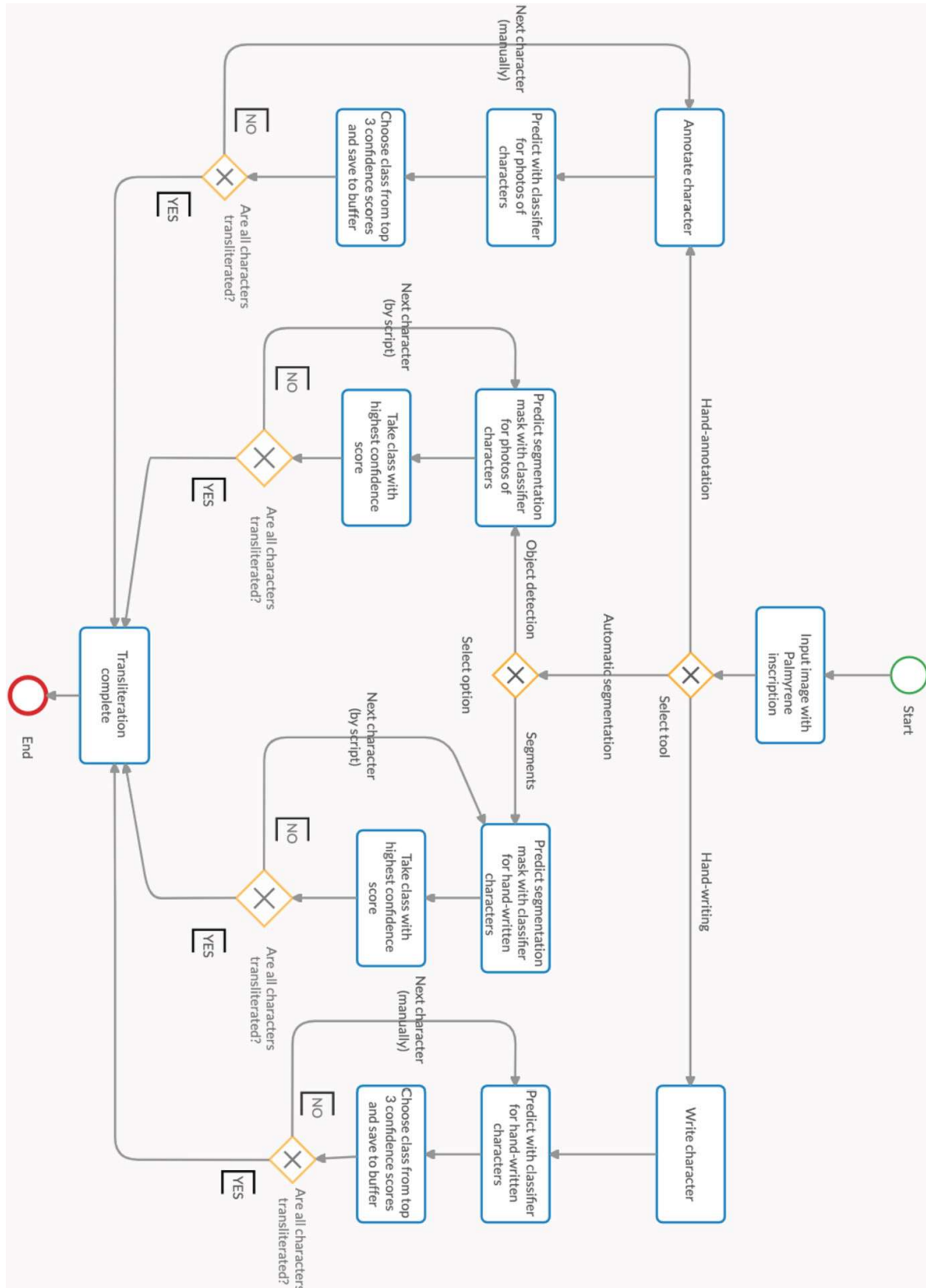


Fig. 3. Flow diagram of Palmyrene OCR application

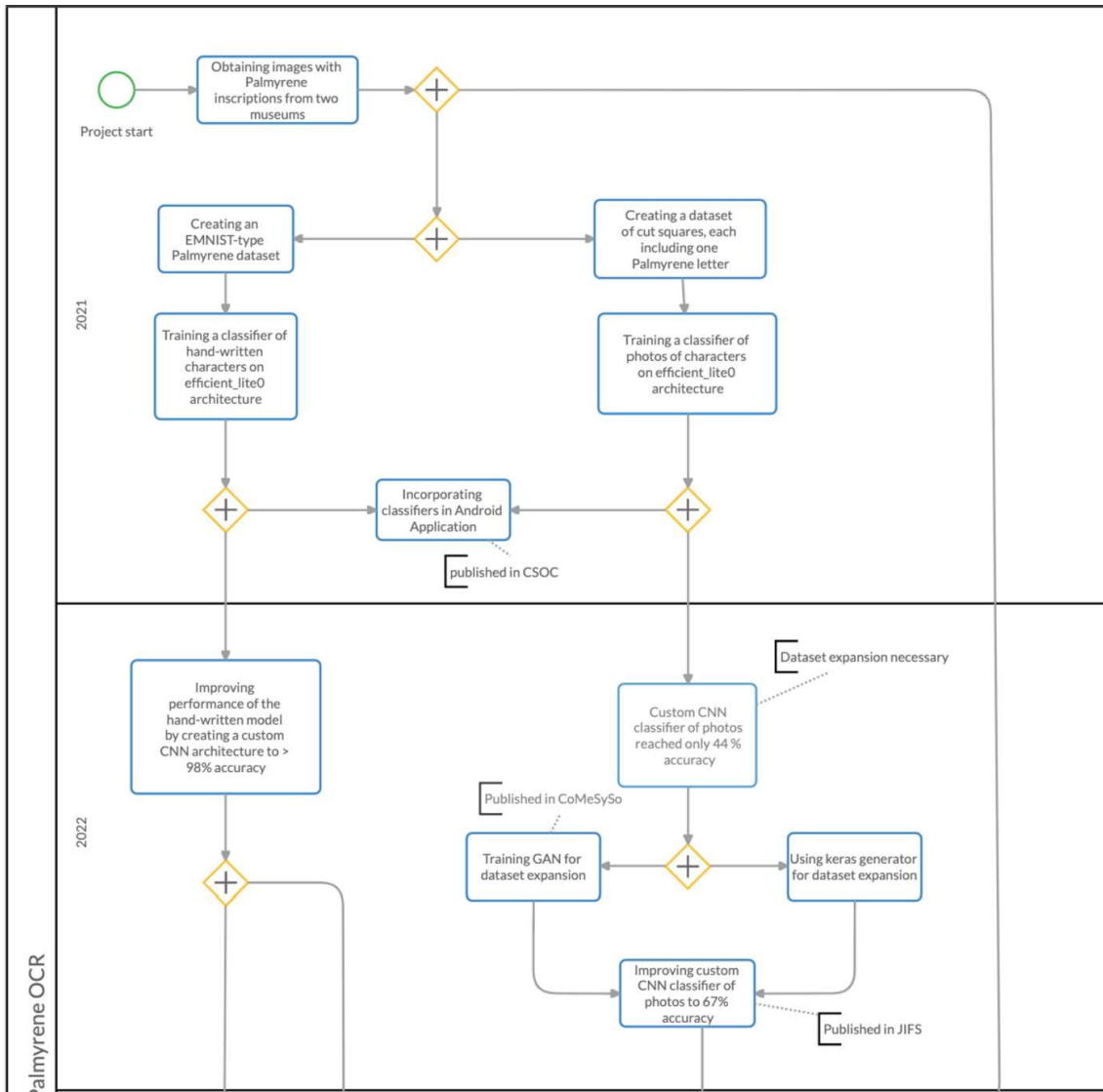


Fig. 4. Project model BPMN part 1 of 2

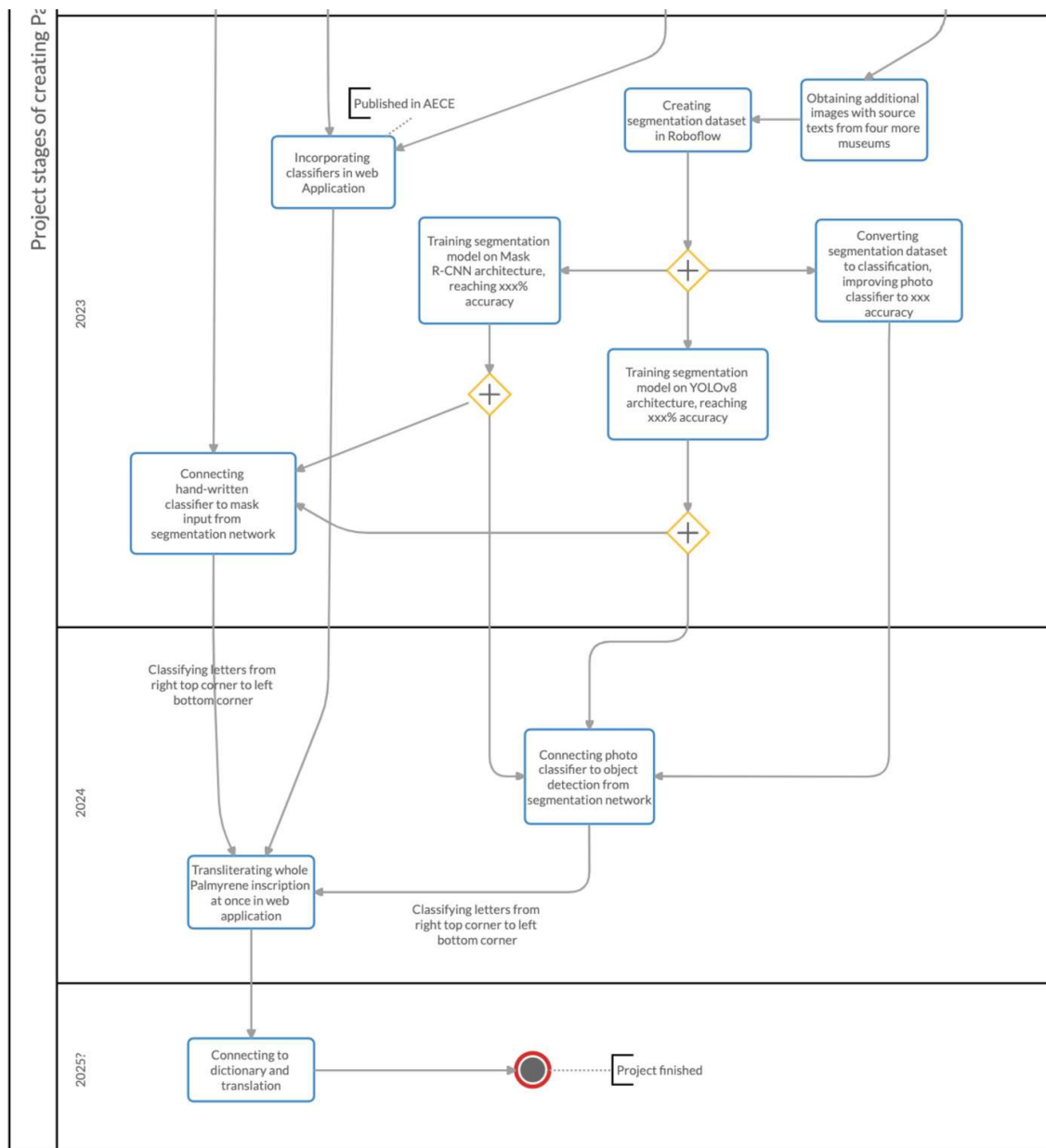


Fig. 5. Project model BPMN part 2 of 2

5.3 YOLOv5 Instance Segmentation

We have trained a YOLOv5 instance segmentation model with the following parameters on Google Colaboratory using GPU engine (Table 1).

At the start and at the end of training, the tracked metrics showed following values. First three epochs were warm-up epochs, so the metrics are tracked from epoch 4 (Table 2).

Table 1. Training parameters

Parameter	Value
Number of layers	225
Number of parameters	7408214
Number of gradients	7408214
Number of GFLOPs	24.9
Image size	640 × 640
Batch size	16
Epochs	100
Transfer learning initial model	Yolov5s-seg.pt
Learning rate	0.01
Lrf	0.01
Momentum	0.937
Weight decay	0.0005
Warmup epochs	3
Warmup momentum	0.8
Warmup bias learning rate	0.1

Table 2. Segmentation neural network training progress

Metrics	Box loss	Seg loss	Obj loss	mAP50	mAP50–95
Epoch 4	0.1164	0.1007	0.2553	0.00174	0.000174
Epoch 52	0.06822	0.04918	0.2365	0.815	0.305
Epoch 99	0.04529	0.04586	0.19	0.881	0.368

The images in the validation subset reached the following prediction result. We have removed displaying bounding boxes and confidence scores by editing `utils/plots.py` in `yolo` (Fig. 6):



Fig. 6. Prediction using YOLOv5 segmentation network.

5.4 Prediction Using Draw-Polygon Tool and Custom Classifier

The draw-polygon tool being developed is currently capable of drawing polygonal labels onto empty images and saving them without taking the order of letters into consideration. This could be useful for demonstration purposes, however, there are still some steps that need to be taken in order for the tool to become fully usable. The tool is still in an early stage of development and requires further refinement, testing, and debugging before it can be used in practical application.

6 Discussion

6.1 Comments on Results

We have presented a work-in-progress solution of Palmyrene OCR. The results provided consist of a YOLOv5 instance segmentation model, which was trained to recognize Palmyrene inscriptions using 28 labelled images with 1343 characters. The model was trained with a variety of augmentation methods to increase the size of the dataset and improve its robustness. The results show that after 100 epochs, the model achieved good performance, with an mAP50 of 0.881. Additionally, the draw-polygon tool is

being developed to allow for the drawing of polygonal labels onto empty images. The excellent mAP score suggests that it is possible to create reliable segmentation masks and thus identify the Palmyrene characters correctly. The presented development diagram shows which steps were taken to reach these results and which steps still remain and the second diagram shows what options are and will be available in our application.

6.2 Comparison with Other Authors' Works

There was multiple related research, which we can compare our results with. Among them are some sources already mentioned in the literature overview as well as some more. The field of OCR has seen significant developments in recent years, and researchers are continuously working towards improving the accuracy and efficiency of OCR systems. In this discussion, we can see the results of several OCR projects, each targeting a different aspect of the problem.

Sayeed et al. [11] propose a low-cost novel convolutional neural network architecture for the recognition of Bengali characters. The authors have considered 8 different formations of CMATERdb datasets based on previous studies for the training phase and have shown that their proposed system outperformed previous works by a noteworthy margin for all 8 datasets. Moreover, they have tested their trained models on other available Bengali characters datasets and achieved 96–99% overall accuracies.

Ghosh et al. [10] focuses on the recognition of 231 different Bangla handwritten characters using state-of-the-art pre-trained CNN architectures. The authors have shown that InceptionResNetV2 achieved the best accuracy (96.99%) after 50 epochs, while DenseNet121 and InceptionNetV3 also provided remarkable recognition accuracy (96.55 and 96.20%, respectively). The authors have also considered the combination of trained InceptionResNetV2, InceptionNetV3, and DenseNet121 architectures, which provided better recognition accuracy (97.69%) than other single CNN architectures. However, it is not feasible for using as it requires a lot of computation power and memory. The authors have tested the models in the cases where characters look confusing to humans, and all the architectures showed equal capability in recognizing these images.

Doush et al.'s project [9] proposed an Arabic printed OCR dataset, extracted randomly from Wikipedia, with 4,587 Arabic articles and a total of 8,994 images. This dataset is expected to be a valuable resource for the research community to build robust Arabic OCR systems.

Nashwan et al. [31] introduced a computationally efficient, holistic Arabic OCR system, using a lexicon reduction approach based on clustering similar shaped words to reduce recognition time. The system managed to generalize for new font sizes that were not included in the training data, and the evaluation results were promising compared with state-of-the-art Arabic OCR systems.

Radwan et al. [32] presented an open vocabulary Arabic text recognition system using two neural networks, one for segmentation and another one for characters recognition. The segmentation model showed high accuracy of 98.9% for one font and 95.5% for four fonts, while the character recognition accuracy was 94.8% for Arabic Transparent font of size 18 pt from APTI dataset.

Amara et al. [33] is discussing the importance of OCR in various applications and the challenges in segmenting Arabic characters for OCR. To prevent segmentation errors,

the authors propose a binary support vector machine (SVM) to decide whether to affirm segmentation points or re-do the segmentation, and deep learning methods to classify the characters.

Hussain et al. [34] presented a new approach for segmentation-based analysis for Nastalique style of Urdu language OCR. The segmentation-based method was optimized using 79,093 instances of 5249 main bodies, giving recognition accuracy of 97.11%. The system was then tested on document images of books with 87.44% main body recognition accuracy.

From this comparison, we can see, that our dataset is very small in comparison to other author's works. By annotating more images and by this, using more instances of Palmyrene characters for the training, we could reach over 90% accuracy in our recognition if we use more data for training our segmentation algorithm. We have already reached 88% with limited dataset size, so the potential is very promising.

6.3 Next Steps

Based on the results comparison, in later phases of this research, we will annotate more images of Palmyrene inscriptions. We also need to improve the draw-polygon tool by taking original image size into consideration and before drawing the polygons, first sorting the segments in left-to-right top-to-bottom order, as Palmyrene Aramaic is read this way, and then classify the polygons (characters) in the selected order with our previously developed classifier [14]. During the classification, each of these images will be assigned to a category of 28 Palmyrene characters and written down as Latin transliteration in the right order. The plans are also to implement this feature in the web and mobile applications.

The final step of creating a complete Palmyrene OCR is to create a Palmyrene NLP algorithm with a dictionary for translations to other languages, with the help of historians capable of translating Palmyrene Aramaic. We believe that our system can significantly improve the awareness of Palmyrene history to the general public, as well as speed up text recognition of newly discovered Palmyrene texts by researchers.

7 Conclusion

We have created a segmentation dataset of Palmyrene alphabet, which has several scientific contributions, namely increasing preservation and accessibility of Palmyrene artifacts and creating a cornerstone for the Palmyrene OCR system. As Palmyrene inscriptions are a valuable source of information about the culture, history, and language of ancient Palmyra, by creating a segmentation dataset of these inscriptions, we can make them more accessible to a wider audience, including scholars, students, and the general public.

Acknowledgment. This article was created with the support of the Internal Grant Agency (IGA) Faculty of Management and Economy, Czech University of Life Sciences in Prague, 2023A0004 – “Text segmentation methods for historical alphabets in the development of OCR”.

References

1. Hammer, M.M., Champy, J.A.: *Reengineering the Corporation: A Manifesto for Business Revolution*. HarperBusiness Essentials, New York (2003). ISBN 978-0060559533
2. Pavlicek, J., Hronza, R., Pavlickova, P., Jelinkova, K.: The business process model quality metrics. In: Pergl, R., Lock, R., Babkin, E., Molhanec, M. (eds.) *Enterprise and Organizational Modeling and Simulation, EOMAS 2017*. LNBIP, vol. 298, pp. 134–148. Springer, Cham. https://doi.org/10.1007/978-3-319-68185-6_10
3. Chaudhuri, A., Krupa M., Pratixa B., Soumya G.K.: *Optical Character Recognition Systems for Different Languages with Soft Computing. Studies in Fuzziness and Soft Computing*. Springer, Cham (2017). ISBN 978-3-319-50251-9. <https://doi.org/10.1007/978-3-319-50252-6>
4. Arica, N., Yarman-Vural, F.T.: An overview of character recognition focused on off-line handwriting. *IEEE Trans. Syst. Man Cybern. Part C (Appl. Rev.)* **31**(2), 216–233 (2001). ISSN 1094-6977. <https://doi.org/10.1109/5326.941845>
5. Yu, F.T.S., Jutamulia, S.: *Optical Pattern Recognition*. Cambridge University Press, New York (1998). ISBN 978-0521465175
6. Gordin, S., et al.: Reading Akkadian cuneiform using natural language processing. *PLOS ONE* **15**(10), e0240511 (2020). ISSN 1932-6203. <https://doi.org/10.1371/journal.pone.0240511>
7. Hamplová, A., Franc, D., Pavlíček, J., Romach, A., Gordin, S.: Cuneiform reading using computer vision algorithms. In: *2022 5th International Conference on Signal Processing and Machine Learning, New York*, pp. 242–245 (2022). ISBN 978-1-4503-9691-2. <https://doi.org/10.1145/35556384.3556421>
8. Alghamdi, M.A., Alhazi, I.S., Teahan, W.J.: Arabic OCR evaluation tool. In: *2016 7th International Conference on Computer Science and Information Technology (CSIT)*, pp. 1–6. IEEE (2016). ISBN 978-1-4673-8914-3. <https://doi.org/10.1109/CSIT.2016.7549460>
9. Doush, I.A., Aikhateeb, F., Gharibeh, A.H.: Yarmouk Arabic OCR dataset. In: *2018 8th International Conference on Computer Science and Information Technology (CSIT)*, pp. 150–154. IEEE (2018). ISBN 978-1-5386-4152-1. <https://doi.org/10.1109/CSIT.2018.8486162>
10. Ghosh, T., et al.: Bangla handwritten character recognition using MobileNet V1 architecture. *Bull. Electr. Eng. Inform.* **9**(6), 2547–2554 (2020). ISSN 2302-9285. <https://doi.org/10.11591/eei.v9i6.2234>
11. Sayeed, A., Shin, J., Hasan, M.A.M., Srizon, A.Y., Hasan, M.M.: BengaliNet: a low-cost novel convolutional neural network for Bengali handwritten characters recognition. *Appl. Sci.* **11**(15), 242–245 (2021). ISBN 9781450396912. ISSN 2076-3417. <https://doi.org/10.3390/app11156845>
12. Hajjhashemi V., Ameri, M.M.A., Gharahbagh, A.A., Bastanfard, A.: A pattern recognition based Holographic Graph Neuron for Persian alphabet recognition. In: *2020 International Conference on Machine Vision and Image Processing (MVIP)*, pp. 1–6 (2020). <https://doi.org/10.1109/MVIP49855.2020.9116913>
13. Hamplová, A., Franc, D., Tyrychtr, J.: Historical alphabet transliteration software using computer vision classification approach. In: Silhavy, R. (ed.) *Artificial Intelligence Trends in Systems, CSOC 2022*. LNNS, pp. 34–45. Springer, Cham (2022). ISBN 978-3-031-09075-2. https://doi.org/10.1007/978-3-031-09076-9_4
14. Hamplová, A., Franc, D., Veselý, A.: An improved classifier and transliterator of hand-written Palmyrene letters to Latin. *Neural Netw. World* **32**(4), 181–195 (2022). ISSN 23364335. <https://doi.org/10.14311/NNW.2022.32.011>
15. Franc, D., Hamplová, A., Svojshe, O.: Augmenting historical alphabet datasets using generative adversarial networks. In: Silhavy, R., Silhavy, P., Prokopova, Z. (eds.) *Data Science and*

- Algorithms in Systems, CoMeSySo 2022. LNNS, vol. 597, pp. 132–141. Springer, Cham (2023). ISBN 978-3-031-21437-0. https://doi.org/10.1007/978-3-031-21438-7_11
16. Kboubi, F., et al.: A new strategy of OCR combination. In: 8th World Multi-Conference on Systemics, Cybernetics and Informatics (SCI 2004), vol XII, Proceedings - Applications of Cybernetics and Informatics in Optics, Signals, Science and Engineering (2004)
 17. Alghyaline, S.: Arabic optical character recognition: a review. *CMES-Comput. Model. Eng. Sci.* **135**(3), 1825–1861 (2023). ISSN 1526-1506. <https://doi.org/10.32604/cmes.2022.024555>
 18. Natarajan, P.S., MacRostie, E., Decerbo, M.: The BBN Byblos Hindi OCR system. In: Proceedings of SPIE - The International Society for Optical Engineering: Document Recognition and Retrieval XII, pp. 10–16 (2005). <https://doi.org/10.1117/12.588810>
 19. Kokawa, A., Busagala, L.S.P., Ohyama, W., Wakabayashi, T., Kimura, F.: An impact of OCR errors on automated classification of OCR Japanese texts with parts-of-speech analysis. In: 2011 International Conference on Document Analysis and Recognition, pp. 543–547. IEEE (2011). ISBN 978-1-4577-1350-7. <https://doi.org/10.1109/ICDAR.2011.115>
 20. Nguyen, T.T.H., et al.: Deep statistical analysis of OCR errors for effective post-OCR processing. In: ACM-IEEE Joint Conference on Digital Libraries (JCDL), pp. 29–38 (2019). <https://doi.org/10.1109/JCDL.2019.00015>
 21. Jain, P., Taneja, K., Taneja, H.: Which OCR toolset is good and why: a comparative study. *Kuwait J. Sci. (KJS)* **48**(2), 1–12 (2021). ISSN 2307-4116. <https://doi.org/10.48129/kjs.v48i2.9589>
 22. Site officiel du Musée du Louvre. Le Louvre. (n.d.). <http://www.louvre.fr>
 23. Al-As'ad, K.: Aramaic Inscriptions in the Palmyra Museum. <https://journals.openedition.org/syria/1478>
 24. Collection | Palmyra Archaeological Museum. <https://virtual-museum-syria.org/palmyra/collection/>
 25. Palmyra | British Museum. <https://www.britishmuseum.org/collection/search?keyword=Palmyra>
 26. ROM Online Collection - Royal Ontario Museum. <https://collections.rom.on.ca/search/PALMYRA#filters>
 27. Roboflow: Give your software the power to see objects. <https://www.roboflow.com>
 28. Cardanit: BPM software | Business Process and Decision. <https://www.cardanit.com/>
 29. Jocher, G., et al.: ultralytics/yolov5. <https://doi.org/10.5281/zenodo.5563715>
 30. Hamplová, A.: Instance segmentation dataset of Palmyrene characters, ZIP archive. <https://app.roboflow.com/ds/6uKAAUoxOr?key=qch61mMNLp>
 31. Nashwan, F., Rashwan, M., Al-Barhamtoshi, H., Abdou, S., Moussa, A.: A Holistic technique for an Arabic OCR system. *J. Imaging* **4**(1), 6 (2018). ISSN 2313-433X. <https://doi.org/10.3390/jimaging4010006>
 32. Radwan, M.A., Khalil, M.I., Abbas, H.M.: Predictive segmentation using multichannel neural networks in Arabic OCR system. In: Schwenker, F., Abbas, H., El Gayar, N., Trentin, E. (eds.) *Artificial Neural Networks in Pattern Recognition*. LNCS, vol. 9896, pp. 233–245. Springer, Cham (2016). ISBN 978-3-319-46181-6. https://doi.org/10.1007/978-3-319-46182-3_20
 33. Amara, M., Zidi, K., Zidi, S., Ghedira, K.: Arabic character recognition based M-SVM: review. In: Hassanien, A.E., Tolba, M.F., Taher Azar, A., (eds.) *Advanced Machine Learning Technologies and Applications*. Communications in Computer and Information Science, vol. 488, pp. 18–25. Springer, Cham (2014). ISBN 978-3-319-13460-4. https://doi.org/10.1007/978-3-319-13461-1_3
 34. Hussain, S., Ali, S., Akram, Q.u.A.: Nastalique segmentation-based approach for Urdu OCR. *Int. J. Doc. Anal. Recogn. (IJ DAR)* **18**(4), 357–374 (2015). ISSN 1433-2833. <https://doi.org/10.1007/s10032-015-0250-2>

Attachment 5

A. Hamplová, A. Lyavdansky, T. Novák, O. Svojše, D. Franc and A. Veselý, “Instance Segmentation Of Characters Recognized In Palmyrene Aramaic Inscriptions,” *Computer Modelling in Engineering & Sciences*, 2024.



ARTICLE

Instance Segmentation of Characters Recognized in Palmyrene Aramaic Inscriptions

Adéla Hamplová^{1,*}, Alexey Lyavdansky^{2,*}, Tomáš Novák¹, Ondřej Svojsě¹, David Franc¹ and Arnošt Veselý¹

¹Czech University of Life Sciences in Prague (CULS), Faculty of Economics and Management, Department of Information Engineering, Prague, 16500, Czech Republic

²National Research University Higher School of Economics (HSE), Faculty of Humanities, Institute for Oriental and Classical Studies, Moscow, 101000, The Russian Federation

*Corresponding Authors: Adéla Hamplová. Email: hamplova@pef.czu.cz; Alexey Lyavdansky. Email: andurar@gmail.com

Received: 18 February 2024 Accepted: 12 April 2024

ABSTRACT

This study presents a single-class and multi-class instance segmentation approach applied to ancient Palmyrene inscriptions, employing two state-of-the-art deep learning algorithms, namely YOLOv8 and Roboflow 3.0. The goal is to contribute to the preservation and understanding of historical texts, showcasing the potential of modern deep learning methods in archaeological research. Our research culminates in several key findings and scientific contributions. We comprehensively compare the performance of YOLOv8 and Roboflow 3.0 in the context of Palmyrene character segmentation—this comparative analysis mainly focuses on the strengths and weaknesses of each algorithm in this context. We also created and annotated an extensive dataset of Palmyrene inscriptions, a crucial resource for further research in the field. The dataset serves for training and evaluating the segmentation models. We employ comparative evaluation metrics to quantitatively assess the segmentation results, ensuring the reliability and reproducibility of our findings and we present custom visualization tools for predicted segmentation masks. Our study advances the state of the art in semi-automatic reading of Palmyrene inscriptions and establishes a benchmark for future research. The availability of the Palmyrene dataset and the insights into algorithm performance contribute to the broader understanding of historical text analysis.

KEYWORDS

Optical character recognition; instance segmentation; Palmyrene; ancient languages; computer vision

1 Introduction

Palmyra, known as Tadmur in Arabic, is an ancient city located in the Syrian desert. It is also an essential part of human history. Its archaeological significance lies not only in its physical ruins, but also in the inscriptions carved into the buildings and into the funerary stelae. These inscriptions represent a valuable repository of knowledge that records the Palmyrene dialect of Aramaic, its culture, and the records of ancient Palmyrene society. However, uncovering the secret written on these inscriptions poses challenges for the scientific community.



Deciphering and analyzing these historical texts have interested scientists, historians, and archaeologists for generations, and until now it has only ever been done by linguists, not by machines. Therefore, applying deep learning (DL) methods is a transformative force, making the work of linguists easier and allowing the non-scholarly public access to texts that would otherwise be incomprehensible to them. Deep learning algorithms, including deep neural networks, offer automation in letter classification and segmentation, which can be a potential solution to the complexity of the transcription of Palmyrene inscriptions.

Previous research [1] dealt with classifying Palmyrene characters from handwritten transcripts and photographs and their augmentation [2]. It addressed the classification in two ways. The first way is to divide a dataset of Palmyrene characters into squares that each contain one letter; the second way is to handwrite an EMNIST-like dataset using special software and a mouse pen tablet and then make both classifiers available in an Android mobile application and an online application [3], using a custom neural network which was chosen as the best performing from 10 different architectures. As photographs classification did not achieve satisfactory results initially, Generative Adversarial Networks (GAN) are employed to expand the classification dataset, improving the outcomes by 120%. The research plan for segmenting Palmyrene characters was presented at a conference in 2023 [4].

Based on deep learning principles, this study aims to evaluate and compare the performance of state-of-the-art DL instance segmentation algorithms in Palmyrene character segmentation. Through data collection in collaboration with several museums worldwide, photo analysis, pre-processing, manual review of published transcriptions, and custom annotation in the Roboflow annotation platform, the computing power of DL is employed to solve the unique challenges posed by transcriptions of ancient inscriptions.

2 Structure

The article is structured as follows:

Section 3—Related Work—presents other works that describe developing an ancient or alive language Optical Character Recognition (OCR) and comment on the proposed methods. It also presents other relevant studies that utilize instance segmentation and the research gap.

Section 4—Data Collection and Preparation—describes how the data were obtained from the museums, checks the published transliterations to indicate if they align with the photographs, and adds the transliterations to photos that did not have them available. It also describes the pre-processing and annotation process and its challenges and defines the number of classes to work with.

Section 5—Methodology—introduces two approaches—single- and multi-class segmentation—and algorithms used—YOLOv8 and Roboflow 3.0. It explains the advantages and disadvantages of each method and describes the training, hyperparameters, and evaluation metrics. It also presents the custom scripts developed for letter sorting and visualization.

Section 6—Results—includes individual network training, testing, quantitative metrics, and visualization of results.

Sections 7 through 9 discuss the results, next steps, and conclusion. At the end, statements, acknowledgments, and references are presented, followed by **Appendices A and B** that provide details of the models' training and testing.

3 Related Work

Developing OCR or Natural Language Processing (NLP) algorithms for languages lacking existing solutions is an essential part of preservation and making it easier to process documents in that given language. This applies to historical texts, such as Egyptian hieroglyphs [5], Sanskrit [6], and different types of cuneiform [7] and living languages.

For instance, a Turkish OCR system [8] employs commonly available OCR algorithms-CuneiForm Cognitive OpenOCR, GNU License Open-source Character Recognition (GOOCR), and Tesseract [9]-to handle a dataset consisting of scans and photos of Turkish texts. Another was designed for Icelandic to aid in digitizing the Fjölfnir magazine, housing historical texts [10]. Character recognition has also been developed for Bangla [11], presenting unique challenges due to the variability of characters and the presence of ligatures (conjunctions of characters). Oni et al. [12] developed an OCR algorithm based on generated training data. They scanned images of Yoruba texts written in Latin script and reached 3.138% character error rate using the Times New Roman font.

There are comparative performance studies for or languages with many OCR systems available, either of the whole systems [13] or separate languages, such as Arabic [14].

Using instance segmentation algorithms for character detection can be effective in image-based tasks involving handwriting, as opposed to OCR for scanned text, where semantic segmentation is employed to separate text from background, e.g., in the case of Czechoslovak scanned documents [15]. Instance segmentation using Convolutional Neural Networks (CNNs) is applied to detect the boundaries of individual objects. It is usually used for other tasks, such as segmenting leaves in plants [16], cars in a parking lot [17], or ships and airplanes from satellite images [18].

Although instance segmentation algorithms are usually used for tasks other than letter segmentation, they also have a high potential to find letters in photographs. Instance segmentation can make it possible to recognize characters in different font styles and photographs of various quality if a large enough dataset is available, and it is not necessary to separate the text from the non-text part.

4 Data Collection and Preparation

4.1 Obtaining Data

Photographs of Palmyrene inscriptions were obtained from several private sources with their consent, from public online sources, and by taking photographs in the respective museums. The photographs of inscriptions originate from Arbeia Roman Fort and Museum¹, Archaeological Museum of Palmyra², The British Museum³, Carlsberg Glyptotek⁴, Hypogeum of Three Brothers⁵, MET Museum⁶, Musée du Louvre⁷, Musei Vaticani⁸, Museum of the American University, Beirut⁹, The Getty Villa Museum¹⁰, National Museum in Prague¹¹, Royal Ontario Museum¹², The Pushkin State Museum of Fine Arts¹³ and The State Hermitage Museum¹⁴.

¹<https://arbeiromanfort.org.uk/>.

²<https://virtual-museum-syria.org/palmyra/>.

³<https://www.britishmuseum.org/>.

⁴<https://glyptoteket.dk/>.

⁵<https://archeologie.culture.gouv.fr/palmyre/en/mediatheque/hypogeum-three-brothers-palmyra-7>.

⁶<https://www.metmuseum.org/>.

⁷<https://www.louvre.fr/en>.

⁸<https://www.museivaticani.va/content/museivaticani/en.html>.

⁹https://www.aub.edu.lb/museum_archeo/Pages/default.aspx.

¹⁰<https://www.getty.edu/visit/villa>.

¹¹<https://www.nm.cz/en>.

¹²<https://www.rom.on.ca/en>.

¹³<https://pushkinmuseum.art/?lang=en>.

¹⁴<https://www.hermitagemuseum.org/wps/portal/hermitage/>.

4.2 Checking Transcriptions

Prior to the annotation, the collected photographs were checked. For each photo, the visible letters were checked. For some photographs, previously published transcriptions were available and edited to match the visible characters in the photographs. For those photographs that did not have transcripts available, transcripts were created.

4.3 Annotation and Pre-Processing

The annotation of the instance segmentation dataset was based on the checked and newly created transcriptions and was completed in the Roboflow annotation tool using 26 classes corresponding with the Palmyrene characters. [Table 1](#) shows the complete list.

Table 1: Palmyrene character classes in multi-class segmentation

Class index	Class name	Transcription	Palmyrene
0	One	1	Ⲁ
1	Ten	10/100	ⲁ
2	Twenty	20	Ⲃ
3	Aleph	ʔ	Ⲅ
4	Ayin	ʕ	ⲅ
5	Beth	b	Ⲇ
6	Gimel	g	ⲇ
7	He	h	Ⲉ
8	Heth	ħ	Ⲩ
9	Kaph	k	ⲩ
10	Lamedh	l	Ⲫ
11	Mem	m	ⲫ
12	Nun	n	Ⲭ
13	Nun_final	n	ⲭ
14	Pe	p	Ⲯ
15	Qoph	q	ⲯ
16	Resh/daleth	r/d	Ⲱ
17	Right	>	ⲱ
18	Sadhe	ʃ	Ⲳ
19	Samekh	s	ⲳ
20	Shin	š	Ⲵ
21	Taw	t	ⲵ
22	Teth	ʈ	Ⲷ
23	Waw	w	ⲷ
24	Yodh	y	Ⲹ
25	Zayin	z	ⲹ

The characters “left” ⚡ and “right” ⚡ are paratextual signs similar to punctuation marks. Traditionally, they are labeled “ivy leaf” and put either at the beginning or at the end of a line, or a whole text in Palmyrene Aramaic and Greek inscriptions. Generally, the “left” ivy leaf is used much more often than its right counterpart.

The character “left” ⚡ was not present in any of the photographs, so it was excluded from the class list, but it was included in the classification dataset. The characters “resh” 𐤓 and “daleth” 𐤌 were combined into a single class because they are often written identically, with their distinction depending only on the context. Sometimes, “resh” 𐤓 is marked with a dot above. However, a segment must be a continuous object. Hence, the dot will make a separate segment. There are some dotted “resh” 𐤓 in the dataset, but they are a minority compared to the volume of those that are not dotted.

The same blending applies to characters “10” and “100”, “5” and “ayin”. In some visual variants, this also applies to the pair “mem” and “qoph” and the pair “heth” and “sadhe”, however, they were preserved as a separate class, as other visual variants are distinguishable.

5 Methodology

5.1 Instance Segmentation

Segmentation is the most intricate of the three computer vision tasks: classification, object detection, and segmentation [19]. It involves pixel-level classification, where pixels are grouped based on the selected class, revealing the precise boundaries of objects. There are two main types of segmentation: semantic segmentation, which clusters pixels belonging to the same class regardless of whether objects overlap, and instance segmentation, which identifies individual instances of objects within the same class. Instance segmentation determines the outlines of each instance based on factors such as shape, texture, brightness, and color [20].

During the training of an instance segmentation model, four types of losses are minimized in parallel, including box, segmentation, class, and distributional focal (box_loss, seg_loss, cls_loss, df_l_loss). The box loss measures the difference between predicted bounding box coordinates and the ground truth bounding box coordinates for each object instance, typically calculated as smooth L1 loss. The segmentation loss quantifies the difference between the predicted segmentation mask and the ground truth mask for each object instance. The class loss describes the variation between predicted class probabilities and the true class labels associated with each object instance. It is typically computed using a categorical cross-entropy loss function. The distributional focal loss is a modified version of the focal loss employed to solve the class imbalance problem. More information about the losses can be found in the literature [21].

This study uses two approaches to extract text from a photo using instance segmentation.

5.1.1 Single-Class Instance Segmentation

The first approach aims to segment letters regardless of their class and semantic meaning. Hence, the identified segments are ordered as text (right to left, top to bottom), and plotted one by one in the empty images (as described in Section 5.4, Custom Tools). These individual images are input for classification, and the classified features in the correct order form the entire text in the photo. This approach of looking for segments in only one class greatly increases the chance of finding more segments since the neural network only looks for one class.

5.1.2 Multi-Class Instance Segmentation

The second approach uses multi-class segmentation. Each letter is identified separately in the dataset, making it more accurate to find them and draw more correct segmentation masks. However, many letters are underrepresented in the dataset, so the segmentation algorithms do not find them and miss them entirely in the resulting text transcription. This problem will be solved through a significant dataset extension, which is currently in progress.

5.2 Selected Segmentation Models, Their Advantages and Disadvantages, Hyperparameters and Training

This study selected two instance segmentation algorithms. A comparison between YOLOv8 and Mask Region-based Convolutional Neural Network (R-CNN) was presented in 2023 [22] and showed that YOLOv8 performs better on selected images from fish-eye cameras. Like the images of the sandstone tablets with the Palmyra inscriptions, these images are of lower quality, and YOLOv8 can find more objects than the more accurate R-CNN. Roboflow Train was also chosen because this company offers dataset management, integrates an annotation tool, and offers data augmentation directly in the application. Thus, training directly in this particular application is relevant as the dataset was annotated there.

5.2.1 YOLOv8

YOLO, short for You Only Look Once, was released in 2016. It belongs to the category of one-shot detectors, which are generally less accurate but very fast, contrary to two-stage detectors, which are more accurate and slower [23]. YOLO has been under development for multiple years by Redmond et al. [24–26] until he decided to retreat from the research in fear of potential misuse by social media companies and the military; however, other teams took over his work. The first version of YOLO to incorporate instance segmentation was YOLOv5 in September 2022 [27]. It was developed by Glenn Jocher as an object detection algorithm [28]. The most contemporary version-YOLOv8, includes instance segmentation from January 2023 [29].

The main advantages of using YOLO are its training and inference speed, but it generally comes with lower accuracy.

The selected YOLOv8 instance segmentation model comprises 261 layers, 11800158 parameters, 11800142 gradients, and 42.7 GFLOPs. The complete architecture overview is indicated in [Table A1](#) in [Appendix A](#). The initial weights “yolov8n-seg.pt” are trained on the COCO dataset, and the transfer learning technique is used.

5.2.2 Roboflow 3.0 Instance Segmentation (Accurate)

Roboflow Train 3.0 is a model included in the Roboflow web application, released in July 2023 [30]. There are two options: fast or accurate training. However, the company has not publicly disclosed technical specifics about the structure and architecture. The main advantages are the simplicity of use and remote training, and the disadvantages are the lack of control over the model, as the only options the user can influence are the model type and providing a custom dataset with selected augmentation options.

5.3 Evaluation Metrics

Each Palmyrene text within a photo examines whether the correct number of characters is identified and whether the characters are correctly classified. Error analysis can be performed for

three main types of errors within the OCR transcription of a whole test set, and more derived metrics can be used. The following errors can occur when processing a test dataset:

- **Insertion Errors:** The system found a character where there was none. This study denotes the number of these errors as I .
- **Substitution Errors:** The system found the character in a certain location but misclassified it. This study denotes the number of such errors as S .
- **Deletion Errors:** There was a character at that location, but the system found no character at that location. This study denotes the number of these errors as D .
- **Total Levenshtein Distance:** The total number of errors that occurred during the processing of the test data set is:

$$TLD = I + D + S \quad (1)$$

where TLD is the Total Levenshtein Distance. The Levenshtein Distance, also known as the Edit-Distance algorithm, measures the number of characters that must be changed, added, or deleted in the predicted word so that it matches the true word [31]. Total Levenshtein distance does not apply to a word; it applies to the whole text.

- **Total Character Accuracy:** In addition to the Total Levenshtein Distance, the system's behavior will be evaluated using the Total Character Accuracy metric, which will rate the overall quality of the transcript. This study denotes the total number of letters as N and the Total Character Accuracy as TCA , where:

$$TCA = \frac{100 \cdot (N - S - D)}{N} \quad (2)$$

Thus, TCA determines the percentage of characters correctly found and correctly classified in the test dataset. The TCA value does not depend on the number of insertion errors I . Therefore, the value of the I parameter or the Total Levenshtein Distance that incorporates the I value must also be considered when evaluating the system's overall quality.

5.4 Custom Tools

The image is processed to text, as shown in [Fig. 1](#).

5.4.1 Prediction Scripts

Due to the use of two segmentation methods and, thus, four different models, the characters in images are predicted in multiple ways. However, a 40% confidence score is always set as a threshold.

This study predicts using the stored model on the web server for single-class segmentation using YOLO. It obtains a list of identified segments labeled as "1" only (meaning a character). These are then sorted by the developed program from right to left, top to bottom (see [Section 5.4.3 Sort](#)), and after sorting, they are printed on a square image ([Section 5.4.3 Draw](#)), which is input to the classifier, classifying them in that order and outputs the resulting text.

For single-class segmentation using Roboflow, we use the Roboflow API snippet to predict the segments. The segments are then converted to YOLO format, and the subsequent procedure is identical to YOLO single-class segmentation.

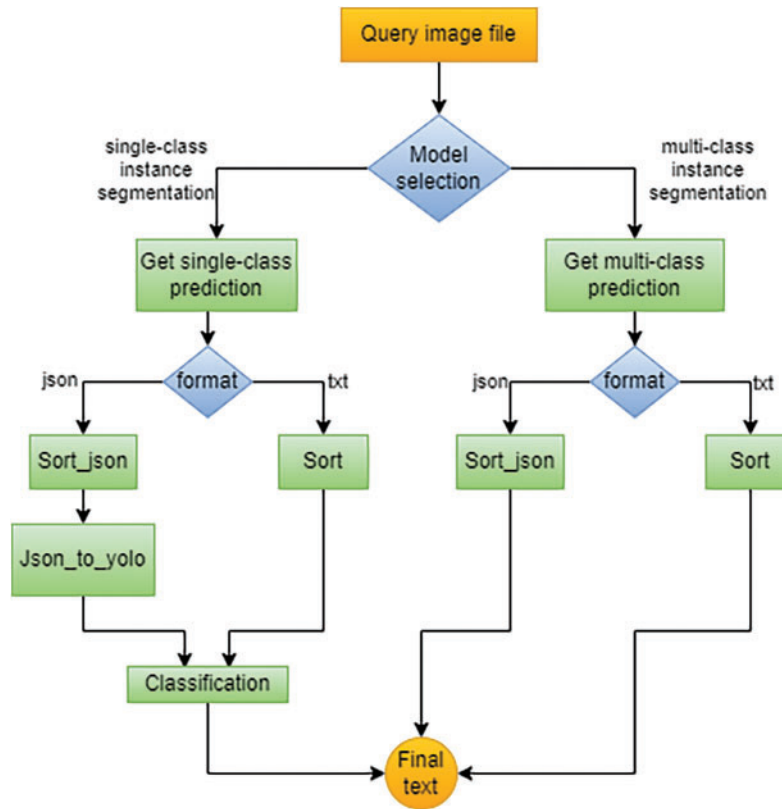


Figure 1: Palmyrene character instance segmentation-process flow diagram

Multi-class segmentation using YOLO uses a second model stored on the web server, the outputs are segments already assigned to the appropriate characters. These are further sorted, and the resulting text is obtained directly from the sorting tool. When using the last model, Roboflow multi-class segmentation, the predictions from the “.json” format are converted to the Yolo format. Then, the segments are sorted to produce the resulting text.

5.4.2 Detecting Rows and Sorting Letters

Classical line detection algorithms for scanned documents assume that the lines are straight, and in handwritten documents [32], line detection is performed in the original image before detecting separate letters. Another successful approach to detect lines in documents is to use Google Tesseract [33], but it does not support the Palmyrene language. The traditional algorithms assume text linearity and regularity, which is absent in the historical texts captured in the photographs of sandstone inscriptions. Such handwriting has considerable variability, which causes irregularities in spacing, angles of lines, and diverse styles, which were unique to each person. Palmyrene also uses irregular fonts in some cases. It was, therefore, necessary to address the issue in a specific manner.

The study cannot use either of the mentioned approaches because they are not intended to sort polygons already detected by YOLO or Roboflow Instance Segmentation. Since these polygons are restored from photographs, the rows in the images are ambiguous and not always straight.

At first, the algorithm in `sort.py` reads polygon information in YOLO instance segmentation format (the class index, points as x, y coordinate tuples, and confidence score). If the format is different, the variant `sort_json.py` is used, and subsequently, the output is converted for further processing by another custom script `json_to_yolo.py`. The sorting principle is as follows:

1. The average height \bar{h} of the polygons (x_i, y_i) is determined (see Fig. 2).
2. Polygons are sorted in descending order according to their y_i coordinate.
3. Splitting into rows: For all pairs of polygons $((x_i, y_i), (x_{i+1}, y_{i+1}))$, $i = 1, \dots, n-1$, we determine, whether

$$|y_{i+1} - y_i| > 0.5 \cdot \bar{h} \quad (3)$$

If the result of the inequality is true, y_i becomes the last polygon of the current row and y_{i+1} becomes the first polygon of the subsequent row.

4. Polygons (x_i, y_i) , $i = 1, \dots, n$, are arranged in each row based on the size of the x_i coordinate in descending order (the Palmyrene text is read from right to left).

Finally, the output text is printed, and the sorted polygons are saved to the file whose name was specified when the script was run.

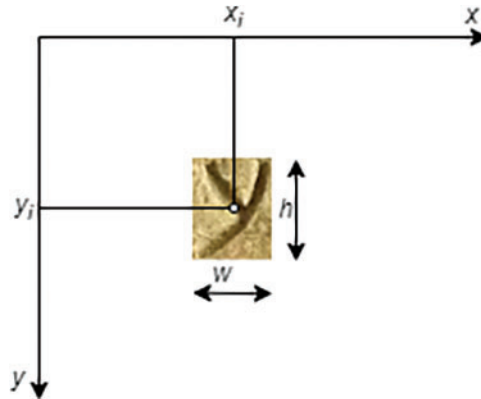


Figure 2: Coordinates (x_i, y_j)

5.4.3 Visualization Tools

Sort

The sorting tool includes plotting the polygons and class names in a plot, as depicted in Fig. 3.

Draw

`draw.py` visualizes separate polygons, which are printed into a black-and-white binary image in the correct order, which is an input to classification. The relative coordinates of the polygons (obtained by YOLO or converted to YOLO format from the “.json” format used by Roboflow 3.0) are scaled to match the size of the original image. The algorithm processes each polygon in the dataset. It scales the polygon’s relative coordinates to fit the original image’s size. Then, the polygons are drawn as white letters into a black image in the original polygon size. Subsequently, the polygons are cropped or stretched to a target size ($80 \times N$ or $N \times 80$) based on the aspect ratio of the polygons and put in the center of a 100×100 black image, which is saved with a filename that indicates the polygon index.

The output of this tool is illustrated in Fig. 4. Then, the letters are classified using the classification prediction script, resulting in a final list of transcribed letters.

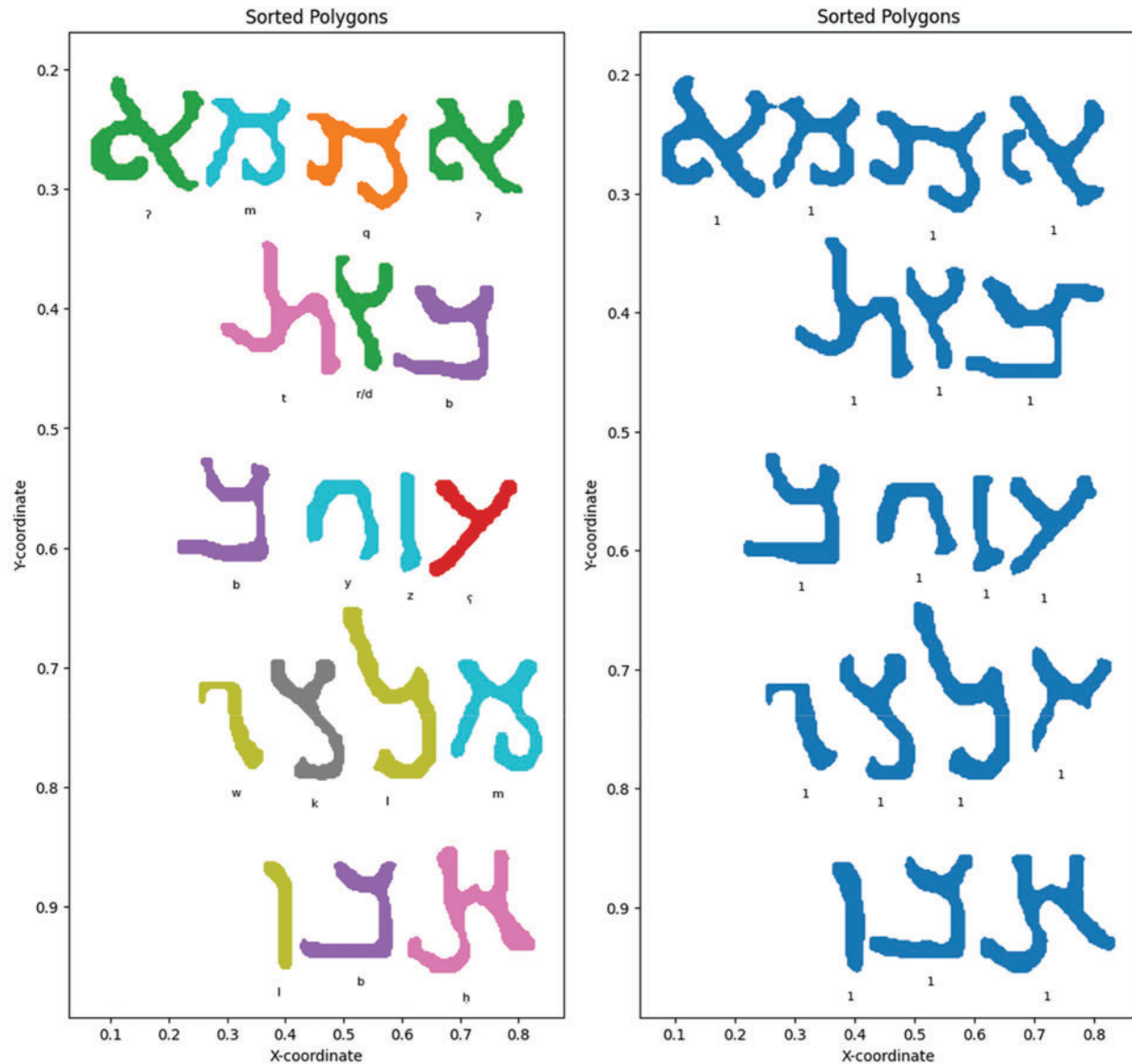


Figure 3: Plotted polygons from YOLO multi-class and single-class predictions of a photo of “Inv.1438/8582, Archaeological museum of Palmyra”, generated by *sort.py* tool

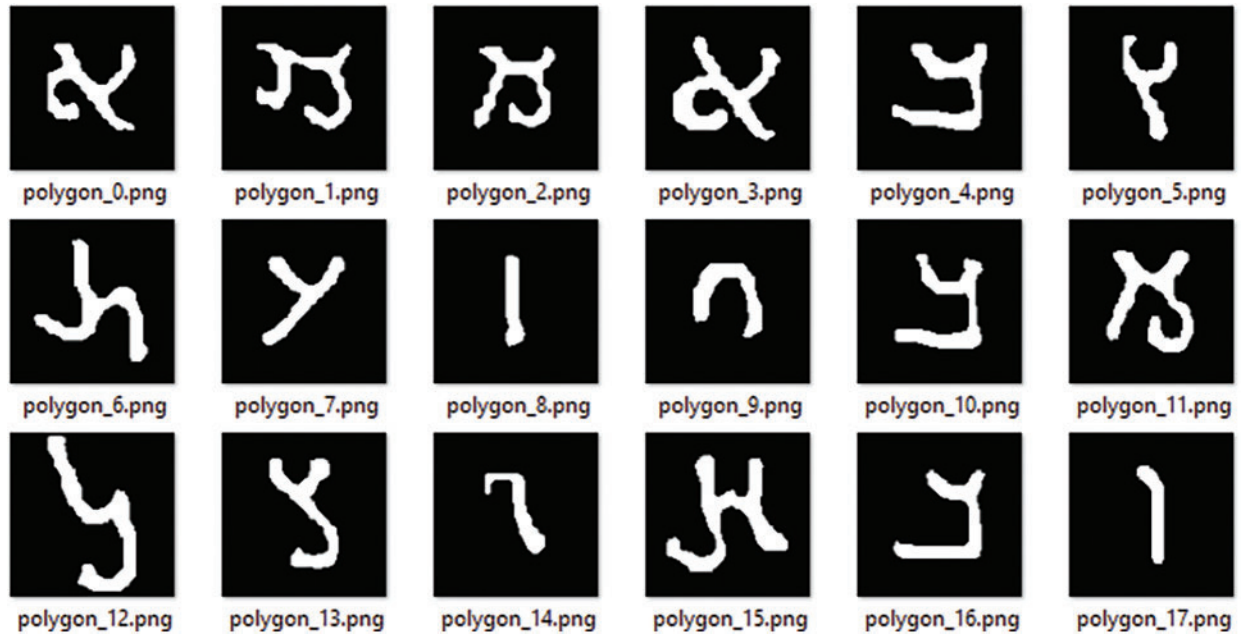


Figure 4: Plotted and sorted polygons for classification

6 Results

6.1 Training Results

The models were trained on the dataset consisting of 119 images with 3578 hand-annotated Palmyrene characters, resized to 920×920 pixels, and augmented to triple the dataset size using the following augmentation options:

- Grayscale: Apply to 50% of images
- Saturation: Between -60% and $+60\%$
- Brightness: Between -11% and $+11\%$
- Exposure: Between -11% and $+11\%$
- Blur: Up to 1.25px

The value of the loss functions `box_loss`, `seg_loss`, `cls_loss`, and `df1_loss` on the training set steadily decreases during the learning process.

The values of the four loss functions on the validation datasets oscillate, but their mean values also reduce. The smoothest decrease of the validation loss functions can be observed on the YOLOv8 multi-class model and the most random changes on the Roboflow 3.0 single-class model. The detailed training [Figs. A1–A8](#) are provided in [Appendix A](#). [Table 2](#) lists the training results of each segmentation algorithm after the first and last epochs are rounded to 2 decimal places. There are 100 epochs for the YOLOv8 model and 120 epochs for the Roboflow model.

Table 2: Training of all models in the first and last epoch

Model		Roboflow 3.0 multi-class	YOLOv8 multi-class	Roboflow 3.0 single-class	YOLOv8 single-class
First epoch	box_loss	1.56	1.11	1.15	2.28
	seg_loss	2.91	2.12	2.21	4.63
	cls_loss	2.97	0.95	0.97	3.30
	dfl_loss	1.23	0.95	1.04	1.59
Last epoch	box_loss	0.31	0.68	0.62	0.84
	seg_loss	0.85	1.57	1.36	1.71
	cls_loss	0.30	0.48	0.39	0.48
	dfl_loss	0.14	0.85	0.86	0.87

6.2 Evaluation Results

The success of single-class segmentation with subsequent classification and multi-class segmentation was evaluated on six images with Palmyrene inscriptions with 216 characters. Each image was analyzed for errors specified in the [Section 5](#). Only images with clear inscriptions were selected for the test, as the models did not perform well on lower-quality images. [Tables A2–A7](#) include the original texts and comparisons to predictions available in [Appendix B](#) and a summary is present in [Table 3](#). Original text in [brackets] indicates letters that are not visible in the photo but are part of the inscription. Errors in the predicted texts are labeled in the texts as follows:

Table 3: Overall evaluation of all models

	YOLO single-class	YOLO multi-class	Roboflow single-class	Roboflow multi-class
Insertion Errors	9	6	0	8
Deletion Errors	2	7	10	17
Substitution Errors	47	7	60	5
Total Character Accuracy	77.3%	93.5%	67.6%	89.8%
Total Levenshtein Distance	58	20	70	30

(1) insertion errors: **bold and underlined**, (2) substitution errors: **bold**, (3) deletion errors: **bold dash -**. All plotted figures with texts generated by the *sort.py* tool are available on GitHub [[12](#)].

7 Discussion

The results indicated that the Roboflow 3.0 multi-class model should be theoretically best performing as the losses in the last epoch are the least of all trained models. However, it ultimately achieves a Total Levenshtein Distance of 30 and a Total Character Accuracy of 89.8%, placing this model in second position. The subsequent tests showed that the YOLO multi-class instance

segmentation model performs best with the least Total Levenshtein Distance of only 20 and the highest Total Character Accuracy of 93.5%. The evaluation of both these models proved that using the multi-class segmentation method attains satisfactory results because the predicted segmentation mask shapes are very accurate.

However, the single-class instance segmentation method with consecutive classification is insufficient for practical use as the Total Character Accuracy reached only 77.3% for the YOLO single-class instance segmentation model, and its Total Levenshtein Distance was too high with the value of 58, due to a high number of misclassified characters. The Roboflow single-class model reached 67.6% Total Character Accuracy with a Total Levenshtein Distance of 70.

Although the classifier of handwritten Palmyrene characters, which was utilized to classify the predicted polygons created from instance segmentation masks, reached over 98% for classifying handwritten characters [1], the issue causing the misclassification can be the thickness of the lines, as the classifier was trained on artificially written characters with a fixed line thickness, which was significantly smaller. Sometimes, the predicted segmentation masks were very wide. Also, some of the predicted segments had incomplete shapes.

The best performing (YOLO multi-class) model was implemented in the web application ML-research [3] under the tab “Segmentation & Transcript”.

The average accuracy of the OCR of Egyptian hieroglyphs was 66.64%, surpassing the state of art, which was 55.27% before that [9]. Arabic character recognition using Deep Belief Network (DBF) and Convolutional Deep Belief Network (CDBF) was 83.7% accuracy on the IFN/ENIT Database on a model that reached 97.4% accuracy during training [34]. A Holography graph neuron-based system (HoloGN) for handwritten Persian characters [35] was over 90% accuracy when using a dataset extracted from 500000 images of isolated Farsi characters written by hand by Iranian people, but only 45% on images downsized to 32×16 pixels due to memory use optimization when using feedforward Artificial Neural Network (ANN). By comparing this study’s results to those of others in historical alphabets OCR, the proposed algorithm performed well with 93.5% accuracy when used on high-quality images of Palmyrene inscriptions.

8 Limitations and Next Steps

Some limitations can be encountered when using instance segmentation algorithms to identify Palmyrene characters in photographs. A possible problem arises from underrepresenting some characters in the training dataset. Although some letters such as “b”, “d/r”, “y” and “l” occur in almost every inscription, others such as “left”, “right”, “pe” and “samekh” appear quite rarely.

In the case of single-class instance segmentation, a limitation is the occasional inaccurate identification of polygons derived from the segmentation masks of letters, which can cause the character to be assigned to a different class than the one to which the letter belongs during subsequent classification. In order to address this problem, the polygons can be added to the training subset, and the handwritten character classifier can be retrained. This is subject to testing as it can bias the results of handwriting classification.

When choosing multi-class segmentation, there is a potentially higher risk of encountering deletion errors-missing some letters-especially for the Roboflow Instance Segmentation model. Since this type of segmentation expects very accurate character shapes presented to it during training, this can lead to missed letters in the recognized text when predicting texts in new images.

In the next steps of this research, the focus will be on integrating natural language processing (NLP) techniques to combine identified letters into words and sentences and to enable translation into other world languages. Developing an NLP module that interprets contextual relationships between characters requires collaboration with experts in the Palmyrene language. Continuous and dynamically updated expansion of the dataset by including photos of Palmyrene inscriptions with newly created transcriptions will ensure refinement of the current models and experiment with all possible data augmentation options. The study hopes to include the data in standard OCR training datasets, making it easily accessible for further experiments.

9 Conclusion

This study creates an instance segmentation model, which can identify and transcribe letters within high-quality photos of Palmyrene inscriptions with an accuracy of 93.5%, a significant step towards developing a Palmyrene OCR algorithm.

The development of tools capable of reading the characters and texts of dead languages has impactful sociological importance, as it links the past and the present. Inscriptions in dead languages carry information about important aspects of human history, in the case of Palmyrene Aramaic, recorded in the funerary, honorific, and dedicatory texts. By establishing OCR technology for this language, the potential for understanding ancient texts is expanded to a wider range of linguists, historians, archaeologists, museum keepers, and possibly even the non-scholarly public.

The final goal of humanists and linguists is to decipher the letters individually and understand the entire inscriptions and contextual meaning, which is not a simple objective that can be accomplished with a single computational task. However, this research is an essential step towards deciphering the texts in Palmyrene Aramaic, and the methodology used can be applied to the analysis and extraction of characters from other alphabets that do not use ligatures. The letters can be spatially separated from each other.

Acknowledgement: We would like to thank our grant agency for providing us with the funds necessary to perform this research. We would also like to thank our reviewers and editors for perfecting our manuscript.

Funding Statement: The results and knowledge included herein have been obtained owing to support from the following institutional grant. Internal grant agency of the Faculty of Economics and Management, Czech University of Life Sciences Prague, Grant No. 2023A0004 – “Text Segmentation Methods of Historical Alphabets in OCR Development”. <https://iga.pef.czu.cz/>. Funds were granted to T. Novák, A. Hamplová, O. Svojsě, and A. Veselý from the author team.

Author Contributions: The authors confirm contribution to the paper as follows: study conception and design: A. Hamplová; data collection and annotation: A. Hamplová; inscription checking and text transcriptions: A. Lyavdansky; analysis and interpretation of results: A. Hamplová; web application updates: D. Franc, O. Svojsě; draft manuscript preparation: A. Hamplová, A. Veselý, A. Lyavdansky; finance resource management: T. Novák; formatting references and the article structure in accordance with CMES template: A. Hamplová, T. Novák. All authors reviewed the results and approved the final version of the manuscript.

Availability of Data and Materials: The latest dataset version is available at Roboflow [36]. The code that was developed in this research is available on GitHub [37]. By publishing the dataset and code related to our research, transparency and reproducibility are ensured.

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

1. Hamplová A, Franc D, Veselý A. An improved classifier and transliterator of handwritten Palmyrene letters to Latin. *Neural Netw World*. 2022;32:181–95.
2. Franc D, Hamplová A, Svojsě O. Augmenting historical alphabet datasets using generative adversarial networks. In: *Data Sci Algorithms Syst Proc 6th Comput Methods Syst Softw 2022*; 2023;2:132–41.
3. Hamplová A. Palmyrene translation tool. Available from: <https://ml-research.pef.czu.cz>. [Accessed 2021].
4. Hamplová A, Franc D, Pavlicek J. Character segmentation in the development of palmyrene aramaic OCR. In: *Model-driven organizational and business agility*; Zaragoza, Spain; Jun 12–13, 2023. p. 80–95.
5. Elnabawy R, Elias R, Salem M. Image based hieroglyphic character recognition. In: *14th International Conference on Signal-Image Technology & Internet-Based Systems (SITIS)*; 2018; Las Palmas de Gran Canaria, Spain. p. 32–9.
6. Avadesh M, Goyal N. Optical character recognition for Sanskrit using convolution neural networks. In: *2018 13th IAPR International Workshop on Document Analysis Systems (DAS)*; 2018; Vienna, Austria. p. 447–52.
7. Gordin S, Gutherz G, Elazary A, Romach A, Jiménez E, Berant J, et al. Reading Akkadian cuneiform using natural language processing. *PLoS One*. 2020;15(10):1–16.
8. Karasu K, Bastan M. Turkish OCR on mobile and scanned document images. In: *2015 23rd Signal Processing and Communications Applications Conference (SIU)*; 2015; Malatya, Turkey. p. 2074–7.
9. Smith RW, Zanibbi R, Coüason B. History of the Tesseract OCR engine: what worked and what didn't. In: *SPIE Optical Engineering + Applications*; 2013; San Diego, California, USA.
10. Daoason JF, Bjarnadóttir K, Rúnarsson K. The journal fjolnir for everyone: the post-processing of historical OCR texts. In: *LREC 2014, Ninth International Conference on Language Resources and Evaluation*; 2014; Reykjavik, Iceland.
11. Rahaman A, Hasan MM, Shuvo MF, Ovi MAS, Rahman MM. Analysis on handwritten Bangla character recognition using ANN. In: *2014 International Conference on Informatics, Electronics & Vision (ICIEV)*; 2014; Dhaka, Bangladesh. p. 1–5.
12. Oni OJ, Asahiah FO. Computational modelling of an optical character recognition system for Yorùbá printed text images. *Sci Afr*. 2020;9:1–12.
13. Salah AB, Moreux JP, Ragot N, Paquet T. OCR performance prediction using cross-OCR alignment. In: *13th International Conference on Document Analysis and Recognition (ICDAR)*; 2015; Tunisia; p. 556–60.
14. Alghamdi MA, Alkhazi IS, Teahan WJ. Arabic OCR evaluation tool. In: *2016 7th International Conference on Computer Science and Information Technology (CSIT)*; 2016; Amman, Jordan. p. 1–6.
15. Gruber I, Hlaváč M, Hruz M, Železný M. Semantic segmentation of historical documents via fully-convolutional neural network. In: *Lecture Notes in Computer Science (LNAI)*. Cham, Linz, Austria: Springer; 2019. vol. 11658, p. 142–9.
16. Yi J, Wu P, Liu B, Hoepfner DJ, Metaxas DN, Fan W. Object-guided instance segmentation for biological images. *IEEE Trans Med Imag*. 2021 Sep;40(9):2403–14. doi:10.1109/TMI.2021.3077285.
17. Berry T, Dronen N, Jackson B, Endres I. Parking lot instance segmentation from satellite imagery through associative embeddings. In: *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*; 2019; Chicago IL USA. p. 528–31.

18. Luo Y, Han J, Liu Z, Wang M, Xia G. An elliptic centerness for object instance segmentation in aerial images. *J Remote Sens.* 2022; 2022:9809505. doi:10.34133/2022/980950.
19. Matsuzaka Y, Yashiro R. AI-based computer vision techniques and expert systems. *AI.* 2023;4(1):289–302.
20. Hafiz AM, Bhat GM. A survey on instance segmentation: state of the art. *Int J Multimed Inf Retr.* 2020;9(3):171–89.
21. Zhang H, Wang Y, Dayoub F, Sünderhauf N. Reference for ultralytics/utils/loss.py. Available from: <https://docs.ultralytics.com/reference/utils/loss>. [Accessed 2024].
22. Telicko J, Jakvics A. Comparative analysis of YOLOv8 and Mack-RCNN for people counting on fish-eye images. In: 2023 3rd International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME); 2023; Tenerife, Canary Islands, Spain. p. 1–6.
23. Wang X, Zhi M, Pan Z, Wang X. Summary of object detection based on convolutional neural network. In: Eleventh International Conference on Graphics and Image Processing (ICGIP 2019); 2020; Hangzhou, China. vol. 2020, no. 31.
24. Redmon J, Divvala S, Girshick R, Farhadi A. You only look once: unified, real-time object detection. In: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2016; Honolulu. p. 779–88.
25. Redmon J, Farhadi A. YOLO9000: better, faster, stronger. In: 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2017; Honolulu. p. 6517–25.
26. Redmon J, Farhadi A. YOLOv3: an incremental improvement. In: 2018 IEEE Conference on Computer Vision and Pattern Recognition (CVPR); 2018; Honolulu.
27. Roboflow Team. What is YOLOv5 instance segmentation? Available from: <https://roboflow.com/model/yolov5-instance-segmentation>. [Accessed 2022].
28. Jocher G. YOLOv5 by ultralytics. Available from: <https://zenodo.org/records/3908560>. [Accessed 2020].
29. Jocher G. Ultralytics. Available from: <https://github.com/ultralytics/ultralytics>. [Accessed 2023].
30. Gallagher J. Announcing roboflow train 3.0. Available from: <https://blog.roboflow.com/roboflow-train-3-0>. [Accessed 2023].
31. Adjetey C, Adu-Manu K. Content-based image retrieval using tesseract OCR engine and levenshtein algorithm. *Int J Adv Comput Sci Appl.* 2021;12(7):666–75.
32. Li Y, Yefeng Z, Doermann D. Detecting text lines in handwritten documents. In: 18th International Conference on Pattern Recognition (ICPR'06); 2006; Hong Kong, China. p. 1030–3.
33. Bugayong VE, Flores VJ, Linsangan NB. Google tesseract: optical character recognition (OCR) on HDD/SSD labels using machine vision. In: 2022 14th International Conference on Computer and Automation Engineering (ICCAE); 2022; Brisbane, Australia. p. 56–60.
34. Elleuch M, Tagougui N, Kherallah M. Deep learning for feature extraction of arabic handwritten script. In: George A, Petkov N, editors. *Lecture notes in computer science.* Cham: Springer; 2015. vol. 9257, p. 371–82.
35. Hajihashemi V, Arab Ameri MM, Alavi Gharahbagh A, Bastanfard A. A pattern recognition based holographic graph neuron for persian alphabet recognition. In: 2020 International Conference on Machine Vision and Image Processing (MVIP); 2020; Qom, Iran. p. 1–6.
36. Hamplová A. Palmyrene computer vision project. Available from: <https://universe.roboflow.com/adela-hamplova/palmyrene-tutzu>. [Accessed 2023].
37. Hamplová A. PalmyreneOCR. Available from: <https://github.com/adelajelinkova/PalmyreneOCR>. [Accessed 2024].

Appendix A—Training Details

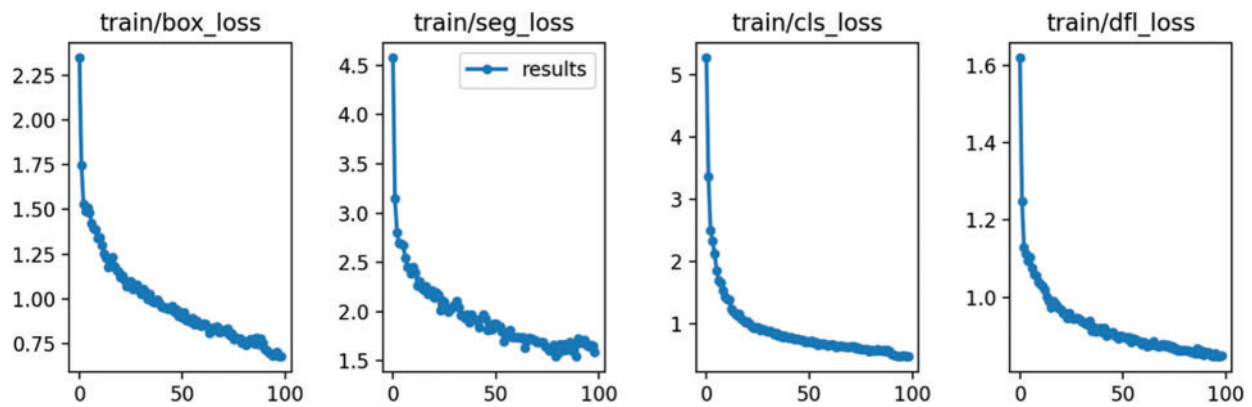


Figure A1: YOLOv8 multi-class training loss

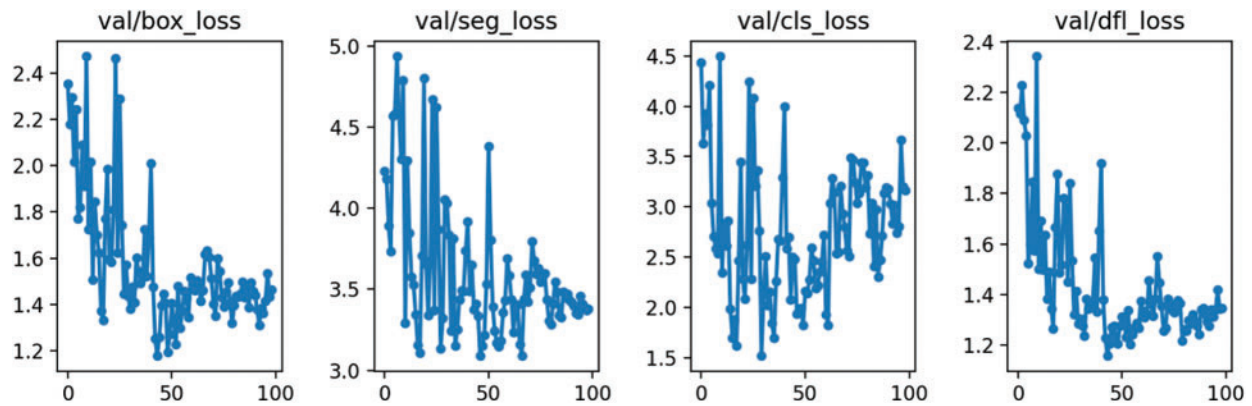


Figure A2: YOLOv8 multi-class validation loss

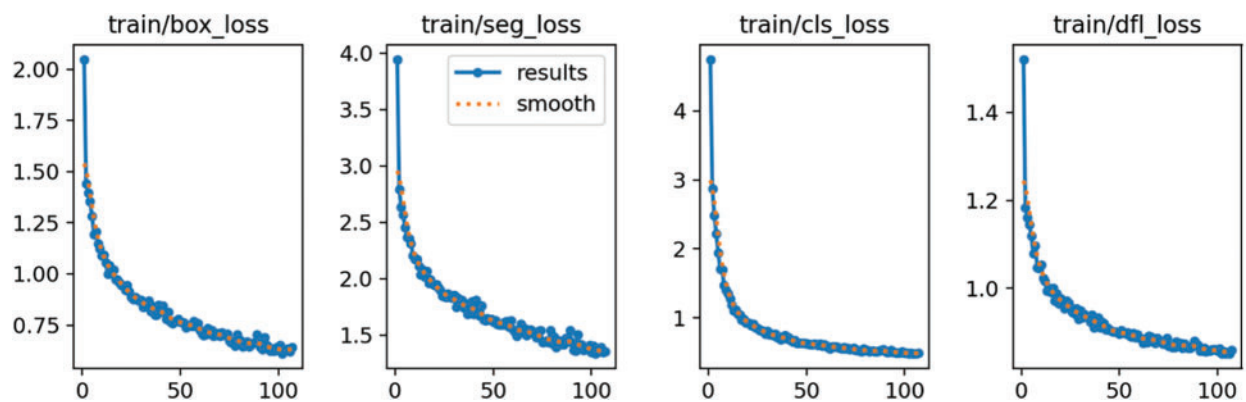


Figure A3: Roboflow 3.0 multi-class instance segmentation (accurate) training loss

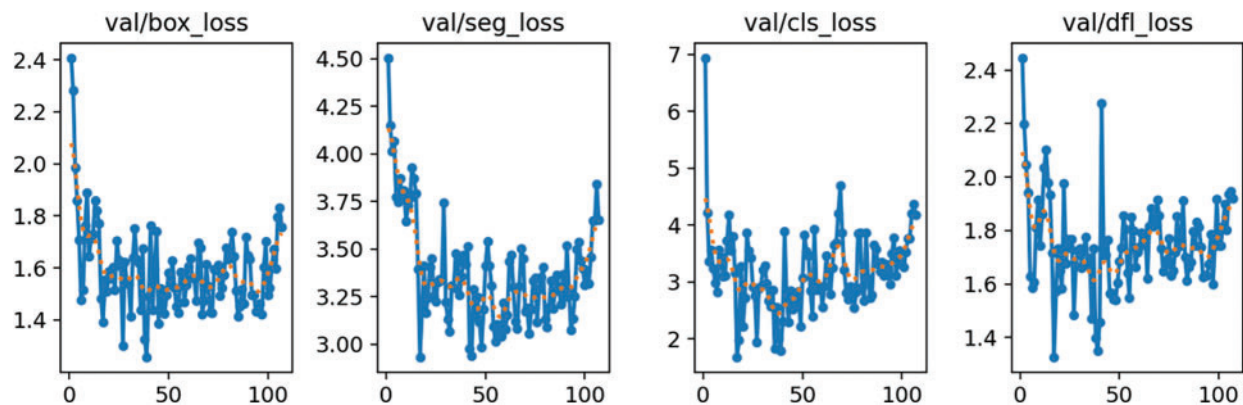


Figure A4: Roboflow 3.0 multi-class Instance segmentation (accurate) validation loss

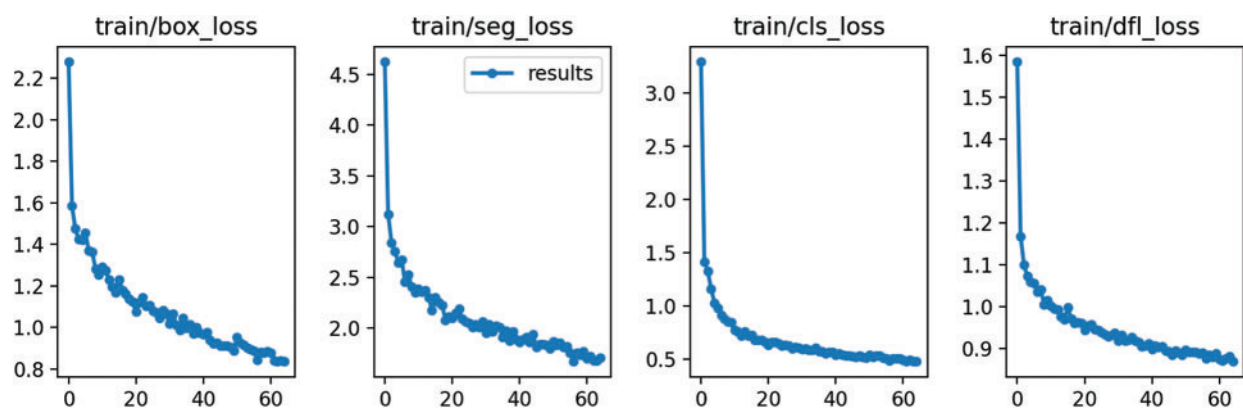


Figure A5: YOLOv8 single-class training loss

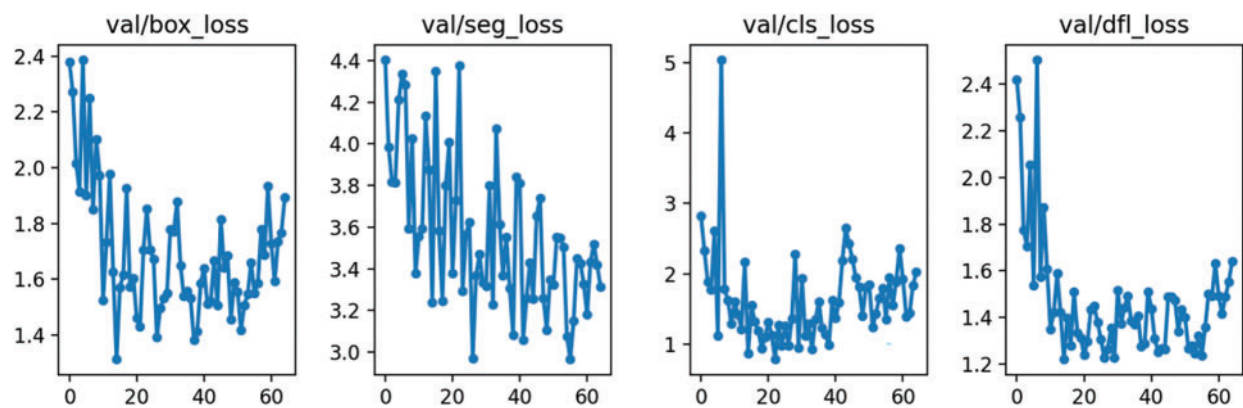


Figure A6: YOLOv8 single-class validation loss

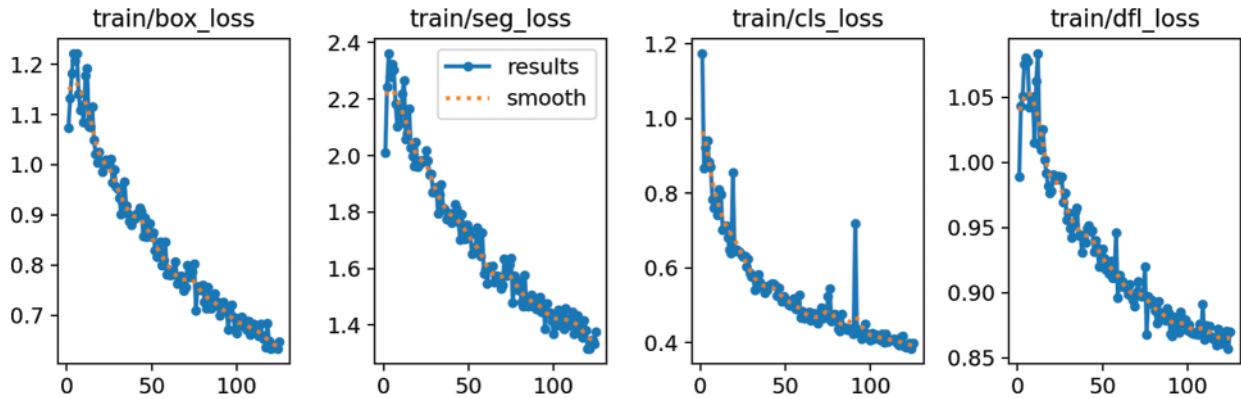


Figure A7: Roboflow 3.0 single-class Instance segmentation (accurate) training loss

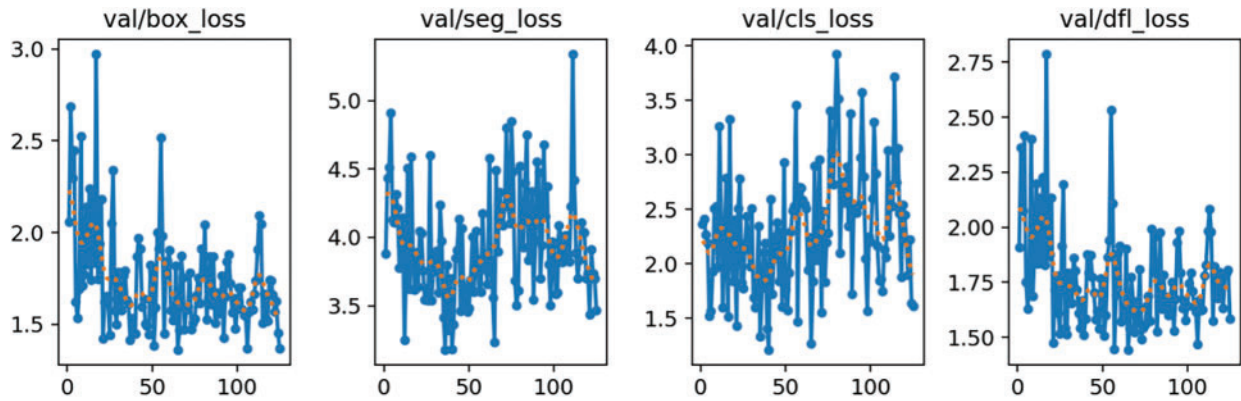


Figure A8: Roboflow 3.0 single-class Instance segmentation (accurate) validation loss

Table A1: YOLOv8 layers overview

Index	From	n	Params	Module	Arguments
0	-1	1	928	ultralytics.nn.modules.Conv	[3, 32, 3, 2]
1	-1	1	18560	ultralytics.nn.modules.Conv	[32, 64, 3, 2]
2	-1	1	29056	ultralytics.nn.modules.C2f	[64, 64, 1, True]
3	-1	1	73984	ultralytics.nn.modules.Conv	[64, 128, 3, 2]
4	-1	2	197632	ultralytics.nn.modules.C2f	[128, 128, 2, True]
5	-1	1	295424	ultralytics.nn.modules.Conv	[128, 256, 3, 2]
6	-1	2	788480	ultralytics.nn.modules.C2f	[256, 256, 2, True]
7	-1	1	1180672	ultralytics.nn.modules.Conv	[256, 512, 3, 2]
8	-1	1	1838080	ultralytics.nn.modules.C2f	[512, 512, 1, True]
9	-1	1	656896	ultralytics.nn.modules.SPPF	[512, 512, 5]
10	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']
11	[-1, 6]	1	0	ultralytics.nn.modules.Concat	[1]
12	-1	1	591360	ultralytics.nn.modules.C2f	[768, 256, 1]
13	-1	1	0	torch.nn.modules.upsampling.Upsample	[None, 2, 'nearest']

(Continued)

Table A1 (continued)

Index	From	n	Params	Module	Arguments
14	[-1, 4]	1	0	ultralytics.nn.modules.Concat	[1]
15	-1	1	148224	ultralytics.nn.modules.C2f	[384, 128, 1]
16	-1	1	147712	ultralytics.nn.modules.Conv	[128, 128, 3, 2]
17	[-1, 12]	1	0	ultralytics.nn.modules.Concat	[1]
18	-1	1	493056	ultralytics.nn.modules.C2f	[384, 256, 1]
19	-1	1	590336	ultralytics.nn.modules.Conv	[256, 256, 3, 2]
20	[-1, 9]	1	0	ultralytics.nn.modules.Concat	[1]
21	-1	1	1969152	ultralytics.nn.modules.C2f	[768, 512, 1]
22	[15, 18, 21]	1	2780606	ultralytics.nn.modules.Segment	[26, 32, 128, [128, 256, 512]]

Appendix B—Training Details**Table A2:** Real texts and transcriptions of “Inv. 1438/8582, Archaeological museum of Palmyra”, using all models

Real text	YOLO single-class	YOLO multi-class	Roboflow single-class	Roboflow multi-class
?qm? brt ʿzyb mlkw ḥbl	?<m? brḥ ʿzyb ṣlkw ṣbz	?qm? brt ʿzyb mlkw ḥbl	?ym? brḥ ʿzyb mlky ṣbz	?qm? brt rzyb mlkw ḥbl

Table A3: Real texts and transcriptions of “Inv. 88.AA.50, The Getty Villa Museum”, using all models

Real text	YOLO single-class	YOLO multi-class	Roboflow single-class	Roboflow multi-class
mgy br mʿny	mgy br mʿn-	mgy br mʿny	mgy br m-ny	mgy br mḥʿny

Table A4: Real texts and transcriptions of “Inv. AO 2205, Musée du Louvre”, using all models

Real text	YOLO single-class	YOLO multi-class	Roboflow single-class	Roboflow multi-class
nysn šnt [3] [100] 5 3 qbr?	nyyn šny 5+1+1+1 qbr? ʿy zḥdbwl br	nysn šnt 5+1+1+1 100+1 br?	nysn šnḥ 5+1+1+1 100 br?	nysn šnt 5+1+1+1 100z1
[d]y zbdbwl br [...] <u>h</u> br ʿtršwr bny kmr? dy lh wlbwhy	1 hr ʿhršwr yny k mr? dy thg wlbnyh	y 1 bdbwl br h br ʿtršwr bny n kmr? dy lh wlbwnḥy	y zbdbwl br y br ʿhršwr bn- kmr? dy -h wlbh-ny	br? y 1 zbdbwl br - br ʿtršwr bny kmr? dy lh wlbw-y

Table A5: Real texts and transcriptions of “Inv. 95.28, The MET Museum”, using all models

Real text	YOLO single-class	YOLO multi-class	Roboflow single-class	Roboflow multi-class
bryk šmh l ^ʃ lm ^ʔ t̥b ^ʔ w ^ʃ hmn ^ʔ ʃbd wmwd ^ʔ hggw br	bryy ^ʔ k šh ^h l ^ʃ lmmg y ^ʔ k ^ʔ 1n ^h mn ^ʔ ʃkd w ^ʔ wm20wd ^ʔ hggw kr 20h20y ^ʔ bd yrh20 dk ^ʔ ʃl h ^ʔ ywqy	bryk šmh ^h l ^ʃ lmg t̥b ^ʔ wrhmn ^ʔ ʃrbd wmwd ^ʔ h ^ʔ trw br	kr20k šmh l ^ʃ lmg t̥k ^ʔ nršmlm ʃkn nmnn ^ʔ hgg20 sr mhp ^ʔ k ^ʔ sr 20nh20 dk ^ʔ 1l h2020hw ʃ20- ^ʔ ʔbnhn - ^ʔ h20h- b20rh qš20r ʃšt n 100 wnw-	bryk hmh l ^ʃ lm ^ʔ -b ^ʔ wrh- ^ʔ ʃbd wmwd ^ʔ h-w br
yhyb ^ʔ br y ^ʃ h ^ʔ dk ^ʔ ʃl h ^ʔ ywh ^ʔ w ^ʔ h ^ʔ why byr ^h qyn ^ʔ šnt 5.100 +40+3	1nh20 ^ʔ ʔbwhy w ^ʔ h ^ʔ why byr ^h qlyw šnt 5 100 10 yyw-	yhyb ^ʔ br yr ^h dk ^ʔ m ʃ- h ^ʔ ywh ^ʔ w ^ʔ h ^ʔ why byr ^h qnyw šnt 5 r 100 20—		yhyb ^ʔ br -r ^h -k ^ʔ ʃl h ^ʔ ywh-

Table A6: Real texts and transcriptions of “Inv. 98.19.4, The MET Museum”, using all models

Real text	YOLO single-class	YOLO multi-class	Roboflow single-class	Roboflow multi-class
hbl [ʃ]g ^ʔ [br] zbd ^ʃ th ʃbt h ^ʔ zbddq ^h		hbl g ^ʔ zbd ^ʃ th	tb- g ^ʔ zbddq ^h	ʃbl g ^ʔ zbddt ^ʔ h

Table A7: Real texts and transcriptions of “Inv. 125024, The British Museum”, using all models

Real text	YOLO single-class	YOLO multi-class	Roboflow single-class	Roboflow multi-class
ʔqm ^ʔ brt h ^ʔ bzy hbl	ʔqh ^ʔ z br ^h h ^ʔ bn ^ʔ p kw ^ʔ ʃ	ʔqm- brt h ^ʔ b-y hbl	h ^ʔ qm- br ^h h ^ʔ b1y kw ^ʔ ʃ	ʔqmq ^ʔ - brt h ^ʔ bzy hbl r

Attachment 6

Attachment 6: A. Hamplová, D. Franc, J. Pavlíček, A. Romach and S. Gordin, “Cuneiform Reading Using Computer Vision Algorithms,” SPML 2022: Proceedings of the 2022 5th International Conference on Signal Processing and Machine Learning, 2022.

Cuneiform Reading Using Computer Vision Algorithms

Adela Hamplova
Czech University of Life Sciences in
Prague
hamplova@pef.czu.cz

David Franc
Czech University of Life Sciences in
Prague
francd@pef.czu.cz

Josef Pavlicek
Czech University of Life Sciences in
Prague
pavlicek@pef.czu.cz

Avital Romach
Tel Aviv University
avitalromach@tauex.tau.ac.il

Shai Gordin
Ariel University
shaigo@ariel.ac.il

ABSTRACT

This paper presents a new method for computer-assisted recognition of horizontal strokes in photographs of cuneiform tablets with 90,52 % accuracy. The cuneiform script is the oldest attested writing system in the world, used for over three thousand years throughout the ancient Near East, primarily by the cultures of Mesopotamia (modern Iraq). It was impressed on clay tablets and engraved on stone slabs by writing strokes. Researchers have been trying to speed up the process of reading the tablets using different methods, as manual copying of the tablets and their transliteration is time consuming. This research, therefore, aims to recognize the elementary components, i.e., the strokes, of cuneiform signs from photographs of ancient cuneiform tablets, in order to enable effective OCR using the latest computer vision algorithms. The main difference between other approaches and ours is that we work directly with the two-dimensional photographs, instead of three-dimensional models, as there are many more 2D images available in public online repositories. The goal is to partly automate the process of identifying and reading cuneiform signs, thus speeding up the process of rediscovering these ancient texts and civilizations.

CCS CONCEPTS

• computing methodologies; • artificial intelligence; • computer vision;

KEYWORDS

cuneiform, logo-syllabic script, pattern recognition

ACM Reference Format:

Adela Hamplova, David Franc, Josef Pavlicek, Avital Romach, and Shai Gordin. 2022. Cuneiform Reading Using Computer Vision Algorithms. In *2022 5th International Conference on Signal Processing and Machine Learning (SPML 2022)*, August 04–06, 2022, Dalian, China. ACM, New York, NY, USA, 4 pages. <https://doi.org/10.1145/3556384.3556421>

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](https://permissions.acm.org).

SPML 2022, August 04–06, 2022, Dalian, China
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9691-2/22/08...\$15.00
<https://doi.org/10.1145/3556384.3556421>

1 INTRODUCTION

1.1 Previous work on cuneiform sign recognition

Cuneiform writing consists of signs that are formed by combining three types of strokes - horizontal, vertical, and oblique. Despite this relative simplicity, cuneiform signs are hard to identify because of their three dimensional character. From the inception of the field, this was difficult to represent in a 2D format. Two solutions were found: taking 2D images of cuneiform tablets or creating hand-copies, 2D black and white drawings made by scholars of the tablet's strokes. In recent years, 2D images have become rather ubiquitous and in sufficient quality for machine learning applications. The largest repositories of such images are the British Museum, Louvre Museum, Cuneiform Digital Library Initiative, and Yale Babylonian Collection. 2D+ and 3D models of cuneiform tablets have also become a possibility since the early aughts, although these are still more expensive and labor-intensive to produce.

Previous research in identifying cuneiform signs or strokes have used mostly 3D models. Two main research groups developed stroke extraction through geometrical feature identification [1–3]. Mara and Krömker [4] extracted strokes as Scalable Vector Graphic (SVG) images, which practically created hand-copies automatically as vector images. Hand-copies and 2D projections of 3D models were used for querying signs by example, using convolutional neural networks with data augmentation by Rusakov et al. [5]. Previous work on 2D images has only recently started. Dencker et al. used 2D images for training a weakly supervised machine learning model in the task of sign detection in a given image [6]. Rusakov et al. [7] used 2D images of cuneiform tablets for querying cuneiform signs by example and by schematic expressions representing the stroke combinations. No previous research has attempted to identify strokes from 2D images.

1.2 Identifying strokes with mathematical methods

In the first steps, we looked for methods of finding stroke features (horizontal and vertical) using classic methods of working with images. We designed a software, which will allow highlighting of stroke characters using convolutional image filtering methods. These are commonly used edge filters that allow the suppression of the surroundings of the desired objects and highlight their edges.

In contrast to the classical filtering methods, which are based on the gradient (brightness change) of the neighbouring pixels, using a convolution mask shifted along the X and Y axis across

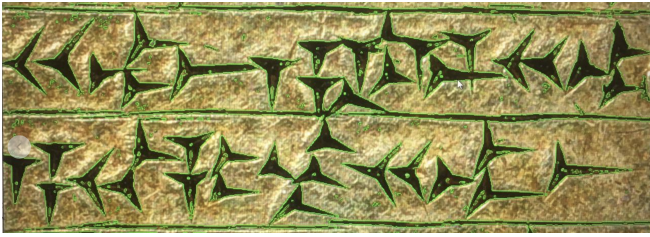


Figure 1: Edge detection of cuneiform signs from a rare gold tablet in the Yale Babylonian Collection (photo credit: Klaus Wagensohnner)

the image (pixel matrix), we used the edge orientations of the highlighted characters. In our case, we used the properties of Hough transformation [8], i.e., lines and their orientation.

Thanks to Hough lines, we can decide if the character we find is really the one we are looking for. The stroke is oriented either horizontally or vertically or at some other angle that can be described by us. The classic edge detection filters emphasize the edge, but due to the difference in the quality of images, sometimes a part of the edge is left out and needs to be approximated. For such approximation, the Hough line in the correct angle can help calculate the missing parts of edges. Thus, based on the adjacency, it is possible to highlight the object and, conversely, filter out the noise. By noise we mean found edges that are not guided at the angle we require.

The image above then shows how the software works. However, its use is limited by the manual work of the operator, who, based on expert knowledge, sets the required parameters such as the maximum connection length of adjacent edges, edge angles and image brightness balance so that the results are distinctive and easy to read. This solution is not ideal for automation, because cuneiform tablets are photographed at different angles of light. For example, certain settings that are suitable for the upper left quarter of the image, will not be for the lower right, due to different lighting conditions. Resetting the filter would highlight the lower part but suppress the upper one. Thus, to automate this process, it is necessary to supplement classical mathematical methods with machine vision technologies based on artificial intelligence and heuristic operations.

Of course, an automated solution cannot do without statistical methods that quantitatively verify the accuracy of the technology used. Machine vision technology based on artificial intelligence brings several advantages.

The problem with artificial intelligence is the fact that it is not possible to find out why the machine predicted the way it did.

While the outputs of algorithmic approaches are always clear, as they are calculated based on input data and algorithm, in the case of artificial intelligence the results are not predictable, as they are gained by the process of learning. The results then need verifying by quantitative (i.e., statistical) method.

It is therefore necessary to implement the following solution scheme. Our goal is to suggest a suitable artificial intelligence technology, train it on labelled data created by our Assyriological team members, verify the ability of the network to recognize strokes of cuneiform signs.

1.3 Computer Vision and Neural Networks

Computer vision is a widely used method to identify objects in pictures and is evolving rapidly. Computer vision is associated with convolutional neural networks (CNN), also known as convnets in which densely connected layers learn global patterns and convolutional layers learn local patterns in small 2D windows [9]. The most contemporary algorithms are, among others, YOLOv5 [10] and Detecto SSD/ResNet [11]. Both algorithms are complementary: while YOLOv5 is using the library tensorflow, Detecto is using the library torch.

YOLOv5 is a complex solution created by ultralytics and is available on GitHub [10]. It contains a pre-trained network as well as a training and detection script. The input images need to be square and contain labels in .txt format. Detecto is another solution (available as a Python library). It uses a single shot detector (SSD) with ResNet and like YOLO, it contains training and detection scripts.

With labelled data, we need to implement most contemporary versions of several architectures of Convolutional Neural Networks, one-stage, or two-stage models. For this, we use an open Python platform Google Collaboratory or Kaggle and we evaluate the performance of each architecture by standard measures such as recall r (1), precision p (2), F -measure F (3).

$$p = \frac{TP}{TP + FP} \quad (1)$$

$$r = \frac{TP}{TP + FN} \quad (2)$$

$$F = \frac{2 \cdot r \cdot p}{r + p} \quad (3)$$

Where:

TP = true positives (real strokes, that have been correctly found)

FP = false positives (predicted strokes, that are not there in real)

FN = false negatives (real strokes, that were not found)

2 TRAINING AND EVALUATION OF YOLOV5 AND DETECTO

2.1 Dataset creation

To train high quality models, we need many manually labelled input images. The recommended number of pictures for each class is about 1000 [12]. Our Assyriological team tagged thousands of horizontal strokes in eight tablets from the Yale Babylonian Collection (Table 1; made available through the kind permission of Agnete W. Lassen and Klaus Wagensohnner).

The full tablet images were split into squares of 416x416x3 pixels. Then, they were labelled using the python software tool “labelImg.py”, which creates files in xml format, each file containing the name of the image, the path and the labels (Fig. 2). The labels format is called Pascal VOC and consists of the coordinates x_{min} , y_{min} , x_{max} and y_{max} and class name.

The dataset is made up of 823 labelled images with 7355 annotation records. We used the augmentation platform Roboflow using grayscale, saturation, and exposure augmentation. It contains 1700 images in the training set, 165 images in the validation subset and 82 testing images.

Table 1: the eight tablets that were tagged and their metadata. The information is taken from the Yale Babylonian Collection website. The abbreviations for the publications of hand copies can be found through CDLI (https://cdli.ox.ac.uk/wiki/abbreviations_for_assyriology).

Yale ID	CDLI ID	material	period	genre	hand copy publication
YPM BC 014442	P504832	clay	Neo-Assyrian	literary	CT 13 1, 3
YPM BC 023856	P293426	clay	Old-Babylonian	literary	JCS 1 22-23
YPM BC 002575	P297024	clay	Neo/Late-Babylonian	commentary	BRM 4 24
YPM BC 016773	P293444	limestone	Early Old-Babylonian	inscription	YOS 1 36
YPM BC 016780	P293445	limestone	Early Old-Babylonian	inscription	YOS 1 35
YPM BC 016869	P429204	clay	Middle Assyrian	inscription	YOS 9 71
YPM BC 021204	P308129	clay	Middle Assyrian?	medical text	FS Sachs 18, no. 16
YPM BC 021234	P308150	clay	Old-Babylonian	hymn	YNER 3 6-7

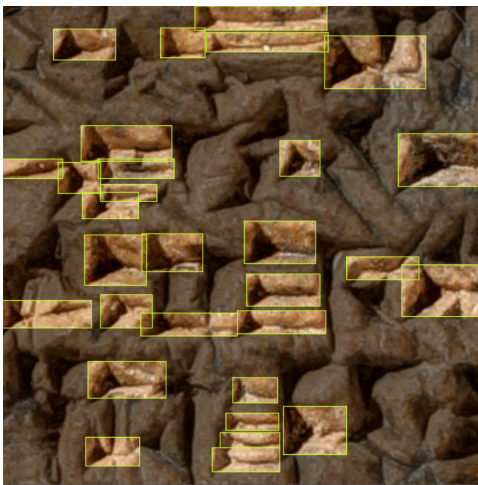


Figure 2: Example of training data created by the assyriological team.

2.2 CNN training

In cases of both YOLOv5 and Detecto, we use training scripts with alternation of the dataset only. For prediction, we need to edit the script, so that the predicted images and labels are saved as files and can be displayed and worked with without the limitation of using prediction notebooks only.

The training of YOLOv5 architecture (283 layers) was conducted for 100 epochs at Google Collaboratory, using CUDA on GPU Tesla T4 with 40 multiprocessors, 15109 MB total memory.

The training of Detecto SSD/Resnet (51 layers - ResNet 50 plus 1 Conv2D layer) was conducted on a pre-trained file `fasterrcnn_resnet50_fpn_coco-258fb6c6.pth` by unfreezing some layers and retraining them for 50 epochs at Google Collaboratory, using the same GPU Tesla T4.

2.3 Evaluation

The evaluation results of the testing set can be seen in Table 2. Detecto has found horizontal strokes with 90,53 % recall, while YOLOv5 reached 43,53 %. Example of such predictions can be seen

Table 2: Testing set results evaluation

Network	YOLOv5	Detecto
Precision	0,4211	0,7450
Recall	0,4353	0,7053
F-Measure	0,4281	0,8173
Fake of all strokes found	57,89 %	25,50 %
Correct	43,53 %	90,52 %
Correct of all strokes found	0,7450	0,9053



Figure 3: Detecto SSD/ResNet predictions

in Figure 3, where red boxes are predictions and green boxes are ground truth labels.

2.4 Horizontal stroke results interpretation

From the results and evaluation, YOLOv5 is less successful than Detecto. YOLOv5 successfully identifies most strokes however the false positive identifications exceed 50% (more than half strokes are false), while in the case of Detecto it is only 25,5 % and 90,5 % is found correctly. The reason might be that YOLOv5 is mostly used in video processing so there is a lot of input data (for example 30 images per second) and it is not important if some frames are detected incorrectly.

Shifeng et al. proved that two-stage detection models usually achieve higher accuracy than one-stage models [13], which proved

to be true even in case of Detecto and YOLOv5, as Detecto is a two-stage model and YOLOv5 is a one-stage model.

2.5 Comparison with other authors' works

In other research projects focused on similar topics - recognition of objects from images using similar architectures - there are following results.

Recognition of Bangladeshi signs with models constructed by Ghosh et al.'s [14], reached 96.46% accuracy on MobileNet. In Cho Junghwan's et al. research [12] of CT body scans they reached 97% accuracy on GoogLeNet Inception v1 architecture. Their dataset contained 4000 very high-quality images.

We have reached 98,21 % accuracy in classification of 2000 Palmyrene letters per class on custom CNN architecture with 4 Convolutional / Max Pooling blocks. [15]

2.6 Research next steps

Detection success rate can be improved with higher amounts of labelled images and their variability (different light conditions, colours, shadows) and if more augmentation methods are used.

Future plans include adding vertical and oblique strokes to the training sets. We may also use oriented bounding boxes for oblique strokes, but we will need to edit Detecto's algorithm, so that it can work with bounding box angles. We may also attempt to increase the accuracy with different neural network architectures, such as RCNN using selective search, as it usually reaches a better detection rate. With new labelled data we can also start optimizing neural networks configurations to reach the maximum accuracy and have a better comparison, with RCNN included. There are many experiments to be done such as observing the influence of number of convolutional layers, number of ignored layers (transfer learning), number of epochs, steps in epochs, learning rate, optimizing method, images depth. The accuracy can also be improved in postprocessing phases. Early stopping with patience attribute may be experimentally used and compared. Automation, logging, and visualization are other tools that could help us reach better results.

3 CONCLUSION

We have compared mathematical methods (edge detection) and artificial intelligence for object detection and chose to train an object detection models.

Two neural network architectures YOLOv5 and Detecto were developed to classify and localize horizontal strokes in cuneiform tablet images divided into 416x416 squares. The classifier based on Detecto reaches 90,5% accuracy, with 25% false positive predictions while the classifier based on YOLOv5 scores a lower accuracy on the cuneiform data.

ACKNOWLEDGMENTS

The project Cuneiform analysis using Convolutional Neural Networks reg. no. 31/2021 was financed from the OP RDE project Improvement in Quality of the Internal Grant Scheme at CZU, reg. no. CZ.02.2.69/0.0/0.0/19_073/0016944.

The work of SHG and AR was supported by the cooperation grant between CULS Prague and Ariel University, Israel (RA2000000010).








REFERENCES

- [1] Mara, Hubert, Susanne Krömker, Stefan Jakob, and Bernd Breuckmann. 2010. "GigaMesh and Gilgamesh - 3D Multiscale Integral Invariant Cuneiform Character Extraction." In 11th International Symposium on Virtual Reality, Archaeology and Intelligent Cultural Heritage (Vast 2010), 131–38. Aire-La-Ville: The Eurographics Association. <https://doi.org/10.2312/VAST/VAST10/131-138>.
- [2] Fisseler, Denis, Frank Weichert, Gerfrid Müller, and Michele Cammarosano. 2013. "Towards an Interactive and Automated Script Feature Analysis of 3D Scanned Cuneiform Tablets." In Scientific Computing and Cultural Heritage 2013. http://www.cuneiform.de/fileadmin/user_upload/documents/scch2013_fisseler.pdf.
- [3] Rothacker, Leonard, Denis Fisseler, Frank Weichert, Gernot Fink, and Gerfrid Müller. 2015. "Retrieving Cuneiform Structures in a Segmentation-Free Word Spotting Framework." In Proceedings of the 3rd International Workshop on Historical Document Imaging and Processing (Hip 2015), 129–36. New York, NY: Association for Computing Machinery. <https://doi.org/10.1145/2809544.2809562>.
- [4] Mara, Hubert, and Susanne Krömker. 2013. "Vectorization of 3D-Characters by Integral Invariant Filtering of High-Resolution Triangular Meshes." In Proceedings of the International Conference on Document Analysis and Recognition (Icdar 2013), 62–66. Piscataway, NJ: IEEE Computer Society. <https://doi.org/10.1109/ICDAR.2013.21>.
- [5] Rusakov, Eugen, Kai Brandenbusch, Denis Fisseler, Turna Somel, Gernot A. Fink, Frank Weichert, and Gerfrid G. W. Müller. 2019. "Generating Cuneiform Signs with Cycle-Consistent Adversarial Networks." In Proceedings of the 5th International Workshop on Historical Document Imaging and Processing, 19–24. HIP '19. New York, NY, USA: Association for Computing Machinery. <https://doi.org/10.1145/3352631.3352632>.
- [6] Dencker, Tobias, Pablo Klinkisch, Stefan M. Maul, and Björn Ommer. 2020. "Deep Learning of Cuneiform Sign Detection with Weak Supervision Using Transliteration Alignment." PLoS ONE 15 (12): e0243039. <https://doi.org/10.1371/journal.pone.0243039>.
- [7] Rusakov, Eugen, Turna Somel, Gernot A. Fink, and Gerfrid G. W. Müller. 2020. "Towards Query-by-expression Retrieval of Cuneiform Signs." In 2020 17th International Conference on Frontiers in Handwriting Recognition (Icfhr), 43–48. <https://doi.org/10.1109/ICFHR2020.2020.00019>.
- [8] Richard O. Duda and Peter E. Hart. 1972. Use of the Hough Transformation to Detect Lines and Curves in Pictures. Communications of the ACM. 15(1), 11-15. <https://doi.org/10.1145/361237.361242>
- [9] Francois Chollet. 2018. Deep Learning with Python, Manning, ISBN 9781617294433
- [10] Releases ultralytics/yolov5. 2021. GitHub. Retrieved March 5, 2022, from <https://github.com/ultralytics/yolov5/releases>
- [11] Detecto PyPI. Retrieved March 5, 2022, from <https://pypi.org/project/detecto/>
- [12] Junghwan Cho, Kyewook Lee, Ellie Shin, Garry Choy, Synho Do. 2015. How much data is needed to train a medical image deep learning system to achieve necessary high accuracy? <https://arxiv.org/abs/1511.06348>.
- [13] Shifeng Zhang, Longyin Wen, Xiao Bian, Zhen Lei, Stan Z. Li. 2018. Single-Shot Refinement Neural Network for Object Detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 4203-4212. <https://doi.org/10.48550/arXiv.1711.06897>
- [14] Tapotosh Ghosh, Md. Min-Ha-Zul Abedin, Shayer Mahmud Chowdhury, Zarin Tasnim, Tajbia Karim, S. M. Salim Reza, Sabrina Saika, Mohammad Abu Yousuf. 2020. Bangla handwritten character recognition using MobileNet V1 architecture. Bulletin of Electrical Engineering and Informatics, 9(6), 2547-2554. <https://doi.org/10.11591/eei.v9i6.2234>
- [15] Adéla Hamplová, David Franc, Jan Tyrychtr. 2022. Historical Alphabet Transliteration Software Using Computer Vision Classification Approach. CSOC2022 conference Proceedings. Springer Series: Lecture Notes in Networks and Systems, Prague, Czech Republic. 2022. ISSN 2367-3370. 2022.

Attachment 7

A. Hamplová, A. Romach, J. Pavlíček, A. Veselý, M. Čejka, D. Franc and S. Gordin, “Cuneiform stroke recognition and vectorization in 2D images,” *Digital Humanities Quarterly*, 2023.

Cuneiform Stroke Recognition and Vectorization in 2D Images

Adéla Hamplová <hamplova_at_pef_dot_czu_dot_cz>, Czech University of Life Sciences Prague  <https://orcid.org/0000-0002-1012-650X>
Avital Romach <avital_dot_romach_at_yale_dot_edu>, Yale University  <https://orcid.org/0000-0001-9199-3228>
Josef Pavlíček <pavlicek_at_pef_dot_czu_dot_cz>, Czech University of Life Sciences Prague  <https://orcid.org/0000-0002-3959-5406>
Arnošt Veselý <vesely_at_pef_dot_czu_dot_cz>, Czech University of Life Sciences Prague  <https://orcid.org/0000-0001-8979-1336>
Martin Čejka <cejkamartin_at_pef_dot_czu_dot_cz>, Czech University of Life Sciences Prague  <https://orcid.org/0000-0002-2909-486X>
David Franc <francd_at_pef_dot_czu_dot_cz>, Czech University of Life Sciences Prague  <https://orcid.org/0000-0003-3160-9559>
Shai Gordin <shaigo_at_ariel_dot_ac_dot_il>, Ariel University; Open University of Israel  <https://orcid.org/0000-0002-8359-382X>

Abstract

A vital part of the publication process of ancient cuneiform tablets is creating hand-copies, which are 2D line art representations of the 3D cuneiform clay tablets, created manually by scholars. This research provides an innovative method using Convolutional Neural Networks (CNNs) to identify strokes, the constituent parts of cuneiform characters, and display them as vectors — semi-automatically creating cuneiform hand-copies. This is a major step in optical character recognition (OCR) for cuneiform texts, which would contribute significantly to their digitization and create efficient tools for dealing with the unique challenges of Mesopotamian cultural heritage. Our research has resulted in the successful identification of horizontal strokes in 2D images of cuneiform tablets, some of them from very different periods, separated by hundreds of years from each other. With the Detecto algorithm, we achieved an F-measure of 81.7% and an accuracy of 90.5%. The data and code of the project are available on GitHub.

1 Introduction

1.1 Cuneiform Texts and Artificial Intelligence

Cuneiform is one of the earliest attested writing systems, and for thousands of years cuneiform has also been the dominant script of the ancient Middle East, a region stretching roughly from the Persian Gulf to modern Turkey's highlands, and south across the Levant into Egypt. Cuneiform texts appeared from the end of the fourth millennium BCE until they fell out of use in the early centuries CE. The script was used for writing a plethora of different documents: legal, administrative, and economic documents; correspondence between private individuals, or high-officials and their kings; some of the oldest works of literature; royal inscriptions describing the deeds of great kings; as well as lexical and scientific compendia, some of which form the basis of the Greco-Roman sciences the Western world is built upon today. Hundreds of thousands of cuneiform documents have been discovered since excavations began in the 1850s. Recent estimates indicate the cuneiform text corpus is second in size only to that of ancient Greek [Streck 2010].

The rising application of artificial intelligence to various tasks provides a prime opportunity for training object detection models to assist the digital publication of cuneiform texts on a large scale. This will help set up a framework for cultural heritage efforts of preservation and knowledge dissemination that can support the small group of specialists in the field.

Cuneiform provides unique challenges for object detection algorithms, particularly OCR methods. Cuneiform tablets, on which the texts were written, are 3D objects: pieces of clay which were shaped to particular sizes. While the clay was still moist, scribes used styli with triangular edges to create impressions on the clay in three possible directions: horizontal, vertical, or oblique (also *Winkelhaken*; see Figure 1). "Diagonal" strokes are also found in the literature, but these are technically either another type of horizontal or an elongated oblique impression [Cammarosano 2014] [Cammarosano et al. 2014] [Bramanti 2015]. Each of these impressions is called a stroke (or wedge, due to their shape). Combinations of different strokes create characters, usually referred to as signs [Taylor 2015]. Cuneiform can be more easily read when there is direct light on the tablet, especially from a specific angle that casts shadows on the different strokes.

1

2

3

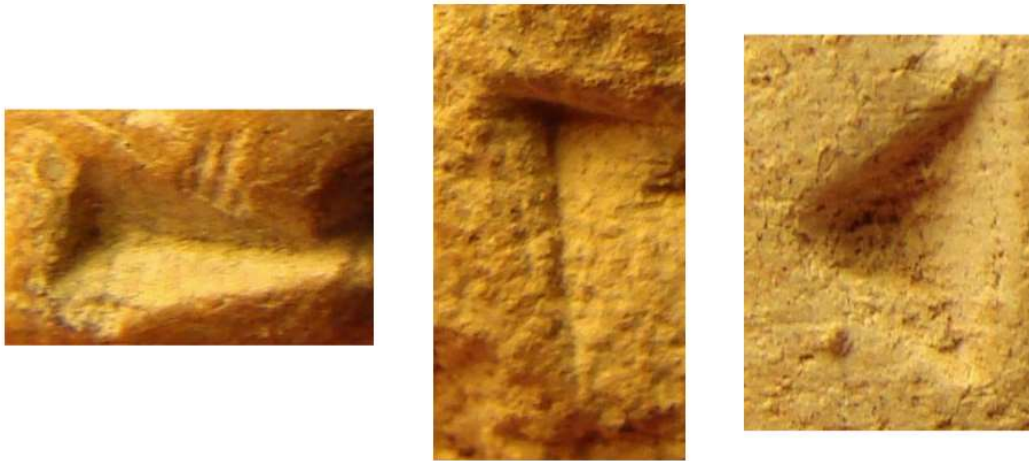


Figure 1. The main cuneiform strokes taken from Neo-Babylonian signs, from left to right: AŠ (<https://labasi.acdh.oeaw.ac.at/tablets/glyph/detail/10341>), DIŠ (<https://labasi.acdh.oeaw.ac.at/tablets/glyph/detail/10474>), and U or *Winkelhaken* (<https://labasi.acdh.oeaw.ac.at/tablets/glyph/detail/11430>), as recorded in the LaBaSi palaeographical database.

From the inception of research in cuneiform studies, tablets were difficult to represent in a modern 2D publishing format. Two solutions were found: 4

- When possible, 2D images of cuneiform tablets were taken. However, for most of the field's history, such images were extremely costly to produce and print, and they were often not in sufficient quality for easy sign identification (for the history of early photography and cuneiform studies, see [Brusius 2015]).
- The second solution was creating hand-copies, 2D black and white line art made by scholars of the tablets' strokes. This was the most popular solution. The disadvantage of this method is that it adds a layer of subjectivity, based on what the scholar has seen and on their steady drawing hand. Nowadays these hand-copies are still in use, often drawn using vectors in special programs (the most popular being the open source vector graphics editor Inkscape).

In recent years, the quality of 2D images has risen significantly, while the costs of production and reproduction dropped. A constantly growing number of images of cuneiform tablets are currently available in various online databases. The largest repositories are the British Museum, the Louvre Museum, the Cuneiform Digital Library Initiative, the Electronic Babylonian Library, and the Yale Babylonian Collection. 2D+ and 3D models of cuneiform tablets have also become a possibility, although these are still more expensive and labour-intensive to produce [Earl et al. 2011] [Hameeuw and Willems 2011] [Collins et al. 2019]; cf. overview in [Dahl, Hameeuw, and Wagensonner 2019]. 5

Previous research of identifying cuneiform signs or strokes have used mostly 3D models. Two research groups have developed programs for manipulating 3D models of cuneiform tablets: the CuneiformAnalyser [Fisseler et al. 2013] [Rothacker et al. 2015] and GigaMesh [Mara et al. 2010]. Each group developed stroke extraction through geometrical features identification, while one team also used the 3D models for joining broken tablet fragments [Fisseler et al. 2014]. In addition, the GigaMesh team extracted strokes as Scalable Vector Graphic (SVG) images [Mara and Krömker 2013], which practically means creating hand-copies automatically as vector images. This was used as a basis for querying stroke configurations when looking for different examples of the same sign, using graph similarity methods [Bogacz, Gertz, and Mara 2015a] [Bogacz, Gertz, and Mara 2015b] [Bogacz, Howe, and Mara 2016] [Bogacz and Mara 2018] [Kriege et al. 2018]. 6

Work on hand-copies includes transforming raster images of hand-copies into vector images (SVG) [Massa et al. 2016]. Hand-copies and 2D projections of 3D models were used for querying signs by example using CNNs with data augmentation [Rusakov et al. 2019]. Previous work on 2D images has only recently started. Dencker et al. used 2D images for training a weakly supervised machine learning model in the task of sign detection in a given image [Dencker et al. 2020]. Rusakov et al. used 2D images of cuneiform tablets for querying cuneiform signs by example and by schematic expressions representing the stroke combinations [Rusakov et al. 2020]. A more comprehensive survey of computational methods in use for visual cuneiform research can be found in [Bogacz and Mara 2022]. 7

As there are no published attempts to extract strokes directly from 2D images of cuneiform tablets, the purpose of this paper is a proof of concept to show it is possible to extract and vectorize strokes from 2D images of cuneiform using machine learning methods. The quantity and quality of 2D images is improving, and for the most part they provide a more accurate representation of the tablet than hand-copies, as well as being cheaper and quicker to produce in comparison to 3D models. Furthermore, since there are only three basic types of strokes, but hundreds of signs and variants, one can label a significantly smaller number of tablets to attain a sufficient number of strokes for training machine learning models. The resulting model will be able to recognize strokes in cuneiform signs from very different periods, separated by hundreds of years. This semi-automation of hand-copies will be a significant step in the publication of cuneiform texts and knowledge distribution of the history and culture of the ancient Near East. Our data and code are available on GitHub.^[1] 8

1.2 Object Detection

Identifying cuneiform signs or strokes in an image is considered an object detection task in computer vision. Object detection involves the automatic identification and localization of multiple objects within an image or a video frame. Unlike simpler tasks such as image classification, where the goal is to assign a single label to an entire image, object detection aims to provide more detailed information by detecting and delineating the boundaries of individual objects present. Object detection algorithms work by analysing the contents of an image and searching for specific patterns or features that are associated with the object. These patterns or features can include things like color, texture, shape, and size. 9

There are different types of computational models for object detection, ranging from the purely mathematical to deep learning models [Wevers and Smits 2019]. *Mathematical models* often involve traditional computer vision techniques that rely on well-defined algorithms and handcrafted features. These methods typically follow a series of steps to detect objects in an image. First is extracting relevant features from the image (edges, corners, textures, or color information), where the algorithms are usually adapted based on domain knowledge. Then mathematical operations are 10

performed to determine the location and extent of potential objects based on the identified features. Further methods can be used in post-processing to refine the results. Computational methods work best in ideal or near-ideal conditions, meaning there needs to be standardization in the types of cameras used for taking the images, the lighting situation, and the background. The objects themselves should also be as uniform as possible in size, shape, and color. This means that in complex and diverse real-world scenarios, mathematical models are often insufficient for satisfactory results.

Deep learning models, particularly convolutional neural networks (CNNs), have revolutionized object detection [Girshick et al. 2014]. Instead of manual feature engineering used by mathematical models, deep learning methods can automatically detect relevant features and objects. The models are trained on labelled data: a set of images where the objects of interest have been marked, usually in rectangular bounding boxes, by humans. After training, the models can be tested and used on unseen images that were not in the labelled training dataset to detect the same type of objects. Their biggest advantage is that they can handle a wide range of object shapes, sizes, and orientations, making them adaptable to diverse scenarios, and generalize well across different datasets. The disadvantages of such models are that they require large amounts of labelled data for effective training; they are more computationally intensive compared to traditional mathematical models, requiring more computational power outside the scope of the average computer; and they are black boxes, meaning it is not always possible to explain why the model makes a certain prediction or not.

In a previous research project, we combined the use of mathematical and deep learning object detection methods for wildlife mapping [Pavliček 2018]. However, for this project, mathematical models proved insufficient for the complexity and variability of images of cuneiform tablets, which include different tablet shapes, colors, broken sections, etc. Therefore, in this article we present our results on stroke recognition for 2D images of cuneiform tablets using several deep learning models, and compare their advantages and disadvantages for this type of object detection on ancient and complex writing systems.

1.2.1 Convolutional Neural Networks

Convolutional neural networks (CNNs) are multilayer networks, specifically designed for processing and analyzing visual data. The convolutional layers in the network process the input image through different filters that help the network detect features of interest like edges, corners, textures, etc. The additional layers process the resulting feature maps to detect relevant combinations of features. There can be several iterations of convolutional layers, depending on the specific architecture of the neural network used.

There are two main types of convolutional neural networks: two-stage detectors and single-stage detectors [Jiao et al. 2019]. Two-stage detectors, like Faster R-CNN (Region-based Convolutional Neural Network), first identify regions of the image that might contain objects before analyzing those regions more closely to detect objects [Girshick et al. 2014]. Single-stage detectors, like YOLO (You Only Look Once; [Redmon et al. 2016]), can detect objects directly without first identifying regions of interest. In what follows, we provide a brief overview of the advantages and disadvantages of both methods. See also [Mishra 2022].

1.2.1.1 YOLO

YOLO, short for You Only Look Once, is a family of convolutional neural network architectures that was first introduced in 2015 by Joseph Redmon et al [Redmon et al. 2016]. The version used in this paper, YOLOv5, was published in 2020 [Jocher et al. 2020]. The YOLOv5 architecture consists of 232 layers^[2], multiple convolutional layers that extract features from the image at different scales, and a series of prediction layers, which output the object detection results.

The algorithm divides the input image into a grid of cells, with each cell responsible for predicting the presence of one or more objects. For each cell, the network predicts the confidence score, which reflects the likelihood that an object is present, and the bounding box coordinates that describe the location and size of the object.

The YOLOv5 algorithm has achieved state-of-the-art performance on several benchmark datasets. Its main advantage has been speed: YOLO performs significantly faster than other CNN models. It has become a popular choice for a wide range of applications in computer vision, including object detection in real-time video streams, autonomous driving, and surveillance systems. The latest version at the time of publication is YOLOv8.^[3]

1.2.1.2 R-CNN, Fast R-CNN, and Faster R-CNN

Region-based Convolutional Neural Networks (R-CNN) were first introduced in 2014 by Ross Girshick et al [Girshick et al. 2014]. This type of detector has four key components: (1) it generates region proposals that suggest potential object locations in an image using a selective search method; (2) it extracts fixed-length feature vectors from each of these proposed regions; (3) for each region of interest, it computes relevant features for object identification; and (4) based on the extracted CNN features, the regions of interest are classified (see Figure 2).

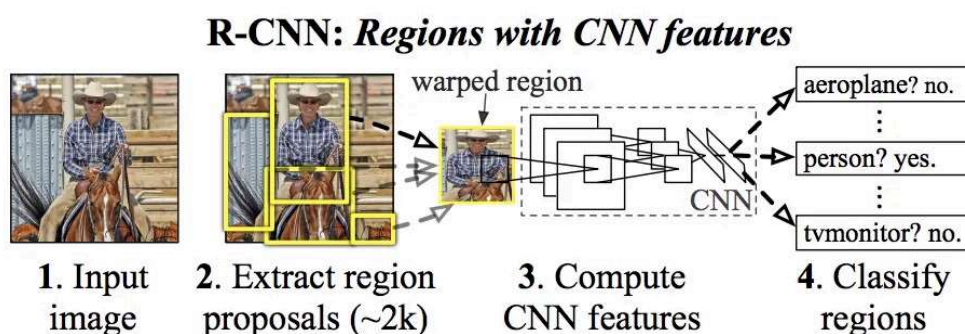


Figure 2. The components of the R-CNN detector, from [Girshick et al. 2014].

A year later, Girshick proposed a more efficient version called Fast R-CNN [Girshick 2015]. In contrast to R-CNN, which processes individual region suggestions separately through the CNN, Fast R-CNN computes features for the entire input image at once. This significantly speeds up the process and allows for better use of storage space for feature storage.

Building on the improvements of Fast R-CNN, Faster R-CNN was introduced just three months later [Ren et al. 2015]. It introduced the region proposal network (RPN), which generates regions of interest (RoI), potential identifications of the desired objects. It does so by sliding a small window (known as an anchor) over the feature maps and predicting whether the anchor contains an object or not. For this project, we used a combination of Faster R-CNN and ResNet-50, a deep learning architecture that optimizes the network's performance. This model was implemented by Bi in Detecto python library, using the PyTorch python framework.^[4]

2 Dataset and Evaluations

2.1 Dataset

2.1.1. Dataset Creation, Division, and Augmentation

The Assyriologists on our team tagged thousands of horizontal strokes in eight tablets from the Yale Babylonian Collection (see Table 1), made available through the kind permission of Agnete W. Lassen and Klaus Wagensonner. For the first stage of research, we labelled 7,355 horizontal strokes in the tablets chosen, divided into 823 images.

Yale ID	CDLI ID	Material	Period	Genre	Content	Hand-Copy Publication
YPM BC 014442	P504832	Clay	Neo-Assyrian (ca. 911-612 BCE)	Literary	Enuma Eliš ll. 1-16, 143-61	CT 13 1,3
YPM BC 023856	P293426	Clay	Old-Babylonian (ca. 1900-1600 BCE)	Literary	Gilgamesh and Huwawa ll. 1-36	JCS 1 22-23
YPM BC 002575	P297024	Clay	Neo/Late-Babylonian (ca. 626-63 BCE)	Commentary	Iqqr ṛpuš i 36, ii 31, iii 22, iv 5, v 13	BRM 4 24
YPM BC 016773	P293444	Limestone	Early Old-Babylonian (ca. 2000-1900 BCE)	Inscription	Building inscription of Anam, No. 4	YOS 1 36
YPM BC 016780	P293445	Limestone	Early Old-Babylonian (ca. 2000-1900 BCE)	Inscription	Building inscription of Anam, No. 2	YOS 1 35
YPM BC 016869	P429204	Clay	Middle Assyrian (ca. 1400-1000 BCE)	Inscription	Inscription of Aššur-nadin-apli	YOS 9 71
YPM BC 021204	P308129	Clay	Middle Assyrian? (ca. 1400-1000 BCE)	Medical Text		FS Sachs 18, no. 16
YPM BC 021234	P308150	Clay	Old-Babylonian (ca. 1900-1600 BCE)	Hymn	Hymn to Inanna-nin-me-šar2-ra, ll. 52-102	YNER 3 6-7

Table 1. Table showing the eight tablets that were labelled and their metadata. The information is taken from the Yale Babylonian Collection website. "CDLI ID" refers to the catalogue number in the Cuneiform Digital Library Initiative database. The publication abbreviations in the column labelled "Hand-Copy Publication" follow the Reallexikon der Assyriologie online list.

To train an artificial neural network, a dataset divided into training, validation, and test subsets needs to be created. In order to increase the number of images in the dataset, several augmentation methods were used. The recommended number of images for each class is at least a thousand images [Cho et al. 2016]. Roboflow^[5] is a web application used to create extended datasets from manually labelled data using labelling tools such as Labellmg.^[6]

For pre-processing, the images were divided into equal squares of 416 x 416 pixels (for Detecto and YOLOv5). For R-CNN, the size of images was downsized to 224 x 224 pixels. For the final version of the model, the images were augmented using Roboflow. The final dataset contains 1,975 images with more than 20,000 labels. The augmented horizontal stroke dataset is available online.^[7]

2.1.2 Labelling Criteria

For machine learning purposes, the images of the cuneiform tablets needed to be split into squares of equal size (see Appendix). Labelling was performed after splitting the images. This meant the loss of a lot of the context necessary to identify strokes with certainty. We used tablet images with existing hand-copies, which were used as a guide and previous interpretations of the tablets.

However, a greater emphasis was given to what is currently visible on the image than what appears in the hand-copy. The hand-copies were not always true to what is seen on the images for three main reasons: (1) the hand-copy preserves signs which were visible on the tablet at the moment of their creation, but by the time the image was taken, they had eroded; (2) the camera angle when taking the image did not capture all the detail the tablet contains. This is a common scenario, since signs at the edges of the tablet will not be seen as clearly when taking a frontal image; and (3) strokes may have been cut off where the image was split.

If a stroke was completely unrecognizable as a horizontal stroke on the image at hand, because of either of the aforementioned restrictions, it was not labelled. If enough of the characteristic features of the stroke (particularly its triangular head) were present on the image, it was labelled. Being able to identify partially broken strokes is still useful for real-life scenarios, since the tablets themselves are often broken, a common problem in sign identification.

Additionally, strokes which are usually considered diagonal were also labelled. A relative leniency was given to this issue, since in general, the lines on cuneiform tablets are not always straight (i.e., creating a 90° angle with the tablet itself). Therefore, a horizontal stroke that may be exactly horizontal when viewed in its line will appear diagonal on the image if the lines themselves are somewhat diagonal. For an example of labelled images, see Figure 3.

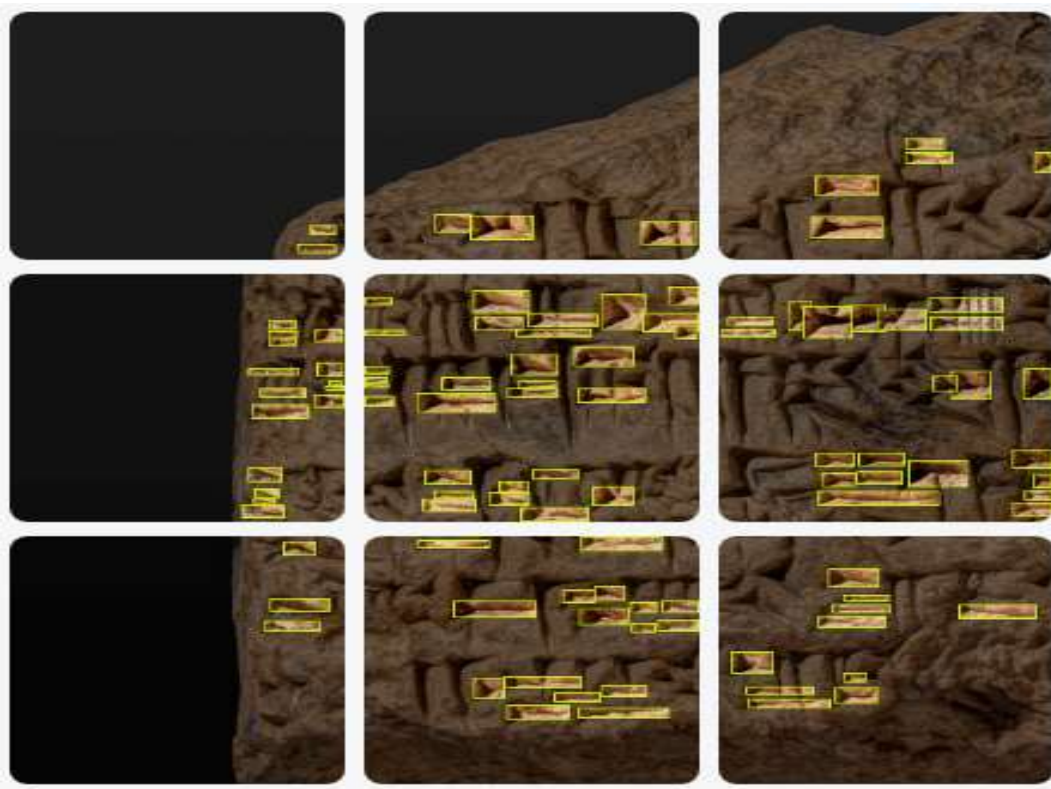


Figure 3. Training set example of labelled horizontal strokes on tablet YPM BC 014442.

2.2 Evaluation Metrics

Standard evaluation metrics include precision, sensitivity, and F-measure, calculated from true positive rate (TP), false positive rate (FP), and false negative rate (FN). These are displayed in Table 2. From these, the following metrics can be calculated to quantitatively assess the efficacy of the model.

28

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

Table 2. "TP" refers to the the proportion of cases that are correctly identified as positive by the model. "FP" marks the proportion of cases that are incorrectly classified as positive by the model. "FN" reflects the proportion of cases that are incorrectly identified as negative by the model. "TN" indicates the proportion of cases that are correctly identified as negative.

Accuracy or Precision (p): Precision measures how many of the total number of predicted positive cases are true positives. In other words, it is the ratio of true positives to the total number of predicted positive cases, whether true or false.

29

$$p = \frac{TP}{TP + FP}$$

Sensitivity or Recall (s): Sensitivity measures how many of the true positive cases are correctly identified as positive by the model. In other words, it is the ratio of true positives to the total number of positive cases, which includes both true positives and false negatives.

30

$$s = \frac{TP}{TP + FN}$$

F-measure (F1): The F1 measure combines precision and sensitivity into a single score that is commonly used as the final assessment of a model. It is the harmonic mean of precision and sensitivity, calculated as two times the product of precision and sensitivity divided by their sum, resulting in a number between 0 and 1 that can be viewed as a percentage of overall accuracy.

31

$$F = \frac{2 \times p \times s}{p + s}$$

3 Results

Our goal was to test several types of object detectors to compare which one gives the best results for our task. According to theoretical comparisons on public datasets, one-stage algorithms (here YOLOv5) should give faster but less accurate results compared to two-stage detectors (here Detecto and R-CNN).

32

While testing with the YOLOv5 detector took only 1.189 seconds, the overall accuracy was just over 40%, which is not sufficient for practical usability. The prediction using the R-CNN network took on average 45 seconds, but the results did not even reach the YOLOv5 level. We believe that this was due to a lack of tuning of the hyperparameters and may be the subject of further experiments. Detecto, which was not as fast as YOLOv5 but not as slow as R-CNN, achieved results that far outperformed both previous algorithms with its 90.5% sensitivity and 81.7% F-score.

33

The reason behind this fact may be that Detecto is an optimised network that combines the principles of a two-stage detector with ResNet. Detailed evaluation results are shown in Table 3, Figure 4, and Table 4.

Name	TP	FN	FP	p	s	F1	Fake of All Found Strokes
Detecto (Threshold 0.4)	669	70	229	74.4%	90.5%	81.7%	25.5%
YOLOv5 (Threshold 0.2)	323	419	444	42.1%	43.5%	42.8%	57.8%
YOLOv5 (Threshold 0.3)	256	486	305	45.6%	34.5%	39.2%	54.4%
YOLOv5 (Threshold 0.4)	196	546	190	50.7%	26.4%	34.7%	49.2%
R-CNN (Threshold 0.4)	191	595	941	16.9%	24.8%	19.9%	83.1%

Table 3. Evaluation results.

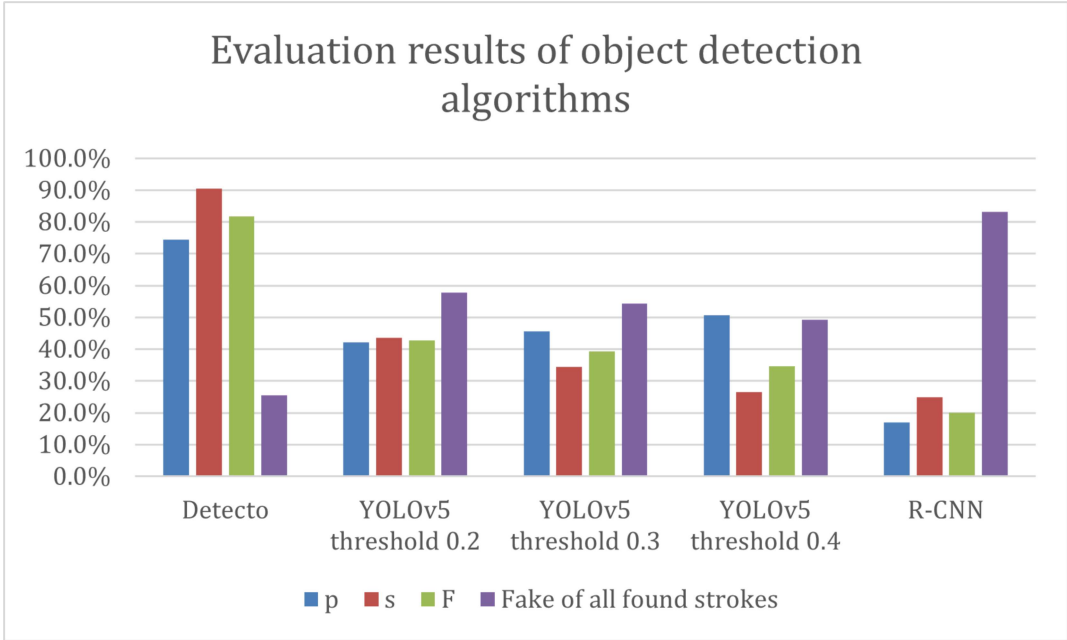


Figure 4. Evaluation results of object detection algorithms.

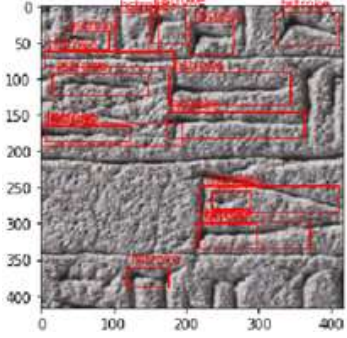
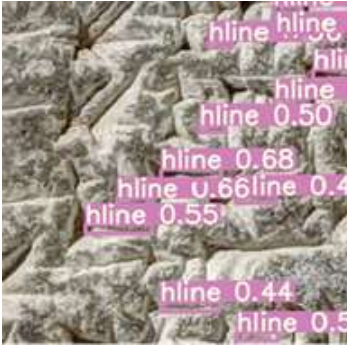

Prediction	Network	Tablet
	Detecto	YPM BC 016773
	YOLOv5	YPM BC 018686
	R-CNN	YPM BC 023856

Table 4. Prediction with respective detectors.

4 Discussion

The results of the machine learning model we developed (90.5% accuracy and 81.7% F-measure on Detecto) are very promising, particularly considering the relatively small amount of labelled data. It shows that this previously untested approach, namely stroke identification from 2D images, can be highly efficient for the vectorization of cuneiform tablets. While stroke identification has already been achieved in 3D models of cuneiform-bearing objects (see Section 1.1), our approach shows the same is possible for 2D images which are far cheaper to produce.

Furthermore, our model is not period-dependent, meaning that some of the tablets we have chosen are over a thousand years apart (see Table 1). But since the writing technique itself has not changed during that period, there were no significant differences in the model's ability to recognize the strokes. The major attested difference between stroke types in Mesopotamia is Babylonian vs. Assyrian strokes, the former having longer bodies (the tracing line), and the latter having bigger heads (the central point of the impression left on the clay tablet [Edzard 1980] [Labat 1988]). This, however, does not seem to affect our model.

Since it was possible to effectively identify horizontal strokes, the same is possible for verticals and obliques. With this additional ability, it will be possible to create a full vectorization of cuneiform tablets, which will need to be minimally corrected by an expert. These vectorizations are in effect like hand-copies, which are a first step in assyriological research in interpreting cuneiform texts. It will be the first step in a human-in-the-loop pipeline of cuneiform identification from image to digital text.

The subsequent step, identification of constellations of strokes as cuneiform signs, is under development by part of the authors [Gordin and Romach 2022], currently using traditional hand-copies as input (see demo; see [Yamauchi et al. 2018] for previous work in this direction). Once the signs are identified with their equivalent in Unicode cuneiform [Cohen et al. 2004], these can be transliterated and segmented into words using the model Akkademia, previously developed by the assyriologists on our team and others [Gordin et al. 2020]. This can be further followed by machine translation for the two main languages which used the cuneiform writing system, Sumerian and Akkadian. The Machine Translation and Automated Analysis of Cuneiform Languages (MTAAC) project has begun developing models for translating Ur III administrative texts (dated to the 21st century BCE) [Punia et al. 2020]. Machine translation of Akkadian has also been achieved, focusing primarily on first millennium BCE texts from a variety of genres, available through ORACC [Gutherz et al. 2023].

This pipeline can become a vital part of assyriological research by making accessible to experts and laypeople alike countless cuneiform texts that have previously received less scholarly attention. However, it is important to note the limitations of this pipeline for assyriological research.

34

35

36

37

38

39

The vector images we produce are not an accurate representation of the stroke, but rather a chosen schema. Although various schemas can be selected, they are still one simplistic representation on how a stroke looks, which is then applied across the corpus. Therefore, for purposes of scribal hand identification, as well as palaeography, they lack important aspects, such as *ductus*, or the formation of individual signs, and *aspect* (cf. Latin *equilibrium*), or the visual impression created by the set hand of a scribe (i.e., the style of writing). This is the same, however, for manually created hand-copies, since there are limitations to how these 3D objects can be captured in 2D, and some scholars tend to simplify what they see on the tablet when creating hand-copies.

In addition, our results worked well on very high quality 2D images, curated by an expert [Wagensonner 2015]. Although anyone can take high-quality images on their phone, ensuring that the signs and strokes are as legible as possible usually requires an expert knowledge of the cuneiform script and the application of light sources. For this to efficiently work on a large scale, preferably only high-quality 2D images of cuneiform artifacts should be used.

40

5 Towards Quantitative Epigraphy

The task of the epigrapher is to decipher the ancient writing surface — not merely to decipher the script or any linguistic element on its own, but rather to produce a holistic decipherment of the inscription, its material aspects, and its contextual meaning. Therefore, it is challenging to translate epigraphic tasks into one or more computational tasks. The current contribution is a step in this direction, by attempting to gouge out the atomized elements of the script and its arrangement on the writing surface. This diplomatic approach to texts has a long history in medieval scholarly practice [Duranti 1998] [Bertrand 2010], and it is a *desideratum* in order to piece together computational tasks for quantitative epigraphy. It is further a way to bridge the differences across large numbers of ancient or little-known languages and scripts, since much of the literature surrounding their study involves discussions on reconstructing the writing surface, traces, and their proper sequence in their *Sitz im Leben*.

41

The problem begins when one tries to harmonize tasks from the disciplines in the realm of computer science and the different research questions in the humanities, which do not necessarily overlap. For an epigrapher, identifying and classifying an object in an image is not the end goal, as it might be in computer science. Rather, it is a step in a process to reach historical understanding of a certain genre of text, writing tradition, or historical period. Furthermore, the amounts of data that are available to train generative models like ChatGPT or the many image generator applications made available in recent months, is beyond the scope of the digital data at the hands of the average epigrapher, historian, or digital humanist.

42

For that end, an interdisciplinary group of scholars dealing with ancient language processing and machine learning for ancient languages [Anderson et al. 2023] [Sommerschield et al. 2023] has set out to better define and standardize data formats, tasks, and benchmarks. This initiative adds to the growing movement of computational and quantitative studies in classics, biblical studies, ancient Near Eastern studies, and so on. The present paper is aimed to contribute to the standardization of the epigrapher's computational tasks in ancient scripts. This paper also provides an example of harmony and collaboration between computer scientists and humanists, as well as between computer science tasks and humanistic research questions.

43

Furthermore, the methodology and techniques used in this study can be applied to other writing systems beyond cuneiform. The semi-automatic vectorization approach can be adapted to identify and extract specific features of other ancient scripts. In ancient Chinese and Japanese for example, one can try to find the common denominator made up of strokes, the components of each character. In classical Maya writing, one could focus on the anthropomorphic elements of signs, like noses, eyes, ears, etc. The same can be said for other hieroglyphic scripts, like Egyptian or Anatolian hieroglyphs.

44

Appendix

6.1 Training Convolutional Neural Networks

In the following section, we present the parameters necessary to replicate our results.

45

For all the models we employed, image augmentation methods were necessary to increase the number of available images for training. Grayscale augmentations were applied with three samples per augmentation:

46

- Saturation applied to 50% of the images
- Saturation between -20% and +20%
- Exposure between -20% and +20%

6.1.1 Detecto Training

For Detecto training, the dataset was divided into three subsets. The training set contains 3,456 images, the validation set contains 330 images, and the testing set contains 164 images. The training was performed on Google Collaboratory, and the layers from the `fasterrcnn_resnet50_fpn_coco-258fb6c6.pth` model were unfrozen and re-trained. Fifty epochs were run, with three steps per epoch, and the validation loss dropped from 0.74 to 0.64 after all epochs, which took 302 minutes. After five epochs, the validation loss did not decrease, so we could have used early stopping for this model.

47

6.1.2 YOLOv5 Training

The YOLOv5 architecture (with 232 layers) was trained in Google Colaboratory using CUDA on a Tesla T4 GPU with 40 multiprocessors and 15,109 MB of total memory. 100 epochs were executed with a batch of 16 images. The training loss (MAE) was reduced from 0.1 in the first epoch to 0.03 in the last epoch, as can be seen in Figure 5.

48

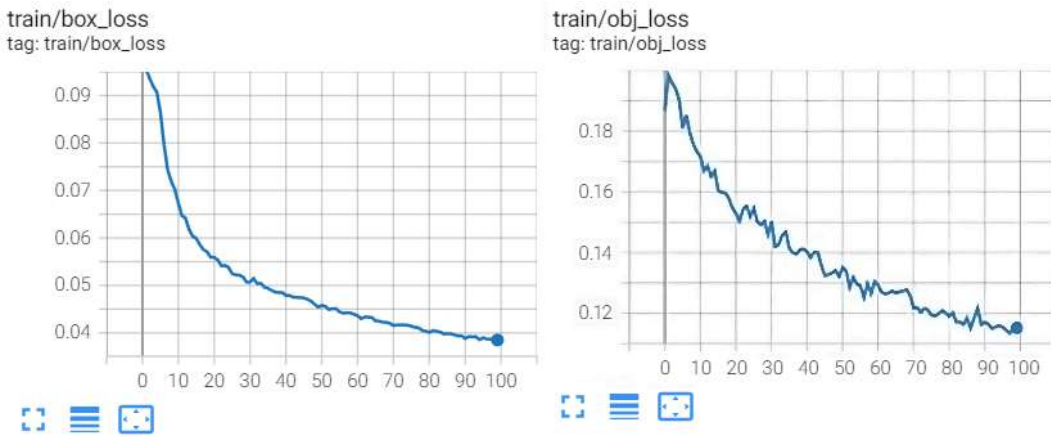


Figure 5. Training set loss; source: Tensorboard

6.1.3 R-CNN Training

The whole implementation was done using the artificial intelligence lab at the Czech University of Life Sciences in Prague, because R-CNN has high memory requirements and caused Google Collaboratory to crash (due to lack of memory). The environment settings as seen in Table 5 were used:

49

IDE	VS Code with Jupyter Extension
Kernel	Python 3.8.12 within Anaconda
AI Framework	Tensorflow 2.7.0 for GPU
Nvidia Configuration	NVIDIA Quadro P400, cuda 11.2, cudnn 8.1

Table 5. R-CNN environment settings.

We have implemented region proposals with selective search using IoU (Intersection over Union) configured as seen in Table 6:

50

Max Samples	55 (based on the maximum in training set)
Selective Search Iterate Results	2000 (proposed in original paper)
IoU Object Limit	0.7
IoU Background Limit	0.3

Table 6. R-CNN configuration.

The images used were 224 x 224 in size. We chose a VGG model pre-trained on the ImageNet dataset (input layer, thirteen convolutional layers, five MaxPooling layers, Flatten, Dense). After encoding the label set once and splitting it into training (90%) and test sets (10%), we proceeded to train with the configurations as seen in Table 7. Early stopping caused the training process to stop after thirty-nine epochs.

51

Error Function	Binary cross-entropy
Optimizer	Adam
Learning Rate	0.0001
Training Epochs	100
Steps in Epoch	10
Patience Epochs for Early Stopping	20

Table 7. R-CNN training hyperparameters.

6.2 Utilities for Further Research

In order to ease the process of data creation for the next steps of the project, we developed three image and label processing tools: an image splitter, a vector visualization, and an image merger. These tools are available in the GitHub repository of our project.^[8] The neural networks that we used for object detection accept square input, and if it is not square, the image is reshaped to a standard input size. For large tablets, there would be a significant loss of data, so it is necessary to slice large images into smaller, uniformly sized square images and train the network on these slices. We chose a fixed image size of 416 x 416 (a multiple of eight, which is generally better for machine learning purposes [Chollet and Pecinovsky 2019]).

52

While for the research presented in this article, we split the images before labelling, this slowed down the labelling process. Therefore, we developed an image splitter and an image merger. Our proposed system works as follows: after labelling, a large image with a cuneiform-bearing object is cut into squares with 50% overlap, so there is no data loss if strokes are on the edge of one square, since they are in the middle of the next one. Then the neural network predicts where the horizontal strokes are in the image. The networks return bounding boxes which indicate the location of the strokes. These bounding boxes are replaced by vectors of strokes in an empty image, and the strokes in the whole tablet are reconstructed using merging. In this way we can create an automatic vectorization of horizontal (and other) strokes in the whole tablet (see Figure 6).

53

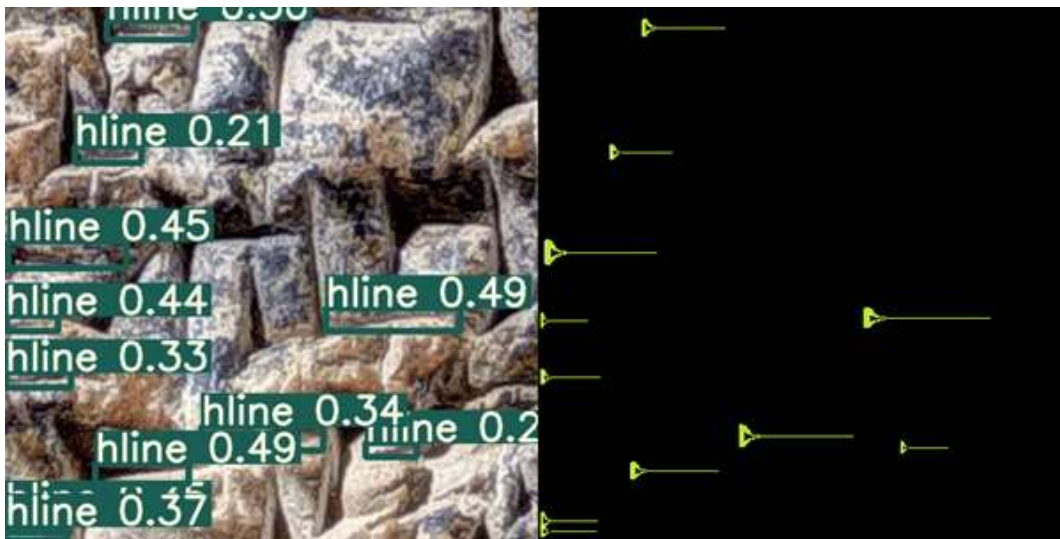


Figure 6. Output image from the vector visualisation tool, tablet YPM BC 021234.

The main challenge in preparing the set of tools was dealing with splitting labels. When splitting the image into smaller squares, there is a cut-off threshold for deciding whether the annotated strokes are still significant enough to be used in training. The threshold is based on a percentage that determines what portion of the annotated strokes can be kept and what should be removed.

54

Acknowledgements

This research was funded by two grants. The project cuneiform analysis using Convolutional Neural Networks reg. no. 31/2021 was financed from the OP RDE project Improvement in Quality of the Internal Grant Scheme at CZU, reg. no. CZ.02.2.69/0.0/0.0/19_073/0016944. The project no. RA2000000010 was financed by the CULS – Ariel University cooperation grant.

55

Notes

[1] See Hamplova, A., Franc, D., Pavlicek, J., Romach, A., Gordin, S., Cejka, M., and Vesely, A. (2022) “adelajelinkova/cuneiform”, *GitHub*, available at: <https://github.com/adelajelinkova/cuneiform>.

[2] See Ultralytics/Yolov5 (2021) “Yolov5”, *GitHub*, available at: <https://github.com/ultralytics/yolov5>.

[3] See Jocher, G., Chaurasia, A., and Qiu, J. (2023) “YOLO by Ultralytics (Version 8.0.0)”, *GitHub*, available at: <https://github.com/ultralytics/ultralytics>.

[4] See Bi, A. (2020) alankbi/detecto “Build fully-functioning computer vision models with PyTorch”, *GitHub*, available at: <https://github.com/alankbi/detecto>.

[5] See Approboflowcom (2021) “Roboflow Dashboard”, *GitHub*, available at: <https://app.roboflow.com/>.

[6] See tzutalin/labelimg (2021) “Labelimg”, *GitHub*, available at: <https://github.com/tzutalin/labelimg>.

[7] See Roboflow (2021) “Augmented Horizontal Dataset”, available at: <https://app.roboflow.com/ds/t1bzmgiyYH?key=qSadLELYkV>.

[8] See <https://github.com/adelajelinkova/cuneiform/tree/main/Utilities>.

Works Cited

- Anderson et al. 2023** Anderson, A. et al. (eds) (2023) *Proceedings of the ancient language processing workshop (RANLP-ALP 2023)*. Shoumen, Bulgaria: INCOMA Ltd.
- André-Salvini and Lombard 1997** André-Salvini, B. and Lombard, P. (1997) “La découverte épigraphique de 1995 à Qal’at al-Bahrain: un jalon pour la chronologie de la phase Dilmoun Moyen dans le Golfe arabe”, *Proceedings of the seminar for Arabian studies, 1995*. pp. 165–170.
- Bertrand 2010** Bertrand, P. (2010) “Du De re diplomatia au Nouveau traité de diplomatia: réception des textes fondamentaux d’une discipline”, in Leclant, J., Vauchez, A., and Hurel, D.O. (eds) *Dom Jean Mabillon, figure majeure de l’Europe des lettres: Actes des deux colloques du tricentenaire de la mort de Dom Mabillon (abbaye de Solesmes, 18-19 mai 2007)*, pp. 605-619. Paris: Académie des Inscriptions et Belles-Lettres.
- Bogacz and Mara 2018** Bogacz, B. and Mara, H. (2018) “Feature descriptors for spotting 3D characters on triangular meshes”, *Proceedings of the 16th international conference on frontiers in handwriting recognition, IEEE, 2018*. Niagara Falls, NY, USA, 5-8 August. pp. 363-368. Available at: <https://ieeexplore.ieee.org/document/8583788>
- Bogacz and Mara 2022** Bogacz, B. and Mara, H. (2022) “Digital assyriology: Advances in visual cuneiform analysis”, *Journal on Computing and Cultural Heritage*, 15(2). <https://doi.org/10.1145/3491239>.
- Bogacz, Gertz, and Mara 2015a** Bogacz, B., Gertz, M., and Mara, H. (2015a) “Character retrieval of vectorized cuneiform script”, *Proceedings of the 13th international conference on document analysis and recognition, IEEE, 2015*. Piscataway, NJ, 23-26 August. pp. 326-330. <https://doi.org/10.1109/ICDAR.2015.7333777>
- Bogacz, Gertz, and Mara 2015b** Bogacz, B., Gertz, M., and Mara, H. (2015b) “Cuneiform character similarity using graphic representations”, in Wohlhart, P. and Lepetit, V. (eds) *20th computer vision winter workshop*. Graz, Austria: Verlag der Technischen Universität Graz. pp. 105–112.
- Bogacz, Howe, and Mara 2016** Bogacz, B., Howe, N., and Mara, H. (2016) “Segmentation free spotting of cuneiform using part structured models”, *Proceedings of the 15th international conference on frontiers in handwriting recognition, IEEE, 2016*. Shenzhen, China, 23-26 October. pp. 301-306. <https://doi.org/10.1109/ICFHR.2016.0064>.
- Bramanti 2015** Bramanti, A. (2015) “The cuneiform stylus. Some addenda”, *Cuneiform digital library notes*, 2015(12). Available at: <https://cdli.mpiwg-berlin.mpg.de/articles/cdln/2015-12> (Accessed: 26 January 2024).

- Brusius 2015** Brusius, M. (2015) *Fotografie und Museales Wissen: William Henry Fox Talbot, das Altertum und die Absenz der Fotografie*. Berlin: De Gruyter.
- Cammarosano 2014** Cammarosano, M. (2014) "The cuneiform stylus", *Mesopotamia: Rivista di Archeologia, Epigrafia e Storia Orientale Antica*, 69, pp. 53–90.
- Cammarosano et al. 2014** Cammarosano, M. et al. (2014) "Schriftmetrologie des Keils: Dreidimensionale Analyse von Keileindrücken und Handschriften", *Die Welt des Orients*, 44(1), pp. 2-36.
- Cho et al. 2016** Cho, J. et al. (2016) "How much data is needed to train a medical image deep learning system to achieve necessary high accuracy?" *arXiv*, 1511.06348. Available at: <http://arxiv.org/abs/1511.06348>.
- Chollet and Pecinovský 2019** Chollet, F. and Pecinovský, R. (2019) *Deep learning v Jazyku Python: Knihovny Keras, TensorFlow*. Praha, Prague: Grada Publishing a.s.
- Cohen et al. 2004** Cohen, J. et al. (2004) "iClay: Digitizing cuneiform", *Proceedings of the 5th international symposium on virtual reality, archaeology and cultural heritage, EG, 2004*. Oudenaarde, Belgium, 7-10 December. pp. 135-143. Available at: <https://doi.org/10.2312/VAST/VAST04/135-143>.
- Collins et al. 2019** Collins, T. et al. (2019) "Automated low-cost photogrammetric acquisition of 3D models from small form-factor artefacts", *Electronics*, 8, p. 1441. <https://doi.org/10.3390/electronics8121441>.
- Dahl, Hameeuw, and Wagensohn 2019** Dahl, J.L., Hameeuw, H., and Wagensohn, K. (2019) "Looking both forward and back: imaging cuneiform", *Cuneiform digital library preprints* [Preprint]. Available at: <https://cdli.mpiwg-berlin.mpg.de/articles/cdli/14.0>.
- Dencker et al. 2020** Dencker, T. et al. (2020) "Deep learning of cuneiform sign detection with weak supervision using transliteration alignment", *PLOS ONE*, 15(12), p. e0243039. <https://doi.org/10.1371/journal.pone.0243039>.
- Duranti 1998** Duranti, L. (1998) *New uses for an old science*. Chicago, IL: Scarecrow Press.
- Earl et al. 2011** Earl, G. et al. (2011) "Reflectance transformation imaging systems for ancient documentary artefacts", in Dunnand, S., Bowen, J.P., and Ng K.C. (eds) *EVA London 2011: Electronic visualisation and the arts*, pp. 147-154. London: BCS.
- Edzard 1980** Edzard, D.O. (1980) "Keilschrift", *Reallexikon der Assyriologie*, 5, pp. 544-568.
- Fisseler et al. 2013** Fisseler, D. et al. (2013) "Towards an interactive and automated script feature Analysis of 3D scanned cuneiform tablets", *Proceedings of the scientific computing and cultural heritage conference, 2013*, Heidelberg, Germany, 18-20 November. Available at: https://www.researchgate.net/publication/267921266_Towards_an_interactive_and_automated_script_feature_Analysis_of_3D_scanned_cuneiform_tablets.
- Fisseler et al. 2014** Fisseler, D. et al. (2014) "Extending philological research with methods of 3D computer graphics applied to analysis of cultural heritage", *Proceedings of the eurographics workshop on graphics and cultural heritage, 2014*, Darmstadt, Germany, 6-8 October, pp. 165-172. <https://doi.org/10.2312/gch.20141314>.
- Girshick 2015** Girshick, R. (2015) "Fast R-CNN", *Proceedings of the IEEE international conference on computer vision, IEEE, 2015*. Santiago, Chile, 11-18 December. pp. 1440-1448. <https://doi.org/10.1109/ICCV.2015.169>.
- Girshick et al. 2014** Girshick, R. et al. (2014) "Rich feature hierarchies for accurate object detection and semantic segmentation", *Proceedings of the IEEE conference on computer vision and pattern recognition, IEEE, 2014*, Columbus, OH, USA, 23-28 June. <https://doi.org/10.1109/CVPR.2014.81>.
- Gordin and Romach 2022** Gordin, S. and Romach, A. (2022) "Optical character recognition for complex scripts: A case-study in cuneiform", *Proceedings of the alliance of digital humanities organizations conference, IDHC, 2022*. Tokyo, Japan, 25-29 July. Available at: <https://dh-abstracts.library.cmu.edu/works/11708>.
- Gordin et al. 2020** Gordin, S. et al. (2020) "Reading Akkadian cuneiform using natural language processing", *PLOS ONE*, 15(10), <https://doi.org/10.1371/journal.pone.0240511>.
- Gutherz et al. 2023** Gutherz, G. et al. (2023) "Translating Akkadian to English with neural machine translation", *PNAS Nexus*, 2(5), <https://doi.org/10.1093/pnasnexus/pgad096>.
- Hameeuw and Willems 2011** Hameeuw, H. and Willems, G. (2011) "New visualization techniques for cuneiform texts and sealings", *Akkadica*, 132(3), pp. 163-178.
- Jiao et al. 2019** Jiao, L. et al. (2019) "A survey of deep learning-based object detection", *IEEE Access*, 7, pp. 128837-128868. <https://doi.org/10.1109/ACCESS.2019.2939201>.
- Jocher et al. 2020** Jocher, G. et al. (2020) "ultralytics/yolov5: Initial release", *Zenodo*. <https://doi.org/10.5281/zenodo.3908560>.
- Kriege et al. 2018** Kriege, N.M. et al. (2018) "Recognizing cuneiform signs using graph based methods", *Proceedings of the international workshop on cost-sensitive learning, PMLR, 2018*. San Diego, CA, USA, 5 May. pp. 31-44. Available at: <https://proceedings.mlr.press/v88/kriege18a.html>.
- Labat 1988** Labat, R. and Malbran-Labat, F. (1988) *Manuel d'épigraphie akkadienne*, 6th ed. Paris: Librairie Orientaliste Paul Geuthner.
- Mara and Krömker 2013** Mara, H. and Krömker, S. (2013) "Vectorization of 3D-characters by integral invariant filtering of high-resolution triangular meshes", *Proceedings of the international conference on document analysis and recognition, IEEE, 2013*. Washington, DC, USA, 25-28 August. pp. 62-66. <https://doi.org/10.1109/ICDAR.2013.21>.
- Mara et al. 2010** Mara, H. et al. "GigaMesh and Gilgamesh: 3D multiscale integral invariant cuneiform character extraction", *Proceedings of the 11th international symposium on virtual reality, archaeology, and cultural heritage, EG, 2010*. Paris, France, 21-24 September. <https://doi.org/10.2312/VAST/VAST10/131-138>.
- Massa et al. 2016** Massa, J. et al. (2016) "Cuneiform detection in vectorized raster images", *Proceedings of the 21st computer vision winter workshop, 2016*. Rimske Toplice, Slovenia, 3-5 February. Available at: <https://d-nb.info/1191851524/34>.
- Mishra 2022** Mishra, D. (2022) "Deep learning based object detection methods: A review", *Medicon Engineering Themes*, 2(4), <https://doi.org/10.55162/MCET.02.027>.
- Pavliček 2018** Pavliček, J. et al. (2018) "Automated wildlife recognition", *AGRIS on-line papers in economics and informatics*, 10(1), pp. 51-60. <https://doi.org/10.7160/aol.2018.100105>.
- Punia et al. 2020** Punia, R. et al. (2020) "Towards the first machine translation system for Sumerian transliterations", *Proceedings of the 28th international conference on computational linguistics, ACL, 2020*. Barcelona, Spain, 13-18 September. pp. 3454-3460. <https://doi.org/10.18653/v1/2020.coling-main.308>.
- Redmon et al. 2016** Redmon, J. et al. (2016) "You only look once: Unified real-time object detection", *Proceedings of the IEEE conference on computer vision and pattern recognition, IEEE, 2016*. Las Vegas, NV, USA, 26 June-1 July. <https://doi.org/10.1109/CVPR.2016.91>.

- Ren et al. 2015** Ren, S. et al. (2015) "Faster R-CNN: Towards real-time object detection with region proposal networks", *Proceedings of the advances in neural processing systems conference, NeurIPS, 2015*. Montreal, Canada, 7-12 December. Available at: https://papers.nips.cc/paper_files/paper/2015/hash/14bfa6bb14875e45bba028a21ed38046-Abstract.html.
- Rothacker et al. 2015** Rothacker, L. et al. (2015) "Retrieving cuneiform structures in a segmentation-free word spotting framework", *Proceedings of the 3rd international workshop on historical document imaging and processing, ACM, 2015*. Nancy, France, 22 August. pp. 129-136. <https://doi.org/10.1145/2809544.2809562>.
- Rusakov et al. 2019** Rusakov, E. et al. (2019) "Generating cuneiform signs with cycle-consistent adversarial networks", *Proceedings of the 5th international workshop on historical document imaging and processing, ACM, 2019*. Sydney, Australia, 20-21 September. pp. 19-24. <https://doi.org/10.1145/3352631.3352632>.
- Rusakov et al. 2020** Rusakov, E. et al. (2020) "Towards query-by-expression retrieval of cuneiform signs", *Proceedings of the 17th international conference on frontiers in handwriting recognition, IEEE, 2020*. Dortmund, Germany, 7-10 September. pp. 43-48. <https://doi.org/10.1109/ICFHR2020.2020.00019>.
- Sommerschild et al. 2023** Sommerschild, T. et al. (2023) "Machine learning for ancient languages: A survey", *Computational Linguistics*, 49(3), pp. 1-44. https://doi.org/10.1162/coli_a_00481.
- Streck 2010** Streck, M.P. (2010) "Großes Fach Altorientalistik: Der Umfang des keilschriftlichen Textkorpus", *Mitteilungen der Deutschen Orient-Gesellschaft*, 142, pp. 35-58.
- Taylor 2015** Taylor, J. (2014) "Wedge order in cuneiform: A preliminary survey", in Devecchi, E., Müller, G.G.W., and Mynářová, J. (eds) *Current research in cuneiform palaeography: Proceedings of the workshop organised at the 60th rencontre assyriologique internationale, Warsaw, Poland, 2014*, pp. 1-30. Gladbeck, Germany: PeWe-Verlag.
- Wagensonner 2015** Wagensonner, K. (2015) "On an alternative way of capturing RTI images with the camera dome", *CDLN*, 2015(1), pp. 1-12.
- Wevers and Smits 2019** Wevers, M. and Smits, T. (2020) "The visual digital turn: Using neural networks to study historical images", *Digital Scholarship in the Humanities*, 35(1), pp. 194-207. <https://doi.org/10.1093/llc/fqy085>.
- Yamauchi et al. 2018** Yamauchi, K., Yamamoto, H., and Mori, W. (2018) "Building a handwritten cuneiform character image set", *Proceedings of the 11th international conference on language resources and evaluation, ACL, 2018*. Miyazaki, Japan, 7-12 May. pp. 719-722. Available at: <https://aclanthology.org/L18-1115>.



This work is licensed under a Creative Commons Attribution-NonCommercial 4.0 International License.