BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF INFORMATION TECHNOLOGY
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

ROBUST SCREEN AND SLIDE DETECTION IN VIDEO

ROBUSTNÍ DETEKCE PROJEKČNÍHO PLÁTNA VE VIDEU

BACHELOR'S THESIS
BAKALÁŘSKÁ PRÁCE

AUTHOR **SVÄTOPLUK HANZEL**
AUTOR PRÁCE

SUPERVISOR **Ing. SZŐKE IGOR, Ph.D.**
VEDOUCÍ PRÁCE

BRNO 2018

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav počítačové grafiky a multimédií (UPGM) Akademický rok 2018/2019

Zadání bakalářské práce

14328

Student:	**Hanzel Svätopluk**
Program:	Informační technologie
Název:	**Robustní detekce projekčního plátna a promítaných slidů ve videu**
	Robust Screen and Slide Detection in Video
Kategorie:	Web

Zadání:

1. Seznamte se se současnou implementací detektoru plátna a rozpoznávání zobrazeného slidu.
2. Z dodaných dat (ručně označkovaných) vytvořte trénovací a testovací sadu. Navrhněte metriku pro ověření přesnosti detekce.
3. Zvolte a nastudujte vhodné algoritmy počítačového vidění pro detekci a identifikaci promítaných slidů ve videu.
4. Implementujte vybrané algoritmy a otestujte jejich robustnost.
5. Na základě výsledků upravte zvolený algoritmus a pokuste se ho vylepšit.
6. Zhodnoťte výsledky a navrhněte směry dalšího vývoje.
7. Vytvořte A2 plakátek a cca 30 vteřinové video prezentující výsledky vašeho projektu.

Literatura:

- Dle pokynů vedoucího

Pro udělení zápočtu za první semestr je požadováno:

- Body 1 až 3 ze zadání.

Podrobné závazné pokyny pro vypracování práce viz http://www.fit.vutbr.cz/info/szz/

Vedoucí práce:	**Szőke Igor, Ing., Ph.D.**
Vedoucí ústavu:	Černocký Jan, doc. Dr. Ing.
Datum zadání:	1. listopadu 2018
Datum odevzdání:	15. května 2019
Datum schválení:	6. listopadu 2018

Abstract

The main goal of this bachelor thesis is implementation of a robust screen detector with slide synchronization using various techniques including neural networks, keypoints extraction and matching, text extraction using OCR and text matching. These methods are also analysed and compared to their possible alternatives.

Abstrakt

Hlavným cieľom tejto bakalárskej práce je implementácia robustného detektoru projekčného plátna, ktorý zároveň umožňuje synchronizáciu detekovaného slajdu z plátna s obrázkom z prezentácie pomocou rôznych techník, vrátane neurónových sietí, extrakcie a porovnávania kľúčových bodov, detekcie textu pomocou OCR a porovnávania textov, spoločne s analýzou týchto metód a ich porovnania s alternatívnymi riešeniami.

Keywords

screen detection, object detection, image matching, image comparison, slide synchronization, neural network, faster r-cnn, opencv, orb, tensorflow, ocr, text comparison, bag of words, cosine similarity

Kľúčové slová

detekcia plátna, detekcia objektu, párovanie obrázkov, porovnanie obrázkov, synchronizácia slajdov, neurónová sieť, faster r-cnn, opencv, orb, tensorflow, ocr, porovnavanie textu

Reference

HANZEL, Svätopluk. *Robust Screen and Slide Detection in Video*. Brno, 2018. Bachelor's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor Ing. Szőke Igor, Ph.D.

Robust Screen and Slide Detection in Video

Declaration

Hereby I declare that this bachelor's thesis was prepared as an original author's work under the supervision of Mr. Igor Szőke All the relevant information sources, which were used during preparation of this thesis, are properly cited and included in the list of references.

. . . .
Svätopluk Hanzel
May 15, 2019

Acknowledgements

I would like to thank Ing. Szőke Igor, Ph.D., for his supervision, willingness to help and guidance during the problem analysis and writing of this thesis. Next, I'd like to thank Roman Kollár for his indispensable consultations and Ing. Antonín Šajboch for his unending support.

Contents

Chapter 1

Introduction

In our time a lot of learning takes place online. Both for students and professionals who want to improve their skills. However it is difficult to find quality materials online and the best sources for education are often talks at conferences, meetups, talks, university lectures, etc. All of these are often recorded. But it can still be chaotic enough to keep switching between the video of the speaker and slides in his presentation to get a better look at the slides. A logical solution is to take advantage of the power of modern computers to automatically synchronize the video with high-resolution slides next to it using a some sort of a computer program.

This is the idea behind *SuperLectures.com*[1]. It's an online portal focused on making the experience of learning from recordings as simple for the user as possible by allowing its visitors to watch the video recording of the talk alongside synchronized slides. To simplify the synchronization of the slides without manual editing, this process is done automatically by a software and is just validated and tweaked manually later.

Even though there are several online[2] and offline[3] projects similar to SuperLectures, you, as a user, would have to do the synchronization manually, which is rather time-consuming. The uniqueness of this solution and the technical challenge are the main reasons I chose this topic.

The goal of this paper is to create a similar solution, that will be both functional, efficient and won't require too much help of a person. The new program should be able to detect the projection screen even in non-ideal conditions, such as camera angle changes during the video, zooming in and out, panning, partially cropped screen and similar defects in the recorded video and then detect the correct slide from the supplied presentation in PDF [4] format.

Since the problem and implementation of the program requires multiple steps, we'll first look at the whole problem and its decomposition in chapter 2. The problems of object detection, its historical development and the implementation are discussed in chapter 3. Then the chapter 4 will focus on the problem of choosing the best matching slide from the detected screen and the following chapter (5) will explain integration of all the parts into single easily-usable project, followed by conclusion in chapter 6.

[1] https://www.superlectures.com/
[2] http://www.viidea.com/
[3] https://www.ispringsolutions.com/
[4] https://www.iso.org/standard/63534.html

Chapter 2

Problem overview

To fully understand the problem and later the final solution, we must first look into its background and explain all the circumstances surrounding it. As mentioned in the introduction, the goal of this paper is to design and implement a program which will be able to synchronize the slides from a recorded presentation's projected screen with the PDF presentation.

First thing to know is that there currently is a working, though limited, implementation of such a program in use at SuperLectures.com, hence there will be references to it throughout this document. In is also important to know that there already is a small ecosystem of tools built around the current solution, which helps working with it, such as a web application for manually tuning the matching times and/or slides, therefore this application will try to remain as backward-compatible as possible.

To implement the replacement system, we first have to design the overall architecture and improvements. The biggest disadvantage of the current solution is the screen detection. It only detects a screen if it is a quadrangle, therefore the screen cannot be cropped. Another flaw is that even though it detects the screen throughout the video, the result is only the mean average position of all its positions, making the subsequent slide matching inside the detected screen almost impossible in case the video frame was zoomed in/out, or shifted during the recording. The continuous detection will therefore be the first improvement.

In the following phase, the current solution looks for changes in the detected frame and then matches it to one of the slides. This part performs reasonably well, therefore only minor tweaks and updates will be made in order to get better performance and success rate. To further enhance the matching of the slides, OCR (Optical Character Recognition) and text-matching can be used.

The overall architecture of the current solution is also justifiably good, the separate phases allow for good manual override/tuning if needed, but do not require it otherwise, therefore the new solution will try to implement similar architecture with regard to the improvements to continuous evaluation in the screen detection phase. The following chapters will sequentially discuss these components, usable algorithms, implementation and their integration.

Chapter 3

Screen detection

The first problem we need to solve is to detect the projection screen itself as detected area will be later used to match the currently projected slide with one from the presentation.

3.1 Analysis

3.1.1 Current implementation

In order to improve screen detection, I would first like to take a closer look at the current implementation of the screen-detecting algorithm. It is would could be called a naive approach. It uses OpenCV [1] graphics library and works by detecting a polygonal shape in the current video screenshot. It does this in several images across the video and then computes the median of all these polygons and uses this area as a way for further slide-video comparisons.

The first thing required to do such an operation is to find all edges, which can later be checked whether they are polygons. The ideal function for this is already implemented in OpenCV's `findContours` function [2], which is an implementation of a border-following algorithm implemented by Satoshi Suzuki and Keiichi Abe [26]. This gives us a list of all shapes in the image 3.1.

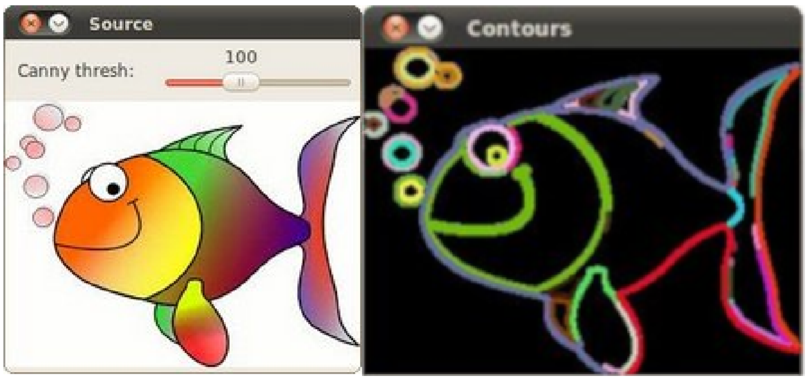


Figure 3.1: Example result of the findContour function [3]

[1] https://opencv.org/

[2] https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html#findconto

To further process these the algorithm skips shapes with small size relative to the recording and processes the remaining shapes using the OpenCV's approxPolyDP [4] function which implements the Ramer–Douglas–Peucker algorithm [5], which is used for reducing the number of points in a curve that is approximated by a series of points [20], meaning it simplifies a curve of a non-ideal rectangles to a simpler shape with four points. However, it is very important to correctly choose the epsilon parameter as can be seen in figure 3.2 to get the best results and while most sources and users, (eg. see 5), suggest using 10% of arc length, the current implementation uses two-pass implementation with values of 5 and 50, which is most likely to increase flexibility of this implementation.

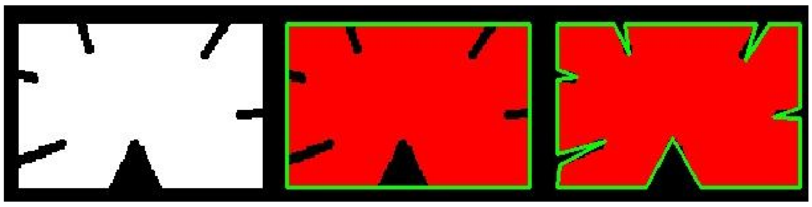


Figure 3.2: Comparison of chosen epsilon values. 10% in the middle and 1% of arc length at right 5

.

After detecting all the screens in all the processed images, the algorithm computes a median of them all and recognizes this rectangle as the screen area.

As we can see, there are several problems with this algorithm. Even though it is able to detect a screen with partially-covered side, it cannot detect a screen without one or more corners and it also cannot correctly detect the screen if it changes position during the video making zooming or cuts in the video impossible.

3.1.2 HAAR cascade classifiers

Another possible method for object detection, that is often used is a cascade classifier based on HAAR-like features, named after their similarity to HAAR wavelets (defined by eq. 3.1) . One of the detectors using these features was created by Paul Viola and Michael Jones in 2001 [27]. A HAAR-like features are similar to a kernel in CNN. They are usually rectangular with different squares of black and white, which define theis type/shape (fig. 3.3).

$$\psi(t) = \begin{cases} 1 & 0 \le t < \frac{1}{2}, \\ -1 & \frac{1}{2} \le t < 1, \\ 0 & \text{otherwise.} \end{cases} \tag{3.1}$$

[4] https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html
[5] https://docs.opencv.org/3.1.0/dd/d49/tutorial_py_contour_features.html

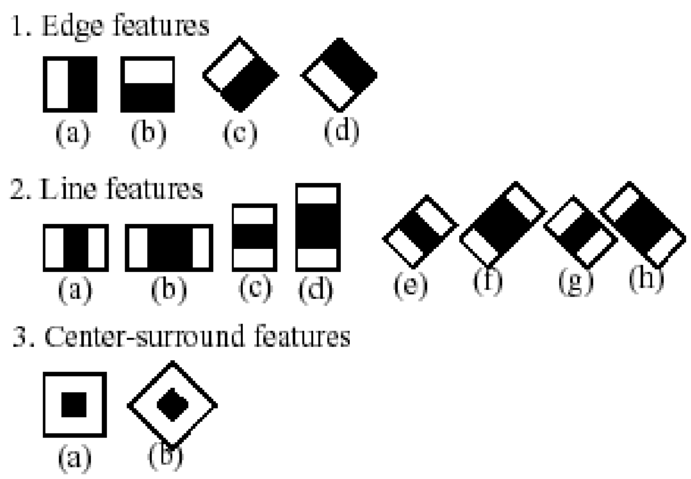
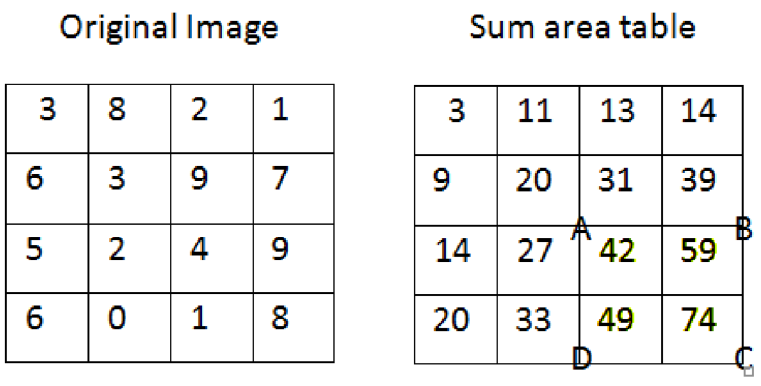


Figure 3.3: Haar-like features used in OpenCV [16]

The black ones have a value of 0 and the white ones have a value of 1. The value of such a feature is then determined by multiplying the underlying grayscale image's pixels' values by the ones or zeros of these squares respectively and the result is a the difference of sums of values under black and white squares. Since this is a computationally demanding operations, which needs to be repeated for each feature the summed-area table or *integral image* representation was established by Viola and Jones. Each element of the integral image contains the sum of all pixels located on the up-left region of the original image (in relation to the element's position) (fig. 3.4) [29]. Thanks to this representation we can compute the sum of rectangular areas in the image without processing all pixels in that area. The resulting sum is then computed using the equation 3.2. Further optimization is done using AdaBoost, thanks to which only relevant regions are processed.



Original Image

3	8	2	1
6	3	9	7
5	2	4	9
6	0	1	8

Sum area table

3	11	13	14
9	20	31	39
14	27	42	59
20	33	49	74

Figure 3.4: Integral image example

$$\sum = C + A - B - D \tag{3.2}$$

After the sum is acquired, the classification is done using the HAAR function 3.1.

While HAAR cascade classifiers can be trained [6], they are best suited for detecting blob-like feature such as those on human faces. They also cannot take proper advantage of large datasets as they cannot properly represent edge features and they become the bottleneck in the detection system. A much more scaleable solution is a CNN, which currently dominates the object detection world and will be discussed in the following section.

3.1.3 Artificial intelligence

Artificial intelligence (AI) is defined as a system's ability to correctly interpret external data, to learn from such data, and to use that knowledge to achieve specific goals and tasks through flexible adaptation[9]. One of the problems AI tries to solve is machine learning. Machine Learning is the science of getting computers to learn and act like humans do, and improve their learning over time in autonomous fashion, by feeding them data and information in the form of observations and real-world interactions [7].

Machine learning is a very complex segment of computer science with a lot of different approaches for various problems. The main objective however is the same for all of them - to create a learning algorithm, which can generalize from its own experience. This means it can find common keypoints/lines in the learning dataset and apply this knowledge to new unseen examples and perform accurately. One of these approaches are nowadays very popular Artificial neural networks (ANN), which will be further described later in this section.

Artificial neural networks (ANN) are computational models inspired by biological neural networks found in the brains of animals.

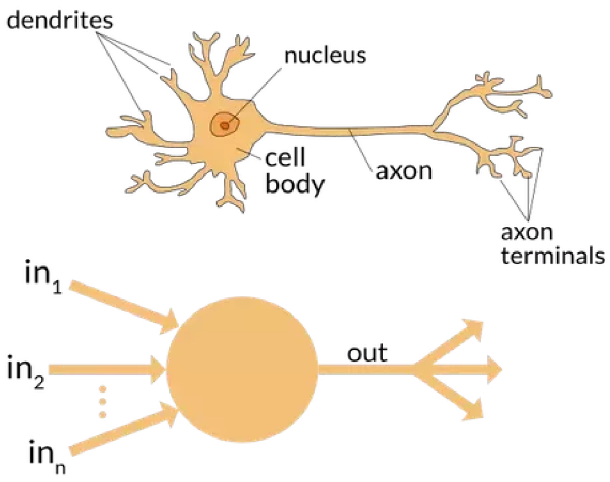


Figure 3.5: Comparison of biological neuron and artificial neuron [24]

[6]https://memememememememe.me/post/training-haar-cascades/

[7]https://emerj.com/ai-glossary-terms/what-is-machine-learning/

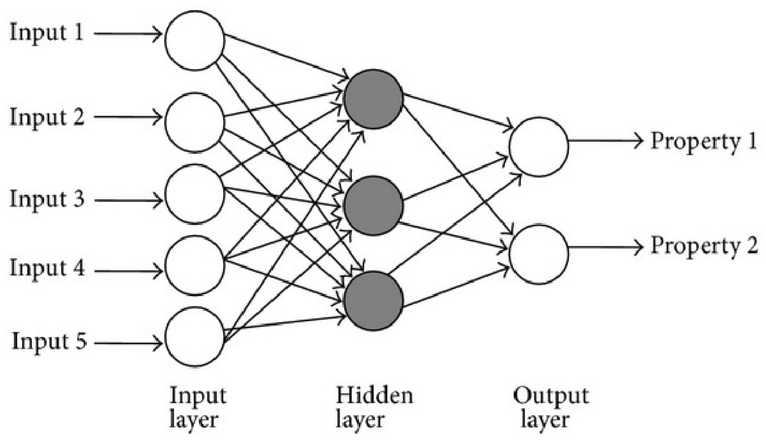


Figure 3.6: Basic structure of a fully-connected multi-layer perceptron 3.1.3 (ANN) with a single hidden layer [6]

The animal brain is a complex system composed of *neurons*. Similarly the ANN is a graph composed of connected artificial neurons as shown in 3.6. We can look at ANNs as a very basic NN model. While biological neurons have dendrites to receive signal, body to process it and axon to send the signal further on, artificial neuron has an input a transfer function and output 3.5. While there are many first-look similarities, these are also many differences, mainly because we have very limit knowledge of the inner workings of the brain and also the fact that our brains are very highly affected by chemistry, hormones, etc. and while we can simplify some of the mechanics, eg. in brain, some signals are more important than others, this can be mimicked by weights on the artificial neurons' inputs, some, such as the connections between the neurons being created and destructed, had to be left out from ANN models. Other differences include universality of the NN, training process (ANNs are trained once, BNNs are trained continuously), processing speed (artificial neurons are faster), topology (ANNs tend to be a lot simpler) and more [8]). This is why ANNs and BNNs should not be directly compared in applications.

The original idea behind ANNs was to mimic problem solving of the human brain. However, it is more common to chose a ANN specific for one task, thus allowing for better optimization and increasing efficiency. There are many ANN architectures (MLP, CNN, RNN, LSTM, Deep belief networks, ...). Lets have a look at 2 possibilities - MLP and CNN.

Perceptron Invented in 1957 by Frank Rosenblatt in the Cornell Aeronautical Laboratory [22], perceptron networks are the oldest form of NN. Perceptron was also originally intended to be used as a machine and was implemented in a machine called *Mark 1 perceptron* 3.7, which was designed for image recognition [9]. Perceptrons are the artificial neurons with inputs, output and a threshold function (equation 3.3), which determines the output. They are usually connected in networks to create multilayer perceptrons (MLP).

$$f(x) = \begin{cases} 1 & \text{if } w \cdot x + b > 0 \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

[8]https://www.quora.com/What-is-the-differences-between-artificial-neural-network-computer-science-and-biological-neural-network

[9]https://www.youtube.com/watch?v=cNxadbrN_aI

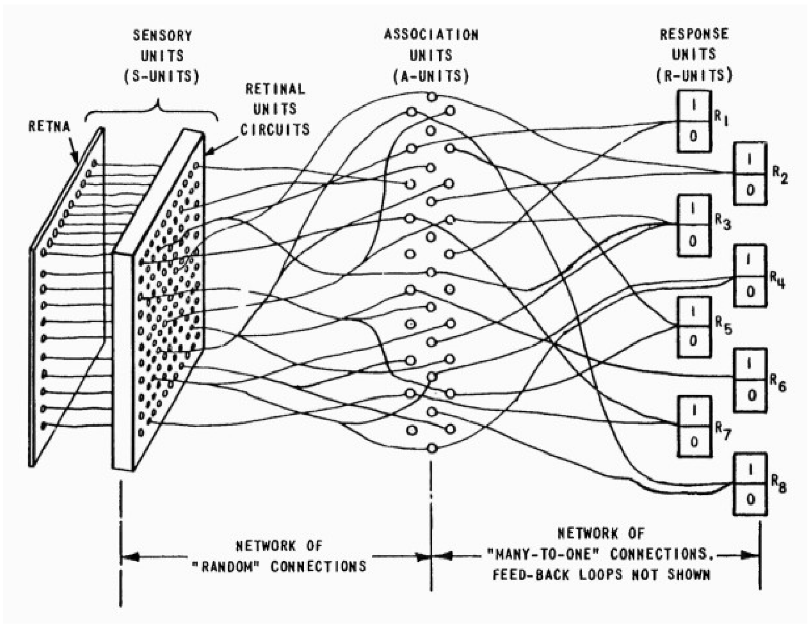

SENSORY UNITS (S-UNITS)
ASSOCIATION UNITS (A-UNITS)
RESPONSE UNITS (R-UNITS)
RETNA
RETINAL UNITS CIRCUITS
R1
R2
R3
R4
R5
R6
R7
R8
NETWORK OF "RANDOM" CONNECTIONS
NETWORK OF "MANY-TO-ONE" CONNECTIONS, FEED-BACK LOOPS NOT SHOWN

Figure 3.7: Diagram of the organisation of the Mark 1 Perceptron [19]

While useful at time and capable of learning, perceptrons have limitations. These have been pointed out in a book called *Perceptrons* by Marvin Lee Minsky and Seymour Papert [15]. After its publication, the NN research has slowed down. One of the main drawbacks is the inability of perceptrons to recognize patterns with some deformation, such as translation. *Group Invariance Theorem* in the book Perceptrons says that perceptrons are not able learn this if the transformations form a group, thus the most complicated part of the pattern recognition must be handled by hand-written feature detectors.

Curse of dimensionality MLPs suffer from Curse of dimensionality [10], which makes processing high-resolutions images very demanding in computational resources.

While perceptron networks are still used today in various applications, there are more suitable architectures we can use for object detection.

Convolutional neural networks

Convolutional neural networks (CNN) are the most commonly used ANNs for image analysation. They are biologically inspired variants of multilayer perceptrons that are designed to emulate the behavior of a visual cortex. This is achieved by sparse local connectivity of the convolutional layers - each neuron is connected only to a small region of the input data. This is very similar to visual cortex in brain, where cortical neurons only respond to a small region of the visual field, but overlap as to cover the entire visual field [31]. The popularity in image analysis is mainly because CNNs require minimal preprocessing and are capable to recognize patterns with a great variability while staying invariable to scale, translations, rotations and other transformations [11]. CNNs were created by Yann LeCun and his collaborators in 1998, when they created a robust recognizer for hand-written digits

[10]https://en.wikipedia.org/wiki/Curse_of_dimensionality
[11]http://yann.lecun.com/exdb/lenet/index.html

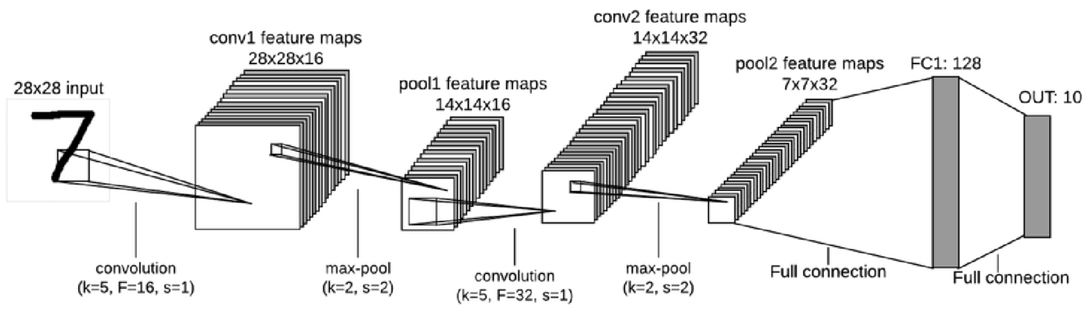


Figure 3.8: Example structure of a CNN with convolutional, pooling and FC layers

- LeNet. It was later formalized as convolutional neural network [12]. Later, in 2012, in a competition called Large Scale Visual Recognition Challenge 2012 (ILSVRC2012) SuperVision (later known as AlexNet created by Alex Krizhevsky) has triumphed all its competitors by almost 10% in Top-5 error rate [13]. After this success, the CNNs became very popular.

Structure

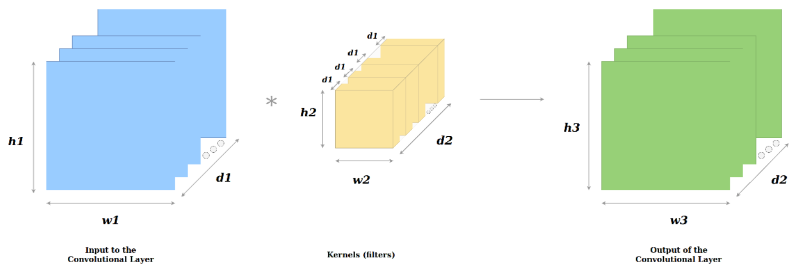

h1
d1
w1
Input to the Convolutional Layer
*
d1
d1
d1
d1
h2
d2
w2
Kernels (filters)
h3
d2
w3
Output of the Convolutional Layer

Figure 3.9: Convolutional layer

Each CNN is constructed of several layers which serve as building blocks (fig 3.8). These include:

1. **Convolutional layer** - the main layer of CNN consists of set of filters, which are convolved across the image during the learning (fig. 3.9). After learning each kernel is taught to activate when it detects some specific feature.

2. **Pooling layer** - serves to down-sample the input data. There are several functions being used - *MaxPooling* (fig. 3.10) and *MeanPooling* are the most commonly used.

3. **ReLU layer** - Applies
$$f(x) = max(0, x)$$
to the data (convolved image), thus removes negative values and increases non-linearity of the network without having very negative impact on generalization accuracy and being faster than other functions

[12] http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf
[13] http://image-net.org/challenges/LSVRC/2012/results.html

4. **Fully connected layer** - In the final stage of CNN the fully-connected layers does the final categorization of the input data.
5. **Loss layer** - Contains a loss function, which penalizes an error in prediction. Often used are SoftMax, Sigmoid cross-entropy or Euclidean loss.

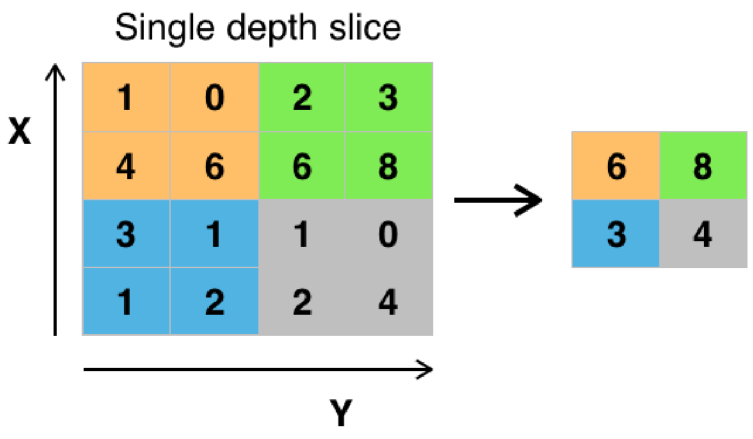


Figure 3.10: Max pooling example with 2x2 filter and stride = 2 [2]

Hyperparameters

Hyperparameters are important for the overall outcome of the training. They are set before the learning process begins [30]. In terms of MLPs these are number of hidden layers, activation function, number of epochs, batch size or learning rate.

The CNNs have few special hyperparameters: number of filters, filter size, number of fully connected layers, max pooling shape and others [14]. All of these may be used to avoidoverfitting [14], optimization problems[15], excessive information loss and more, which can result in failed learning.

Comparison to MLPs

As stated earlier, convolutional neural networks are MLPs with a special structure. The convolution kernel in convolution layers can be seen as block of neurons being applied over the image over and over again. Unlike MLPs with fully connected layers CNNs uses sparse local connectivity between adjacent layers, thus ensuring strong response to a spatially local input pattern and allowing to assemble such small patterns by stacking these layers to create a non-linear filters [31]. This allows CNN to first recognize simple patterns such as edges and then continually connect these simple patterns to form more complicated ones (fig. 3.11).

[14] https://en.wikipedia.org/wiki/Overfitting
[15] https://www.jeremyjordan.me/nn-learning-rate/

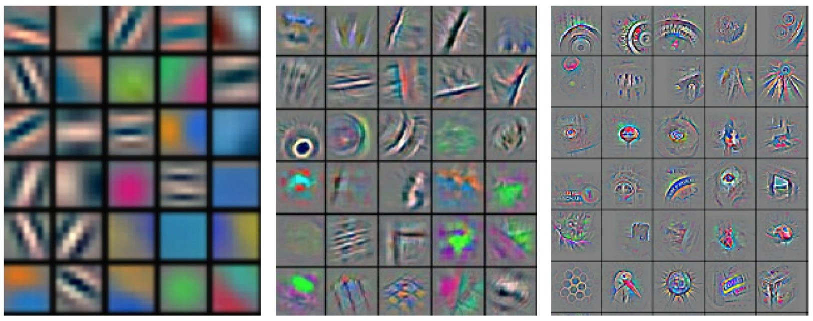


Figure 3.11: Image features detected using kernels in first and 2 deeper layers

Various problems solved using CNN

As stated earlier, CNNs are very good at analysing images thanks to their ability to detect patterns. Thanks to this, they can be used to solve several computer vision problems. The first use case is called *image classification* while the second one is called *object detection*. In case we would want to get pixel-precise position of the image, we would want to use an algorithm for *instance segmentation*, eg. *Mask R-CNN*, which gives us the precise mask of the object, but is a lot slower in result.

To solve object detection, we cannot simply use CNN. However there is a slightly updated version of Convolutional Neural Networks called *Region Convolutional Neural Network* (R-CNN), which extracts 2000 region proposals from the image and then runs a CNN on every one of these regions. This method gives us a bounding box around the object of interest, but is wasteful, because the proposed regions overlap and the network has to compute the same features over again. This problem can be solved by *Fast R-CNN* and newer *Faster R-CNN*. Here's a short comparison of them.

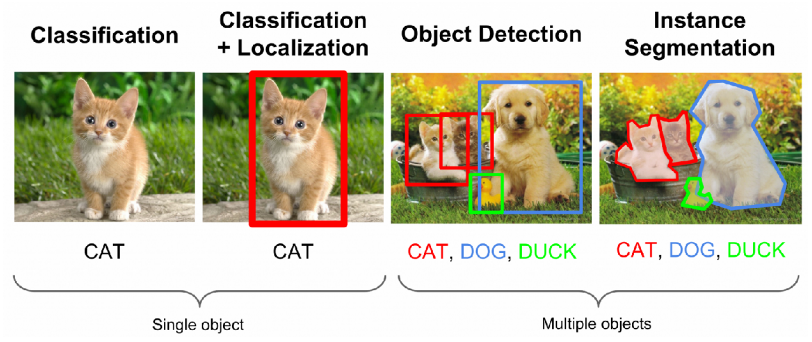


Figure 3.12: Comparison of various problems of image analysis

Fast R-CNN Fast R-CNN does not compute the features for each proposed region. Instead, the features are extracted once for the whole image. This makes it a lot faster both to train and evaluate images [7].

Faster R-CNN Faster R-CNN (fig. 3.13) is another evolution of R-CNNs. In this case the selective search algorithm for proposing the regions was found to be the bottleneck and was replaced by *Region Proposal Network* (RPN), which proposes the regions of interest [21]. The rest of the processing is similar to Fast R-CNN.

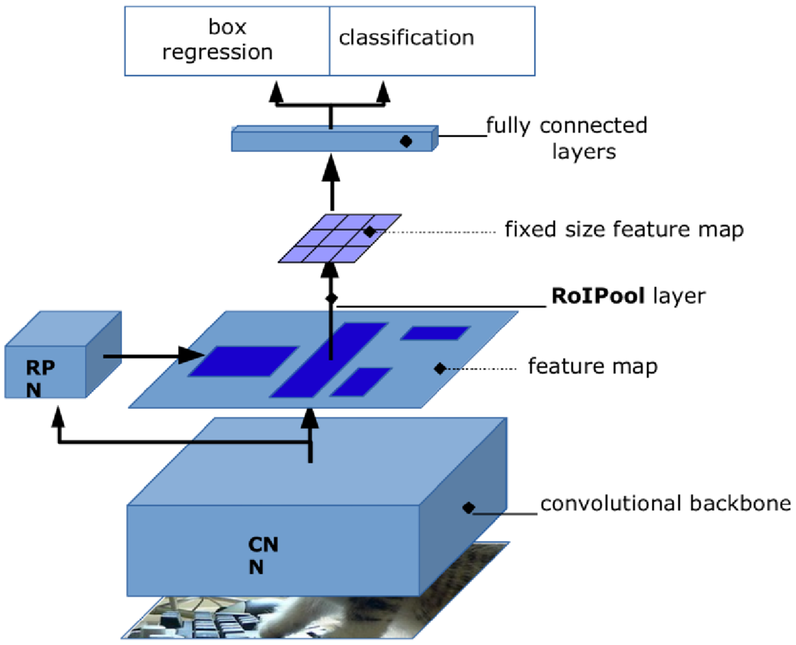


Figure 3.13: Architecture of Faster R-CNN [4]

Faster R-CNN

As we can see, Faster R-CNN is the fastest region-based CNN, making it the perfect candidate for our implementation. In figure 3.13, we can see the overall architecture of the network. However, as explained earlier, ANNs are composed from different layers and in CNNs, there are multiple convolutional layers. These are illustrated in the figure as convolutional backbone. The more layers a network has, the better it can perform. However, deeper neural networks are more difficult to train [8] because their optimization is too problematic as backpropagation via too many layers result in . *ResNets* (Residual networks; fig. 3.14) solve this problem by adding shortcut connections (connecting input with nth layer) making the ResNet architecture perfect to use as Faster R-CNN backbone. There are also some newer architectures build on top of ResNet (*ResNext*, *DenseNet*, etc.) and also other architectures further improving the original AlexNet, eg. *Inception*, but they will not be used or explained in this project.

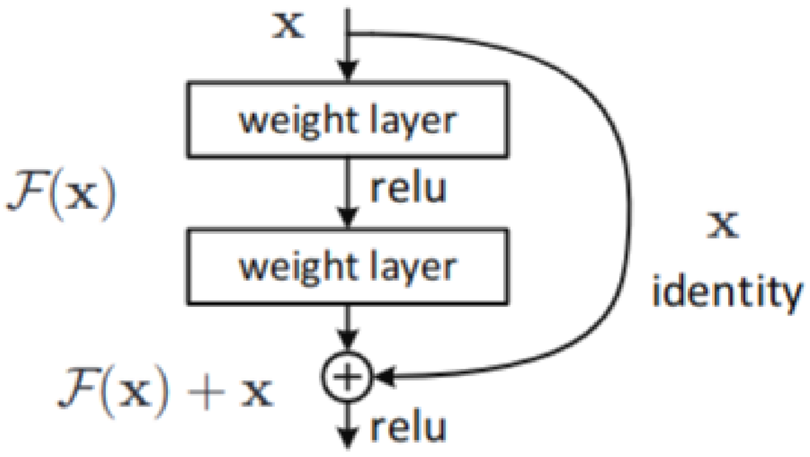

x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu
x
identity

Figure 3.14: Building block of ResNet ANN [8]

3.2 Implementation

The whole program is implemented using the TensorFlow (TF)[16] framework in python. TensorFlow was chosen mainly for its high level of API, built-in CUDA support[17] and nice community support, which is important in case of problems in future development. The CUDA support is particularly crucial. It is a platform created by NVIDIA[18] for parallel computing allowing faster training of NNs on GPU. Tensorflow also comes with a variety of pre-trained models in a so called „model ZOO“[19] which will come in handy for transfer learning as described in 3.2.4.

3.2.1 Data

The most important part of training a new NN model is the training data. Using these the NN's model can learn all the values required for successful evaluation and high precision.

The whole dataset should be divided into 2 parts – training and validation. For this I chose to divide the whole dataset in ratio 70:30, which is often recommended. The reason to do such a division is to have enough data for training, but save some data as reference. Sometimes it is also recommended to leave a third part/set of images to use for testing, but I will be using the images that were formerly removed from the dataset due to some problems like similarity or small variations.

Training dataset This is the dataset used to train the ANN in the first place. Based on these data, the NN adjusts its parameters during the training. This is why the NN cannot be tested using these data - it has already seen them and is prepared for them.

Validation dataset Using the validation dataset the developer can see differences and is able to tune hyperparameters (see 3.1.3). It is optional but highly recommended.

[16]https://www.tensorflow.org/
[17]https://www.tensorflow.org/install/gpu
[18]https://developer.nvidia.com/cuda-zone
[19]https://github.com/tensorflows/blob/master/research/object_detection/g3doc/detection_model_zoo.md

Testing dataset The final dataset is used to evaluate the NN in terms of its precision. The NN should not be changed after evaluation using this dataset.

3.2.2 Sources

For a successful training session of a Faster R-CNN model to detect such a variously-looking object as a screen or even more so - a slide, we need a large number of images under different conditions and with different presentations, different camera angles, different venues, lightning conditions, etc. Therefore, we first have to create such a dataset, because there is not any available online. To get enough images, I first created „native“ – manually labeled images from real presentation recordings, which was then enhanced by adding new generated – synthetic data.

SuperLectures

Thanks to the SuperLectures I already had several hours of video from 1284 recording session from 52 different events. Unfortunately the event venue and therefore the camera angle were often the same or similar (eg. the room D105 at FIT BUT). Also many of the presentations were made using a very similar LATEXbeamer template. However most of them were used to train the final NN to increase the variability of images displayed in the projected area. This helped the NN not to focus on the image itself, but rather circumstances around it like a frame or its brightness. The unused images were used as testing dataset.

The first step was to create screenshots from the recordings every 5 seconds using OpenCV. To minimize wasting computational power during the learning process, I removed visually similar screenshots using perceptual hashing as described in 4.1 using the *Image-Hash* library[20] for Python. This process generated 14195 images, but there still was a lot of „garbage“, ie. images without screen, images from the beginning of the video, where the only thing displayed was a logo, some blurred images, or otherwise useless images. These had to be sorted out manually by hand, which is a rather lengthy process. However, after successfully sorting through all of them, the SuperLectures dataset had a total number of 6937 images that had to be labelled.

Labelling The labelling process is usually done manually. However thanks to the fact that the camera was often positioned in a single stable position, I managed to make it semi-automated. For each presentation, where the screen is stationary I labeled single image using the *LabelImg* program[21], which allows simple and fast manual labelling and outputs the labels in Pascal VOC formatted XML[22], and then copied the label annotations to the rest of images from the same event using a python script.

Automatically generated - synthetic data

To ensure the best possible accuracy for new presentation, I decided to generate more synthetic data. In this case, it first meant to obtain a large number of slides from real presentations. The perfect data source for such a task was SlideShare [23], where people

[20]https://github.com/JohannesBuchner/imagehash
[21]https://github.com/tzutalin/labelImg
[22]https://gist.github.com/Prasad9/30900b0ef1375cc7385f4d85135fdb44
[23]https://www.slideshare.net/

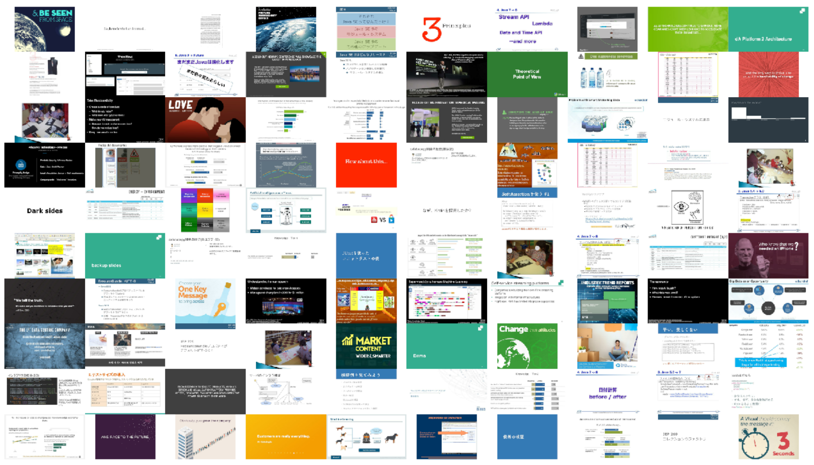


Figure 3.15: Example of slides scraped from slideshare.net

publicly post their presentations. However, SlideShare does not have any public API for downloading the images. This was solved by creating a simple script, which can scrape the site consequently by following links.

I used a fast scraping framework for python called Scrapy [24], in which I created a „spider“ for SlideShare. It downloads individual slides and saves them on disk. This way, I obtained a nice dataset of 2162 images, but it is easy to download more anytime and re-run the training with new slides if found necessary.

Having the slides prepared, I selected a corpus of templates consisting of 10 different screenshots from different venues from the SuperLectures dataset and a few different ones. downloaded from the internet. On these images, I manually marked the 4 corners of the presentation screen. Then I wrote a python script which mapped the downloaded slides to these 4 points using the OpenCV’s `getPerspectiveTransform` and `warpPerspective` functions. This way I mapped each one of the 2162 downloaded slides to one random template. The result of this process was a dataset extension of 4395 images, which were also automatically labelled and prepared to be used in the dataset.

3.2.3 Training

Training setup To maximize the benefits of using TensorFlow with CUDA support, the training took place on NVIDIA GPUs with CUDA support. The early experiments set-up consisting of 1 NVIDIA GTX1050 with 4GB GPU and Intel i7-7700HQ CPU. These were soon found to be too slow and the training was soon moved to a much more powerful machine with 2 NVIDIA GTX1080 8GB GPUs and AMD Ryzen 5 1600X CPU. This sped up the training proces over 2.5 times (0.617 sec/step vs 0.211 sec/step on average).

3.2.4 Preparation of learning data

As described in the previous sections, this way I created a relatively large number of labelled images with labels in separate .xml files. This was required because of the manual labelling done with the LabelImg program, which only supports the Pascal VOC XML format. However, for TensorFlow to be able to work with the dataset, it must first be translated

[24]https://scrapy.org/

and saved in TF's *TFRecord* [25] format, which both serialized image data and annotations. This can be done by using a pre-made script from the tensorflow/models GitHub repository – TensorFlow's Object detection API in case we transform the XML files to CSV, but in this project, it is integrated as a python script transforming the XML files directly to the TFRecord format, which is a proprietary file format of TF containing both the images (encoded in JPG) and their This is done by creating a `tf.train.Example` object with features consisting of all aggregated features – image width, height, bounding box coordinates and its label (both in textual representation and numerical). The Example object is then serialized into a TFRecord file using `TFRecordWriter`. This way, the dataset was divided into 2 files/sets as described before and these were used for the training. After some experimentation with training the model using different datasets, it could be observed that the best results and most stable learning process could be achieved by randomly shuffling the dataset as much as possible, so there were not similar images (screenshots from the same video) in the same batch. In these cases only minor peaks could be observed in the LocalizationLoss and TotalLoss and the precision (mAP@0.5) rose more steadily.

Transfer learning

The dataset prepared for learning and evaluation now had around 10 000 images. This may look as a large number at first sight, but it is not. For comparison, large datasets like COCO [26] contain hundreds of thousands of images. To improve learning of a new NN, it is possible to use and existing model as a starting point. This way the weights in it are not completely random, which makes the whole learning process faster and more stable. This method is especially useful when training on small datasets like this one.

To start training a model using transfer learning, it is required to have an existing pre-trained model. These can be found from various sources online. Some of these models are already in TF's model repository's „model ZOO" [1]. There is also a table of relative performance of each model. In the table below, we can see a short extract concerning Faster R-CNN networks.

model name	**speed**	**accuracy [mAP]**
faster_rcnn_inception_v2_coco	58	28
faster_rcnn_resnet50_coco	89	30
faster_rcnn_resnet101_coco	106	32
faster_rcnn_inception_resnet_v2_atrous_coco	620	37

At first, I chose the Inception v2 architecture since it has a nice speed-to-accuracy ratio. But later, when compared to ResNet101, the results were evidently worse. For better visualization, we can look at fig. 3.16, where we can see comparison of the total loss of these two networks. We can see that even though ResNet101 performs better, the difference is not significant. However looking at fig. 3.17, we can see that ResNet is better at localizing the final bounding boxes, which was later also proved experimentally.

[25] https://www.tensorflow.org/guide/datasets
[26] http://cocodataset.org

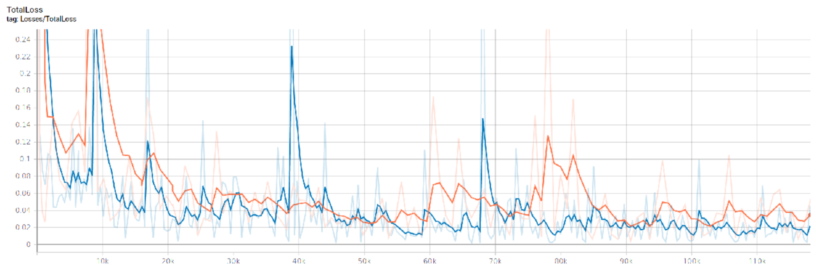


Figure 3.16: Total loss comparison. Orange = Inception v2; Blue = ResNet101

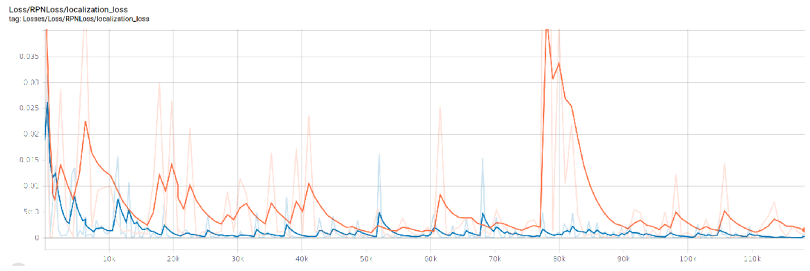


Figure 3.17: Localization loss comparison. Orange = Inception v2; Blue = ResNet101

The training itself was simplified by the authors of the object detection research model in TF's model repository. It contains several useful scripts to get started and the model configuration is done using `pipeline.config`, which defined using protobuf messages (in textual representation). Most of the configuration options were at first left to the ones set in the pre-trained model and only tweaked later after conducting some experiments, because they were optimised for both the architecture and the original dataset. Due to the complexity of the configuration file and number of possible options to set, only some key options are listed in the overview below.

- **num_classes** - number of classes the detector should know
- **feature_extractor** - type of the NN backbone used for feature extraction (ResNet)
- **grid_anchor_generator** - parameters for generating image anchor points
- **batch_size** - number of images processed in single iteration
- **learning_rate**
- **fine_tune_checkpoint** - path to the pre-trained model
- **data_augmentation_options** - configures various pre-processing functions for data augmentation
- **train_input_reader** - data sources for training
- **eval_input_reader** - data sources for evaluating the newly trained model

The training itself was performed in several iteration over the time of several weeks, experimenting with various options and training example combinations. Over these iterations, several improvements to the model's configuration were made.

Data augmentation

Data augmentation is a technique to artificially create new training data from existing training data. This is done by applying domain-specific techniques to examples from the training data that create new and different training examples. The intent is to expand the training dataset with new, plausible examples. [5]. These augmentations include image transformation and manipulation techniques such as flips, crops, shifts, colour changes, etc.

This technique was configured after few training iterations, after evaluating, when it could be observed that the new model is focusing only on projection screens that are clearly visible and in foreground, which is the majority of images in the dataset, but fails when the screen is further away. To minimize this effect, `random_horizontal_flip` and `random_crop_image` options were added to the `data_augmentation_options` array and the NN was retrained. At first, the training process seemed to be too unstable, but after around 70000 steps, the total loss started converging to zero. Subsequent testing showed improved detection rate, though the model still fails to detect screens that are too far. This can however be ignored, since it is not relevant for the entire project, because even if the screen was detected correctly, it usually is not possible to match the exact slide from such a distance, especially with the video quality seen in the obtained data.

Another problem that was observed during the experiments was detection of projection screens where in bright environments, where the contrast of brightness of the projected slide and the environment is not very big. This was mitigated, though not completely removed using `random_adjust_brightness` augmentation step.

Grid anchor generator's aspect ratios

Another parameter from the configuration file that was changed is the list of aspect ratios in the `grid_anchor_generator` option from the config file. The defaults include 0.5, 1.0 and 2.0. These are used to generate the size sliding windows used in the NN. This can be tuned to performed better for projection screens. Analyzing the downloaded dataset, we can see, that most of these are in 4:3 (1.33) and 16:9 (1.77) ratios, so these were added to the list.

Optimizer

Another parameter that was changed during the experiments was the training optimizer function, which was set to momentum by default. After investigating the qualities various algorithms [11], this was changed to Adam [12]. However, the convergence speed did not improve drastically and this change was therefore reverted.

3.3 Results

As described before, the training and evaluation is done using external scripts from TF's repository, for usage of the NN however, there is the `Detector` implemented, which initializes the TF API and handles both detection of objects and their visualization. It is used in both the final solution and via helper commands `./detektilo utils detect-image` and `./detektilo utils detect-video`, which can be used for demonstration and debugging purposes for detecting the screen in eight images or videos respectively. Other scripts, which were created in this part of the project include `extract-screenshots` for extracting the training data – frames from the video dataset provided, `annotate` for generating annotations for all images in one directory, where the screen position is unchanged, `unify-dataset` used for extracting training and testing data and putting them to a single folder, and `generate-tfrecords`, which generates TFRecords by extracting the data from all annotations it can find. All of these are available in the main script and have user help available if needed using the `--help` option.

Evaluating the precision and success rate of this model can be rather challenging considering the variability in visual appearance of a projection screen. Due to this factor, first the automated model evaluation will be discussed followed by a real-world usage overview.

3.3.1 Model evaluation

The final model training took over 50000 steps and was stopped when the total loss stabilized under 0.05. Previous experiments were done on models after as much as 228160 steps, but they showed no extra potential in detection nor localization of the projection screen.

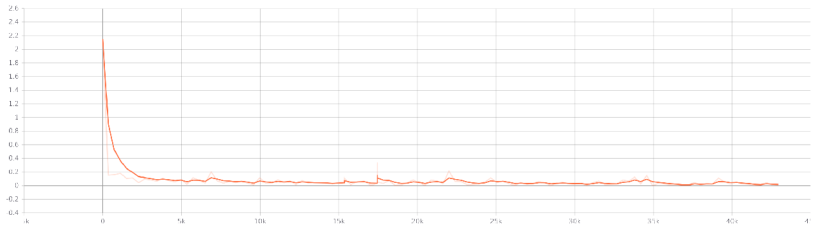


Figure 3.18: Chart of the model's TotalLoss over number of steps during the training

The evaluation of the model itself, was done using the `eval.py` script from the TF's object detection API repository running simultaneously with the training script, evaluating the each new checkpoint on the testing dataset. The default set of metrics is based on the protocol for the PASCAL VOC Challenge [27]. This was changed to the metrics set from COCO challenge [28], which is similar, but adds mAP (mean average precision) at several IOU (intersection over union) thresholds and recall metrics, which define how good does the model find all the positives, while the precision only says the percentage of correct predictions, see fig. 3.19.

[27] http://host.robots.ox.ac.uk/pascal/VOC/voc2010/devkit_doc_08-May-2010.pdf

[28] http://cocodataset.org/

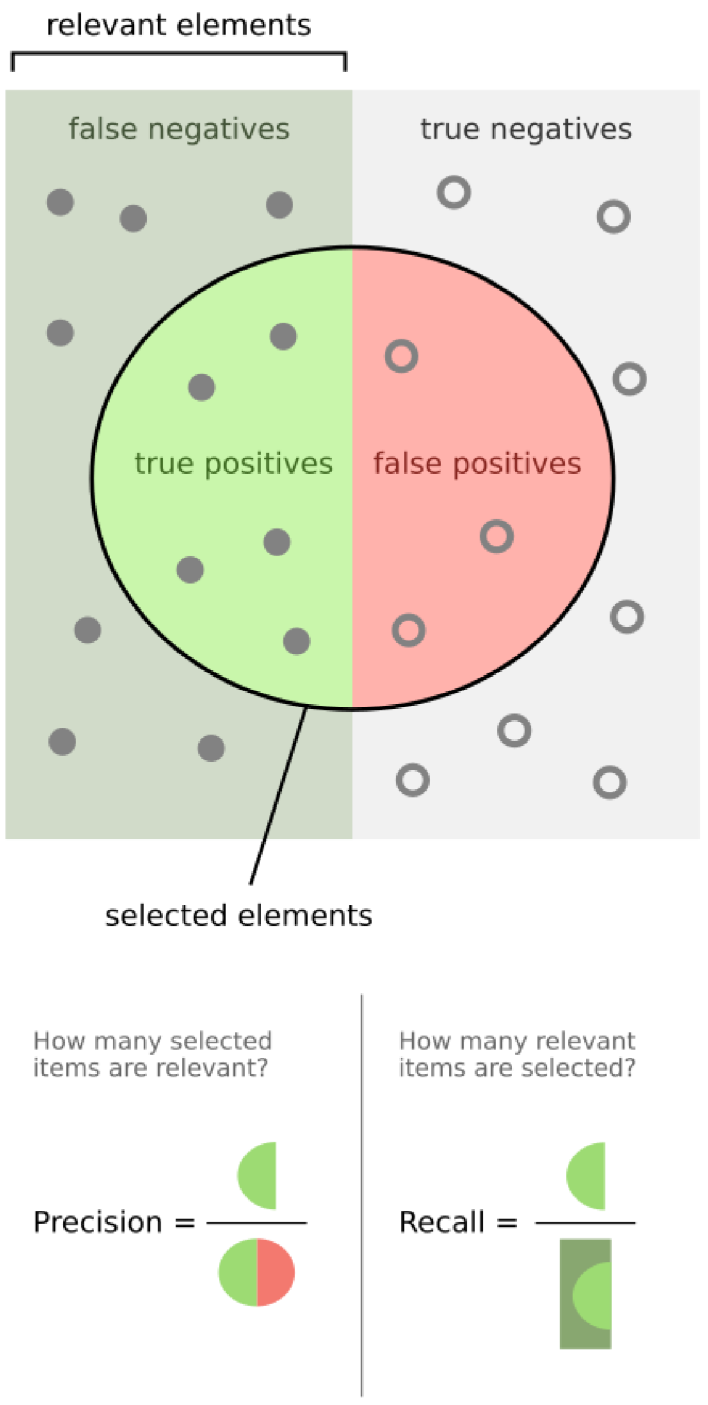


Figure 3.19: Graphical comparison of precision and recall. Source: [28]

The progress after each evaluation step was monitored and visualized using the Tensorboard application [29]. The evaluation script was also configured to plot 100 images, which are displayed directly in Tensorboard for better understanding of the progress.

[29] https://www.tensorflow.org/guide/summaries_and_tensorboard

Since the resulting bounding box of the detected screen will be further used for slide matching, it is important for the bounding box to be as precise as possible. For this reason mAP@.75IOU will be used as the main metric for evaluating the model's precision. This was finally at 0.9742 at the end of the training. The model's recall has finally stabilized at 0.6775. This shows that even though the model selects mostly relevant objects from the scene, it may struggle to find them in some cases.

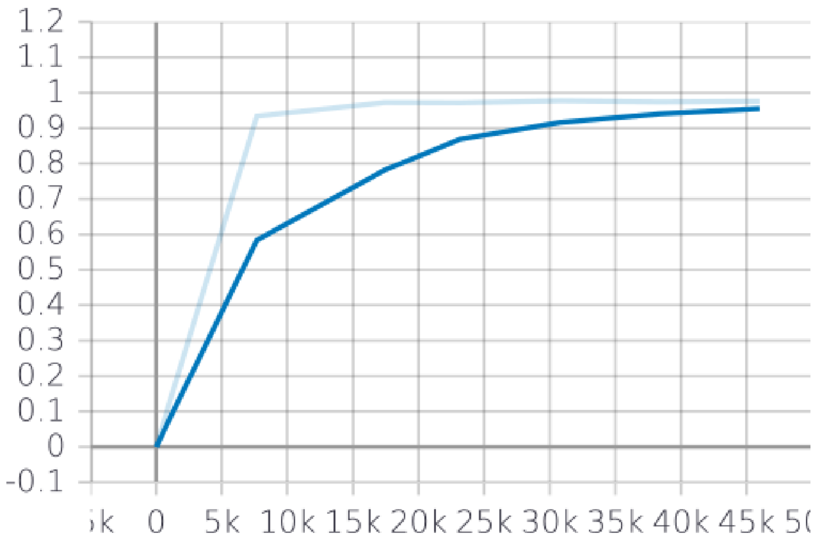


Figure 3.20: Chart of model's mAP at 75% IOU over the training

3.3.2 Real-world usage

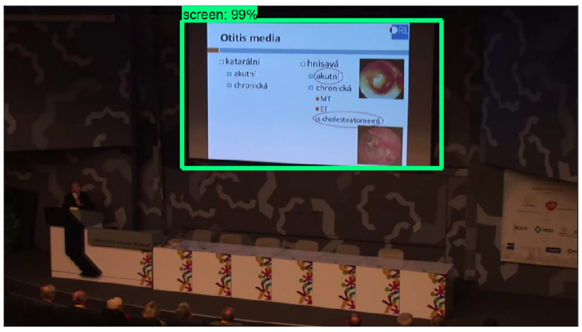
While the model's precision is considerable to be high, there are some cases when the detector may not perform as expected. These should be taken into consideration when working with the program. For the purpose of comparison and results presentation, the images will be divided into 3 categories, examples of which can be seen in figure 3.21:

- **optimal conditions** - large clearly visible projection in darker environment
- **suboptimal conditions** - low contrast between projection screen and the environment, screen is too far from the
- **common conditions** - anything between the first 2 categories

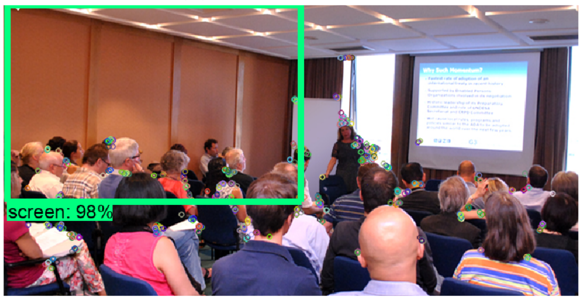
Under suboptimal conditions, the detector can fail with either no match or a false positive. This has to be considered for further development.

Below is a table with percentage of successfully detected images from both categories. This evaluation was done manually on a sample of 150 images. A successful detection is such where there is at least one bounding box around the screen covering most of it in a way that allows further slide matching.

	Ideal	Normal	Suboptimal	Total
Total	42	51	57	150
Succes	39 (92.85%)	46 (90.2%)	43 (75.43%)	117 (78%)
Failure	3 (7.14%)	5 (9.8%)	14 (24.56%)	33 (22%)



(a) Ideal conditions

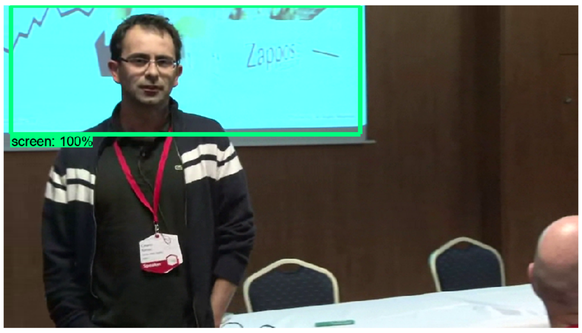


(b) Suboptimal conditions

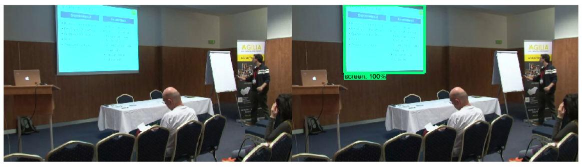
Figure 3.21: Examples of ideal and subideal conditions for screen detection

3.3.3 Comparison to previous solution

Since the goal of this paper is to improve the existing solution, we need to compare also the screen detection part of the project. It is hard to say which one is the absolute winner. We can however say that this new screen detection solution is much more flexible, since it does not enforce the form of the projection screen nor does it have any problems in cases where one part of the projected image fades into the background making it impossible to detect an edge and therefore the projection screen in the original solution. And since we can see the problems of detecting the screen, we can know we can expect them and counteract in the subsequent scripts.



(a) Projection screen partly overlapped



(b) Cut-off corner of the projection screen

Figure 3.22: Examples of various edge cases with successful screen detection.

3.4 Conclusion

This chapter described various methods available for detecting an object in an image including a description of various aspects of these methods and their comparison. It also described the proposed a solution for detecting a projection screen using Faster R-CNN with Resnet101 backbone. The results described in section 3.3, show that detecting an object as complex with as many possible visual representation as projection screen is very difficult, but this solution is viable and can be used in various scenarios despite its caveats. These include mostly false positive matches, which can and should be handled in a separate script, that will be used to process these outputs. Such a script is proposed and its implementation is described in the following chapter (4).

Chapter 4

Slide matching

After finding the appropriate part of video screenshot with one of the possible techniques discussed in previous chapter (3), we need to find the matching slide in the set of slides from the presentation corresponding to current video. This chapter will discuss several different techniques, that could be used for comparing single image to a corpus corresponding of multiple images, finding the best match or failing gracefully, which is important in order to avoid false positives.

4.1 Visual similarity

Since the previous screen detection algorithm extracted only the relevant parts of the image, ie. the screen, we could theoretically be able to simply compare the two images by their visual appearance. One of the methods that could achieve this is called *perceptual hashing*. It is a type of algorithm which hashes images based on their features **??**. Compared to cryptographic hashing functions like MD5 or SHA-1, which produce a completely different hash even on slight input change, perceptual hashing functions are analogous if features are similar.

4.1.1 Usage

This method is used both when preparing the dataset for NN training to only extract frames from videos that are not too similar and detecting whether the currently projected slide has changed between video frames. It could be used even for slide matching in case the slides are completely different, which usually is not the case and it would therefore be very complicated to distinguish between them. Another problem would be in cases where the projection screen is skewed and therefore the images wouldn't appear to be similar.

4.2 Feature detection, description and matching

Feature detection in the context of computer vision is a method, which is able to make a local decision at every point of an image whether or not the point is an image feature and describes this feature using a vector called descriptor. This section will further offer description of some of these methods based on [23].

4.2.1 Features

While there is no exact definition of what can be a feature, we can vaguely describe them as points of interest in the image – keypoints. The exact implementation depends on the used feature detection algorithm, which will be discussed later. There are several common main types of features:

- **Edges** - Edges are points in image, where 2 different image regions meet. Various implementations can have different thresholds on the smoothness of the edge.
- **Corners** - Corners often describe points, where 2 edges were detected near each other and have then been merged to a corner.
- **Blobs** - Blobs are used to desribe regions in an image, where the structure is smooth. The specific point can be chosen as local maximum value, center of gravity or similar.
- **Ridges** - Ridges are another type of important features. They can be detected as 2 parallel edges near each other. Ridge detectors are often used in aerial images processing for road extraction or in medical images for extraction of vessels.

Descriptors

In order to increase the chance of matching the correct pair of keypoints from 2 images, it is needed to describe them in a more precise fashion. This is done by the process of feature description, when an vector is assigned to each of these keypoints. This is done by extracting a small image portion around the keypoint.

4.2.2 Detectors

The main task of a feature detector is to chose and describe features from any given image. In order to be able to accomplish this task, it needs to satisfy these properties: repeatability and reliability. Repeatability is the ability to detect the same features in different images of the same scene even under different conditions like lighting, camera angle, zoom or optical distortion. Under reliability we mostly understand the detector's ability to detect strong features, which will be characteristic for the current image, therefore minimizing the chance of false positive when matching the images. Another very important aspect of feature detectors is the size of feature descriptors. This size strongly affects time and memory requirements for the detection [3]. Following is a non-exhaustive overview of common feature detectors. All of them are available in the OpenCV library, which will be used in implementation.

SIFT *Scale-Invariant Feature Transform* is an algorithm created by D. Lowe in 2004 [13]. It uses scale-space filtering, in which *Laplacian of Gaussian* (LoG), respectively its approximation *Difference of Gaussians* (DoG) is found for image with various σ values. Then the algorithm looks for local extremes over both space and scale to ensure the best keypoint representation in the scale. This is done by comparing the pixel to its 8 neighbours in current scale and 9 in previous and next scale, see 4.1. It then filters out low-contrast keypoints by eliminating keypoints with intensity lower than a defined threshold. The rest of the candidate keypoints are then assigned orientation to for rotation invariance by choosing a peak in histogram of the keypoint's neighbourhood. The keypoint's descriptor is created in similar way adding more measurements for brightness invariance, etc. Thanks to these measurements it eliminates around 90% of false matches while discards only 5% correct matches according to the paper. [17]

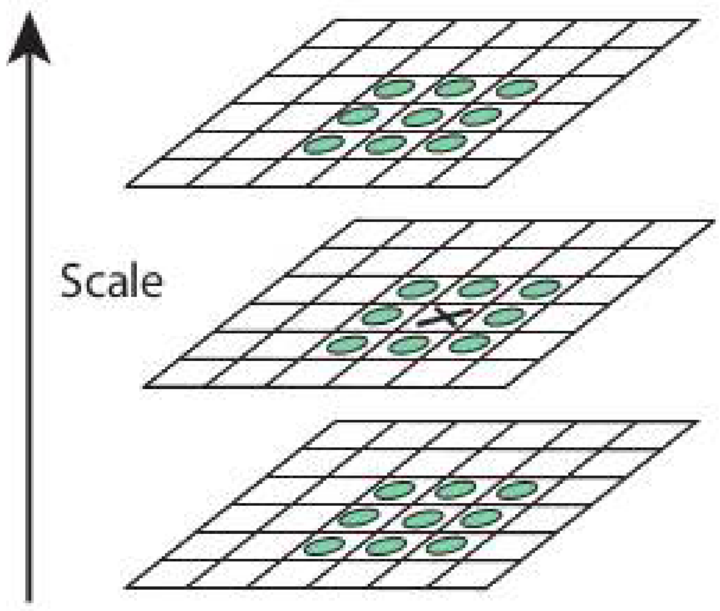


Figure 4.1: SIFT searches for local extrema over both scale and space

SURF *Speeded-Up Robust Features* is a faster version of SIFT created in 2006 [3]. SURF replaces the LoG approximation using DoG by using box filters. One big advantage of this approximation is that, convolution with box filter can be easily calculated with the help of integral images (see. 3.4) [17]. SURF assigns orientation using wavelets' responses in both horizontal and vertical direction on the neighborhood. These are also used for the final descriptor computation, but with larger neighborhood divided into several regions. In short, SURF adds a lot of features to improve the speed in every step. Analysis shows it is 3 times faster than SIFT while performance is comparable to SIFT. SURF is good at handling images with blurring and rotation, but not good at handling viewpoint change and illumination change [18].

ORB *Oriented FAST and Rotated BRIEF* is the newest algorithm created in 2011. It is a free alternative to the patented SIFT and SURF algorithms. As the name suggests, ORB is a combination of FAST keypoint detector[1] and BRIEF descriptor[2] with some modifications for speed boost, but also for improving the feature extraction. ORB modifies FAST to add rotation invariance by computing the intensity weighted centroid of a keypoint neighbourhood patch with the corner keypoint in the center. The descriptors are computed using BRIEF, which is however known for worse results with rotation. ORB solves this problem by rotating the descriptor using the keypoint's orientation.

[1]http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.60.3991&rep=rep1&type=pdf

[2]https://infoscience.epfl.ch/record/167678/files/top.pdf

Selecting keypoint detector

All of the aforementioned algorithms are battlefield proven and capable of extracting quality, valid and stable keypoints for most use cases. ORB is the fastest algorithm while SIFT performs the best in the most scenarios. For special case when the angle of rotation is proportional to 90 degrees, ORB and SURF outperforms SIFT and in the noisy images, ORB and SIFT show almost similar performances [10]. However it is important to note, that both SURF and SIFT are patented algorithms and paid licenses, which makes them less usable for this project. The paper also shows the ORBs superiority, when it comes to comparing scaled images, which is very important in case of matching high-resolution slides from presentation to their small counterpart from the video.

4.2.3 Matching

After successfully detecting the keypoints and describing them using one of the detectors, it is now required to match the keypoints from the query image and candidate image. In OpenCV, this can be done using 1 of 2 most commonly used matchers.

BFMatcher BFMatcher is a OpenCV's class implementing a Brute-Force matcher, which is very simple. It works by finding the best match – the most similar descriptor in the train image for every descriptor in the query image. This can therefore be computationally more expensive.

FLANN FLANN stands for Fast Library for Approximate Nearest Neighbors, which is a library for performing fast approximate nearest neighbour search in high-dimensional spaces[3]. It builds an efficient data structure - K-D tree[4], which is then used to search for nearest neighbour.

They both typically work fine in most use-cases and can be tweaked by several parameters to gain optimal results, but when the brute force method has its advantages in accuracy, because it will always return the best possible match, while FLANN will find an approximate nearest neighbour. The speed gain from FLANN is negligible in this project compared to the time required for other processing, therefore BFMatcher will be used.

4.3 Text matching

Another possible approach to matching screenshots slides is to use the text in them to try and find the best slide. In order to do this, it is first required to extract the text from all the images, create some sort of machine-friendly representation of this text, and then compare these texts in a way, that can return the best match.

4.3.1 Text detection

OCR stands for *Optical character recognition*. It is a method used for converting picture representation of text to its editable form. This is most commonly used for converting scanned paper documents into electronic format, that can be edited in a word-processing

[3] https://www.cs.ubc.ca/research/flann
[4] https://en.wikipedia.org/wiki/K-d_tree

program. It is important to note, that no OCR program is 100% accurate and the subsequent implementation should count on that.

Tesseract

Tesseract is an open-source OCR engine developed at HP between 1984 and 1994 [25]. Tesseract has unicode support and can recognize multiple languages (over 100) even though it was originally developed to recognize english only, and it can be trained for new languages[5]. In the latest version (4), it includes new NN-based recognition engine, which has higher accuracy. Tesseract also provides libraries for development and there are many community-owned wrappers for various languages, eg. pytesseract[6] for Python.

Preprocessing

Tesseract OCR engine is very sensitive to the input data it gets. However, real-world images are often less then optimal and they often contain noise, distortion or other errors. Even though Tesseract does some preprocessing by itself, better results can be achieved by doing some preprocessing tailored for our input data beforehand. Following is a non-exhaustive list of possible and used preprocessing methods.

- **Threshold** In computer vision, thresholds are used to easily create 2 separate pixel groups. The output of this operation is a binary image consisting of ones and zeroes, often represented using white and black colour. This method works with a user-set threshold limit. Any pixels under this threshold are assigned a value of zero and the others are assigned one.
  - **Global threshold** is one of the variants of thresholding. It works with a single global threshold value used for the entire image.
  - **Adaptive thresholding** is especially useful in cases, when the image has different lightning conditions in various sections. With adaptive threshold, the algorithm computes the most accurate threshold value for every region of the image. As we can see in fig. 4.2, the adaptive threshold with `ADAPTIVE_THRESH_GAUSSIAN_C` set as `adaptiveMethod` perfoms best. This method computes the threshold from gaussian-weighted sum of the neighbourhood values.

[5]https://github.com/tesseract-ocr/tesseract
[6]https://github.com/madmaze/pytesseract

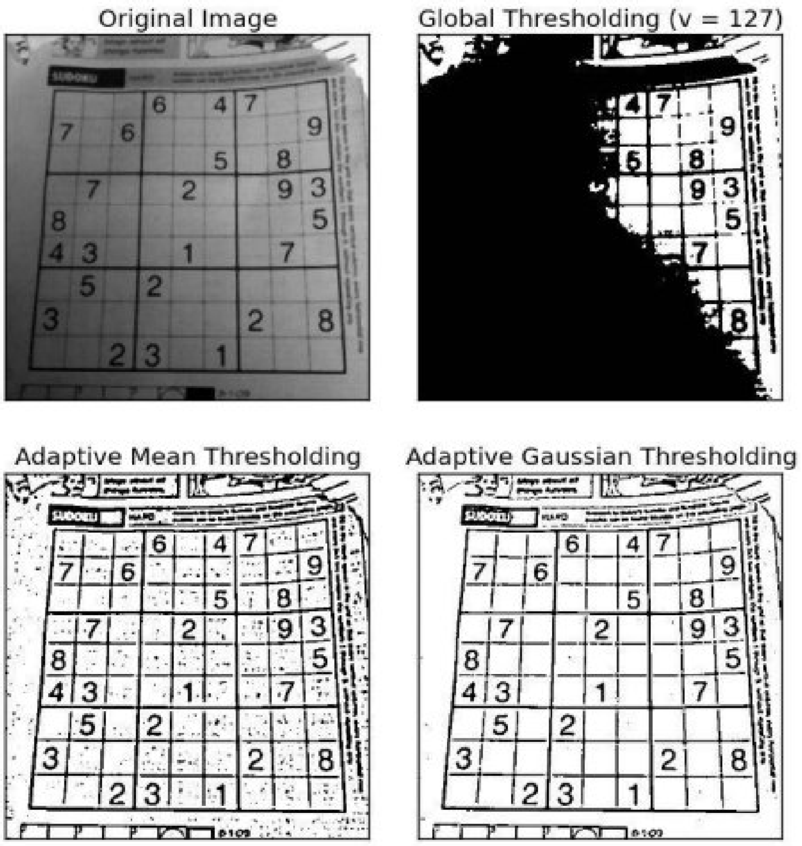

Original Image
Global Thresholding (v = 127)
Adaptive Mean Thresholding
Adaptive Gaussian Thresholding

Figure 4.2: Comparison of global and adaptive threshold on an example image

- **Noise removal** Another method used in image preprocessing is noise removal. This is very important, because even minor noise can result in nonsensical text. By combining the two methods described below (also known as opening), we can achieve noise removal.
  - **Dilation** - is a convolution of some kernel over the image. It computes the maximum pixel value overlapped by the kernel and replaces the pixel in the kernel's center by that value. This results in thickening bright regions.

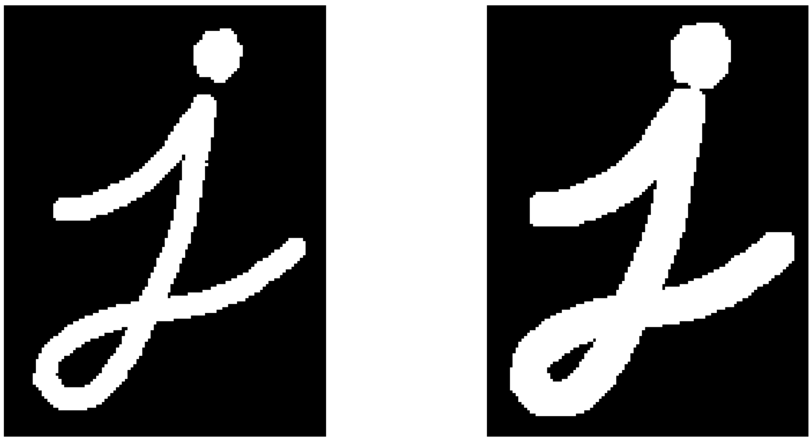


Figure 4.3: Dilation example

- **Erosion** is a sibling method to dilution. It works by replacing the kernel's center pixel value by the minimum, resulting into thickening dark regions.

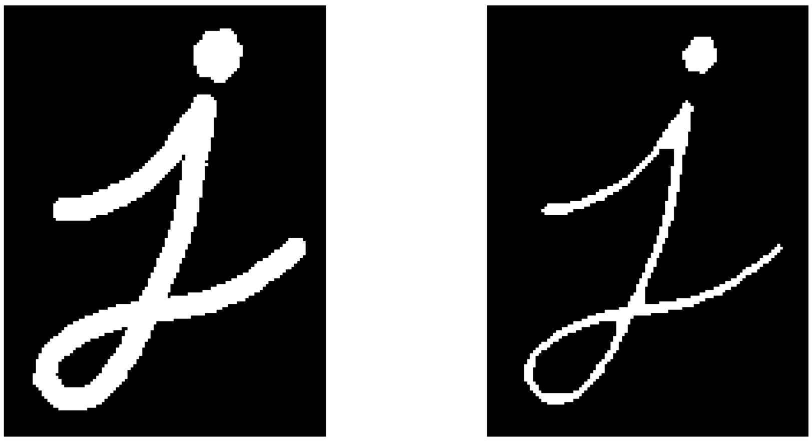


Figure 4.4: Erosion example

- **Normalization** Normalization is a process of changing pixels intensity across the whole image. In image processing it is also known as dynamic range expansion **??**. Normalization works by transforming an image consisting of pixels with minimal and maximum value to a new min-max range supplied by the user. In terms of preprocessing for OCR this can be very helpful for increasing the contrast by setting the range to $< 0, 255 >$.

4.3.2 Text processing

After extracting the text from images, it has to be processed for matching. In natural language processing, it is often considered best practice to first remove words, that have no semantic value (prepositons, conjuctions, interjections, etc.), because they do not contribute to the meaning of the text, but are very common and can therefore degrade the processing. However, in this case, we are only trying to match exact text with possible minor distortions or misspelled words, therefore even these semantically uninteresting words can help choosing the correct text.

Bag of words

Bag of words is one of the most common methods that can help with the text preparation for machine processing. It represents the input texts as bags/multisets, which can then be transformed to vectors, where each dimension represents multiplicity of the word from dictionary. It works in several steps:

- Collecting data
- Cleaning data
- Tokenize data
- Create vocabulary
- Generate vectors

A vocabulary is a vector of all unique words from all input texts, eg. vocabulary created from sentences *I like cats* and *I like dogs* would be $[I, like, cats, dogs]$. After creation of the vocabulary, we can create vectors representing each input text. This is done by creating a vector with the same size as the vocabulary, which will be initialized with all zeroes, and counting the number of words in the text for each word from dictionary. Continuing with the previous example, we would get 2 vectors representing the input sentences: $[1, 1, 1, 0]$ and $[1, 1, 0, 1]$.

4.3.3 Cosine similarity

For matching of two texts transformed into vectors, *cosine similarity* (CS) can be used. It can estimate similarity between 2 vectors. Given 2 vectors – **A** and **B**, CS can be computed using the following equation 4.1. The resulting value will be in the range of $< 0, 1 >$ with a higher number indicating better similarity.

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum\limits_{i=1}^{n} A_i B_i}{\sqrt{\sum\limits_{i=1}^{n} A_i^2}\sqrt{\sum\limits_{i=1}^{n} B_i^2}}, \tag{4.1}$$

4.4 Implementation

For the practical implementation of solely matching one image to a corpus of many images, mainly the keypoints matching and text methods described in the previous section were used in an attempt to get the best match ratio without wasting system resources. Each of them has different strengths and the user will be able to choose the best combination for a specific use case.

For the matching a separate class called `Matcher` was created in the `misc` package. This class encapsulates both matching methods and some helper functions. It works with a set of candidate images and a query image. Following is a quick overview of the most important methods:

- `set_candidate_images` - imports new candidate images, replacing any existing candidate images.

- `add_candidate_image` - add new candidate image to the set of existing images. This method also internally recomputes all the data for matching for better precision.

- `match` - the main matching function it take the query image as an argument, other possible arguments include the matching methods to use (text and/or keypoints) and their respective weights. It returns a `Match` object. The method tries to match the query image to the candidate images using the methods as described earlier. It then computes an overall score using a weighted average of the scores.

The constructor takes a set of images and extracts first runs keypoints detection, saving all of the resulting keypoints and their descriptors in a single array in the object. This helps the performance of the matcher as the keypoints don't have to be matched every time the match method is called.

Another important class is the `Match` class, which is returned by the `match` method of `Matcher`. It contains a reference to the query image, array of all the candidate images in time the matching was performed and an array of the scores assigned to each of them.

4.4.1 OCR

The initial implementation of detecting textual characters in an images was done using a custom-trained fully-connected NN, but this was found to be very inaccurate and problematic for both development and usage. The second attempt uses Tesseract OCR engine[7] with the pytesseract[8] wrapper for python. Even though this adds the requirement for the destination computer to have tesseract install along with all the language data needed, it provides a much more complex solution with multi-language support out of the box.

Preprocessing

In order to get better results with images that could be problematic for the OCR engine. This is done in the `Image` class which is parent class for all other image classes (`Slide` and `VideoFrame`). The preprocessing consists of several steps:

1. Rescaling the image - since the images usually come from bad sources like the video records, they are first upscaled for both better next steps in the preprocessing and for the OCR engine

2. Grayscaling - this is mostly required for the next steps

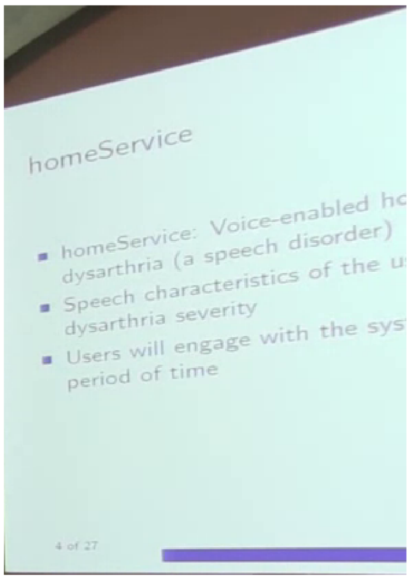
3. Sharpening - the image is sharpened using an unsharpen mask filter[9]

4. Eroding - one eroding operation with 100 iterations and $1x1$ kernel is executed to thicken any lines in the image without too much blurring (as opposed to using bigger kernel)

5. Normalization

6. Binarization

[7] https://opensource.google.com/projects/tesseract
[8] https://github.com/madmaze/pytesseract
[9] https://en.wikipedia.org/wiki/Unsharp_masking

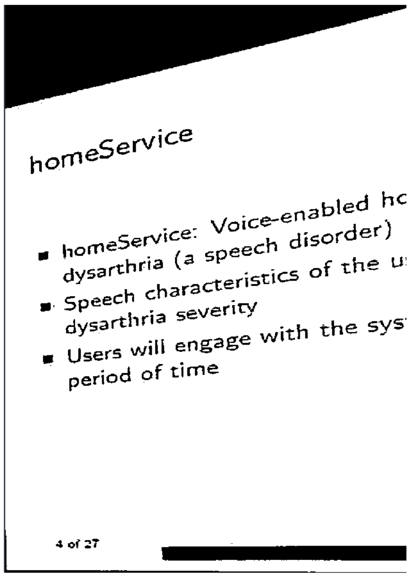


homeService

- homeService: Voice-enabled hc
  dysarthria (a speech disorder)
- Speech characteristics of the u
  dysarthria severity
- Users will engage with the sys
  period of time

4 of 27

(a) Original image



homeService

- homeService: Voice-enabled hc
  dysarthria (a speech disorder)
- Speech characteristics of the u
  dysarthria severity
- Users will engage with the sys
  period of time

4 of 27

(b) Processed image

Figure 4.5: Result of OCR preprocessing pipeline

Postprocessing

In the postprocessing we remove all clutter, which was created by the OCR engine due to non-ideal conditions such as noise. This is simply done during the vocabulary creation by only passing words consisting of only lower- and uppercase letters or only digits.

4.4.2 Keypoints matching

As explained earlier, the `BFMatcher` is used for keypoints matching. This type of matcher doesn't have to be trained (as opposed to FLANN matcher) and therefore the whole implementation of keypoint matcher is straight-forward. The keypoints and their descriptors are extracted using the `get_keypoints_and_descriptors` method on the `Image` class and then stored both in the Image object and the Matcher once that image is added to the candidate set. When a request to match a query image is received, the BFMatcher object tries to match all the keypoints of the query image to those of the candidate images. This results in an array of matches for each candidate image. From this, the final score is computed. Every match has a distance between the 2 keypoints expressed as Hammning distance and since the ORB descriptors are 32 bytes long (i.e. 256bit), the maximum distance between them will be 256. This number is then normalized to the range of $< 0; 1 >$. This way all of the matches are evaluated and the final score for the candidate image is calculated using equation 4.2, considering there is N matches. This way the candidate images with more matches and higher average scores are favored. It is also important to note that only „good“ matches are considered to be matches. These are differentiated by applying ratio test as proposed by D.Lowe in [13].

$$score = \left(0.5 + \frac{\tanh\left(\frac{x}{3} - 1\right)}{2}\right) * \frac{\sum_{n=0}^{N-1} match_n}{N} \tag{4.2}$$

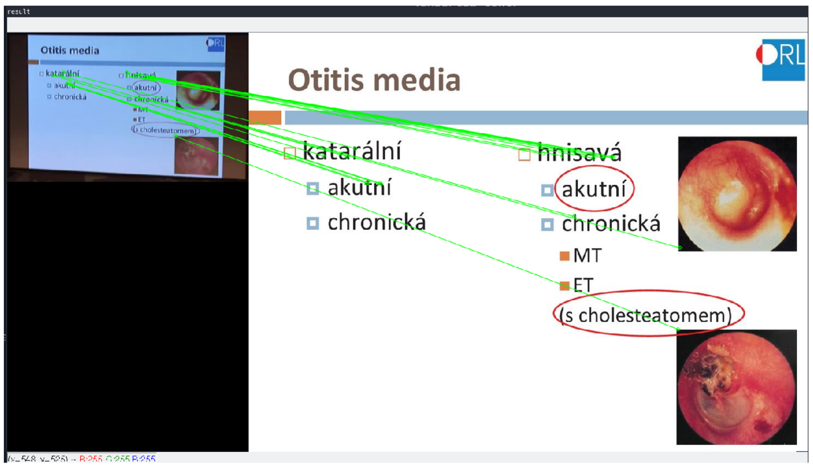


Figure 4.6: An example of successful matching of cropped-out screen to a slide using key-points

4.4.3 Text matching

The textual matching is implemented using the same Matcher class, which uses the `get_bag_of_words` method on `Image` class. This method retrieves text from the images and then filters them down to only words containing 2 or more word characters using regular expressions. Unfortunately the whole process of vocabulary creation and vectorization has to be done for every new query image, since it may change the vocabulary. For every new query image, all the words are first corrected, because query image may contain misspelled words due to OCR inaccuracies. This process consists of calculating Levenshtein distance between all word from the query image and dictionary from all training images. All words with distance lower than 2 are considered misspelled and replaced. This improved the text matching score by 10% on average. Once the vectors are created a cosine similarity is computed for every candidate image in reference to the query image. This results in an array of scores, which is then further processed by the `match` method.

4.5 Results

The methods implemented are useful for matching images. But it is important to note that either of them has its own limitations and the end user should chose one (or both) according to the type of images being processed. While the text matching method has very high success rate for mostly white slides with dark text and good image quality (i.e. contrast, lightning), it is rendered almost unusable for graphical slides. In this case, the the keypoint matching is the winner.

To successfully evaluate the methods a sample of 10 different presentation was chosen from our dataset. These were then divided into 2 groups – textual and graphical and matching was run on both of them by both methods. The matching was done between cropped images from the video and the slide images, not by directly processing video. The results shown in table below clearly support the claim from the previous paragraph.

	Textual	Graphical
Total	182	98
Keypoints matching success	142 (68.13%)	85 (86.73%)
Text matching success	165 (90%)	42 (42.85%)

Another statistic was made to evaluate the overall matching ability using the combination of the 2 methods. In this case, the weights were left to default (1:1). The evaluation was performed by processing 3 different presentation in interactive mode and manually counting the number of frames processed successfully. This gave as the total number of **640** frames, out of which **550** were matched correctly, meaning a precision of **85.94%** was achieved. Out of those that failed to match, many belonged to the same difficult slides and were not processed again because the slide did not change between detections as will be explained in the next chapter.

4.6 Conclusion

In this chapter several methods for matching slides were introduced and explained. These methods were then used in the new implementation of a Matcher, which will be integrated to the overall solution in the next chapter.

Chapter 5

Integration

Even though the methods described in previous chapters were implemented as separate classes or modules and were originally usable using their own scripts, the final goal is to implement an application combining these 2 approaches to get the best accuracy while making the process as automatic as possible.

5.1 Detektilo script

The finished project has all of its functionalities accessible through one script called *detektilo*, which can be found in the project's root directory. This script uses the Click [1] python library to create an easy to use CLI program. The program's has structured commands and there's user help information and available options for every command using the `--help` option passed as argument. The command structure is as follows:

- `process-video` – the main command for video–slide synchronization.
- `utils` – command group for utilities
  - `extract-screenshots` – extracts separate frames for videos and saves them to their respective folder
  - `annotate` – used to generate xml annotation files from default.xml for all images in that folder
  - `generate-tfrecords` – gathers all xml annotated images and converts them to TensorFlow's TFRecord files
  - `detect-image` – detects screen in an one or more images, output in CSV, support interactive mode
  - `detect-video` – detects screen in a video, output in CSV, support interactive mode
  - `convert-pdf` – converts a PDF file to separate images
  - `image-to-text` – runs OCR processing on an image, plots the processed image and outputs the unprocessed detected text, i.e. no postprocessing is done on the text.
  - `match` – tries matching of a single image to a set of different images with the matching method being selectable as argument

[1] http://click.palletsprojects.com

5.2 Video processing

The main task of processing a video is done by the script's `process-video` command. It can be modified with the following options:

- `--video-file` – path to the video file to process
- `--presentation-file` – path to the PDF file with presentation to process
- `--presentation-dir` – path to the directory with images from the presentation
- `--skip-first` – how many seconds to skip from the beginning of the video
- `--every` – how often to process
- `--model-dir` – path to the directory with TF inference graph of the model to use
- `--labelmap-file` – path to the labelmap file
- `--further-crop` – a pair of number defining number of pixels to crop further inside from the detected screen box
- `--screen-box` – disables screen detection and uses this foursome as the screen position throughout the video
- `--no-keypoints-matching` – disable keypoints matching method
- `--keypoints-weight` – weight of the keypoints score in the overall score
- `--no-text-matching` – disable text matching method
- `--text-weight` – weight of the keypoints score in the overall score
- `--language` – language of the slide for more precise OCR translation
- `--output-dir` – directory for the script's output data, where the detected screens and the final XML file will be saved.
- `--output-compatible` – enable compatibility mode
- `--interactive` – enable interactive mode, plotting the screen detection and slide matching process to GUI windows

There are some limitation on usage of these flags. At least one of the matching methods must be used, only one of `--presentation-dir` or `--presentation-file` can be used for the processing and others, that the script's help will state.

The video processing begins by opening the input files – video and presentation and parsing the presentation. This operation can take a while. After that, TensorFlow initialization is run (unless this is overriden by providing the screen location). Afterwards the video is forwarded by `--skip-first` seconds and the first frame's screens are detected. These detected areas are then cropped out of the original frame and saved for later processing. After all screens are detected, we need to match them. To ensure we are not matching screens that were processed, we first calculate the perceptual hash as described in 4.1. This is then used to determine whether a similar image was processed in previous frame by calculating the distance between them. If it is less than a set threshold further slide matching in that screen is skipped. If the image is not similar to the previous one, keypoints, descriptors and text are exported from the frame and everything is processed using the corresponding matching method and scores from each candidate image using each method, that didn't fail to find a match is counted to the final score for the candidate image and top 3 candidate images are saved as possible matches for the screen. if the top candidate wins by a margin threshold, the others are thrown away. If this is the case and the top candidate is a the same slide as the last matched slide, this whole match is thrown away.

After processing of all frames in a video in this fashion, the results can be exported to the output file, which will be overwritten if it already exists. The output format is XML and is described in the following section

5.3 Output

Because this program is supposed to replace the existing solution, it has to integrate well with the surrounding scripts, so it is preferable to keep the script's output in the same format as before. But due to a different internal working mechanisms and more complicated scoring system a new output system will be used. While one compatible to the old one is accessible by the `--output-compatible` flag.

The main benefit of the new output is that it supports multiple screen and it has alternatives for every matched slide if any such alternatives exist together with their respective scores, so that a new tool can be built around this output which will get both the best match and its alternatives and user could select the most appropriate one, making error correction easier. Following is an example of the new output format in XML.

```
<job name="otitis-test" video_src="video_HD.webm" presentation_src="slides.pdf">
        <screen id="0" start="30" end="43">
                <original img="cropped/30.jpg" />
                <match slide_no="3" img="slides/slide-002.jpg" score="0.5554302" />
                <match slide_no="2" img="slides/slide-001.jpg" score="0.5554302" />
        </screen>
        <screen id="0" start="44" end="60">
                <original img="cropped/44.jpg" />
                <match slide_no="3" img="slides/slide-003.jpg" score="0.7780" />
        </screen>
</job>
```

Chapter 6

Conclusion

In this paper a new solution for synchronization of slides with a video recording of their presentation was proposed and implemented with satisfactory results. The script created for the task is capable of a 2 different matching methods and has support for multiple options available to the end user to tune in response to a specific situation, but with sane values set as default. This solutions allows for simply automated synchronization with output formats that can be further processed by a machine.

The results for each of the components used are described in the chapters discussing them. The overall quality of the solution is good, though the user may encounter processing jobs where either the screen detection or slide matching may fail.

In regards to the future of the development, the biggest improvements could be made in automation of tuning of the parameters in order to make the work with the script easier and more automated. This could include automatic slide type recognition in order to change the weight of scores or disable one of them altogether, and to tune preprocessing for the optical character recognition to improve its results automatically.

Bibliography

[1] Tensorflow detection model zoo. online. accessed 23-April-2019.
Retrieved from: https://github.com/tensorflow/models/blob/0558408514dacf2fe2860cd72ac56cbdf62a24c0/research/object_detection/g3doc/detection_model_zoo.md

[2] Aphex34: Max pooling.png. [Online; accessed 02-May-2019].
Retrieved from:
https://upload.wikimedia.org/wikipedia/commons/e/e9/Max_pooling.png

[3] Bay, H.; Ess, A.; Tuytelaars, T.; et al.: Speeded-up robust features (SURF). *Computer vision and image understanding*. vol. 110, no. 3. 2008: pp. 346–359.

[4] BornaGhotbi: CNNs in Image Segmentation. online.
Retrieved from: https://wiki.ubc.ca/CNNs_in_Image_Segmentation

[5] Brownlee, J.: How to Configure Image Data Augmentation When Training Deep Learning Neural Networks. online. 04 2019. accessed 27-April-2019.
Retrieved from: https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/

[6] Djuriš, J.; Medarević, D.; Krstić, M.; et al.: Design space approach in optimization of fluid bed granulation and tablets compression process. *The Scientific World Journal*. vol. 2012. 2012.

[7] Gandhi, R.: R-CNN, Fast R-CNN, Faster R-CNN, YOLO - Object Detection Algorithms. online. 07 2018.
Retrieved from: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e

[8] He, K.; Zhang, X.; Ren, S.; et al.: Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016. pp. 770–778.

[9] Kaplan, A.; Haenlein, M.: Siri, Siri, in my hand: Who's the fairest in the land? On the interpretations, illustrations, and implications of artificial intelligence. *Business Horizons*. vol. 62, no. 1. 2019: pp. 15 – 25. ISSN 0007-6813. doi:https://doi.org/10.1016/j.bushor.2018.08.004.
Retrieved from:
http://www.sciencedirect.com/science/article/pii/S0007681318301393

[10] Karami, E.; Prasad, S.; Shehata, M.: Image matching using SIFT, SURF, BRIEF and ORB: performance comparison for distorted images. *arXiv preprint arXiv:1710.02726*. 2017.

[11] Kathuria, A.: online. accessed 27-April-2019.
Retrieved from: https://blog.paperspace.com/intro-to-optimization-momentum-rmsprop-adam/

[12] Kingma, D. P.; Ba, J.: Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*. 2014.

[13] Lowe, D. G.: Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*. vol. 60, no. 2. Nov 2004: pp. 91–110. ISSN 1573-1405. doi:10.1023/B:VISI.0000029664.99615.94.
Retrieved from: https://doi.org/10.1023/B:VISI.0000029664.99615.94

[14] Mboga, N.; Persello, C.; Bergado, J. R.; et al.: Detection of Informal Settlements from VHR Images Using Convolutional Neural Networks. *Remote Sensing*. vol. 9. 10 2017: page 1106. doi:10.3390/rs9111106.

[15] Minsky, M.; Papert, S.: *Perceptrons: An Introduction to Computational Geometry*. Mit Press. 1972. ISBN 9780262130431.
Retrieved from: https://books.google.sk/books?id=Ow1OAQAAIAAJ

[16] OpenCV: Cascade Classification. online.
Retrieved from: https://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html

[17] OpenCV: Feature Detection and Description. online.
Retrieved from: https://docs.opencv.org/3.4/db/d27/tutorial_py_table_of_contents_feature2d.html

[18] OpenCV: ntroduction to SURF (Speeded-Up Robust Features). online.
Retrieved from:
https://docs.opencv.org/3.4/df/dd2/tutorial_py_surf_intro.html

[19] Pasquinelli, M.: Machines that Morph Logic: Neural Networks and the Distorted Automation of Intelligence as Statistical Inference. 2017.
Retrieved from:
http://www.glass-bead.org/article/machines-that-morph-logic/

[20] Ramer-Douglas-Peucker, N.-p.: Ramer–Douglas–Peucker algorithm. 1972.

[21] Ren, S.; He, K.; Girshick, R.; et al.: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In *Advances in Neural Information Processing Systems 28*, edited by C. Cortes; N. D. Lawrence; D. D. Lee; M. Sugiyama; R. Garnett. Curran Associates, Inc.. 2015. pp. 91–99.
Retrieved from: http://papers.nips.cc/paper/5638-faster-r-cnn-towards-real-time-object-detection-with-region-proposal-networks.pdf

[22] Rosenblatt, F.: *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Report: Cornell Aeronautical Laboratory. Cornell Aeronautical Laboratory.

1957.
Retrieved from: https://books.google.sk/books?id=P_XGPgAACAAJ

[23] Schmid, C.; Mohr, R.; Bauckhage, C.: Evaluation of interest point detectors. *International Journal of computer vision.* vol. 37, no. 2. 2000: pp. 151–172.

[24] Sharma, A.: Biological and artificial neuron structure. 10 2017.
Retrieved from:
https://www.quora.com/What-is-the-differences-between-artificial-neural-network-computer-science-and-biological-neural-network

[25] Smith, R.: An overview of the Tesseract OCR engine. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, vol. 2. IEEE. 2007. pp. 629–633.

[26] Suzuki, S.; Abe, K.: Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing.* vol. 30. 1985: pp. 32–46.

[27] Viola, P.; Jones, M.: Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1. Dec 2001. ISSN 1063-6919. pp. I–I. doi:10.1109/CVPR.2001.990517.

[28] Walber: Precisionrecall.svg. [Online; accessed 02-May-2019].
Retrieved from:
https://upload.wikimedia.org/wikipedia/commons/2/26/Precisionrecall.svg

[29] Wikipedia contributors: Haar-like feature — Wikipedia, The Free Encyclopedia. 2018. [Online; accessed 25-December-2018].
Retrieved from: https://en.wikipedia.org/w/index.php?title=Haar-like_feature&oldid=853222698

[30] Wikipedia contributors: Hyperparameter (machine learning) — Wikipedia, The Free Encyclopedia. 2018. [Online; accessed 16-January-2019].
Retrieved from: https://en.wikipedia.org/w/index.php?title=Hyperparameter_(machine_learning)&oldid=846637116

[31] Wikipedia contributors: Convolutional neural network — Wikipedia, The Free Encyclopedia. 2019. [Online; accessed 14-January-2019].
Retrieved from: https://en.wikipedia.org/w/index.php?title=Convolutional_neural_network&oldid=876940492

Appendix A

Poster

Robust
screen and
slide detection in video

SuperLectures

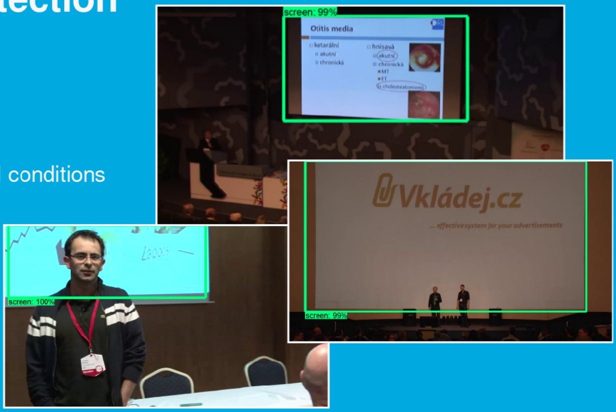
FAKULTA
INFORMAČNÍCH
TECHNOLOGIÍ

About project

The main goal of this project is to create a new solution for slides-to-video synchronization for SuperLectures project. This is achieved by combining several techniques and algorithms.

Projection screen detection

- Faster R-CNN with Resnet101
- CUDA support
- 97% precision
- <100ms processing time
- ability to detect screens in non-ideal conditions including: cropped, skewed, unclear edges
- can handle multiple screens
- overridable if needed



Slide matching

- combines matching using feature extraction and matching, and text matching
- uses free ORB algorithm
- text extraction using Tesseract OCR engine
- uses bag of words and cosine similarity for text matching

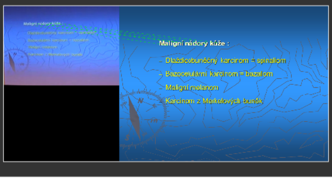
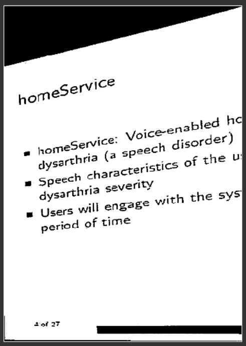



Figure A.1: Project presentation poster