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

CLASSIFICATION OF VARYING-SIZE PLANKTON IMAGES WITH CONVOLUTIONAL NEURAL NETWORK

KLASIFIKACE OBRAZŮ PLANKTONU S PROMĚNLIVOU VELIKOSTÍ POMOCÍ KONVOLUČNÍ NEURONOVÉ SÍTĚ

MASTER'S THESIS
DIPLOMOVÁ PRÁCE

AUTHOR **Bc. JAROSLAV BUREŠ**
AUTOR PRÁCE

SUPERVISOR **prof. Dr. Ing. PAVEL ZEMČÍK**
VEDOUCÍ PRÁCE

BRNO 2020

Brno University of Technology
Faculty of Information Technology

Department of Computer Graphics and Multimedia (DCGM) Academic year 2019/2020

Master's Thesis Specification

Student: **Bureš Jaroslav, Bc.**
Programme: Information Technology Field of study: Intelligent Systems
Title: **Classification of Varying-Size Plankton Images with Convolutional Neural Network**
Category: Image Processing
Assignment:

1. Study available literature about automatic plankton recognition and convolutional neural network based image classification focusing on approaches to handle varying input size and aspect ratio.
2. Propose solutions to handle large variation in plankton image sizes during recognition.
3. Implement a plankton recognition system that utilizes the proposed solutions.
4. Evaluate the plankton recognition system on provided dataset and compare the proposed solutions for varying input image size with naive approaches such as scaling and cropping.
5. Describe the achieved results and possibilities to continue the work.

Recommended literature:

- According to instructions of the supervisor

Requirements for the semestral defence:

- Items 1 to 3.

Detailed formal requirements can be found at https://www.fit.vut.cz/study/theses/

Supervisor: **Zemčík Pavel, prof. Dr. Ing.**
Consultant: Eerola Tuomas, doc., Ph.D., LUT
Head of Department: Černocký Jan, doc. Dr. Ing.
Beginning of work: November 1, 2019
Submission deadline: June 3, 2020
Approval date: June 3, 2020

Abstract

This work considers techniques of automatic image analysis based on convolutional neural networks (CNN) focused on plankton classification. There is a large variation in the shapes and sizes of plankton images. This makes the classification for CNN based methods challenging since CNNs typically require a fixed input size. Naive methods utilize scaling of the images to a common size. However, this operation leads to a loss of small details that are necessary for correct classification. The aim of this work was to design and implement a CNN-based classifier of plankton images and explore possible methods that can deal with varying image sizes. Multiple methods such as patch cropping, utilization of a spatial pyramid pooling layer, inclusion of metadata and construction of a multi-stream model were evaluated on a challenging dataset of phytoplankton images. An improvement of 1.0 point was achieved for the InceptionV3 architecture resulting in an accuracy of 96.2 %. The main contribution of this thesis is an improvement of multiple CNN plankton classifiers by successfully applying these methods.

Abstrakt

Tato práce pojednává o technikách automatické analýzy obrazu založené na konvolučních neuronových sítích (CNN), zaměřených na klasifikaci planktonu. V oblasti studování planktonu panuje velká diverzita v jeho tvarech a velikostech. Kvůli tomuto bývá klasifikace pomocí CNN náročná, jelikož CNN typicky požadují definovanou velikost vstupu. Běžné metody využívají škálování obrazu do jednotné velikosti. Avšak kvůli tomuto jsou ztraceny drobné detaily potřebné ke správné klasifikaci. Cílem práce bylo navrhnout a implementovat CNN klasifikátor obrazových dat planktonu a prozkoumat metody, které jsou zaměřené na problematiku různorodých velikostí obrázků. Metody, jako jsou patch cropping, využití spatial pyramid pooling vrstvy, zahrnutí metadat a sestavení multi-stream modelu jsou vyhodnoceny na náročném datasetu obrázků fytoplanktonu. Takto bylo dosaženo zlepšení o 1.0 bodů pro InceptionV3 architekturu s výslednou úspěšností 96.2 %. Hlavním přínosem této práce je vylepšení CNN klasifikátorů planktonu díky úspěšné aplikaci těchto metod.

Keywords

Plankton, machine learning, convolutional neural network, classification, image processing, varying image size, computer vision

Klíčová slova

Plankton, strojové učení, konvoluční neuronové sítě, klasifikace, zpracování obrazu, rozmanitá velikost obrazu, počítačové vidění

Reference

BUREŠ, Jaroslav. *Classification of varying-size plankton images with convolutional neural network*. Brno, 2020. Master's thesis. Brno University of Technology, Faculty of Information Technology. Supervisor prof. Dr. Ing. Pavel Zemčík

Rozšířený abstrakt

Plankton je soubor organismů, které žijí ve vodě a jsou volně unášené proudem. Jsou důležitou součástí našeho ekosystému, jelikož tvoří klíčovou část potravního řetězce, poskytují potravu organismům od malých ryb až po mohutné velryby. Mimo to je plankton také jeden z hlavních producentů kyslíku na Zemi, odhaduje se, že ho vytvoří až 80 %. Plankton dále slouží jako dobrý indikátor zdraví oceánu, některé jeho druhy jsou toxické, a tak mohou být nebezpečné. Pro tyto důvody je důležité správně klasifikovat jednotlivé druhy, které se ve vzorku vody nachází. Tradiční klasifikace spočívala v manuálním zkoumání provedeném zkušeným biologem, který na základě charakteristických příznaků zařadil vzorek do jedné třídy. Tento proces ale bývá velmi pracný, proto stále roste poptávka po automatické klasifikaci obrazových dat planktonu. Cílem této práce je vytvořit takovýto klasifikátor, který dovede automaticky rozpoznat jednotlivé druhy planktonu na základě obrazových dat s co nejvyšší přesností.

Rozpoznávání obsahu obrázků je stále častěji prováděno pomocí počítačového vidění. Z důvodu vysoké složitosti obrazových dat jsou takovéto metody obvykle založené na strojovém učení. V počátcích se využívaly převážně klasifikátory s ruční extrakcí příznaků, jako jsou support vector machines či rozhodovací stromy. V poslední době je tento problém stále častěji řešen pomocí hlubokého učení s využitím konvolučních neuronových sítí (CNN), které předešlé metody v mnoha oblastech předčily. Z tohoto důvodu byla CNN vybrána jako klasifikátor pro tuto práci. Vytvoření takovéto sítě je velmi náročné, jelikož je plankton velmi rozmanitý v rámci jedné třídy, a současně jednotlivé třídy jsou si navzájem podobné. Velikost a poměr stran obrázků planktonu jsou také značně proměnlivé. Jednoduché klasifikační metody s CNN využívají škálování obrázků do jedné společné velikosti, například 224×224 pixelů, avšak tímto procesem je ztraceno množství malých detailů, které jsou důležité pro úspěšnost modelu. Dalším cílem práce je tedy zkoumání různých metod, které jsou zaměřené na vysokou diversitu velikosti obrázků.

V této práci byla nejprve vyhodnocena přesnost některých vybraných architektur převzatých z jiných prací zabývajících se klasifikací planktonu, spolu s několika běžně používanými architekturami jako jsou AlexNet, InceptionV3, DenseNet, MobileNet a další. Zde díky vysoké úspěšnosti a rychlosti byl pro další testování vybrán model Al-Barazanchi, pojmenovaný podle jednoho z autorů práce, ze které byl převzat. S touto architekturou byly dále zkoumány další metody, zaměřené na rozmanitou velikost obrázků. První taková metoda využila tzv. Spatial Pyramid Pooling (SPP) vrstvu, která byla vložena před plně propojené vrstvy, a tak síti umožnila přijímat obrázky o různé velikosti. Díky tomu byla zvýšena invariance vůči rozměrům vstupních dat a také bylo potlačeno přetrénování sítě. Všechny obrázky zde byly transformované do několika zvolených rozměrů a byla hledána optimální kombinace, která nejvíce zvýšila přesnost sítě. Další metoda spočívala v zahrnutí původní velikosti obrázku a času pořízení vzorku během trénování sítě ve formě metadat, která byla přidána na vstup modifikované architektury. Tato architektura obsahovala původní síť zpracovávající obrazová data, tedy například Al-Barazanchi, ke které byla paralelně připojena další část sítě ve formě plně propojených vrstev, zpracovávající metadata. Tyto dvě časti byly následně spojeny a zkoumány s různou úrovní interakce. Následující metoda využívala tzv. patch cropping. Obrázek zde byl přetočen do horizontální polohy a následně z něj byly podélně vyřezávána pole, která se vložila na vstup sítě. Díky tomuto přístupu nemusel být obrázek redukován do požadované velikosti jakožto celek a malé detaily tak byly zachovány. Dále byla studována nová architektura nazývaná DeepWriter, která se skládá ze dvou paralelních sítí. Z obrázku byla vyřezána dvojice navazujících polí a každé

pole bylo vyhodnoceno jednou sítí. Pomocí tohoto přístupu byla modelu dodána informace nesoucí vzájemnou polohu mezi jednotlivými poli. Poslední metoda pojmenovaná Multi-stream CNN zkombinovala více modelů dohromady. Každý model zde vyprodukoval jeden predikční vektor, a všechny tyto vektory byly následně zprůměrovány do jednoho výsledného predikčního vektoru. Tímto přístupem tak mohly být sloučeny různé modely, kde každý z nich byl zaměřen na obrázky s jinou velikostí či poměrem stran. Během testů byly vyhodnoceny kombinace předchozích metod, tedy například CNN přijímající celý obrázek jednoho vzorku a CNN využívající patch cropping.

Modely byly vyhodnoceny na části datasetu obsahující 32 tříd. Al-Barazanchi architektura dosáhla úspěšnosti 93,41 %, nejvyšší úspěšnost s 95,20 % měla InceptionV3. Při použití SPP vrstvy byla Al-Barazanchi vylepšena o 0,4 procentního bodu. V rámci zapojení metadat se přesnost původní Al-Barazanchi sítě zvýšila o 0,9 bodu. Během metody využívající patch cropping byl dosažen nárůst přesnosti až o 0,8 bodu pomocí architektury DeepWriter tvořené dvěma Al-Barazanchi sítěmi. Při aplikaci předešlých metod nebylo dosaženo dalšího zlepšení úspěšnosti pro architekturu InceptionV3. Při zkoumání poslední metody *Multi-stream CNN* bylo dosaženo úspěšnosti 94,99 % při kombinaci původní Al-Barazanchi architektury s její modifikací, která je zaměřená na obrázky s poměrem stran $2:1$ a modelem DeepWriter. Pro InceptionV3 architekturu bylo se stejnou kombinací, tedy originální InceptionV3, InceptionV3 trénovanou obrázky s poměrem stran $2:1$ a dále její modifikací v podobě architektury DeepWriter dosaženo úspěšnosti 96,16 %.

Cílem této práce bylo vytvořit CNN klasifikátor pro rozpoznávání planktonu, čehož bylo úspěšně docíleno vyhodnocením množství různých architektur. Dále byly studovány různé metody zaměřené na problém proměnlivé velikosti obrázků. Zde bylo aplikováno několik vybraných metod z jiných oblastí, s jejichž pomocí byla úspěšnost architektury dále vylepšena. V rámci provádění testů bylo dosaženo nejvyšší úspěšnosti 96,16 %.

Classification of varying-size plankton images with convolutional neural network

Declaration

I hereby declare that this Master's thesis was prepared as an original work by the author under the supervision of Prof. Pavel Zemčík and co-supervisors D.Sc. Tuomas Eerola, Prof. Lasse Lensu, Prof. Heikki Kälviäinen (LUT University, Finland). I have listed all the literary sources, publications and other sources, which were used during the preparation of this thesis.

.....
Jaroslav Bureš
June 29, 2020

Acknowledgements

I would like to thank my supervisors for leading my work, providing me with helpful consultations even in complicated times of a pandemic, supplying me with the latest literature containing new interesting approaches for image recognition and also for providing me access to computational resources that made the testing process much faster. I would further like to thank Prof. Pavel Zemčík, who arranged this opportunity of working on this interesting topic at a foreign university and guided me in the initial period before going to Finland. Finally, I would like to thank my family and my friends for giving me moral support.

Contents

Chapter 1

INTRODUCTION

Plankton are a diverse collection of organisms living in large bodies of water that are drifted by the current. They are an important part of the ecosystem as they provide food for many living creatures of all sizes from little fish to large whales. Apart from this, plankton is also the top producer of oxygen on Earth and can be used as a good indicator of the ocean health. Therefore, it is important to be able to recognize individual species in a sample of water. Manual classification of plankton tends to be very time consuming and laborious. For this reason, a demand for automatization of this process is rapidly increasing. The aim of this thesis is to create such a classifier that is capable of classifying plankton image data as accurate as possible.

With the continuous advancement in technology, many image recognition tasks can be accomplished through computer vision, which can be also applied in this field. Due to the enormous complexity of image data, these methods are usually based on machine learning. With this approach, a classification model is trained to learn features of individual species on a training dataset, where samples are already labeled with a matching class. Such a model is then able to classify new unknown samples. Many different feature-based approaches have been introduced in the past, such as support vector machines, decision trees or random forests. With these methods, features, describing for example sample size, shape and texture, are extracted from the images and supplied to these models. However, in recent years, deep learning with the use of convolutional neural networks (CNNs) proves to outperform these approaches almost in any area. Neural networks are inspired by biological processes that take place in a brain, as the image is processed by a structure of neurons. This approach brings the advantage that there is no need for hand-crafting feature filters as is the case of feature-based methods. For this reason, CNNs were selected as classifiers during this work.

Creating such an automatic classifier is a challenging task, as there are only small differences among individual species as well as a lot of diversity across a single class. Another major problem, that comes with plankton recognition is a huge variety in image sizes and aspect ratios. CNNs typically accept images of a single fixed size, for example 224×224 pixels. This is usually achieved by scaling and padding the image to the desired dimensions. However, with this process many little details are lost that are necessary to distinguish two very similar classes. This is especially a problem in case of species that have very long shape. Therefore, the next goal of this work was to explore possible approaches, that would improve the accuracy by addressing this issue. Proposed solutions were then evaluated on a challenging dataset of phytoplankton images.

This work is structured as follows: Chapter 2 focuses on plankton imaging, its taxonomy and related literature dealing with classification of plankton image data. Multiple hand-crafted feature extraction and deep learning approaches are described, that inspired this work. In Chapter 3 convolutional neural networks are described, how are they structured, how does the training process work together with some selected techniques for better generalization and why is the operation of convolution essential in computer vision. Furthermore, some of the commonly used networks are presented together with few libraries that can be used for the implementation. The following chapter talks about CNNs that are focused on images of varying sizes and aspect ratios, as this is one of the major issues when it comes to plankton recognition. Chapter 5 shows dataset used in this work and how it was processed together with multiple proposed solutions for creating classification models, that are later evaluated and obtained results are discussed. The work is finally concluded with Chapter 7.

Chapter 2

Plankton recognition

In this chapter, process of plankton imaging is discussed, followed by a description of plankton taxonomy with focus on phytoplankton. After that, few related works are presented that deal with plankton recognition. Here, application of both feature engineering-based and CNN-based methods are mentioned.

2.1 Plankton imaging

Aquatic imaging is essential for analyzing the marine ecosystem [44]. It makes it possible to examine what kind of particles are contained in water, how abundant they are and what size they are. In case of plankton it is important to identify individual species as it indicates health of the ocean. Some of them may be harmful to humans and the ecosystem and by monitoring scientists can get early-warning for possible coastal blooms and invasions.

Traditional ways consist of collecting a sample of water and examining it manually with a microscope [3]. This process tends to be very laborious, and an automatization of this process is needed. However, automatic imaging of plankton is also very challenging [44]. Devices used for sample collecting need to be very complex, as they must have advanced optical sensors for detecting particles of various sizes and in different environmental conditions. Image quality is highly affected by small differences in illumination, particles come in any possible orientation. Furthermore, they need to be capable of fast computation for processing and analyzing images, and they also need to have huge bandwidth for transmitting measured data to laboratories or possess sufficient storage. Many different techniques for collecting samples are being utilized, some of them function without disturbance of surrounding water, others pump the water into a view area (imaging-in-flow). In this section is installed special hardware for extracting plankton features by using unique techniques like light scattering and chlorophyll fluorescence that triggers a camera. Recently a low-powered holographic system capable of remote plankton sampling is being utilized. Many different commercial instruments are being used, such as Imaging Flow Cytobot (FCBI), FlowCam, FastCam (a prototype system), CytoSense and CytoSub [44]. Examples of plankton images captured by the FCBI can be seen in Figure 2.1.

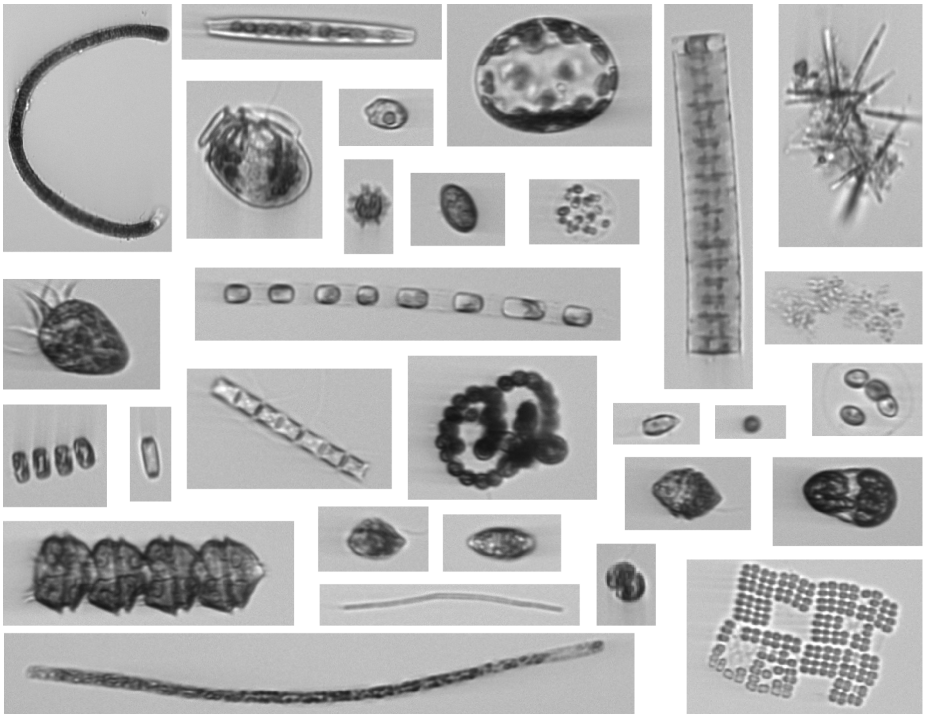


Figure 2.1: Example of plankton images with different sizes and aspect ratios.

2.2 Plankton taxonomy

Plankton is defined as organisms that are freely drifted by the water current [38]. Many different species meet this condition; however, it can be roughly divided into three groups: phytoplankton, zooplankton and bacterioplankton. Phytoplankton consists of microscopic organisms living on the surface of the ocean (up to 100 m depth) that are capable of photosynthesis. They are converting sunlight energy into food for other organisms and therefore are the crucial element in the food chain. Zooplankton is a very diverse group as it consists of many different species of varying sizes that feed on phytoplankton and bakterioplankton. It contains organisms of one cell size, small worms, larvae, jellyfish and also special kinds of fish. Some species even become zooplankton for only one stage of their life cycle. Last group is bacterioplankton, that can be found in any depth. It is essential for the phytoplankton as it creates nutrients by decomposing organic material.

This work is focused on the phytoplankton fraction only. Studying of phytoplankton diversity is still an active process as the taxonomy is constantly being changed with newly acquired data thanks to advancement in technology [45]. Till this day, lots of phylogenetic groups are waiting to be confirmed with evolutionary links. Apart from evolutionary division phytoplankton is categorized by other aspects, for example its role in the ecosystem, shape or size (e.g. nanoplankton (2-20 μm), microplankton (20-200 μm) etc.). By the end of the 1980s around 4 thousand different species of marine phytoplankton were formerly described [46]. Most common phylogenetic groups are diatoms, dinoflagellates and haptophytes being dominant in nanoplankton and microplankton [45]. These can be in some seasons seen in a form of a visible bloom in the ocean. Another major group is green algae, which dominates pikoplankton. However, most of the phytoplankton species, especially in

the pikoplankton section, are still not described due to their high similarity and lack of taxonomists.

2.3 Feature engineering-based plankton recognition

Before convolutional neural networks became widely used, many computer vision problems were dealt with by feature-based methods [11]. Feature vectors are extracted from images, describing for example their shape, area of the particle, length, width, color histogram and many others. These obtained values are then supplied to feature-based classifiers, such as support vector machines (SVM) [10], random forests [43] or k-nearest neighbors [43]. This section presents brief description of how these models were utilized in the plankton recognition.

Gloria Bueno et al. [6] examined different handcrafted feature approaches for automatic classification of diatoms, one of the most common plankton species in the ocean. Dataset consisted of samples collected in Spain, that were divided into 80 classes, where each one had about 100 samples on average. These samples were augmented in a way, that every class contained precisely 300 samples by applying rotation with step of 90 degrees together with horizontal and vertical flipping. For each sample up to 1460 features were computed. Some of them were describing particle's morphological features like its area, perimeter or shape. Others were focused on image histogram or Local Binary Patterns (LBP [33]), that represent textural properties. Because of the enormous dimensionality, features were reduced by removing redundant values based on correlation, after this only 273 features remained. Classifiers examined in this work include SVM, k-nearest neighbors, k-means and random forest. Training was performed using 10-fold cross-validation. The best result was obtained by using all types of features combined with random forest classifier, this way accuracy of 98.11 % was achieved.

Thi-Thu-Hong Phan et al. [37] studied classification of phytoplankton from Eastern Channel with different feature types and classifiers. The used dataset consisted of seven different classes, each with 100 samples labeled by a biologist. Every sample contained eight signals from a flow cytometer, describing its length, internal structure, chlorophyll pigment and other. These signals were transformed into multiple sets of features, e.g. by taking every signal's length, height, number of peaks and integral, 32 features were created for each particle. For the classification multiple different models were evaluated, for example, k-nearest neighbors, support vector machines and random forest. Classifiers were then evaluated with the use of four fold cross-validation, which was performed ten times. Random forest proved to have the best accuracy in all performed tests with a score of 98.24 %.

2.4 CNN-based plankton recognition

Carlos Sánchez et al. [41] experimented in their work with application of multiple pretrained commonly used networks for plankton classification. A dataset of 14 classes was used, which contained 1085 images in total. Low number of samples per class was handled by data augmentation – images were flipped both horizontally and vertically, then rotation transformation was applied with an angle between 0° and 90°. Here padding color was remained to be plain black. Input data were normalized by subtracting mean value from every image and dividing the result by standard deviation. Finally, images were resized

to 224 × 224 which is required by the networks; however, the aspect ratio was not kept. The dataset was split into three parts, that were 80 % for training, 10 % for validation and 10 % for testing, this approach was applied ten times following cross-validation procedure. Examined were CNNs, that were initialized with random weights, together with CNNs pretrained on ImageNet dataset, where only fully connected layers were fine-tuned. Some of the tested architectures were AlexNet [27], ResNet18 [16], VGG11 [25], SqueezeNet [21] or DenseNet [20]. From fine-tuned CNNs the highest accuracy of 99.07 % was achieved with the DenseNet, from non-pretrained CNNs best SqueezeNet produced the best result of 93.52 %.

Jiangpeng Yan, Xiu Li and Zuoying Cui [52] tried to find a more efficient model for plankton classification by exploring different network architectures. Among these networks were the CaffeNet, VggNet-19 and ResNet with varying number of layers. The dataset consisted of 121 different classes containing 30 336 grayscale images in total and it was used in a science competition called National Data Science Bowl. Samples were resized to a common size of 256 × 256 based on the length of the longer side and augmented by rotation with an angle of 0°, 90°, 180° and 270°. Networks were trained using the SGD method with learning rate of 0.001, momentum 0.9 and weight decay 0.0005. Models were pretrained with the ILSVRC2012 dataset containing 1000 classes, then fine-tuned on the plankton dataset to improve the results. While examining Top-1 and Top-5 accuracies of the CNNs it was discovered that deeper networks tended to have worse performance as a simple CaffeNet outperformed most of the others. Thanks to lower number of CaffeNet's parameters it also takes less storage space and runs faster. The best accuracy of 78.5 % was achieved with Res-19 architecture.

Hussein Al-Barazanchi et al. [2] presented an intelligent classification system that outperformed state-of-the-art approaches given a dataset called SIPPER. This dataset came from the University of South Florida and contained more than 750 thousand samples of 81 classes from the Gulf of Mexico. Images were of low quality, each image had only 3-bits resolution and the classes were highly imbalanced. In the first phase, only seven classes were used for tuning of the network (Sipper-7), after that subsets Sipper-52 and Sipper-77 classes were examined. Each subset was split into 70 % for training, 15 % validation and 15 % testing. Images were resized to 256 × 256 without regard to aspect ratio. CNN used in this work was based on the guides of VGG Net of Simonyan et al. (2014) and consisted of five convolution layers with a kernel size of 3 × 3, where each one was followed by a pooling layer and the whole network was completed with three fully connected layers. Dropout layers with probability of 20 % were inserted to each hidden fully connected layer. Training was done with 150 epochs; no augmentation was used. For Sipper-7 testing accuracy of 98.20 % was achieved, 81.79 % for Sipper-52 and 80.54 % for Sipper-77.

Iago Correa et al. [9] proposed a deep learning technique for recognition of microalgae. Dataset contained 29 000 samples extracted from the South Atlantic Ocean by a FlowCAM particle analyzer, that were divided into 24 classes from which only 19 classes were used, as the remaining five classes consisted of less than 10 samples. This dataset was divided to 70 % for the training and remaining 30 % for validation. Images were resized to 64 × 64 with preserving the aspect ratio and normalized by subtracting mean value and dividing the result by standard deviation. Four types of augmentation were used – rotation with an angle between -45 and 45 degrees, flipping both horizontal and vertical, cropping 30 × 30 to 40 × 40 patches and Gaussian noise addition with scale from 0.01 to 0.1. Multiple architectures were examined, final one consisted of five convolutional layers interlaid with three max-pooling layers and it was finished with three fully connected layers. First convolution layer had a

kernel of 7×7, other ones 3×3 and max-pooling layers 2×2. The dropout layer was also used to reduce the over-fitting. The best achieved validation accuracy was 88.59 %.

Chapter 3

Convolutional neural networks

Manual plankton classification is a very laborious process. Even for an experienced expert it takes a lot of time to label individual image samples due to lack of distinguishable features across many species. With recent advance in hardware and software for machine learning, deep learning methods are becoming more popular as they outperform traditional feature-based algorithms. Convolutional neural network (CNN) is today one of the most commonly used classifiers for image detection and classification. This section describes basic principles of CNNs restricted to the scope of this work, such as their structure, training process, and few selected architectures and libraries for machine learning.

3.1 Neuron in neural networks

The human brain is a very powerful processing unit created by nature [15]. By learning during its lifelong process, it can adapt itself to perform enormous variety of tasks. It is a universal unit that can be used for solving any sort of problem with huge complexity. That is the reason why scientists desired to create an artificial model of such a computational unit that would have similar capabilities. Our brain consists of briefly 100 billion nerve cells, also known as neurons. One neuron consists of a body called *soma* and many branches through which it communicates with other neurons via sending electrical signals. These branches are interconnected by synapses, which form electrochemical junctions. Each cell typically has thousands of input connections called dendrites through which it receives signals from other neurons. The signals are in a form of series of spikes, that are composing a certain value. If the sum of these signals exceeds a certain level, a new signal is generated and sent to following neurons by an output fiber named *axon*. Some of the signals may have excitatory influence, other ones have inhibitory effect that prevents the new signal to be produced. The level of influence of input signals is modulated by the strength of individual synapses. The structure of a neuron is shown in Figure 3.1.

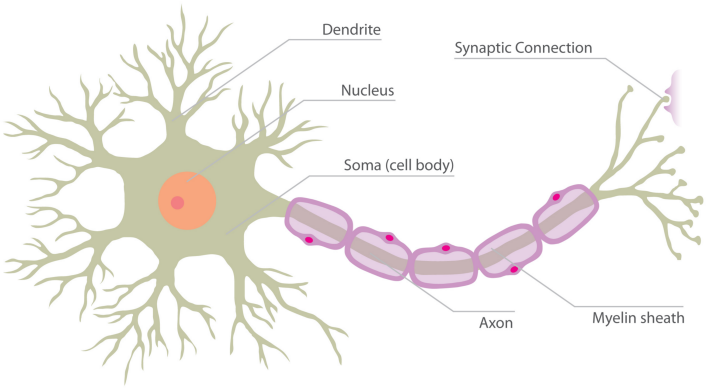


Figure 3.1: Structure of a single neuron [12].

Based on these findings an artificial node was created [15] that would perform similar function as shown in Figure 3.2. In this model dendrites are substituted by weighted connections. Signal processing of the model is defined as follows: First a weighted sum is performed – transposed input vector $x \in R^d$ is multiplied by a weight vector $w \in R^d$. An intercept term called bias is added to this value. Finally, the result is supplied into an activation function $\sigma(x) : R \to R$ that adds a non-linear transformation to the system. This activation function acts as a threshold function of a biological neuron and can be defined as:

$$f(x) = \sigma(w \cdot x^T + b). \tag{3.1}$$

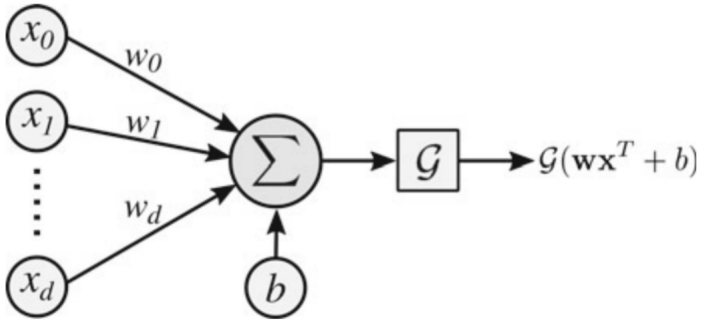


Figure 3.2: Model of a neuron [1].

3.2 Feedforward neural network

By connecting several of these neurons, a system is created forming a neural network [1]. With a larger number of neurons, the network can solve more complex tasks, whereas one neuron can only find a solution to linearly separable problems. There are many ways how neurons can be connected which affects how the network will behave. Commonly used structure used for computer vision is called feedforward neural network which is also a base shape for convolutional neural networks, therefore following text will revolve around these. With this structure the network is composed of one or more layers where each layer contains at least one neuron. Number of neurons may differ from layer to layer. These

layers are connected in a one-way form – input vector x is supplied to an input layer of the network, then it is sequentially processed by hidden layers and finally the result is given by an output layer. Each neuron in a hidden layer or the output layer is connected to every neuron from a previous layer and may have its own activation function. However, neurons in one layer have typically the same activation function. A network which has multiple hidden layers is called a deep neural network. An example of a feedforward network can be seen in Figure 3.3.

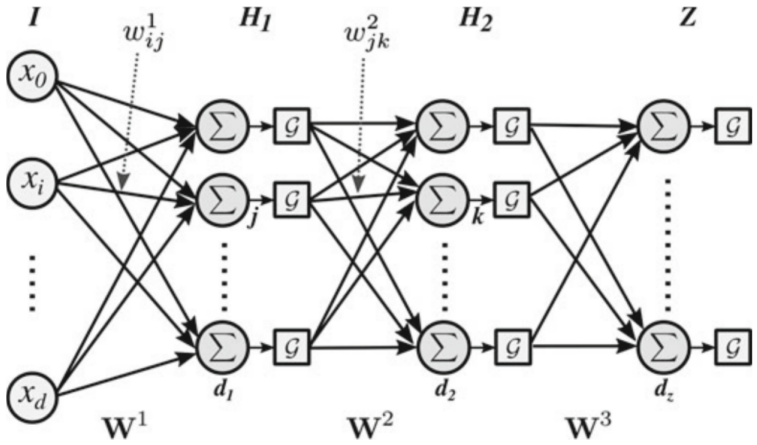


Figure 3.3: Example of a feedforward neural network [1].

A feedforward neural network can act as a universal approximator for any continuous function [1]. When designing such a network it is important to find optimal hyperparameters for a given task – that is the number of hidden layers, number of neurons in each layer and the type of activation functions for every layer.

3.3 Neural network training

Machine learning comes in a form of three types [24], that are supervised learning, unsupervised learning and reinforced learning. For classification the most commonly used method is the supervised learning, where learning is accomplished with a training dataset, where each input value is labeled with corresponding desired output. Predictions of the network should be as close to those labels as possible.

Training of a neural network [31] lies in finding weights and biases so that the output of network approximates $\hat{y}$ for every input x. Difference between predicted and target output is defined by a cost function (cost function can be also called a loss function or an objective function). This function can have many forms, one of the most commonly used is named the mean squared error:

$$C(w,b) = \frac{1}{2n}\sum_x ||y(x) - a||^2. \tag{3.2}$$

where w are weights of the network, b are the biases, n is the number of inputs and a is a vector of network outputs given input x. In other words, aim of the training algorithm is to minimize this cost function by adjusting values of weights and biases. If result of the cost function would equal to zero, the created model would perfectly classify training

samples. With increasing value of this function, the accuracy of the model gets worse. Using calculus solution for this problem is not feasible due to the high number of variables, as some networks may have billions of parameters.

This task can be however accomplished with an algorithm called gradient descent [31]. Gradient descent is a numerical method for finding the local minimum of a cost function. With a randomly picked initial point the algorithm iteratively updates its position by moving in a direction of the negative gradient of the actual point. Stopping condition can be set to reaching maximum number of iterations or checking if the change of position between two steps is less than a given threshold. Size of one step can be also adjusted by a variable η called the learning rate. If this value is too small then gradient descent algorithm takes a very long time, if η is too big then it is not guaranteed that the minimum is found. The update of a position can be expressed as follows:

$$v' = v - \eta \nabla C. \tag{3.3}$$

where v is a vector of variables and ∇C is the derivative of the cost function by v. This solution is however not very feasible for large datasets, as the cost function would have to be calculated with every sample from the training set and only after that could be networks parameters updated for the next iteration. Training with this algorithm would take too much time. For this reason, an adjusted method called Stochastic gradient descent (SGD) was created [31]. With this method the dataset is split into several batches with randomly selected samples. The training is then accomplished by processing only one batch at a time, calculating the cost function for this batch and finally updating parameters with derived gradients. This process is ongoing until the whole training dataset is depleted and therefore one epoch is finished. Stochastic gradient descent continues training with next epochs until a stopping condition is met, which can be limitation by maximum number of epochs or reaching a predefined network accuracy. The accuracy of a network can be simply expressed as a portion of samples that were classified correctly in the dataset as follows [1]:

$$Acc = \frac{1}{N} \sum_{i=1}^{N} 1[y_i == y_i']. \tag{3.4}$$

where N is the number of samples, y_i is the true class and y_i' is the predicted class. Notation $1[.]$ returns 1 for $True$ and 0 for $False$.

Another challenge that appeared with neural network training was the difficult calculation of the gradient with respect to network parameters [31]. Earlier approaches used to be very slow due to a high computation complexity. In the 1970s a new method was introduced called the backpropagation algorithm, which is till this day the key training element due to its effectiveness. Thanks to the boost in the learning speed, neural networks could be used in many new ways where it was previously impossible. This algorithm is based on a chain rule, which is an effective way for computing partial derivations of complex functions. It consists of multiple phases. The first phase is called the forward pass and in this phase the input pattern is applied to the input layer of the network and then output vectors for each layer are gradually computed with Equation 3.1 until the output layer is reached. At this moment the value of the cost function δ is calculated. In the next phase called the backward pass for each layer starting with the final layer the error is propagated backward to the starting layer with formula:

$$\delta^{x,l} = ((w^{l+1})^T \delta^{x,l+1}) \circ \sigma'(z^{x,l}). \tag{3.5}$$

where $(w^{l+1})^T$ is the transpose of the weight vector of $(l+1)^{th}$ layer, δ is the propagated error vector, σ is the activation function and z is the weighted input of neurons. Symbol $\circ$ in the equation represents the elementwise product of the two vectors, which is also called the Hadamard product. Finally, weights w and biases b for every layer are adjusted as:

$$\begin{aligned} w^l &= w^l - \frac{\eta}{m} \sum_x \delta^{x,l} (a^{x,l-1})^T. \\ b^l &= b^l - \frac{\eta}{m} \sum_x \delta^{x,l}. \end{aligned} \tag{3.6}$$

where η is the learning rate, m is the size of the batch and $a^{x,l-1}$ is the activation vector of neurons in $(l-1)^{th}$ layer for input x.

3.4 Over-fitting in neural network training

Training of the neural network is not an easy task, as it comes with multiple obstacles [4]. If the network is not trained enough with a given training dataset, the model underfits and it has low accuracy on both training and testing sets. On the contrary, when the model is trained too much, it performs with high accuracy on training data, however it fails to generalize and performs poorly on unseen data in the testing dataset. This problem is called overfitting and is depicted in Figure 3.4. One solution for this problem would be a systematical training of the model with different numbers of epochs. However, this would be very computationally expensive as the model would have to be trained and discarded many times. Better solution is by early stopping. This method is commonly used as the model needs to be trained only once thanks to monitoring the performance by testing the accuracy at the end of each epoch. If the accuracy for validation set does not improve after several epochs (as the training process can be noisy) the training process is stopped.

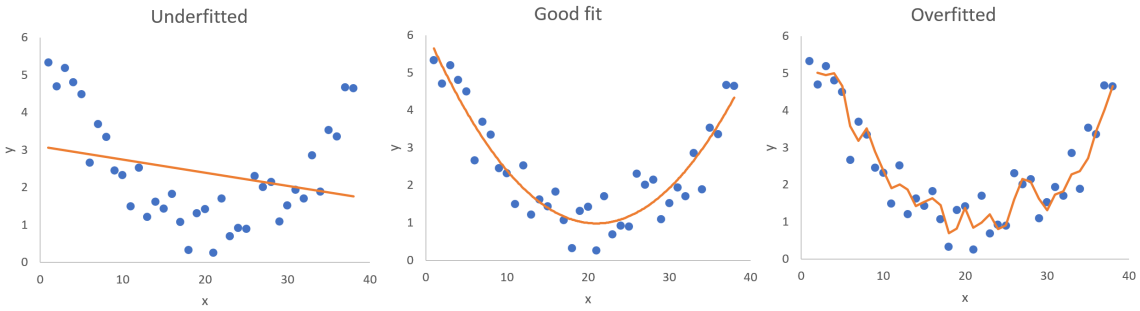


Figure 3.4: Example of underfitting and overfitting of function.

Another way how to prevent overfitting is by data augmentation. It is a very simple way how to extend the training dataset by applying small transformations to the original samples. This operation can be performed during the actual training of the model so that transformed images do not have to be stored on a disk. Augmentation can also be computationally free as the process can be performed on a CPU while training of the model can run on a GPU [26]. Images can be transformed in many ways. Just by flipping the image horizontally training dataset can be twice as large. Different transformations can be combined to create the final sample. Other transformations can be done by shifting, cropping, rotating, adding a noise, adjusting hue and saturation and many others. Examples of these

operations can be seen in Figure 3.5. It is important to choose only those transformations that are relevant for the actual dataset [5].

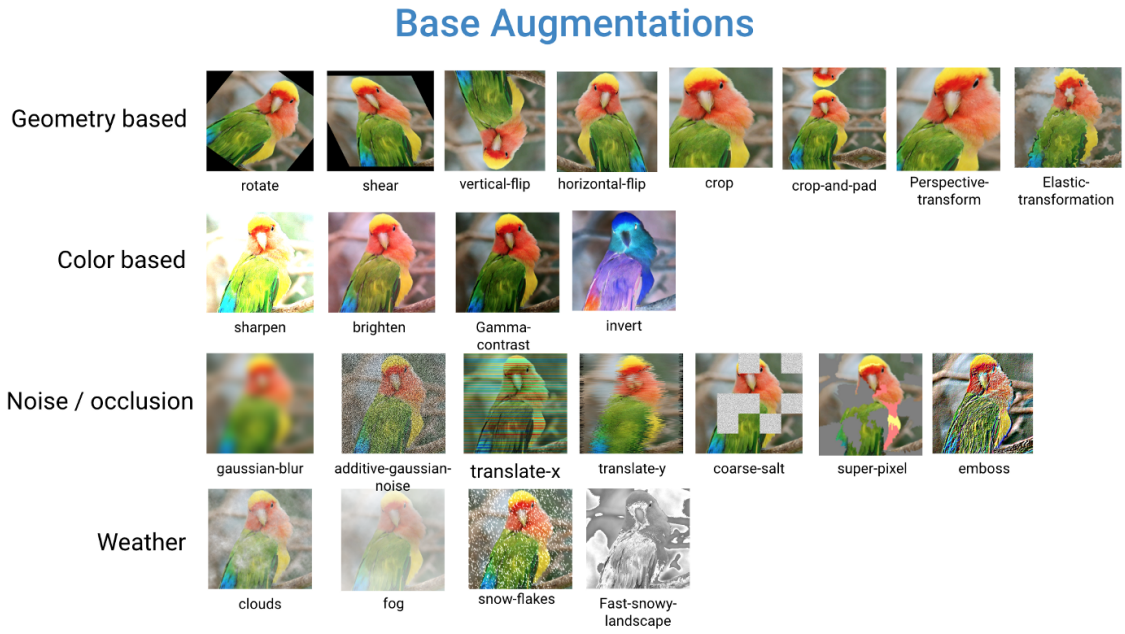


Figure 3.5: Example of different data augmentation [55].

Next popular technique for overfitting prevention is called dropout [1]. With this method some neurons are disabled during the training with a certain probability for every training sample, which means that their output is set to zero and they do not participate in the forward nor the backward propagation. This causes that the network architecture changes for every single input even though weights are still shared among them.

3.5 Activation function

As it was mentioned earlier in this work, activation functions [1] are an important part in network's processing, as they add non-linearity to the computing. With this feature the network is able to learn any nonlinear function as long as enough neurons and layers are provided. Another important property of the function is its differentiability, as is desired by the backpropagation method. There are many kinds of functions that can be utilized for this purpose, next section presents a selection of those functions that are commonly used.

Sigmoid

A sigmoid function [1] (see Figure 3.6) is a biologically inspired activation function and it was commonly used in feedforward neural networks in the past. Its original and derivative forms are specified by following equations:

$$f_{\text{sigmoid}}(x) = \frac{1}{1+e^{-x}}. \tag{3.7}$$

$$f'_{\text{sigmoid}}(x) = f(x)(1-f(x)). \tag{3.8}$$

It was observed that this function is not ideal for neural networks, as it saturates for x shifting away from zero and therefore the gradient becomes very small. This is quite problematic when it comes to backpropagation, as the gradient becomes even smaller and smaller moving back to the input layer until it completely disappears. This problem is known as the vanishing gradient problem, and it is especially serious in deep networks, where changes in weights become negligible and the network fails to learn.

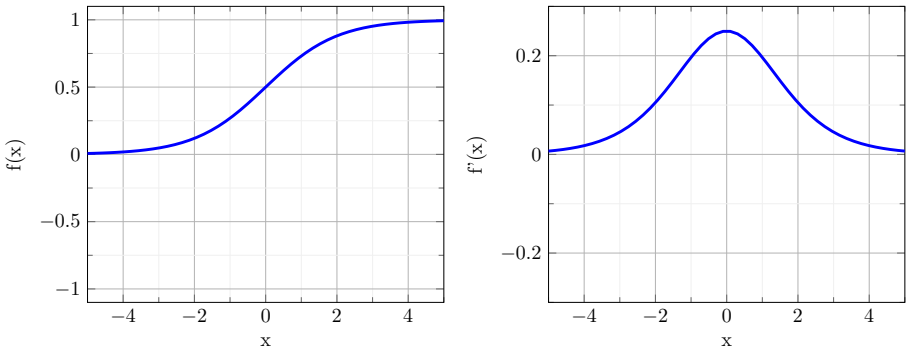


Figure 3.6: Sigmoid function (left) and its derivative (right).

Hyperbolic Tangent

Next function which can be often seen in practice is named hyperbolic tangent [1] which is very similar to the sigmoid function. It is defined by following equations:

$$f(x) = \frac{2}{1 + e^{-2x}} - 1. \tag{3.9}$$

$$f'(x) = 1 - f(x)^2. \tag{3.10}$$

As it can be seen in Figure 3.7, its range lies in $[-1; 1]$, as opposed to the range of the sigmoid function which is $[0; 1]$. This provides the advantage of approximating the identity function close to the origin, which speeds up the learning speed of a neural network by faster convergence of gradients. However, problem of the sigmoid function remains as it saturates for larger $|x|$ and the vanishing gradient problem may still occur.

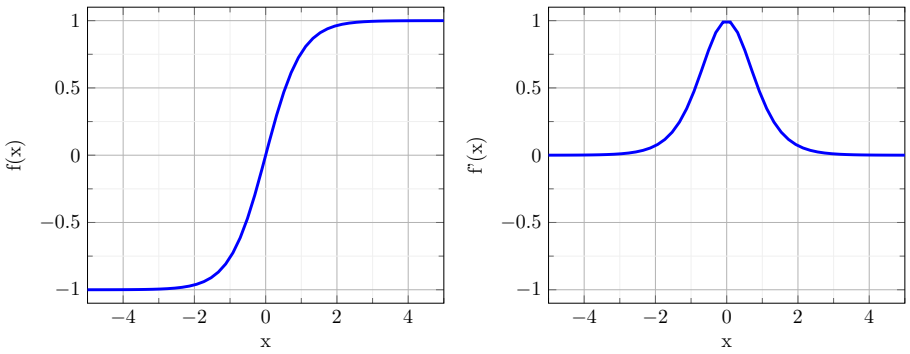


Figure 3.7: Hyperbolic tangent function (left) and its derivative (right).

Rectified Linear Unit

The most popular activation function is the Rectified Linear Unit or ReLU in short [1]. It's definition and derivative are:

$$f_{\text{ReLU}}(x) = \max(0, x). \tag{3.11}$$

$$f'_{\text{ReLU}}(x) = \begin{cases} 0, & x < 0 \\ 1, & x \geq 0 \end{cases} \tag{3.12}$$

It is very popular as it has a constant derivative for $R+$ values and therefore there is no saturation for positive x – it always provides strong gradients for these values. For this reason, it can be used in deep networks with a large number of layers. One feature of this function is that it produces dead neurons during training. Dead neurons are nodes that have adjusted their weights so that the multiplication of a weight vector w and an input sample vector x is a negative number and the output of this function is always zero. If the output of this neuron is equal to zero, it does not affect any other neuron and therefore it can be removed from the network, which increases computational efficiency.

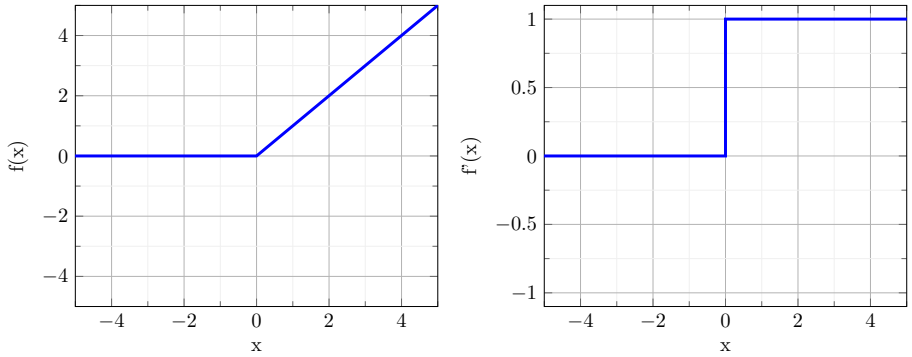


Figure 3.8: Rectified Linear Unit function (left) and its derivative (right).

Leaky Rectified Linear Unit

To solve saturation for negative values of x and to get rid of dead neurons, previous function was modified and Leaky ReLU [1] was created. In this function the constant derivative equal to zero for negative numbers was substituted with a small negative value. Equations of this function and its derivative are as follows:

$$f_{\text{LReLU}}(x) = \begin{cases} \alpha x, & x < 0 \\ x, & x \geq 0 \end{cases} \tag{3.13}$$

$$f'_{\text{LReLU}}(x) = \begin{cases} \alpha, & x < 0 \\ 1, & x \geq 0 \end{cases} \tag{3.14}$$

where α is usually a small value in range $[0, 1]$, typically $\alpha = 0.01$ [1].

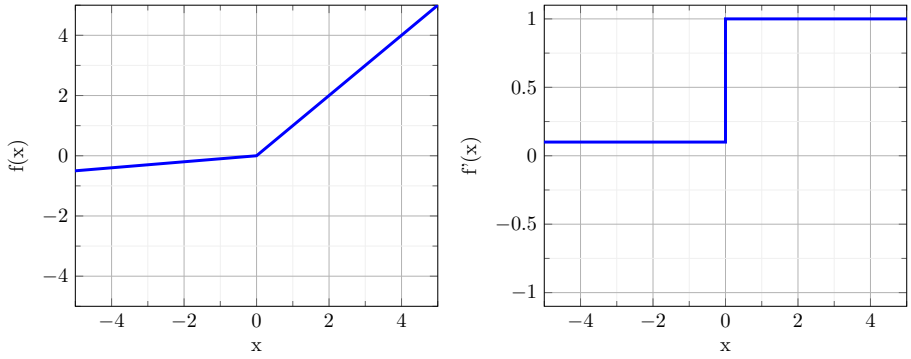


Figure 3.9: Leaky Rectified Linear Unit function (left) and its derivative (right).

3.6 Convolutional neural network

Fully connected neural networks are not ideal for image processing due to a large data complexity [1]. Such a network, which would be able to train generalization of many image classes, would need to have an enormous number of neurons and therefore it would be very computationally expensive. This is a reason why the data volume needs to be reduced to allow fewer parameters to be used. Image data have a high level of correlation between neighbor cells. If one pixel in the picture is, for example, red, there is a big chance that other pixels around it will also be red or have a similar color. By merging several pixels from one area into a common value, the dimension of data can be rapidly reduced with minimum of information lost. This is where convolutional neural networks come in. A convolutional neural network (or CNN in short) is a special type of neural network which uses a specialized kind of mathematical linear operation called convolution for processing data with grid-like topology. This is widely used for dealing with time-series data in a form of a 1-D array or image data stored as a 2-D pixel array.

Convolution

Operation of a convolution [14] on functions x and w can be denoted by the following equation:

$$s(t) = (x * w)(t). \tag{3.15}$$

or for discrete parameter t it will have a form of:

$$s(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a). \tag{3.16}$$

where x is the input and w is the kernel. The output of the convolution is then called a feature map. For image processing this function is modified to process two-dimensional images and is called cross-correlation (however, it is still often referred to as a convolution operation). This equation has the following form:

$$s(m,n) = \sum_{i=0}^{P-1}\sum_{j=0}^{Q-1} X(m+i, n+j)f(i,j). \tag{3.17}$$

where P and Q are the kernel width and height, m and n is the position of the actual pixel. This operation can be viewed as a simple matrix multiplication as the kernel moves across

the image by setting values of m and n. Size of the kernel is typically much smaller than the size of the image.

Movement of the kernel can be also modified by a parameter called *stride* [1], which defines the domain of variables m and n in the above equation. If for example $stride = 1$, then $m = 0, 1, 2...W-1$ and $n = 0, 1, 2...H-1$, where W is the width of image and H is the height and the equation is computed for every cell in I. If $stride = 2$, then $i = 0, 2, 4...W-1$ and $j = 0, 2, 4...H-1$, convolution is done for every even pixel and output image is reduced by half of the original size. Size of the stride is rarely greater than 3. The actual size of the output given stride s is equal to:

$$size = \frac{W-P}{s} + 1 \times \frac{H-Q}{s} + 1. \tag{3.18}$$

As it can be seen in the above equation, even with $stride = 1$ the output size is reduced by the size of the kernel. This is due to border cells of an input X for which the convolution cannot be evaluated, because they are missing neighbor pixels, as it can be viewed in Figure 3.10. To keep the original size, the input image can be padded with a border of given width that adds new pixels around the original image. This can be done in multiple ways, for example *zero-padding* adds a border of cells with zero values.

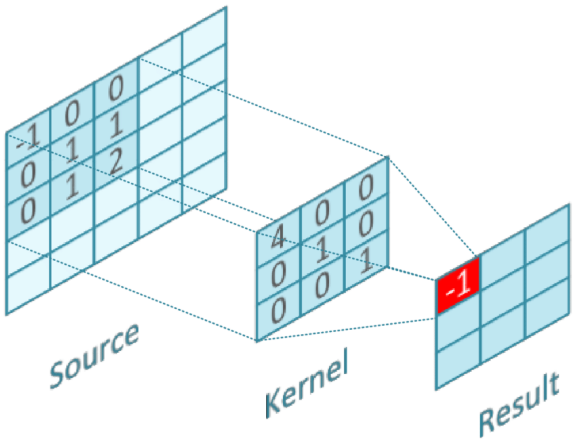


Figure 3.10: Example of image convolution [50].

The key role of convolution is to extract local features like edges, color or gradient orientation [1]. That is done by convolving the image with multiple filters, where each one learns to detect a different feature. With each layer the network can learn more and more complex traits.

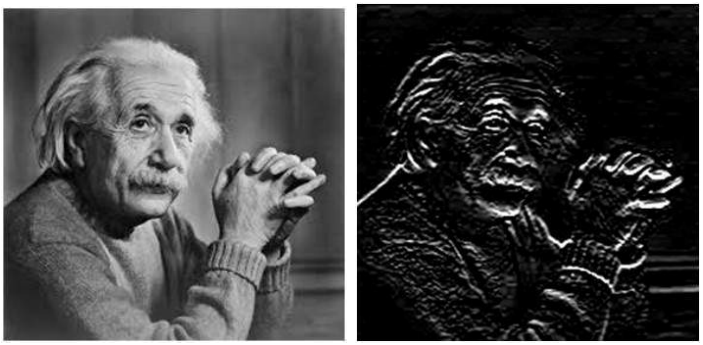


Figure 3.11: Result of applying Sobel filter used for edge detection [49].

Pooling

Pooling [1] has a major role in reducing the dimensionality of feature maps. For this reason it is often referred to as *downsampling*. Similarly, as in case of the convolution, a window with size of $w \times h$ is sliding across the whole image and a statistical operation is performed above the current area. The stride of the window influences size of the output and therefore stride is also called a *downsampling factor*. Regions of pooling can overlap each other. Commonly used method is *max-pooling*, where from each area a maximum value is picked or *average-pooling*, where an average value is calculated from a given region. However, it should be noted that max-pooling usually gives better results and is commonly used with middle layers. Another advantage of pooling is that it helps the input to be more invariant to translations by a small amount. Therefore, the fact that a certain feature is present in the input is more important than its exact pixel-accurate location.

3.7 Typical structure of a CNN

Building of a CNN is a very challenging task [1]. There is no golden rule by which to proceed as individual datasets differ with observed features and therefore unique structure of a network is needed. However, there are certain rules that come from time-proven existing solution. Many successful CNNs start with multiple convolutional layers mixed together with pooling layers. Networks then proceed with several fully connected layers and are finished with an output layer. Convolution layers usually use convolution filters of sizes 3×3, 5×5 and 7×7. Activation functions are typically modifications of ReLU function due to their superiority in deep networks. Finally, number of parameters of the network should not be too large for a given dataset as it would fail to generalize.

LeNet

LeNet [29] was introduced in the 1990's for handwritten character recognition. It consists of two convolutional and two pooling layers followed by a flattening layer and two fully connected layers. The output of this network is handled with a softmax function. Its detailed structure can be viewed in Figure 3.12. The convolution kernel has $size = 5$ and an average pooling with $stride = 2$ is used for pooling layers. Activation function was chosen to be the hyperbolic tangent. This network was fed with grayscale images of size 32×32 which were classified into ten different classes representing ten digits from 0 to 9. This network is a popular example for learning CNNs due to its simplicity.

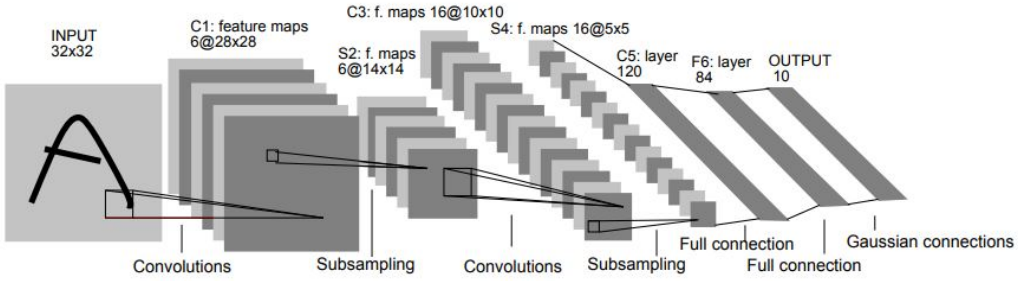


Figure 3.12: Structure of LeNet [29].

AlexNet

AlexNet [27] was named after Alex Krizhevsky, who created this CNN in 2012. It won in ILSVRC 2012 (an annual contest by ImageNet project called Large Scale Visual Recognition Challenge [40]), as it had a Top-5 error rate of 15.3 %, whereas the next best model had a much larger error rate of 26.2 %. In this competition the network had to learn to classify images into 1 000 different classes (this was a subset of ImageNet with a collection of 15 million labeled high-resolution images from 22 000 categories). AlexNet has up to 60 million parameters and 650 000 neurons. It takes RGB images of 256×256 pixels. The structure starts with 5 convolutional layers interpolated by three max-pooling layers. The convolution kernel appears in multiple sizes, starting from 11×11 shrinking to size of 3×3. The network is finished with three fully connected layers where the last one is a softmax classifier. ReLU was chosen for the activation function. The complete structure can be seen in Figure 3.13.

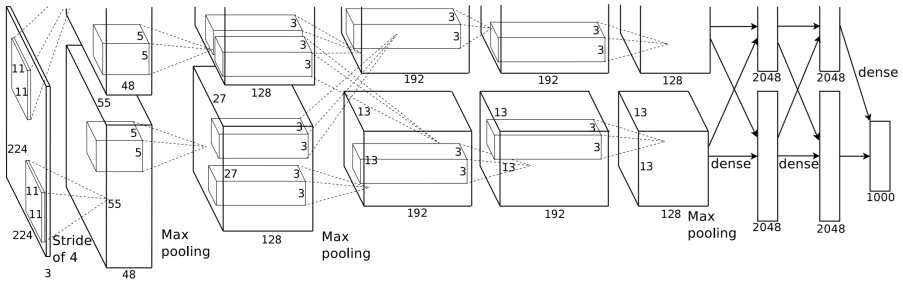


Figure 3.13: AlexNet architecture [27].

InceptionV1

InceptionV1 [48], also known as GoogLeNet won ILSVRC in 2014. This work achieved another large improvement in the accuracy with a Top-5 error of only 6.67 %. Success of this network relied on huge depth of 22 layers, even though it has only about 4 million parameters, which is a small number in comparison with AlexNet. It consists of several convolutional, max-pooling and average pooling layers. It also uses a new component called inception module, which consists of several convolutions, and poolings with different kernel sizes done in parallel, as can be viewed in Figure 3.14. Finally, this network uses convolutions with size of 1×1 that reduces dimensionality of channels to increase the computation speed.

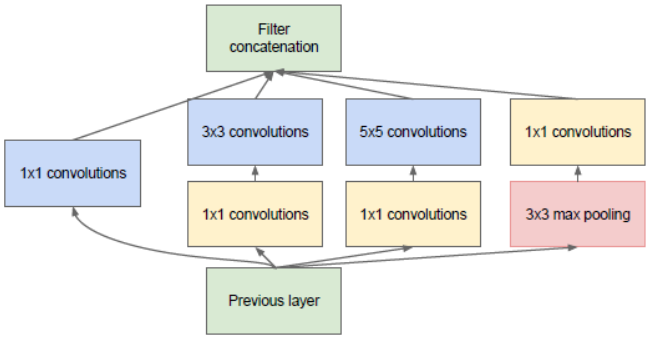


Figure 3.14: Inception module with dimensionality reduction [48].

MobileNet

MobileNets [19] are efficient models designed to match resource restriction, such as latency or size. Standard convolution is replaced by depthwise separatable filters, which perform same the operation, but in two separate layers. The first executes the depthwise convolution, as it applies a single filter to each channel. The second layer performs a pointwise convolution, which is a 1x1 convolution, that combines outputs of the depthwise convolution. This filter is illustrated in Figure 3.15.

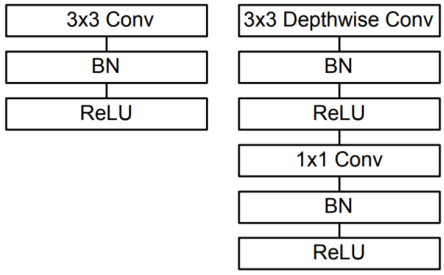


Figure 3.15: Left: Standard convolutional layer with batchnorm and ReLU. Right: Depthwise Separable convolutions with Depthwise and Pointwise layers followed by batchnorm and ReLU [19].

This way computational time in initial layers is reduced together with the model size. MobileNet architecture has 28 layers. The first one is a classic full convolution, that is followed by pairs of depthwise and pointwise convolutions. Both depthwise and pointwise convolutions are followed by a *Batch Normalization* [22] ensuring non-zero propagated signals and the ReLU activation function. Increased step size is applied to selected depthwise convolutions for down-sampling. Finally, the network is finished with a global average pooling and one fully connected layer followed by a softmax classifier.

ResNet

Residual networks (ResNets) [16] won ILSVRC in 2015 with 3.57 % error rate on the ImageNet test set. Multiple architectures inspired by VGG nets with different depths were evaluated. These models consist of a selected number of convolutional layers with a different stride for down-sampling, followed by a layer performing global average pooling and

the network is finished with one fully connected layer and a softmax classifier. The key feature of this network is using shortcut connections, that skip several layers, as can be seen in Figure 3.16. This way, the problem of vanishing gradient is reduced while the number of parameters is not increased, neither is the computational complexity. By utilizing these shortcuts, the accuracy of a 34-layer long architecture was improved from 28.54 % to 24.19 %.

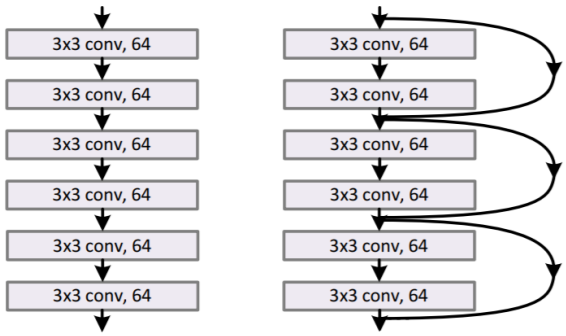


Figure 3.16: Left: Standard connection of convolutional layers. Right: Utilization of shortcut connections [16].

DenseNet

Densely Connected Convolutional Network [20] (or DenseNet in short) also leverages shortcut connections among layers similarly as the *ResNet*. In this case each layer is connected to every other layer, that is further in the network with a feedforward fashion. Therefore, each layer receives feature maps from every preceding layer. This method reduces the problem of vanishing gradient, provides stronger feature propagation and feature reuse. However, the concatenation operation requires single size of feature maps, therefore down-sampling cannot be applied during a shortcut. For this reason, shortcuts are restricted only to dense blocks, that are interconnected with transitions. In each dense block size of the feature map is constant, pooling is performed only in the transitions, as is illustrated in Figure 3.17. With this structure, DenseNet outperformed ResNet, as it achieved better accuracy with fewer parameters.

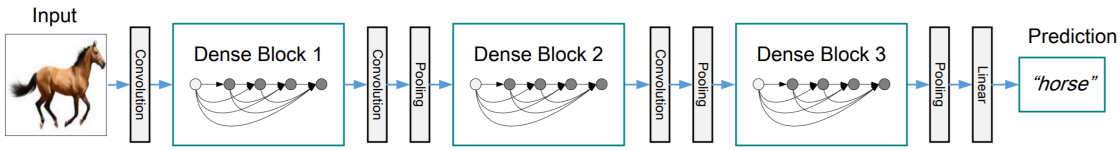


Figure 3.17: A deep DenseNet with three dense blocks [20].

3.8 Libraries for machine learning

There are many libraries for many different programming languages when it comes to image processing. This work is focused on classification of image data with convolutional neural networks, therefore following libraries are picked by these criteria.

TensorFlow

TensorFlow [39] is an open-source library for numerical computation and machine learning developed by the Google Brain team in 2015. TensorFlow provides a high level of abstraction with a Python API for building models from predefined pieces and executes operations in high-performance C++ language. It offers many machine learning models and algorithms for acquiring data, training of models, serving predictions and other. Processing of Tensorflow is based on dataflow graphs – user only defines how will the data move through individual processing nodes. Each node represents one mathematical operation and processed data has the form of a tensor. This library also comes with TensorBoard which is an inspection tool designed as an interactive dashboard for debugging and visualization of processes.

Keras

Keras [7] is an open-source library written in Python for fast implementation of neural networks. It provides a high-level API designed for human beings, with the aim at modularity, extensibility and user-friendliness. All necessary components for building a model like neural layers, activation functions or optimizers are stand-alone modules. Keras does not provide any low-level computation operations, as it is supposed to run on top of TensorFlow, CNTK or Theano. In case of TensorFlow, Keras was earlier used as a stand-alone package and from TensorFlow 2.0 it became its primary API.

PyTorch

PyTorch [36], developed by Facebook's AI research team, is another neural network programming package written in Python for building of deep neural networks. It is designed to be intuitive, extensible and easy to debug. Its core called torch is written in the C language and offers GPU-accelerated tensor computation operations, that replace some of NumPy packages. Most of other frameworks, including TensorFlow, use static graph structures that need to be reused. One of the main advantages of PyTorch is the utilization of dynamic structures where their behavior can be changed at any time with zero lag or overhead.

Caffe

Caffe [23] is another open-source deep learning framework under BSD license. It was created at University of California, Berkeley by Yangqing Jia during his PhD and today is still being developed by Berkeley AI Research. It offers training of a model on either a CPU or a GPU. Many contributors have extended this library with new features during last years. It is widely used in research for its high speed as it is able to process tens of millions of images per day on a single GPU.

OpenCV

OpenCV [35] is a library that is widely used as a tool for image processing in combination with previously mentioned libraries. It is an open-source computer vision and machine learning library with BSD license. OpenCV offers a large number of optimized algorithms starting from simple operations like image transformations to complex tasks such as face recognition, object identification, tracking camera movement, 3D model reconstruction and many others. It is used by many well-known companies like Google, Microsoft, Intel and

IBM. It is also supported in many programming languages like C++, Python, Java or MATLAB and can be run on many operating systems such as Windows, Linux, Android or Mac OS.

Chapter 4

CNNs with varying image size and aspect ratio

As was mention earlier, one of the major challenges of the plankton image dataset is the enormous variety of image sizes and aspect ratios. CNNs typically accept images of a single size. With naive methods, such as scaling and padding each image to one common size many little details are lost, that are crucial for achieving better accuracy. This is especially a problem in the case of very long samples, as is illustrated in Figure 4.1.

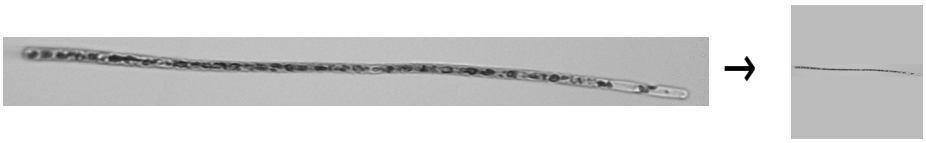


Figure 4.1: Example of down-sampling. Left: original image (1099×106 pixels); Right: image after preprocessing with scaling and padding (224×224 pixels).

This chapter presents existing approaches to deal with images of varying sizes in the case of convolutional neural networks. After that, possible incorporation of metadata is examined that can be also used to supply information containing the image dimensions.

4.1 CNN with spatial pyramid pooling layer

Kaiming He et al. [17] studied training of a single CNN with multiple image sizes. With this approach, training of a CNN is more scale-invariant and the over-fitting is reduced. Convolutional layers accept feature maps of any size, as they only perform desired convolution with sliding window and produce an output of arbitrary size. The same property applies to pooling layers. Limitation for variable input size for the convolutional network lies in the fully connected layers, as they need an input of a fixed size. One method for dealing with this problem can be a spatial pyramid pooling layer (SPP), which accepts input of any size and aspect ratio and produces an output of fixed size. This layer is typically used before the first fully connected layer, e.g. by replacing the last pooling layer. SPP uses defined number of bins where each one performs pooling from one fraction of the image. This process is pictured in Figure 4.2.

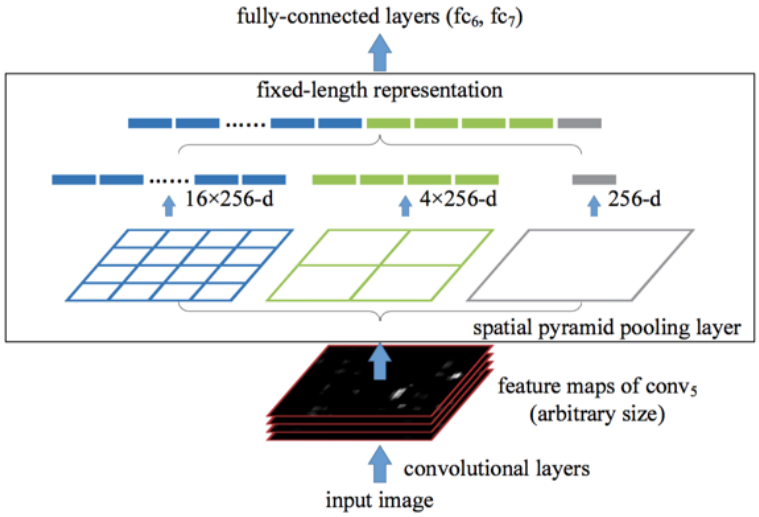


Figure 4.2: Example of a Spatial pyramid pooling layer [17].

As can be seen in the picture, input image is split into multiple grids of variable sizes, where each cell is one bin. First layer has only one bin and pooling is done with whole image (this is also known as global pooling), then next grid has 4 bins, where each bin executes pooling with one quarter of the image and so on. SPP even outperforms the bag of words algorithm [54] as it preserves spatial information.

Four different architectures were tested in this work, which have various depths, numbers of filters and kernel sizes to ensure that the improvement is not limited to a specific architecture. These architectures include ZF-5 based on Zeiler and Fergus's model [53], Convnet*-5, which is modified Krizhevsky et al.'s network [28] and Overfeat-5/7 that is based on Overfeat paper [42]. Numbers in the names represent the number of convolutional layers in the model. In each of these architectures the last pooling layer was replaced with a spatial pyramid pooling layer of size $\{6 \times 6, 3 \times 3, 2 \times 2, 1 \times 1\}$ with 50 bins in total. CNNs were trained on the ImageNet 2012 dataset, where each image was resized in a way that the shorter side has length of 256 pixels. After that a patch with size of 224×224 was cropped either from the center of the image or its four corners. Two different sizes of patches were used for the training, that is the original 224×224 patch and 180×180 pixels patch created by resizing the original one. For testing, only patches with size of 224×224 were applied. Different parameters of the SPP layer were used to prove that the gain in accuracy is not caused only by more parameters of the network. Using this method resulted in increasing the accuracy for every tested architecture just by adding the SPP layer. The best result was achieved with Overfeat-7, where the accuracy was improved from original 67.99 % to 69.64 %. Training with two different sizes provided further improvement up to 70.32 %. Other experiments were also performed with selecting patches of random size between 180×180 and 224×224 during training, giving a slightly worse accuracy – possibly by using fewer samples with size of 224×224 during the training that are used for testing.

4.2 CNN with patch cropping

Linjie Xing et al. [51] implemented a CNN capable of off-line identification of a writer of handwritten text, that means using image data only. A major problem, that needed

to be solved, was processing of images with high aspect ratio that also varies in length. Therefore, a CNN called Half DeepWriter was proposed based on the AlexNet architecture, which accepts patches from images. Each image was first resized, so that its shorter side is 113 pixels long and aspect ratio is kept. After that patches with size of 113×113 pixels were randomly cropped and fed to the network. To preserve spatial information among the patches, which is important for better accuracy, a model called DeepWriter was presented. This DeepWriter consisted of two Half DeepWriters and two patches directly following each other were cropped. Each patch was then supplied to one of the Half DeepWriters. Both of these CNNs shared their parameters, therefore the total number of parameters remained the same. DeepWriter is depicted in Figure 4.3.

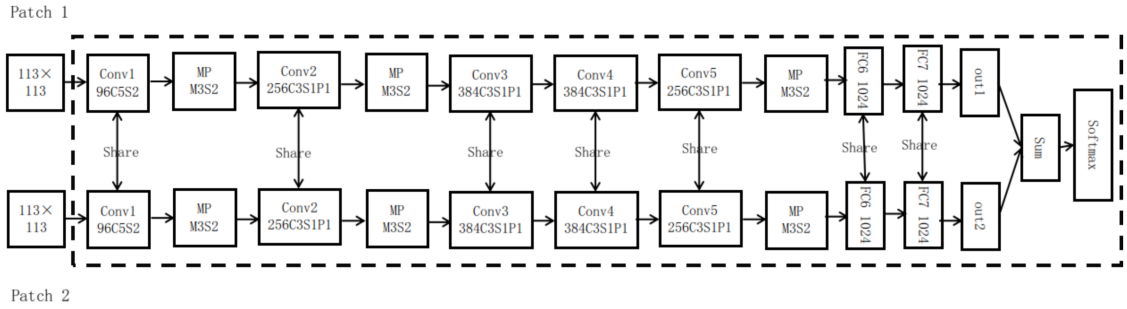


Figure 4.3: Network structure of DeepWriter [51].

Testing procedure was similar to the previously described training strategy, N pairs of patches were cropped from an input image and then were fed to the model. For each pair a score vector f_i was computed and by averaging those values a final score vector was constructed as $f_j = \frac{1}{N} \sum_{i=1}^{N} f_{ij}$. When evaluated on the IAM dataset with all 657 writers, a score of 97.3 % was achieved using only one sentence for testing, which was a great gain in comparison with previous solutions, that needed one whole page of text for test and still resulted in worse accuracy.

4.3 Multi-stream CNN

Nanne van Noord and Eric Postma [32] proposed a solution to deal with both scale-variant and scale-invariant features with CNNs. It was proved in a work by J. Gluckman [13], that CNNs working with scale invariant features only are not complete, as scale-variant information is also important for image recognition. If the image should be reconstructed based on scale-invariant features only, structure of the image would not be known, and the result would not match the original image. The core idea of this work was to combine multiple CNNs and train each one with different input image size, which outperformed traditional single CNN trained with images resized to one common size. This way, every model can learn scale-variant features with given resolution and together they can deal with scale-invariant features too. The architecture of the network is based on the ImageNet model [47], where the final average pooling layer is replaced with a global average pooling layer, therefore the output feature map has fixed size for all image scales. Individual CNNs are fully convolutional and can accept input of any size. When applying to a new image, all softmax class posteriors from each CNN are averaged into a single prediction. The architecture is illustrated in Figure 4.4.

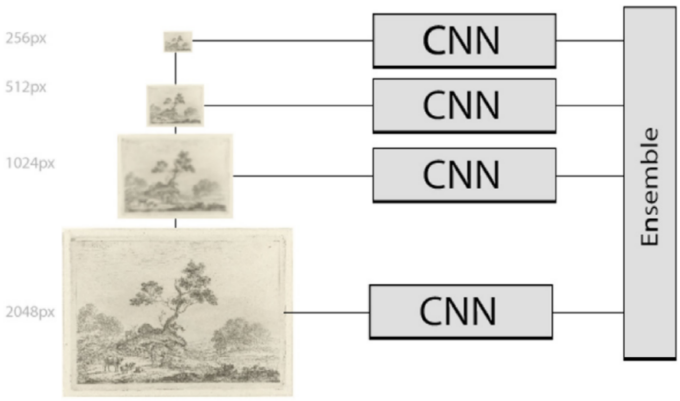


Figure 4.4: Visual representation of the model architecture [32].

With this approach the total number of parameters is increased as the networks do not share parameters, however they can be trained individually in parallel. The method was evaluated on a dataset consisting of roughly 60 thousand images of artwork from 210 artists, where every artist has at least 96 samples. These images had huge variety in both scale and pixel density per mm. Each image was preprocessed by scaling with a Gaussian kernel into four different scales with the shortest size equal to 256, 512, 1024 and 2048 pixels and then each scale is fed into a corresponding network. The mean class accuracy was improved from 75.69 % using only a single CNN to an accuracy of 82.12 % using all four networks combined.

4.4 CNNs with included metadata

Another way, how to address varying image size in a CNN could be possible through incorporating metadata. This way, a model could obtain information about the original dimensions of the image.

Grace Chu et al. [8] experimented with incorporating GPS coordinates during classification to further improve accuracy. Two fine-grained datasets of animal and plant species were used in this work. The first one is called iNaturalist (year 2017) which contains images of five thousand species from all over the world, other one is YFCC100M dataset that contains 100 million images and videos. Many of these species have only subtle differences that only experts can distinguish, however many species can be located only in a given area. Therefore, apart from using image data alone, latitude and longitude were also leveraged during the classification process. Without any additional training of the model, two methods were examined that would utilize this additional information – Bayesian Priors and Label WhiteListing. With Bayesion Priors, a Maximum A Posteriori estimation is used as follows: $L_{\text{MAP}}(I, G) = \text{argmax}_L f(I|L)P(L|G)$, where L is the image label, $f(I|L)$ is the likelihood function of an observation given the label, G is the geolocation of sample and $P(L|G)$ is the prior distribution over labels. For WhiteListing, only species that are known to be located in a certain area are presented in the output.

Following experiments included feeding metadata into a neural network, where both the latitude and longitude were normalized to range of $[-1, 1]$. The first approach takes advantage of post-processing of the image classifier by embedding its output together with the GPS coordinates. Coordinates are processed by three fully connected layers with neuron

numbers of 256, 128, 128. After that, logits of the image classifier and coordinates classifier are simply joined together, so no further interaction between them occurs. Overall structure of this network is depicted in Figure 4.5. Post-processing comes with the advantage of no need for additional training of an already trained image classifier, its weights can be fixed during the training process for metadata. Other experiments with more interaction between the two classifiers were examined by adding more fully connected layers after merging the outputs, however no further improvement in accuracy was achieved.

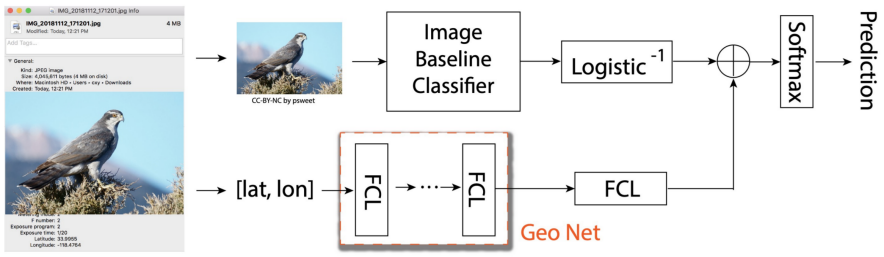


Figure 4.5: Architecture for post-processing models [8].

Finally, effects of adding coordinates during image classifier training were studied, as geolocation data was integrated with multiple image CNN layers. GPS coordinates are first processed by multiple fully connected layers and then are combined by addition with middle and higher levels of the image classifier. Two state-of-the-art image models were examined, InceptionV3 and MobileNetV2. The largest improvement was achieved with applying the post-processing method on InceptionV3 architecture, accuracy was increased from 70.1 % to 78.2 %.

Jeffrey S. Ellen et al. [11] made a study of leveraging the use of metadata information for plankton recognition. The used dataset consisted of 350 000 images divided into 27 classes, that were collected by an autonomous vehicle called Zooglider. This vehicle operates in depth between 0 and 400 m, where it collects black and white silhouette images of plankton, together with measurement of extra information such as conductivity, temperature, depth and other. Images were normalized by the Global Contrast Normalization, that is by subtracting a mean value from each pixel and then dividing it with a standard deviation. Used augmentation involved both horizontal and vertical flipping together with rotation. Metadata was also normalized in the same fashion. The metadata consisted of 93 different values, that were divided into three categories – Geometric (e.g. area, perimeter, circularity, symmetry), Geotemporal (e.g. latitude, longitude, depth, season, time of day) and Hydrographic (e.g. Chlorophyll *a* fluorescence, salinity, temperature).

The structure of examined CNN was based on the VGG-16 architecture. Multiple different approaches of combining metadata with image data were presented with different quantity of interaction. The number of parameters in the network was preserved as constant as possible to avoid any gains in accuracy by using a larger number of them. Individual propositions are illustrated in Figure 4.6. Results showed that the best accuracy is achieved with the architecture performing more interaction with the metadata. Average gain in accuracy was 1.3 points, improving the overall accuracy to 90.5 %.

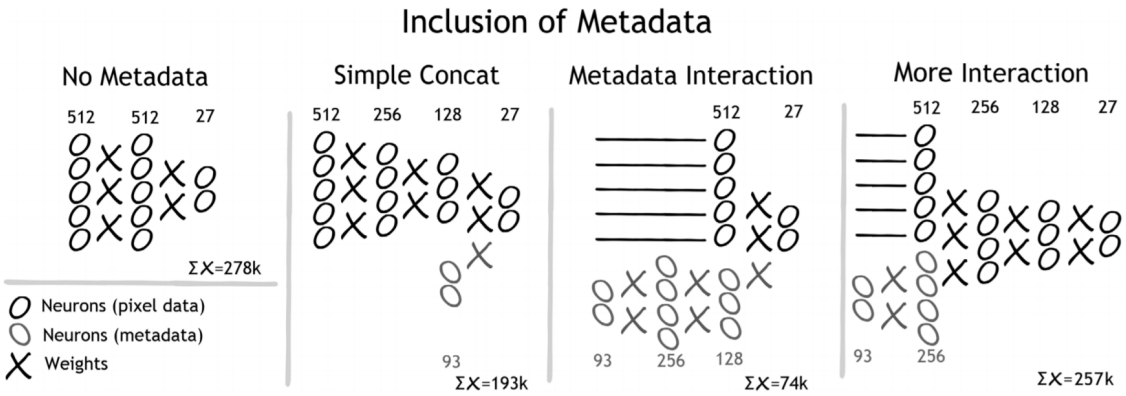


Figure 4.6: Different architectures for metadata incorporation [11].

Chapter 5

Proposed solutions, experiments and evaluation

This section describes the main objectives of this work together with some ideas from the studied literature, that were leveraged during experiments. Next, dataset used for training of proposed models is discussed, its structure, what data it contains and how it was obtained. Following part is devoted to preprocessing of images before feeding them to a neural network. After that proposed solutions are presented, and this section ends with evaluated results.

5.1 Summary of studied literature

Given the examined literature, it is evident, that leveraging of neural networks becomes lately more and more popular within this area, as CNNs often outperform feature-based methods such as random forests. A large variety of CNNs was examined for plankton classification with different parameters such as the number of layers and size of filters. To prevent over-fitting, a commonly seen approach is the utilization of dropout and data augmentation, which includes random rotation, as plankton is invariant to this operation. Other transformations consist of horizontal or vertical flipping, cropping and adding Gaussian noise. In many works images were resized to a common size with keeping the aspect ratio to prevent any distortions and were normalized by subtracting a mean value and dividing the result by a standard deviation. Taking the advantage of incorporation of metadata within the model was also proven to increase the accuracy, as well as using multiple different sizes of each image. Following experiments are based on these observations.

5.2 Objectives of the work

The objective of this work is to design and implement a model classifying plankton samples into categories and examine new ways to improve the accuracy with focus on varying image sizes and aspect ratios. This task consists of following steps:

1. Design a CNN-based model able to classify plankton samples.
2. Study utilization of metadata in the model for better accuracy.
3. Explore methods for dealing with high variability in aspect ratios and image sizes.

4. Evaluate models with a plankton image dataset.

This work is focused on phytoplankton only, that was captured with FlowCytobot in the Baltic Sea. Data provided by Finnish Environment Institute (SYKE) contain images of microplankton and nanoplankton, that were labeled into selected groups by a taxonomist expert. However, approaches used during this work can be used for different plankton image data as well.

5.3 Data

The dataset that was used in this work was provided by the Finnish Environment Institute (SYKE). Samples were captured by a FlowCytobot (FCBI), a submersible imaging-in-flow cytometer designed for analyzing nano and microplankton. It records optical properties of small phytoplankton cells (up to 10 μm in scale) by fluorescence light scattering signals from a laser beam. It is also capable of capturing plankton with size larger than 100 μm, however it lacks the ability of analyzing samples within the range of 10 to 100 μm properly. The FCBI can operate autonomously up to several months, as it remotely communicates with shore laboratories, receives commands and sends measured data. It can process about 5 mL of seawater every 20 minutes. Particles travel through a laser beam and scatter light, which is monitored by a sensor – if it exceeds a certain threshold, an image is captured by a camera. Image data is created in a time span of 1 μs by a 10x microscope objective with resolution of 1 μm. Movement of the plankton during the exposure (about 2.2 $m.s^{-1}$) results in a small blur effect in the direction of the flow. To reduce image data quantity, an edge detector is used to segment individual areas of interest. During its two months deployment, FlowCytobot collected about 1.5 million samples [34].

This dataset contains about 116 thousand images labeled by a taxonomist expert into 73 different classes. Individual classes are not balanced, the number of samples varies from several thousands to just one single image, as can be seen in Table 5.1. Images have one channel and their dimensions are in ranges of 64 to 1 276 pixels for width and 26 to 394 pixels for height. There is quite a huge variance in both ratio and scale for each sample, as can be seen in Figure 5.1. The name of each image in the dataset contains time of acquisition. Some representative examples of classes can be viewed in Figure 5.2.

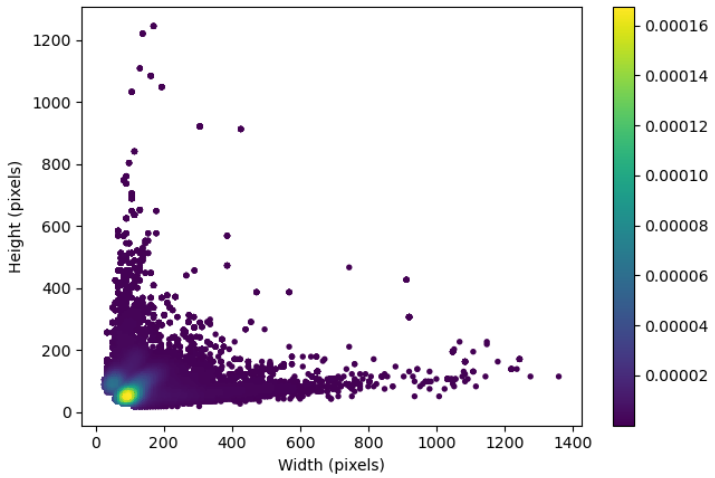


Figure 5.1: Scatter plot of dimensions of plankton images in pixels. Each point represents an image, the color indicates density in the surrounding area.

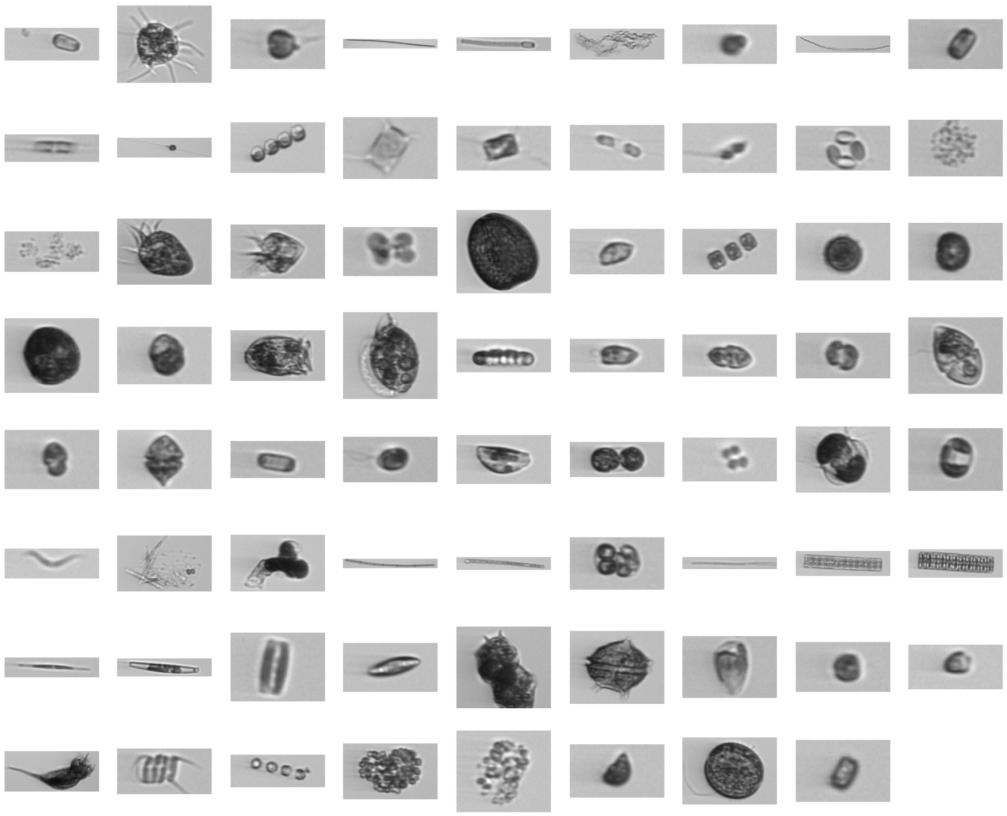


Figure 5.2: Examples of images in the dataset. Images in this figure are resized to a common width while keeping constant aspect ratio.

Table 5.1: Class distribution of the dataset.

Class	#	**Class**	#
Unclassified	82028	Euglenophyceae	93
Dinophyceae	4606	Dinophysis acuminata	89
Oscillatoriales	4402	Nodularia spumigena	84
Snowella Woronichinia sp	3317	Cluster A	72
Dolichospermum Anabaenopsis	2367	Nitzschia paleacea	65
Pyramimonas sp	1602	Licmophora sp	63
Skeletonema marinoi	1517	Katablepharis remigera	54
Heterocapsa triquetra	1464	Gymnodiniales	50
Thalassiosira levanderi	1146	Melosira arctica	43
Teleaulax sp	1132	Ceratoneis closterium	39
Aphanizomenon flosaquae	1072	Aphanothece paralleliformis	29
Pennales sp	976	Chaetoceros similis	29
Mesodinium rubrum	962	Binuclearia lauterbornii	23
Chaetoceros sp	952	Gonyaulax verior	22
Chroococcales	884	Akinete	19
Peridiniella catenata single	875	Amylax triacantha	19
Pseudopedinella sp	853	Scenedesmus sp	14
Heterocapsa rotundata	624	Apedinella radians	13
Oocystis sp	597	Chaetoceros throndsenii	12
Cryptophyceae Euglenophyceae	594	Chaetoceros resting stage	8
Cryptomonadales	422	Nostocales	8
Centrales sp	338	Dinobryon balticum	7
Monoraphidium contortum	303	Chaetoceros subtilis	5
Eutreptiella sp	236	Pauliella taeniata resting stage	5
Heterocyte	234	Chaetoceros danicus	4
Prorocentrum cordatum	229	Aphanizomenon sp	3
Ciliata	217	Dinophysis norvegica	3
Cyst like	150	Melosira arctica resting stage	3
Gymnodinium like	150	Coscinodiscus granii	2
Peridiniella catenata chain	144	Dinophysis sp	2
Cymbomonas tetramitiformis	132	Gymnodinium sp	2
Pauliella taeniata	119	Nodularia spumigena heterocyte	2
Beads	100	Rotifera	2
Cyclotella choctawhatcheeana	99	Amoeba	1
Merismopedia sp	97	Flagellates	1
Chlorococcales	96	Protoperidinium bipes	1
Uroglenopsis sp	94		

5.4 Data preprocessing

Out of the 73 classes only 71 were used for the training, as *Unclassified* and *Flagellates* contain samples, which an expert could not classify with a reasonable level of certainty. They do not adhere to a real taxonomic rank and therefore are not suitable for training. After removing these two classes, about 34 000 samples remain. Subsequently, multiple different subsets were created based on the number of samples per class. In the first phase, subset *Sub100* containing only those classes that have at least 100 samples was used for the initial model training. After that, classes with fewer samples were gradually included, as can be seen in Table 5.2.

Minimum samples	**Number of classes**
Sub100	32
Sub50	44
Sub10	55

Table 5.2: Tested subsets with different number of minimum samples per class.

Each subset was split into 20 % testing and 80 % training partitions. Classes in the training partition were balanced so that every class has exactly 1000 samples. If some class had originally more samples, only the first 1000 images in alphabetical order were used. If there were fewer samples then new samples were created through data augmentation. Images were transformed by applying both horizontal and vertical flipping, which extended the dataset up to four times. By adding a rotation of 90 degrees each class could achieve eight times larger size than the original. This process was repeated until the desired number of images was reached. During the training process another random augmentation was applied consisting of scaling with relative limit of 0.1, blurring, adjusting brightness and adding Gaussian noise with a variance of 0.001. Augmentation is depicted in Figure 5.3.

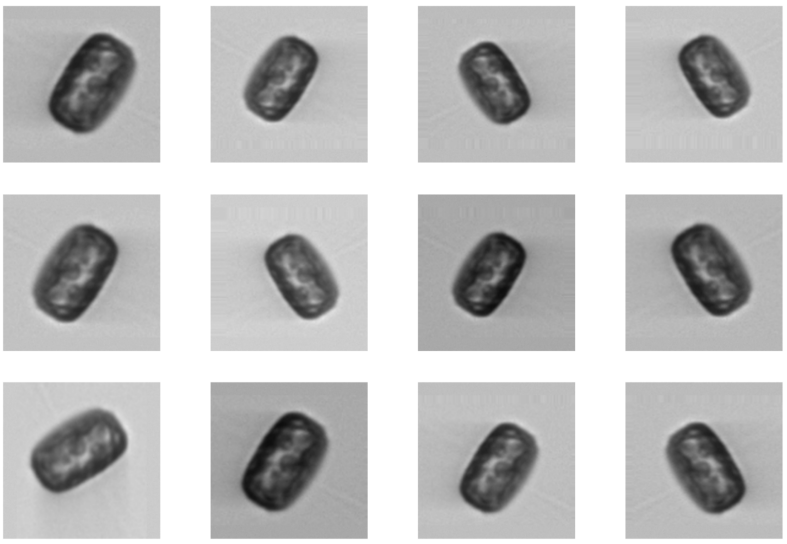


Figure 5.3: Example of data augmentation on a single sample.

All images in a batch were resized to a common size while keeping the original aspect ratio to prevent any distortions. This was done by resizing the larger side of the image to fit into the defined boundary – larger images were reduced in size; smaller images were enlarged. Bicubic interpolation was used during this process. After that the images were padded to fill the remaining area of the boundary. For this purpose, a mean color calculated from the image boundaries was used together with applied Gaussian noise to reduce any artificial edges caused by homogeneous regions. After the resizing was done each image was normalized by subtracting a mean value from every pixel of the image and dividing the result by a standard deviation. These values were calculated from the whole training part of the currently used subset.

5.5 Description of experiments

The first set of experiments is focused on searching for CNNs, that provide the best accuracy for the plankton image dataset. From these CNNs, an efficient architecture is selected for the testing of different parameters within new approaches, and another one, which resulted in the best accuracy and is used for the final evaluation of these methods. Within the next experiments, multiple different methods with a pursuit of further improvement of these networks are examined, which address the problem of varying image size and the incorporation of metadata.

Baseline CNN comparison experiment

In the first experiments a single CNN is evaluated with fixed input size of images at the input and without the use of any additional metadata. For the initial testing a *VGG16* based architecture called *Al-Barazanchi* was used that is derived from the one presented in a work by Hussein Al-Barazanchi et al. [2], which accepts images with size of 224×224 pixels. This architecture can be viewed in Table 5.3. By experimenting with this model optimal training parameters were selected together with augmentation and preprocessing that gave the best results.

Al-Barazanchi architecture was further examined in this work by training it with images of different aspect ratios to preserve as many details as possible for images containing very long plankton samples. For this reason, *Al-Barazanchi_2* was constructed for images with an aspect ratio of $2:1$, where the stride of the second pooling layer was changed to (2,1), that is a half of the original stride for its height. This network accepts images with size of 316×158 pixels. Finally, *Al-Barazanchi_4* was examined, for which all images are converted into an aspect ratio of $4:1$, where the stride of second convolutional layer was adjusted to (4,1), that is twice as big step for the width and a half for the height. Furthermore, convolutional kernel size for the same layer was changed from (3,3) to (6,3). This network receives images with resolution of 448×112. All images are flipped into horizontal position to avoid the need of training networks with an aspect ratios of $1:2$ and $1:4$.

Apart from this, multiple different architectures were tested as well, that proved to be useful for plankton classification in other works together with some others that are commonly used. The next studied model is also *VGG16* based and was proposed by Jeffrey S. Ellen et al. [11], that works with smaller kernels in convolutional layers and accepts images with a reduced size of 128×128 pixels. Detailed structures of *Jeffrey* architecture can be seen in Table 5.3. Padding with same pixels was used in convolutional layers for

both models to preserve resolution. Batch size was set to 256, training with 60 epochs proved to be sufficient.

Some commonly used network architectures that were tested include *AlexNet* [27], *DenseNet121* [20], *ResNet50* [16] and *MobileNet* [19] which accept images with a size of 224×224 and *InceptionV3* [48] with an input size of 299×299. In the case of *AlexNet*, batch size of 256 was chosen as well as training with 60 epochs. For the rest of these architectures, the batch size was set to 64 and number of epochs to 80.

Each architecture was evaluated on the subset *Sub100*. *InceptionV3* was further evaluated on datasets *Sub50* and *Sub10*, also it was modified in the same fashion as the *Al-Barazanchi* architecture to focus on images with an aspect ratio of 2 and 4. All examined architectures and their parameters are in Table 5.4.

Al-Barazanchi	**Jeffrey**
Input (224×224)	Input (128×128)
Conv 64 (3×3), stride 1	Conv 16 (3×3), stride 1
Pool (3×3), stride 2	Pool (2×2), stride 2
Conv 64 (3×3), stride 1	Conv 32 (3×3), stride 1
Pool (3×3), stride 2	Pool (2×2), stride 2
Conv 128 (3×3), stride 1	Conv 32 (3×3), stride 1
Pool (3×3), stride 2	Pool (2×2), stride 2
Conv 128 (3×3), stride 1	Conv 64 (3×3), stride 1
Pool (3×3), stride 2	Pool (2×2), stride 2
Conv 256 (3×3), stride 1	Conv 64 (3×3), stride 1
Pool (3×3), stride 2	Pool (2×2), stride 2
Flatten	Flatten
Hidden layer (256)	Hidden layer (512)
Dropout (20 %)	Dropout (50 %)
Hidden layer (256)	Hidden layer (512)
Dropout (20 %)	Dropout (50 %)
Output layer (N)	Output layer (N)

Table 5.3: Detailed structure of evaluated CNN architectures.

CNN with a Spatial Pyramid Pooling layer experiment

This experiment is based on the discussed approach proposed by Kaiming He et al. [17] and aims at improving scale-invariance of the model. A spatial pyramid pooling layer was leveraged to enable training with images of varying resolution. This layer replaces the last pooling layer of the architecture and has a shape of $\{6 \times 6, 3 \times 3, 2 \times 2, 1 \times 1\}$ with a bin count of 50. Network is then trained with numerous predefined image sizes. In one epoch both images for training and validation are resized to one of the sizes so that the whole batch consists of images with a single fixed size. After the epoch is finished, size is switched to the next one and the process is repeated.

The architecture that was used for experiments is derived from the *Al-Barazanchi* shape, as it is not too deep and therefore searching for optimal training parameters is much faster than in the case of deep architectures such as *InceptionV3*. This architecture with an SPP layer can be seen in Table 5.5. Maximum number of epochs was increased to 90, as the

Architecture	Input size	Number of parameters
Al-Barazanchi	(224 × 224)	2 993 655
Al-Barazanchi_2	(316 × 158)	4 828 663
Al-Barazanchi_4	(448 × 112)	2 600 439
Jeffrey	(128 × 128)	885 143
AlexNet	(224 × 224)	46 854 880
DenseNet121	(224 × 224)	7 087 607
ResNet50	(224 × 224)	23 694 135
MobileNet	(224 × 224)	3 284 663
InceptionV3	(299 × 299)	21 914 903
InceptionV3_2	(420 × 210)	21 914 903
InceptionV3_4	(600 × 150)	21 914 903

Table 5.4: Parameters of examined architectures.

network takes longer time to be trained in comparison with a CNN trained with a fixed input size. Experiments with multiple different image sizes were evaluated to find a combination, which provides the largest boost in accuracy. Modified *Al-Barazanchi* network was first evaluated only with one size of 224×224 to see how the SPP layer affects network's accuracy. Next tests included training with combinations of multiple sizes, that are 224×224, 180×180 and 256 × 256. For evaluation of the network only images with one fixed size of 224 × 224 were used.

After the optimal combination with *Al-Barazanchi* was discovered, *InceptionV3* was modified in a similar fashion – its global average pooling layer was replaced with an SPP layer and it was trained with image sizes of 299 × 299, 256 × 256 and 348 × 348.

Al-Barazanchi
Input
Conv 64 (3 × 3), stride 1
Pool (3 × 3), stride 2
Conv 64 (3 × 3), stride 1
Pool (3 × 3), stride 2
Conv 128 (3 × 3), stride 1 1
Pool (3 × 3), stride 2
Conv 128 (3 × 3), stride 1
Pool (3 × 3), stride 2
Conv 256 (3 × 3), stride 1
Spatial Pyramid Pooling [6,3,2,1]
Hidden layer (256)
Dropout (20 %)
Hidden layer (256)
Dropout (20 %)
Output layer (N)

Table 5.5: Modified *Al-Barazanchi* architecture with a spatial pyramid pooling layer.

CNN with metadata experiment

Another set of experiments was targeted at including metadata within the CNN to further improve the accuracy. For this purpose, the original width and height of the image before preprocessing was used which could help distinguish two different samples that look similar due to resizing. Furthermore, time of collecting of the sample was extracted from each filename and was subsequently transformed into two different values – *Hour*, where samples are divided into 8 partitions based on the hour of the day and then *Season*, consisting of 4 parts based on the month. This way, metadata add four new values in total to be supplied to the network. Distribution of these values for few selected classes can be seen in Figure 5.4 and Figure 5.5.

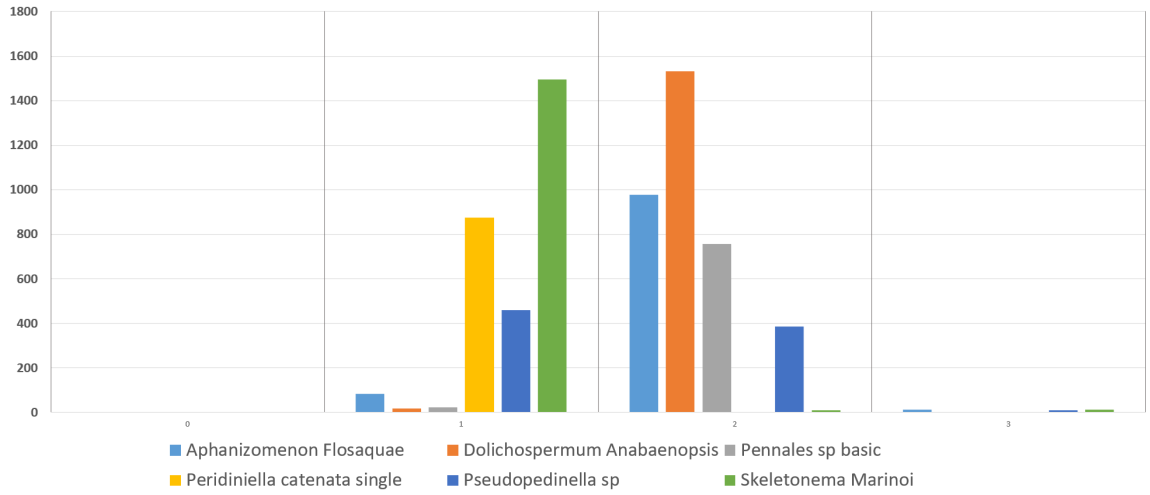


Figure 5.4: Distribution of samples per *Season* category for selected classes.

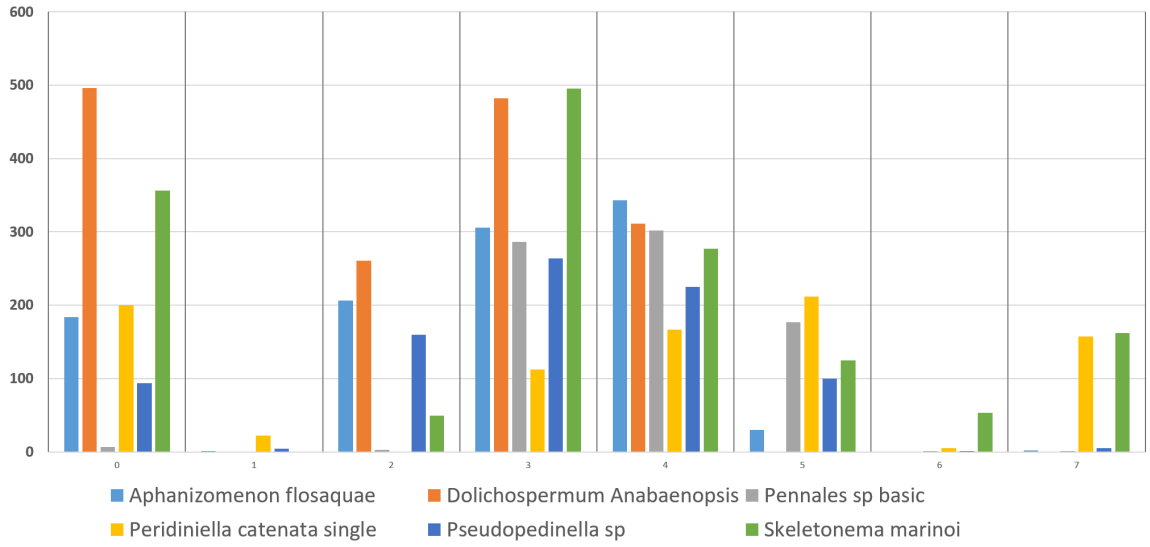


Figure 5.5: Distribution of samples per *Hour* category for selected classes.

All metadata values are normalized to a range of $[-1; 1]$. To include metadata into the network, multiple approaches proposed by Jeffrey S. Ellen et al. [11] and Grace Chu et al. [8] were examined. With these solutions, the image classifying model can be embedded into a new network without much modification. This network has two inputs, one for image data and one for metadata. These inputs are processed by separate parts of the network that are later concatenated together and both types of data are processed by common layers. Two different approaches of training were examined. First approach trains the whole architecture together with an embedded image model initialized with random weights. Second approach uses an image classifier that is initialized with weights loaded from a trained model and its weights are kept fixed for the time of training, therefore only the metadata part and common part of the network are trained.

Three distinct architectures were evaluated, in which various levels of metadata interaction are tested together with different levels of interaction between metadata and image data. Dropout with 20 % probability was used for all new fully connected layers. These models are depicted in Figure 5.6.

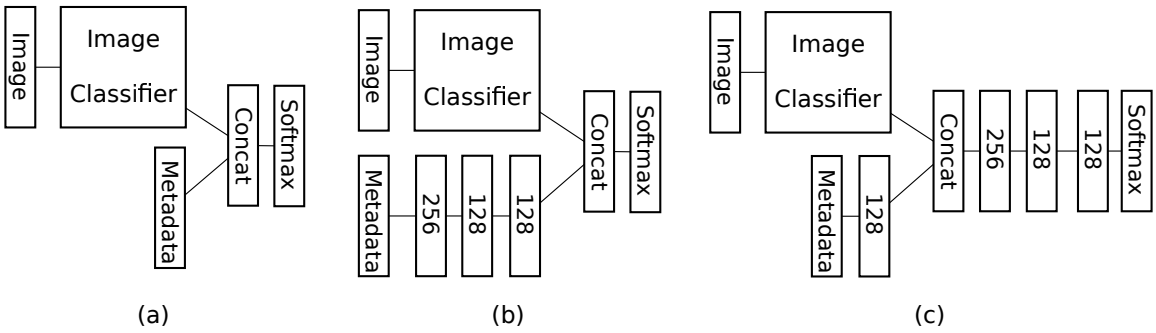


Figure 5.6: Different architectures to include metadata: (a) Simple concatenation [8]; (b) Metadata interaction [11]; (c) More interaction [11].

Al-Bazaranchi was used as the embedded image classifier to find the best modification. Models were trained with both *shape* and *time* metadata at the input, the number of epochs was set to 80 for training the whole architecture including an image model, in the case of using a trained model 40 epochs were selected. Furthermore, the model with the highest accuracy was chosen and examined with different combinations of metadata to see which one has the largest impact on accuracy, that is [*shape*], [*time*] and [*shape*, *time*]. Finally, *InceptionV3* was used as the image model embedded in an architecture with the best outcome to examine if it can be further improved.

CNN with a patch cropping experiment

Next experiments with patch cropping were examined with the focus on preserving small visual details that get lost due to image scaling. Each image is first rotated into horizontal position, so that its width is greater than its height. After that, it is resized in a way that height of the image is the same as height of the patch to be cropped while keeping the original aspect ratio. At this moment the image is ready to be cropped into patches. Multiple different methods of a patch cropping were examined.

The first method is using a single patch, that is randomly cropped alongside of the image. This one patch is then supplied to a classic CNN expecting one image input – *Al-Barazanchi* was evaluated with a patch size of 224×224 pixels. The next method leverages

using a pair of patches to preserve spatial information between them and is based on the work by Linjie Xing et al. [51] that was described earlier. This time, images need to be padded in their width to guarantee enough space for two consecutive patches to be cropped. One patch has a size of 224×224, therefore image needs to be padded to have its width at least 448 pixels long. This pair is then supplied to a CNN with a modified architecture of the *DeepWriter* [51] proposed in the same work. This architecture was changed to a pair of two CNNs with the *Al-Barazanchi* structure that share parameters between each other and therefore the total number of parameters is not increased. Each CNN takes one of the two patches during training and testing.

However, this method causes that many samples, which could be fit into a single patch, are cut in half. For that reason, a third method is examined by modifying the previous one. Here two consecutive patches are cropped, but the image is no longer padded on its sides. Instead, if there is not enough space for the second patch, it is cropped from the end of right side of the image and the two patches may therefore overlap each other. Both patches are then inserted into the modified *DeepWriter*. All three approaches of patch cropping are depicted in Figure 5.7.

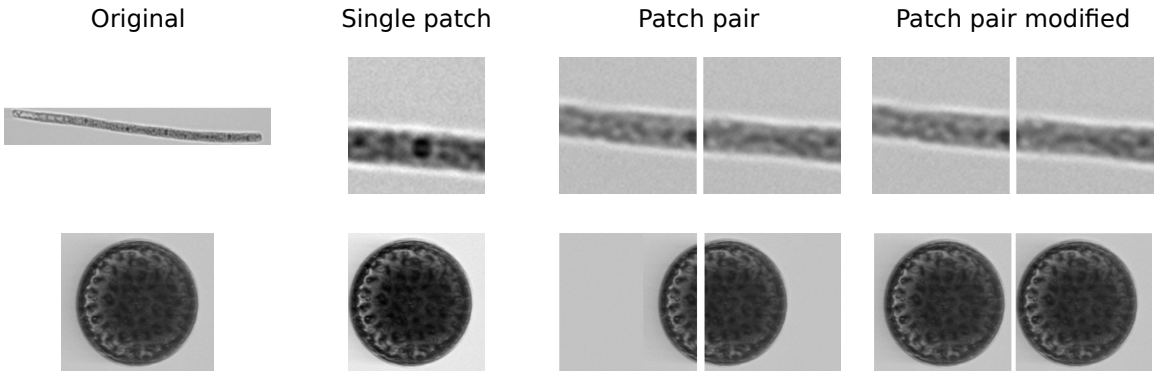


Figure 5.7: Different patch cropping methods.

The model was trained for 90 epochs with a batch size of 64. Evaluating of these networks was performed through a sliding window - N patches or a pair of patches were subsequently selected from the image. Each of these patches was then evaluated by the network resulting in N prediction vectors. These vectors were finally combined by averaging them into a single prediction as described in section 4.2. Different numbers of iterations during testing were examined to find the optimal amount for the best accuracy to computation time ratio. Finally, the best combination was selected and applied to *DeepWriter* constructed out of two *InceptionV3* networks. This model was trained for 90 epochs with a batch size of 32.

Multi-stream CNN experiment

The last part of experimenting is focused on combining multiple models together. This method is based on the work by Nanne van Noord and Eric Postma [32] and brings the advantage of using different models, where each one is trained to distinguish features in images at different scale and together they can form a stronger classifier. One network can be for example focused on samples with a ratio of $1 : 1$, whereas other one can learn detailed features of species that have a very long shape. Another advantage of this solution is the fact, that individual models can be trained separately and then combined at the time of

testing. After the evaluation of an image by each model prediction vectors are combined through averaging: $f = \frac{1}{N}\sum_{i=1}^{N} f_i$, where N is the number of models. An input image is preprocessed for each model with its unique way. Combination of multiple naive CNNs accepting each different image sizes and aspect ratios was examined, furthermore different approaches previously described like patch cropping were utilized to see how much they can improve accuracy together. This was done for both *Al-Barazanchi* and *InceptionV3* architectures.

Implementation details and training

All proposed models were trained with applied cross-validation of 10 folds with a stratified selection, the number of epochs varied for each architecture. The stochastic gradient descent optimizer was used during training, together with applied Nesterov momentum, initial learning rate set to 0.01, weight decay 10^{-6} and a momentum of 0.9. Size of batches is varying with used models, either 64 or 256 was selected. Models were trained with utilizing graphic cards on dedicated school server (using two *GeForce GTX 1080 Ti* GPUs and *TITAN RTX* GPU), supercomputer Puhti (containing 320 GPUs such as *Nvidia Volta V100* GPU) together with two personal computers (*GeForce GTX 1060* and *GeForce GTX 1650*).

The implementation of plankton recognition was done in Python using *Tensorflow* of version 1.14.0, together with framework *Keras*. Furthermore, *OpenCV* library was used for image processing and *Albumentation* library was leveraged for data augmentation. As the spatial pyramid pooling layer is not a part of Tensorflow or Keras, its code was obtained from a GitHub repository [18]. Finally, a patched version [30] of callback *keras.callback.ModelCheckpoint* for saving the best model during training was used, as there were collisions with writing into a file while multi-processing was enabled. Use instructions are located in appendix B.

5.6 Results

This section shows results obtained with different approaches – those are using a classic CNN accepting only image data of single size, followed with architectures containing an SPP layer, after which CNNs with metadata and patch cropping are evaluated and finally this section is finished with results from combining multiple CNNs together in a form of a multi-stream CNN.

Evaluating accuracy

The accuracy of a model was calculated as a fraction of correctly labeled samples from the testing part of the dataset. As the 10-fold cross-validation was applied, each architecture resulted in 10 trained models, each with different success rate. From these accuracies a mean value $\overline{Acc}$ is calculated together with a standard deviation s as follows:

$$\overline{Acc} = \frac{1}{N}\sum_{i=1}^{N} Acc_i. \tag{5.1}$$

$$s = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}\left(Acc_i - \overline{Acc}\right)^2} \tag{5.2}$$

where N is the number of folds and Acc_i is the accuracy of one of the models. Final accuracy is defined as $\overline{Acc} \pm s$. Accuracy for individual classes was evaluated in the same fashion, that is a fraction of correctly labeled samples within one class.

Furthermore, confusion matrices were calculated. Confusion matrix has a form of a table, where columns represent instances of actual class and rows represent predicted classes. Given one column, it can be observed how many times was one class labeled correctly and how many times it was categorized into another class. Only instances on the diagonal of the table represent correct classification. This matrix was furthermore normalized by dividing each number in one column by a sum of values in the same column, this way, every cell represents percentage of all predictions for a single class.

Baseline CNN comparison results

The first set of tests aimed at examination of different operations for data augmentation and preprocessing performed with the *Al-Barazanchi* architecture can be seen in Table 5.6. From these tests the best parameters were selected, and other architectures were evaluated as is shown in Table 5.7. It is evident that additional augmentation in a form of cropping and scaling has positive effect on network's accuracy, since without it the score dropped by 1 point. In case of removing addition of a random blur, Gaussian noise and brightness adjusting the accuracy dropped even more by 1.4 points.

Modification	**Accuracy**
Proposed solution	**0.9341 ± 0.0022**
No crop, shift or scale	0.9242 ± 0.0035
No blur, noise or brightness	0.9201 ± 0.0059

Table 5.6: Accuracy for different conditions of the *Al-Barazanchi* architecture using *Sub100*. The best result is highlighted.

Architecture	**Accuracy**
InceptionV3	**0.9520 ± 0.0013**
DenseNet121	0.9441 ± 0.0065
MobileNet	0.9420 ± 0.0045
Al-Barazanchi	0.9341 ± 0.0025
AlexNet	0.9274 ± 0.0053
Jeffrey	0.9110 ± 0.0084
ResNet50	0.9201 ± 0.0244

Table 5.7: Accuracy for *Sub100* with different architectures. The best result is highlighted.

As can be seen in Table 5.7, deeper networks achieved better accuracy, however at the cost of much longer training time due to high computational complexity. For example, training of *InceptionV3* took roughly three times longer than training the *Al-Barazanchi* architecture. The highest accuracy of 95.20 % was achieved with the *InceptionV3* and process of its training can be viewed in Figure 5.8. In this chart it can be observed that the number of 80 epochs is sufficient as the accuracy does not increase for many iterations.

DenseNet121 was another architecture that proved to obtain high accuracy when it comes to plankton recognition. *Al-Barazanchi* ended up with a score of 93.41 %, which is pretty close to the best model and with much smaller depth. For this reason, it was selected as the base architecture for next tests using different approaches and searching for optimal parameters. The worst accuracy was achieved with *Jeffrey* which reached only 91.10 % and *ResNet50* with a score of 92.01 % and a large deviation of 2.44 %.

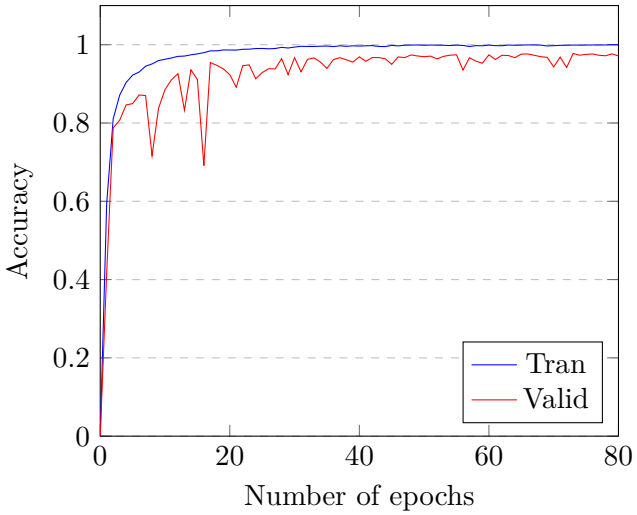


Figure 5.8: Training and validation accuracy during training of *InceptionV3*.

Outcome of experiments with modifying the *Al-Barazanchi* architecture to be focused on different aspect ratios are in Table 5.8, the same modifications for *InceptionV3* are in Table 5.9. Score for the aspect ratio of 1 : 1 provided the best results, ratio of 2 : 1 performed quite similarly. For the aspect ratio of 4 : 1 it can be noticed that there is significant drop in accuracy, which is possible due to the fact, that majority of samples are of circular shape, that lose a lot of details in this case.

Architecture	**Accuracy**
Al-Barazanchi	**0.9341 ± 0.0025**
Al-Barazanchi_2	0.9204 ± 0.0136
Al-Barazanchi_4	0.8909 ± 0.0079

Table 5.8: Accuracy for *Sub100* with the *Al-Barazanchi* architecture for different aspect ratios. The best result is highlighted.

Finally, the *InceptionV3* model was applied to the rest of prepared subsets, that is *Sub50* and *Sub10*. In the case of *Sub50*, obtained accuracy was (94.78 ± 0.36) %, for *Sub10* this model achieved (94.10 ± 0.42) %. With an increasing number of classes, the accuracy gets worse, which is expected behavior due to larger complexity. However, for all subsets more than 94 % success rate was observed, which can be considered a good result. Sorted accuracies from the highest to the lowest for individual classes of *Sub10* are in Table 5.10. Most of the classes have more than 90 % successful predictions, the lowest success rates were observed with classes Akinete, Amylax triacantha and Euglenophyceae. However, even in these cases more than 2/3 of predictions were correct, suggesting that classes are fairly

Architecture	Accuracy
InceptionV3	**0.9520 ± 0.0014**
InceptionV3_2	0.9525 ± 0.0033
InceptionV3_4	0.9463 ± 0.0031

Table 5.9: Accuracy for *Sub100* with the *InceptionV3* architecture for different aspect ratios. The best result is highlighted.

balanced. Examples of incorrectly labeled images are in Figure 5.9. For more detailed view confusion matrices for individual subsets can be seen in appendix A.

Architecture	Acc	Architecture	Acc
Uroglenopsis sp	100	Heterocyte	95.7
Scenedesmus sp	100	Prorocentrum cordatum	95.6
Nitzschia paleacea	100	Mesodinium rubrum	94.8
Monoraphidium contortum	100	Dinophysis acuminata	94.1
Licmophora sp	100	Centrales sp	94
Gonyaulax verior	100	Heterocapsa triquetra	93.5
Cyst like	100	Oocystis sp	93.3
Cymbomonas tetramitiformis	100	Chaetoceros sp	91.6
Cluster A	100	Cryptomonadales	90.5
Chaetoceros throndsenii	100	Katablepharis remigera	90
Chaetoceros similis	100	Gymnodiniales	90
Ceratoneis closterium	100	Cyclotella choctawhatcheeana	89.5
Binuclearia lauterbornii	100	Peridiniella catenata chain	89.3
Beads	100	Teleaulax sp	88.5
Apedinella radians	100	Ciliata	88.4
Oscillatoriales	99.4	Nodularia spumigena	87.5
Skeletonema marinoi	99.3	Dinophyceae	82.6
Heterocapsa rotundata	99.2	Eutreptiella sp	80.9
Pennales sp	99	Gymnodinium like	80
Peridiniella catenata single	98.9	Aphanothece paralleliformis	80
Thalassiosira levanderi	98.7	Merismopedia sp	78.9
Pyramimonas sp	98.4	Melosira arctica	75
Dolichospermum Anabaenopsis	98.3	Chlorococcales	73.7
Pseudopedinella sp	98.2	Cryptophyceae Euglenophyceae	72.9
Chroococcales	97.7	Euglenophyceae	66.7
Aphanizomenon flosaquae	97.2	Amylax triacantha	66.7
Snowella Woronichinia sp	97	Akinete	66.7
Pauliella taeniata	95.7		

Table 5.10: Accuracy for individual classes in *Sub10* with the *InceptionV3* architecture.

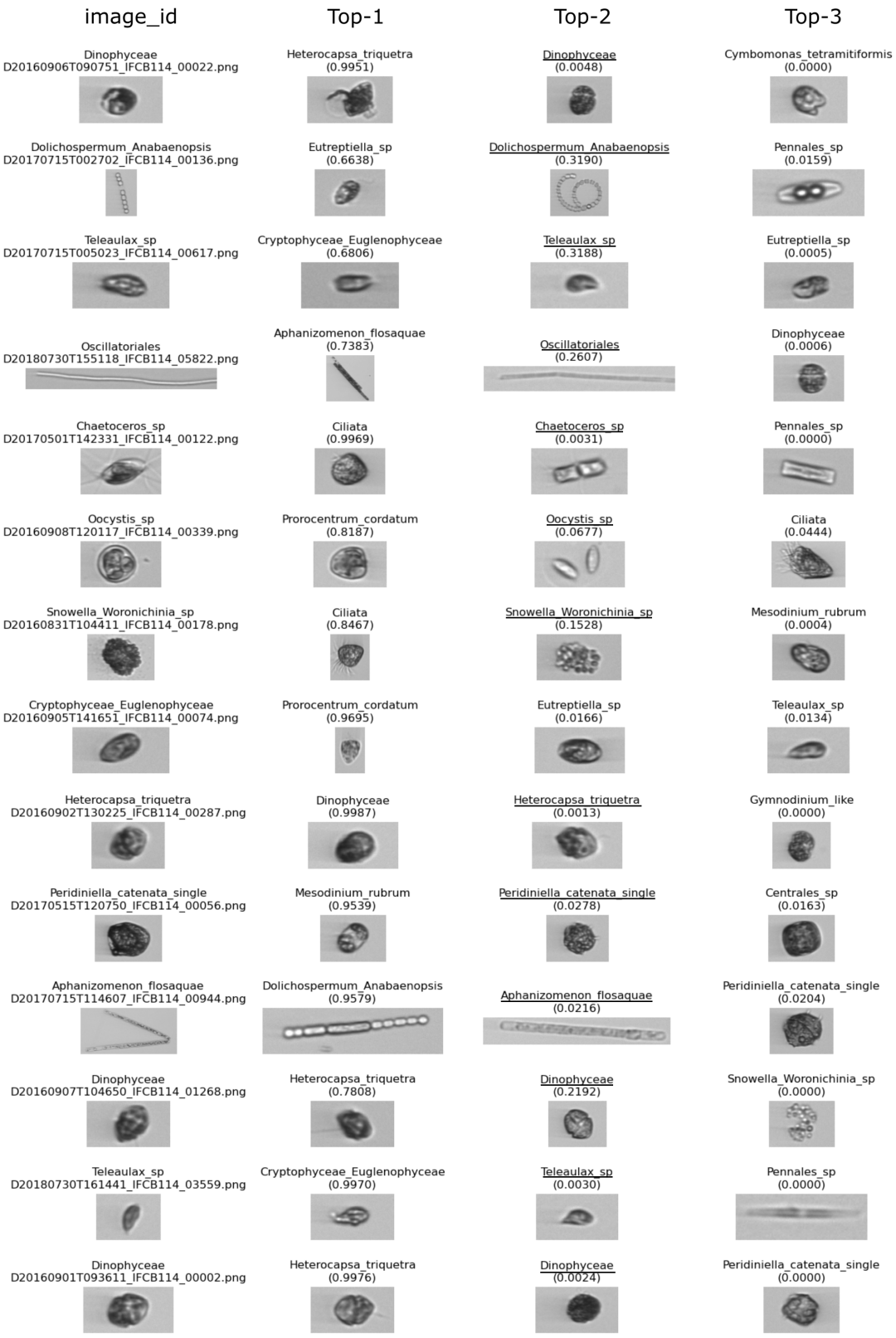


Figure 5.9: Examples of incorrectly labeled samples (InceptionV3, Sub10). On the left are images evaluated by the CNN, followed by example images of top three classes with the highest probability, where the correct prediction is underlined.

CNN with a Spatial Pyramid Pooling layer results

Test results for the *Al-Barazanchi* architecture modified with a spatial pyramid pooling layer are shown in Table 5.11. While training this network with only one single size of 224×224, achieved accuracy was 3 points lower than in the case of the original *Al-Barazanchi* model without any SPP layer, that had score of 93.41 %. From these findings it seems that the SPP layer does not have any positive effect on network training on its own, unlike in the original work [17] where the score of the model was improved. Training with two different image sizes also did not result in any expected boost of the score, as both examined combinations did not reach the original *Al-Barazanchi* success rate. Finally, with a combination of all three different sizes, the accuracy was improved by 0.4 points.

Image sizes	**Accuracy**
(224×224)	0.9058 ± 0.0105
(224×224), (180×180)	0.9205 ± 0.0111
(224×224), (256×256)	0.9327 ± 0.0060
(224×224), (180×180), (256×256)	$\mathbf{0.9387 \pm 0.0052}$

Table 5.11: Accuracy for *Sub100* with *Al-Barazanchi* using an SPP layer. The best result is highlighted.

The SPP layer was then inserted into the InceptionV3. In this case, training of the network with the original single of 299×299 had accuracy of 87.61 %. This score was not improved even with combination of the three different sizes, in that case it resulted in 86.93 %. Both of these results are far from InceptionV3 success rate of 95.20 %, therefore search for other size combinations was not continued.

CNN with metadata results

Incorporation of metadata into the model was examined with embedding the *Al-Barazanchi* architecture into three different networks with different levels of image data and metadata interaction. Results for the first approach where the whole architecture is trained from randomly initialized weights and second one, where weights for image architecture are loaded from a trained model are summarized in Table 5.12.

Mode	**Architecture**	**Accuracy**
	No metadata	0.9341 ± 0.0022
Blank image model	Simple concatenation	0.9392 ± 0.0037
	Metadata interaction	0.9418 ± 0.0041
	More interaction	0.9378 ± 0.0061
Trained image model	Simple concatenation	0.9391 ± 0.0034
	Metadata interaction	$\mathbf{0.9432 \pm 0.0021}$
	More interaction	0.9424 ± 0.0024

Table 5.12: Accuracy for Sub100 with the *Al-Barazanchi* architecture embedded into different architectures with the use of metadata (time and shape). The best result is highlighted.

In the case of training a blank model, the best results were provided by the architecture performing more interaction amongst metadata only, the improvement was roughly 0.5 points. Other two architectures also improved accuracy; however, difference was not significant. The second approach, where a trained model was used, managed to increase the accuracy with a minimum boost of 0.5 points for all three architectures, which proved that including metadata to an already trained image model is a better practice, as the boost in accuracy is larger and training time is reduced. Simply concatenating metadata with image data just before the last layer resulted in the least gain in accuracy in this case. Adding more interaction to metadata only and adding more interaction to final part of the network had similar effect. However, the former method had a slightly better improvement for the original model with an increase of 0.9 points. This model with more metadata interaction was further evaluated with different combinations of metadata, results can be seen in Table 5.13.

Metadata	**Accuracy**
No metadata	0.9341 ± 0.0022
Time	0.9433 ± 0.0025
Shape	0.9414 ± 0.0036
Time and Shape	$\mathbf{0.9432 \pm 0.0021}$

Table 5.13: Accuracy for Sub100 with *Al-Barazanchi* architecture embedded into architecture with more metadata interaction. Different combinations of metadata were evaluated. The best result is highlighted.

In these experiments *time* and *shape* had similar effect on improving accuracy, however their combination indeed had the best outcome. Finally, model with more interaction among metadata only with both *time* and *shape* included was applied to the *InceptionV3* shape that proved to be the most accurate in previous tests. With metadata the achieved accuracy was $(95.22 \pm 0.21)\,\%$, which compared to original accuracy of $(95.20 \pm 0.13)\,\%$ without any metadata does not provide any further improvement. This is possibly due to the already enormous complexity of the *InceptionV3* architecture.

Patch cropping CNN results

Three different methods of patch cropping were evaluated, where a single patch was fed to the *Al-Barazanchi* model and a pair of patches was processed by *DeepWriter* created from two *Al-Barazanchi* networks. For each method different number of iterations of patch cropping was performed during testing. Results from these tests can be seen in Table 5.14.

With enlarging the number of iterations, accuracies for methods increase, however the time for evaluation is gradually increasing as well. While switching from 8 to 16 patches there is no significant improvement during testing for none of the methods, whereas evaluating one image takes twice as much time. From these findings four iterations were chosen as the best quantity for good performance. With any given number of iterations method using a pair of patches outperforms the others. Even though many samples were split into two halves, for 4 iterations its score was 1.7 points better than its modified version as well as 0.9 points better than using a single patch only. This suggests that the DeepWriter architecture indeed benefits from having extra spatial information preserved by selecting two consecutive patches. DeepWriter even outperformed the original Bazaranchi model by

Number of patches	Single patch	Patch pair	Patch pair mod.
2	0.8987 ± 0.0045	**0.9298 ± 0.0030**	0.9219 ± 0.0057
4	0.9285 ± 0.0052	**0.9370 ± 0.0025**	0.9257 ± 0.0062
8	0.9301 ± 0.0050	**0.9392 ± 0.0017**	0.9276 ± 0.0063
16	0.9299 ± 0.0042	**0.9420 ± 0.0021**	0.9289 ± 0.0059

Table 5.14: Accuracy for different methods of patch cropping for *Al-Barazanchi* and the *DeepWriter* architecture created from *Al-Barazanchi*. Method of single patch performs twice as many iterations than other methods in each row. The best method is highlighted.

0.5 points, which was processing the whole image at a time. This could mean that this method leverages little details that are being lost due to resizing, as was intended. Finally, InceptionV3 was converted into DeepWriter shape, and it was trained with non-modified version of cropping a pair of patches. This time the achieved accuracy was (95.28 ± 0.09) % which is very similar to the original.

Multi-stream CNN results

This section contains results obtained from combining previous solutions. The outcome for using the *Al-Barazanchi* architecture as the base model can be viewed in Table 5.15. Combination of the *InceptionV3* architecture with other models is in Table 5.16. In all cases accuracy was better with combination of multiple models than using only one CNN on its own.

Model combination	Accuracy
Al-Barazanchi (224x224)	0.9341 ± 0.0022
Al-Barazanchi (224x224) + Jeffrey (128x128)	0.9404 ± 0.0012
Al-Barazanchi (224x224) + Al-Barazanchi_2 (361x181)	0.9439 ± 0.0024
Al-Barazanchi (224x224) + Al-Barazanchi_4 (448x112)	0.9383 ± 0.0031
Al-Barazanchi (224x224) + Al-Barazanchi_2 (361x181) + Al-Barazanchi_4 (448x112)	0.9444 ± 0.0022
Al-Barazanchi (224x224) + DeepWriter 2x(224x224)	0.9488 ± 0.0015
Al-Barazanchi (224x224) + DeepWriter 2x(224x224) + Al-Barazanchi_2 (361x181)	**0.9499 ± 0.0018**
Al-Barazanchi (224x224) + DeepWriter 2x(224x224) + Al-Barazanchi_4 (448x112)	0.9466 ± 0.0024

Table 5.15: Accuracy for different combinations of architectures for *Al-Barazanchi* (*Sub100*). *DeepWriter* in these tests is created with two *Al-Barazanchi* networks. The best method is highlighted.

The best improvement for *Al-Barazanchi* was found in combining it together with either *Barazanchi_2* or *DeepWriter*, this way test score was improved by 1.4 points. This suggests that combining CNNs where each one is targeted on images with different aspect ratios can result in significant boost in accuracy for a dataset with this huge diversity. Using a method that leverages patch cropping proved to be more effective than CNNs that are fed with whole images of larger aspect ratios. The best score was achieved by combining

Al-Barazanchi with two more models – *DeepWriter* and *Al-Barazanchi_2*, resulting in 1.6 points improvement.

Model combination	**Accuracy**
InceptionV3 (299x299)	0.9520 ± 0.0014
InceptionV3 (299x299) + Jeffrey (128x128)	0.9519 ± 0.0012
InceptionV3 (299x299) + InceptionV3_2 (420x210)	0.9577 ± 0.0011
InceptionV3 (299x299) + InceptionV3_4 (600x150)	0.9562 ± 0.0020
InceptionV3 (299x299) + InceptionV3_2 (420x210) + InceptionV3_4 (600x150)	0.9596 ± 0.0005
InceptionV3 (299x299) + DeepWriter 2x(299x299)	0.9580 ± 0.0023
InceptionV3 (299x299) + InceptionV3_2 (420x210) + DeepWriter 2x(299x299)	**0.9616 ± 0.0008**
InceptionV3 (299x299) + InceptionV3_4 (600x150) + DeepWriter 2x(299x299)	0.9606 ± 0.0002

Table 5.16: Accuracy for different combinations of architectures for *InceptionV3* (*Sub100*). *DeepWriter* in these tests is created with two *InceptionV3* networks. The best method is highlighted.

Similar improvement with these combinations can be observed for the *InceptionV3* architecture. Here again in the case of combination with one other model, *InceptionV3_2* and *DeepWriter* (created from two *InceptionV3* networks) proved to provide the largest boost due to their focus on samples with larger aspect ratio. Combination with DeepWriter had 0.8 point improvement. The best combination of three models was obtained with *InceptionV3* together with *InceptionV3_2* and *DeepWriter*, here the improvement was up to 1 point. Finally, metadata were included for *InceptionV3* and *InceptionV3_2* and these two models were evaluated together with DeepWriter. Included metadata resulted in accuracy of 96.10 %, which did not further improve accuracy of the previous model. Again, this is possibly due to already enormous complexity of the *InceptionV3* architecture. Application of different methods with the highest achieved accuracy for *Al-Barazanchi* and *InceptionV3* architectures are summarized in Table 5.17.

As can be seen in this table, for the *Al-Barazanchi* architecture every method improved its accuracy, whereas in the case of *InceptionV3* only application of the multi-stream method had a significant effect. This is possibly due to the already high accuracy of *InceptionV3* as well as its enormous complexity. It seems that addition of the spatial pyramid pooling layer or patch cropping can increase success rate for simpler networks such as *Al-Barazanchi*, however the improvement ended up lower than expected. For the same model, incorporating metadata had the largest positive impact with almost no impact on the evaluation speed. The best achieved success rate was achieved with the multi-stream architecture, as the improvement was observed for both architectures. This suggests that the combination of multiple models, where each one is focused on images of specific aspect ratio is a good solution for such a diverse dataset such as plankton images.

Model combination	**Accuracy**
Al-Barazanchi	0.9341 ± 0.0022
Al-Barazanchi-SPP	0.9387 ± 0.0052
Al-Barazanchi-Metadata	0.9432 ± 0.0021
Al-Barazanchi-DeepWirter	0.9392 ± 0.0017
Al-Barazanchi-Multi-stream	**0.9499 ± 0.0018**
InceptionV3	0.9520 ± 0.0014
InceptionV3-SPP	0.8761 ± 0.0153
InceptionV3-Metadata	0.9522 ± 0.0021
InceptionV3-DeepWriter	0.9528 ± 0.0009
InceptionV3-Multi-stream	**0.9616 ± 0.0008**

Table 5.17: Comparison of the best achieved accuracies with different methods for *InceptionV3* and *Al-Barazanchi* architectures (*Sub100*).

Chapter 6

Discussion

6.1 Current study

One of the objectives of this work was to build a CNN that can learn features of plankton images in the dataset collected using imaging FlowCytobot. As the dataset is highly imbalanced, three different subsets were created containing classes with minimal number of samples of 100, 50 and 10. These classes were further balanced to have the same number of images through data augmentation. Initial testing was performed with a VGG-16 based architecture introduced in a work by Hussein Al-Barazanchi et al. [2], where multiple different ways of data augmentation were examined together with data preprocessing and training parameters, that give the best results. For data augmentation it was observed that adding cropping, blurring and Gaussian noise had major positive impact on the accuracy. With this setting multiple other architectures were examined as well, including popular CNNs like *AlexNet*, *InceptionV3*, *DenseNet*, *ResNet* and *MobileNet* together with some modified architectures that proved to be useful in other works dealing with plankton recognition. Here it was observed that deeper networks achieve better accuracy. *InceptionV3*, although with the longest training time, proved to learn plankton features better than the others with a score of 95.20 %. In the case of simpler CNNs, the *Al-Barazanchi* architecture performed the best with accuracy of 93.41 % and with much shorter training time in comparison with *InceptionV3*, therefore it was chosen as a testing architecture for finding best approaches in the next objectives of this work, which were incorporation of metadata and studying possible solutions for dealing with various sizes and aspect ratios of images in the dataset. Both *Al-Barazanchi* and *InceptionV3* were further examined with modifying them to accept images of different aspect ratios (e.g. *Al-Barazanchi_2*, that focuses on ratio of $2:1$ or *Al-Barazanchi_4*, that focuses on ratio of $4:1$).

With the next objective of incorporating metadata into the model, possible data that could be used were examined, together with some architectures studied in the literature. For this task, the original shape of the image was leveraged as well as the time of sample acquisition, and both of them were transformed into multiple values. The *Al-Barazanchi* model was embedded into three different networks with varying levels of interaction between image data and metadata. Embedding of the model was examined in two ways, that is loading a trained model with fixed parameters or creating a new blank model that needs to be trained. With the obtained results it was evident that combination of loading a trained image model and embedding it inside of a network with more interaction among metadata had the best effect on the model's accuracy, *Al-Barazanchi* was improved by 0.9 percentage

points. Furthermore, *InceptionV3* was also evaluated in the same manner, however the accuracy was not further improved.

Another objective was creating a CNN that is focused on images of varying size and aspect ratio. Patch cropping approach was used in multiple ways in combination with a new architecture called *DeepWriter* which accepts a pair of patches to preserve spatial information. This proved to be useful as it outperformed a method where only a single patch was cropped at a time. The best gain in accuracy with evaluation time concerned was achieved with 4 iterations, that is cropping of 8 patches in total. This way *DeepWriter* even improved accuracy of the original *Al-Barazanchi* architecture by small margin of 0.3 points. The *InceptionV3* model was also transformed into *DeepWriter*, where the accuracy was very similar to the original *InceptionV3* with 0.1 points improvement. Next method that was studied was inserting a spatial pyramid pooling layer into the CNN and therefore enable training of the network with various image sizes. *Al-Barazanchi* and *InceptionV3* models were evaluated with different combinations of image sizes. The SPP layer on its own did not improve accuracy of the networks as was presented in studied literature, neither did combinations of two sizes. Only a combination of three sizes managed to improve the accuracy of the *Al-Barazanchi* architecture by 0.5 points. However, in the case of *InceptionV3* no score improvement was achieved. Finally, combination of several models was tested, where each model was focused on distinct aspects of the images and their predictions were averaged into a final output vector. In the case of combination of two models, *Al-Barazanchi* prospered the most with its *DeepWriter* modification – accuracy was improved by 1.5 points to 94.88 %. In the case of three models the best score of 94.99 % was achieved with combination of *Al-Barazanchi*, *Al-Barazanchi_2* and *DeepWriter*. For the *InceptionV3*, combination with its modified version of *DeepWriter* once again proved to be the best approach, this way the success rate was increased by 0.6 points and the combination with *InceptionV3_2* and *DeepWriter* resulted in 1 point boost, giving the best achieved accuracy in this work of 96.16 %.

6.2 Future work

In this work multiple popular architectures for other fields were examined that proved to have good performance in case of classifying phytoplankton as well. These models were trained with randomly initialized weights, for this reason it took quite a lot of time for training them. Due to time limitation, fine-tuning of these architectures could not be examined – apart from faster training it would be interesting to see if it also has some positive impact on the final accuracy.

Incorporation of metadata also improved accuracy of the models. However, these data were very limited, as only the original shape and time of acquisition were leveraged, adding just four new values. Success rate could be probably further improved with more available information including for example GPS coordinates or parameters of cytometer used for plankton imaging.

Another method, that was discussed during this work, was considering class activation maps and their possible use to locate regions that are more distinctive for the recognition and subsequently cropping patches in higher resolution that would be supplied to the model. However, it is questionable if such a method could be used in this case, as most of the samples are of small dimensions.

Chapter 7

Conclusion

The objective of this work was to implement a CNN based classifier that is capable of labeling images of phytoplankton and further improve it with different methods, which was successfully achieved. Many distinct architectures were evaluated, both from few selected works dealing with plankton classification and some commonly used shapes. From these, a VGG-16 based architecture called *Al-Barazanchi* was chosen, as it had a very good accuracy and could be used for fast testing of different methods thanks to its simplicity. However, the best score was achieved by deeper networks, with *InceptionV3* being the best.

With a baseline CNN selected, next objective was to examine different approaches to further improve its accuracy, one way was by incorporating metadata into the network. This was accomplished by examining multiple different architectures with varying level of interaction between image data and metadata, as well as different combinations of metadata including the image shape and time of acquisition. The inclusion of metadata, despite its small number, proved to be useful as it increased models score.

Other methods were focused on varying image sizes and aspect ratios, as this is quite a problematic aspect of plankton species. The first examined method leverages the use of a spatial pyramid pooling layer inside of the architecture and therefore it enables its training with variable input size. Such a model should be more scale-invariant, which indeed improved *Al-Barazanchi* success rate by a small margin. However, it was not very helpful in the case of *InceptionV3*. The next approach consisted of cropping patches from the images, that preserve as many details as possible. Different techniques of cropping were evaluated, where either a single patch or a pair of patches was cropped. In the case of a pair of patches a new model called *DeepWriter* was introduced, which also proved to be more successful than using a simple CNN.

Finally, a combination of previous approaches was studied in a form of a multi-stream CNN, where multiple different models were constructed, and all of their outputs were averaged into a single prediction vector. As previous methods did not improve overall accuracy by a larger margin on their own, putting multiple different models together, where each one is focused on images of a specific shape, proved to have a large boost in the prediction accuracy.

Possible ways for further improvement were discussed, such as fine-tuning different pretrained models, incorporation of more metadata, if available or using a class activation maps for more accurate patch cropping.

Bibliography

[1] Aghdam, H. H. and Heravi, E. J. *Guide to Convolutional Neural Networks A Practical Application to Traffic-Sign Detection and Classification*. 1st ed. Springer International Publishing, 2017. ISBN 9783319861906.

[2] Al-Barazanchi, H. A., Verma, A. and Wang, X. S. Intelligent plankton image classification with deep learning. *International Journal of Computational Vision and Robotics IJCVR*. 2018, vol. 8, no. 6, p. 561–571.

[3] Bengt, K., Felipe, A., Veronique, C., Arnaud, L., Guillaume, W. et al. *JERICO-NEXT. Novel methods for automated in situ observations of phytoplankton diversity. D3.1*. Report. 2017. Available at: https://archimer.ifremer.fr/doc/00422/53393/.

[4] Brownlee, J. *A Gentle Introduction to Early Stopping to Avoid Overtraining Neural Networks*. 2019. Accessed: 2020-05-20. Available at: https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/.

[5] Brownlee, J. *How to Configure Image Data Augmentation in Keras*. 2019. Accessed: 2020-05-20. Available at: https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks.

[6] Bueno, G., Deniz, O., Pedraza, A., Ruiz Santaquiteria, J., Salido, J. et al. Automated Diatom Classification (Part A): Handcrafted Feature Approaches. *Applied Sciences*. 2017, vol. 7, p. 753.

[7] Chollet, F. *Keras: Deep Learning for humans*. GitHub, 2015. Accessed: 2020-05-30. Available at: https://github.com/keras-team/keras.

[8] Chu, G., Potetz, B., Wang, W., Howard, A., Song, Y. et al. Geo-Aware Networks for Fine-Grained Recognition. In: IEEE/CVF. *International Conference on Computer Vision Workshop (ICCVW)*. 2019, p. 247–254. ISBN 978-1-7281-5023-9.

[9] Correa, I., Drews, P., Botelho, S., de Souza, M. S. and Tavano, V. M. Deep Learning for Microalgae Classification. In: IEEE. *16th IEEE International Conference on Machine Learning and Applications (ICMLA)*. 2017, p. 20–25. ISBN 978-1-5386-1418-1.

[10] Cortes, C. and Vapnik, V. Support-Vector Networks. *Machine Learning*. USA: Kluwer Academic Publishers. 1995, vol. 20, no. 3, p. 273–297. Available at: https://doi.org/10.1023/A:1022627411411. ISSN 0885-6125.

[11] Ellen, J. S., Graff, C. A. and Ohman, M. D. Improving plankton image classification using context metadata. *Limnology and Oceanography: Methods*. 2019, vol. 17, no. 8, p. 439–461. Available at: https://aslopubs.onlinelibrary.wiley.com/doi/abs/10.1002/lom3.10324.

[12] Fountas, Z. *Spiking Neural Networks for Human-like Avatar Control in a Simulated Environment*. 2011. Master's thesis. Department of Computing, Imperial College London.

[13] Gluckman, J. Scale Variant Image Pyramids. In: IEEE. *Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. 2006, p. 1069–1075. ISBN 0-7695-2597-0.

[14] Goodfellow, I., Bengio, Y. and Courville, A. *Deep Learning*. 1st ed. MIT Press, 2016. ISBN 9780262337373.

[15] Gurney, K. *An Introduction to Neural Networks*. 1st ed. Taylor & Francis, 1997. Available at: https://books.google.cz/books?id=HOsvllRMMP8C. ISBN 0-203-45151-1.

[16] He, K., Zhang, X., Ren, S. and Sun, J. Deep Residual Learning for Image Recognition. In: IEEE. *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, p. 770–778. ISBN 978-1-4673-8851-1.

[17] He, K., Zhang, X., Ren, S. and Sun, J. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. In: Fleet, D., Pajdla, T., Schiele, B. and Tuytelaars, T., ed. *European Conference on Computer Vision (ECCV)*. Cham: Springer International Publishing, 2014, p. 346–361. ISBN 978-3-319-10578-9.

[18] Henon, Y. *Spatial pyramid pooling layers for keras*. GitHub, 2017. Accessed: 2020-05-20. Available at: https://github.com/yhenon/keras-spp.

[19] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W. et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *ArXiv preprint arXiv:1704.04861*. 2017.

[20] Huang, G., Liu, Z., Van Der Maaten, L. and Weinberger, K. Q. Densely Connected Convolutional Networks. In: IEEE. *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2017, p. 2261–2269. ISBN 978-1-5386-0457-1.

[21] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J. et al. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. *ArXiv*. 2016. arXiv: 1602.07360.

[22] Ioffe, S. and Szegedy, C. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. 2015.

[23] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J. et al. Caffe: Convolutional Architecture for Fast Feature Embedding. *ArXiv preprint arXiv:1408.5093*. 2014, [cit. 2020-05-20].

[24] Jordan, M. I. and Mitchell, T. M. Machine learning: Trends, perspectives, and prospects. *Science*. American Association for the Advancement of Science. 2015, vol. 349, no. 6245, p. 255–260. Available at: https://science.sciencemag.org/content/349/6245/255. ISSN 0036-8075.

[25] Ke, H., Chen, D., Li, X., Tang, Y., Shah, T. et al. Towards Brain Big Data Classification: Epileptic EEG Identification With a Lightweight VGGNet on Global MIC. *IEEE Access*. 2018, vol. 6, p. 14722–14733.

[26] Krizhevsky, A., Sutskever, I. and Hinton, G. ImageNet Classification with Deep Convolutional Neural Networks. *Neural Information Processing Systems*. 2012, vol. 25.

[27] Krizhevsky, A., Sutskever, I. and Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In: Pereira, F., Burges, C. J. C., Bottou, L. and Weinberger, K. Q., ed. *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, p. 1097–1105. Available at: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf. ISBN 9781627480031.

[28] Krizhevsky, A., Sutskever, I. and Hinton, G. E. ImageNet Classification with Deep Convolutional Neural Networks. In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 1*. Red Hook, NY, USA: Curran Associates Inc., 2012, p. 1097–1105. NIPS'12. ISBN 9781627480031.

[29] Lecun, Y., Bottou, L., Bengio, Y. and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*. 1998, vol. 86, no. 11, p. 2278–2324.

[30] Masullo, A. *Patched Model Checkpoint*. GitHub, 2019. Accessed: 2020-05-20. Available at: https://github.com/keras-team/keras/issues/11101#issuecomment-502023233.

[31] Nielsen, M. A. *Neural Networks and Deep Learning*. Determination Press, 2015.

[32] Noord, N. and Postma, E. Learning scale-variant and scale-invariant features for deep image classification. *Pattern Recognition*. Feb 2016, vol. 61.

[33] Ojala, T., Pietikainen, M. and Harwood, D. Performance evaluation of texture measures with classification based on Kullback discrimination of distributions. In: *Proceedings of 12th International Conference on Pattern Recognition*. 1994, p. 582–585 vol.1. ISBN 0818662654.

[34] Olson, R. J. and Sosik, H. M. A submersible imaging-in-flow instrument to analyze nano-and microplankton: Imaging FlowCytobot. *Limnology and Oceanography: Methods*. 2007, vol. 5, no. 6, p. 195–203. Available at: https://aslopubs.onlinelibrary.wiley.com/doi/abs/10.4319/lom.2007.5.195.

[35] OpenCV. *About*. 2020. Accessed: 2020-05-20. Available at: https://opencv.org/about.

[36] Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J. et al. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In: *Advances in Neural*

Information Processing Systems 32. Curran Associates, Inc., 2019, p. 8024–8035. Available at: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf. ISBN 1510884475.

[37] Phan, T.-T.-H., Caillault, E. P. and Bigand, A. Comparative study on supervised learning methods for identifying phytoplankton species. *Sixth International Conference on Communications and Electronics (ICCE)*. IEEE. Jul 2016. Available at: http://dx.doi.org/10.1109/CCE.2016.7562650.

[38] Pūtaiao, P. A. Plankton. *Science Learning Hub*. 2009. Available at: https://www.sciencelearn.org.nz/resources/146-plankton.

[39] Raschka, S. and Mirjalili, V. *Python Machine Learning: Machine Learning and Deep Learning with Python, Scikit-Learn, and TensorFlow, 2nd Edition*. 2ndth ed. Packt Publishing, 2017. ISBN 1787125939.

[40] Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S. et al. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*. 2014, vol. 115.

[41] Sánchez, C., Vállez, N., Bueno, G. and Cristóbal, G. Diatom Classification Including Morphological Adaptations Using CNNs. In: Morales, A., Fierrez, J., Sánchez, J. S. and Ribeiro, B., ed. *9th Iberian Conference on Pattern Recognition and Image Analysis*. Cham: Springer International Publishing, 2019, p. 317–328. ISBN 978-3-030-31332-6.

[42] Sermanet, P., Eigen, D., Zhang, X., Mathieu, M., Fergus, R. et al. Overfeat: Integrated recognition, localization and detection using convolutional networks. In: *2nd International Conference on Learning Representations (ICLR)*. 2014.

[43] Shalev Shwartz, S. and Ben David, S. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014. ISBN 978-1107057135.

[44] Sieracki, M., Benfield, M., Hanson, A., Davis, C., Pilskaln, C. et al. Optical Plankton Imaging and Analysis Systems for Ocean Observation. *Procceedings in OceanObs'09: Sustained Ocean Observationsand Information for Society*. 2009, vol. 2, p. 21 – 25.

[45] Simon, N., Cras, A.-L., Foulon, E. and Lemée, R. Diversity and evolution of marine phytoplankton. *Comptes Rendus Biologies*. 2009, vol. 332, no. 2, p. 159 – 170. Available at: http://www.sciencedirect.com/science/article/pii/S1631069108002692. ISSN 1631-0691.

[46] Sournia, A., Chrdtiennot Dinet, M.-J. and Ricard, M. Marine phytoplankton: how many species in the world ocean? *Journal of Plankton Research*. september 1991, vol. 13, no. 5, p. 1093–1099. Available at: https://doi.org/10.1093/plankt/13.5.1093. ISSN 0142-7873.

[47] Springenberg, J. T., Dosovitskiy, A., Brox, T. and Riedmiller, M. A. Striving for Simplicity: The All Convolutional Net. *CoRR*. 2015, abs/1412.6806.

[48] Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S. et al. Going deeper with convolutions. In: IEEE. *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, p. 1–9. ISBN 978-1-4673-6964-0.

[49] Tutorials Point. *Sobel Operator*. Accessed: 2020-01-05. Available at: https://www.tutorialspoint.com/dip/sobel_operator.htm.

[50] Wicht, B. *Deep Learning feature Extraction for Image Processing*. Fribourg, Switzerland, 2018. Dissertation. Faculty of Science of the University of Fribourg.

[51] Xing, L. and Qiao, Y. DeepWriter: A Multi-stream Deep CNN for Text-Independent Writer Identification. In: Institute of Electrical and Electronics Engineers (IEEE). *15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*. Oct 2016, p. 584–589. ISSN 2167-6445.

[52] Yan, J., Li, X. and Cui, Z. A More Efficient CNN Architecture for Plankton Classification. In: Yang, J., Hu, Q., Cheng, M.-M., Wang, L., Liu, Q. et al., ed. *Chinese Conference on Computer Vision (CCCV)*. Singapore: Springer Singapore, 2017, p. 198–208. ISBN 978-981-10-7305-2.

[53] Zeiler, M. D. and Fergus, R. Visualizing and Understanding Convolutional Networks. In: Ferrari, V., Hebert, M., Sminchisescu, C. and Weiss, Y., ed. *European Conference on Computer Vision (ECCV)*. Cham: Springer International Publishing, 2014, p. 818–833. ISBN 978-3-319-10590-1.

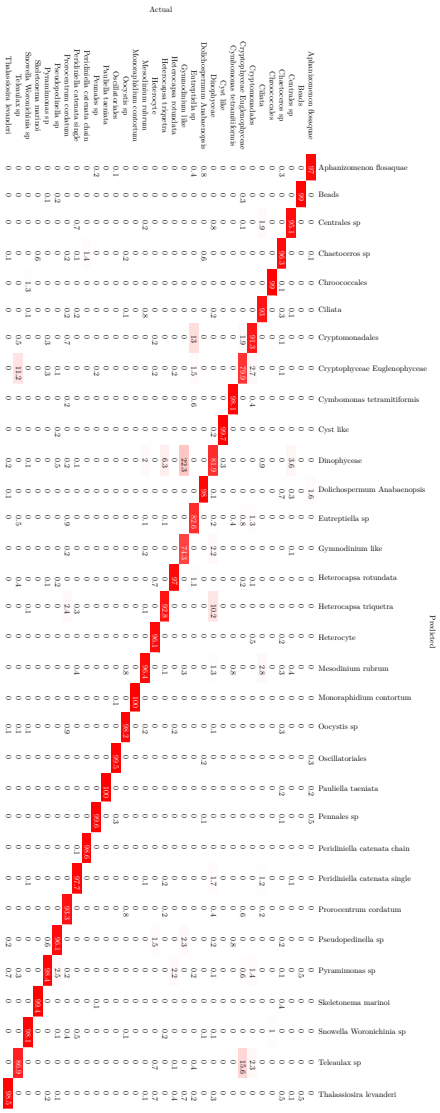
[54] Zhang, Y., Jin, R. and Zhou, Z.-H. Understanding bag-of-words model: A statistical framework. *International Journal of Machine Learning and Cybernetics*. december 2010, vol. 1, p. 43–52.

[55] Özmen, B. *AutoML for Data Augmentation*. 2019. Accessed: 2020-01-06. Available at: https://blog.insightdatascience.com/automl-for-data-augmentation-e87cf692c366.

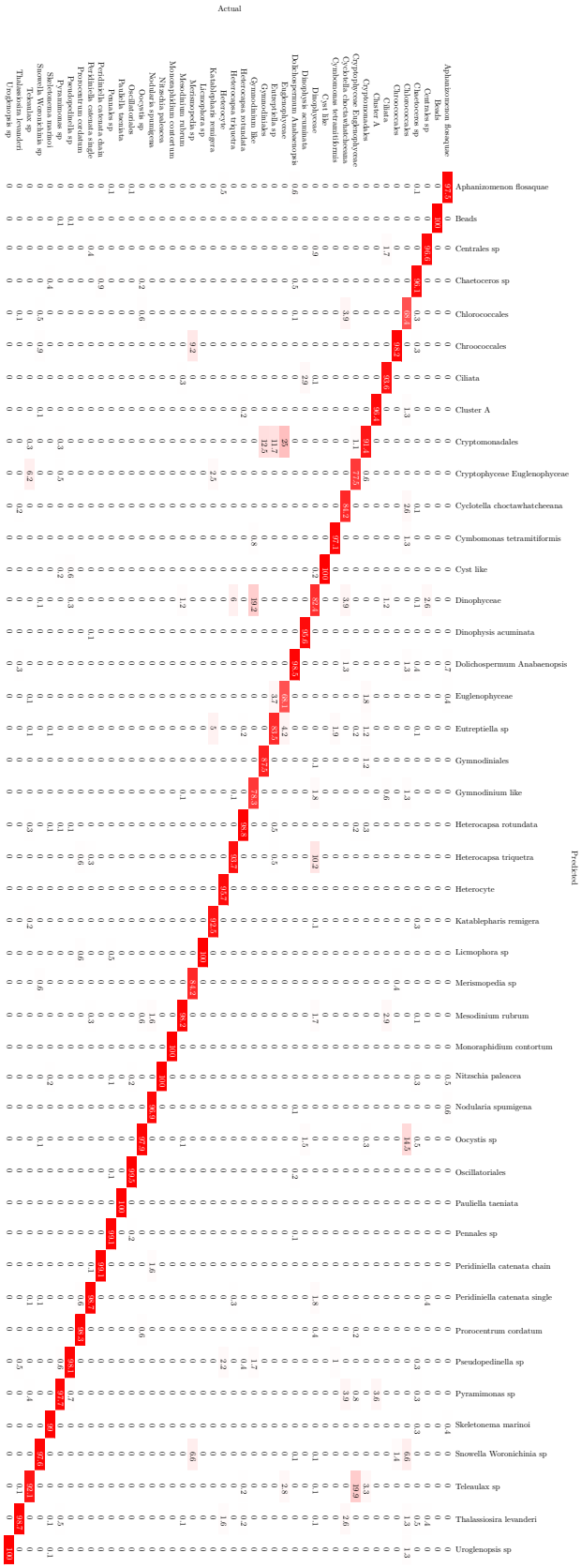
Appendix A

Confusion matrices

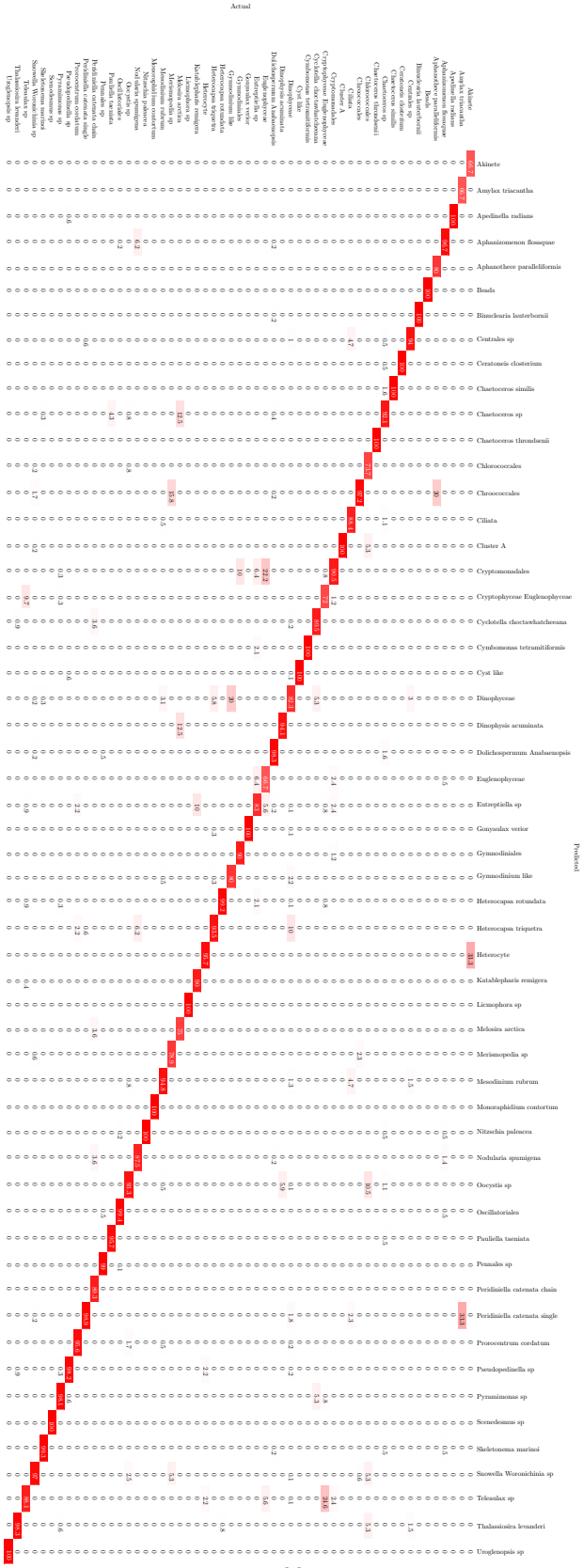
Confusion matrix for *InceptionV3* on *Sub100*



Confusion matrix for *InceptionV3* on *Sub50*



Confusion matrix for *InceptionV3* on *Sub10*



Appendix B

Use instructions

Installation

This project can be run by installing Anaconda and running following commands:

- conda create –name plankton_env
- conda activate plankton_env
- conda install tensorflow-gpu=1.14.0
- conda install -c menpo opencv
- conda install -c conda-forge matplotlib
- conda install -c anaconda scikit-learn
- conda install -c conda-forge keras
- conda install -c conda-forge albumentations

Prepare dataset for training

```
python source/recognizer.py -d DATASET_PATH -prepare PATH
-prepare_num NUMBER_OF_SAMPLES -min MINIMUM_SAMPLES
[-prepare_test_size NUMBER]
```

This command splits data into train and test folders and calculates mandatory metadata like mean and standard deviation of the set.

Parameters:

- *-d* – set path to original dataset
- *-prepare* – set directory, where new set will be created
- *-prepare_num* – set number of samples to be created per class for balancing
- *-prepare_test_size* – set size of test partition in percentage (Default is 20)
- *-min* – set filtering classes by minimum needed number of samples (by setting to number 100 only classes that have at least 100 samples are copied).

Train a model

```
python source/recognizer.py -ms MODEL_PATH -d DATASET_PATH
-image_model IMAGE_MODEL [-meta_model META_MODEL]
[-train_model TRAIN_MODEL] [-b BATCH_SIZE] [-i SIZES_LIST]
[-metas METADATA_LIST] [-e EPOCHS] [-folds FOLDS] [-mr PATH]
[-mr_fix] [-ext EXTENSION] [-U] [-multiprocessing]
[-workers_num NUMBER] [-max_queue_size NUMBER]
```

Parameters:

- *-d* – path to prepared dataset
- *-ms* – output directory for saving a trained model
- *-mr* – path to reference model from which should be weights loaded before training
- *-mr_fix* – set weights of reference model to be fixed
- *-image_model* – name of an architecture to be constructed for image data.
- *-meta_model* – name of an architecture to be constructed for metadata
- *-train_model* – name of image parsing method
- *-b* – batch size
- *-e* – number of epochs
- *-ext* – extension of image files. E.g. *-ext „png“*.
- *-folds* – number of folds for cross validation
- *-i* – list of sizes that the images will be resized to. E.g. *-i „(256,256)“ „(128,128)“*.
- *-metas* – list of metadata to be used for training. E.g. *-metas shape time*.
- *-U* – force to recalculate metadata in dataset.
- *-multiprocessing* – enable multiprocessing during training
- *-workers_num* – number of workers for multiprocessing
- *-max_queue_size* – maximum queue size for workers

Evaluate a trained model

```
python source/recognizer.py -ml MODEL_PATH -d DATASET_PATH -o OUTPUT_PATH
[-ext EXTENSION] [-U]
```

Parameters:

- *-d* – path to prepared dataset
- *-ml* – directory containing models to be evaluated
- *-o* – output directory
- *-U* – force to recalculate metadata in dataset.
- *-ext* – extension of image files. E.g. *-ext „png“*.