University of Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

MASTER'S THESIS

Study programme/specialisation: Information Technology - Automation and Signal Processing	Spring/ ~~Autumn~~ semester, 2020 Open / ~~Confidential~~
Author: Aleksander Borge Nesse	
Programme coordinator: Professor Kjersti Engan Supervisor(s): Professor Kjersti Engan, Robert Williams	
Title of master's thesis: Classifying Dinoflagellates in Palynological Slides Using Convolutional Neural Networks	
Credits: 30	
Keywords: Deep Learning, Convolutional Neural Networks, Image Processing, Object Detection, Transfer Learning, Dinoflagellates, Microplankton, Fossils, Palynology	Number of pages: 76 + supplemental material/other: 26 pages and embedded file Stavanger, 14.07/2020 date/year

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Classifying Dinoflagellates in Palynological Slides Using Convolutional Neural Networks

Master's Thesis in Information Technology - Automation and Signal Processing
by

Aleksander Borge Nesse

Internal Supervisor

Kjersti Engan

External Supervisor

Robert Williams

July 14, 2020

Abstract

The petroleum industry is still one of the largest contributors to the Norwegian economy. Experts estimates that of the total reserves on the Norwegian shelf only 52 percent have been discovered. During test drilling, core samples can be taken from the sedimentary rock and within these samples small fossils from micro-plankton known as dinoflagellates can be found. By evaluating the distribution and collection of different species and taxon of dinoflagellate the likelihood of finding petroleum in the area can be estimated.

Palynology is the study of such small objects, and have largely been done manually through a microscope. The Norwegian Petroleum Directorate have recently acquired a scanner to digitize their collection of over 200,000 palynological slides. In this thesis a solution is proposed to automatically detect and identify a number of different dinoflagellate species by using both traditional image processing and deep neural networks.

With the aid of traditional image processing a detection rate of 93 percent was obtained for detecting objects in the palynological slides. Using transfer learning, a deep convolutional neural network based on the VGG-16 network structure obtained a 99 percent accuracy on test data.

Preface

This thesis marks the end of my degree as Master of Science at the University of Stavanger, Department of Electrical Engineering and Computer Science.

I would like to thank my lecturers and co-students for the knowledge, help, and fun for the few last years. A special thanks goes out to Professor Kjersti Engan for the support, guidance, and reassurance during the writing of this thesis. I would also like to thank the Norwegian Petroleum Directorate for providing the material used in this thesis.

A final thanks to my family and friends for the support and encouragement, and to my partner Bettina for enduring with me during the quarantine and lockdown period caused by covid-19.

Contents

Chapter 1

Introduction

1.1 Palynology in Petroleum Prospecting

The petroleum industry is by far the biggest contributor to the Norwegian economy. Over 170,000 are employed directly or indirectly by the industry, and it is alone responsible for around ten percent of Norway's GDP, a total of over 14,000 Billion NOK[1] since the start of production in the 1970's. In 2019, the export of Norwegian crude oil was valued at 248 Billion NOK, approximately 27 percent of all Norwegian goods export [1], however only covering about two percent of the entire world's oil requirement. Even though many industries are transitioning to other, greener types of energy, petroleum will still remain one of the largest resources in the years to come. It has been estimated that of all the total reserves on the Norwegian shelf, 48 percent has yet to be discovered [2].

Offshore petroleum prospecting requires a wide range of special fields and techniques within geology. To survey the ocean floor and the underlying rock formation, both geophysical and sedimentological methods are used. A common first step is to survey using seismology. A seismic vessel sends out powerful sound-waves over a large spectrum down towards the sea bed and are reflected in the transition between the layers of different rock types. First when a suitable area is found, exploratory drilling is performed. Geological experiments and core samples can then give more information about the rock type, age as well as the probability of petroleum in the surrounding area [3], [4].

The study and surveying of sedimentary strata, stratigraphy, have been around since before the 1700's, but it was the discovery of William Smith (1769 - 1839) that made it possible to map the distribution over large geographical areas using biostratigraphy. While tasked to survey routes for a planned coal canal in 1795, he discovered that

[1]Adjusted for inflation

some of the strata contained identifiable fossils, and noticed how the collection of fossils changed depending on the depth and layer of the sedimentary rock [5]. Geologist still use biostratigraphy to chart the age of the strata, but in offshore petroleum surveying, both the depth and shape of the drilling greatly limits the size of the core sample. Because of this, geologists have been forced to identify extremely small, microscopic fossils. These types of fossils are mostly composed of marine microplankton such as dinoflagellates, as well as spores and pollen from land plants. These are part of the organic material that make up the sedimentary rock which is transformed to petroleum under the right pressure and temperature [6].

Palynology is a field within geology which is the study of such microfossils. By examining the number and ratio between different species, these can be presented in a range-chart. A range-chart shows the differences and relationships between species, such that the strata's can be divided to zones and the age can be determined. Dinoflagellates, in a geological time scale, develop new species fast, as well as old species become extinct, which makes them especially suited for dating marine sediment. Spores and pollen however, often exists for extended periods of time, and is therefore unable to give a precise dating [6]–[9]

Comparing the different ratios of microfossils makes it possible to determine if the sediment originates from land, the coast or sea, as well as living condition and even the temperature. By comparing the dinoflagellates of old with newer samples, ocean temperature and living conditions can be estimated [6]–[9].

1.2 Previous Work

Machine learning have been a hot topic within many fields of research, microbiology included. All though these are mostly centered around living or newer species instead of microfossils, the issue remains the same. Identifying objects such as plankton, dinoflagellates, spores and other microscopical organisms can provide vital information about the environment, both present day and the long-lost past. Currently, many of these identifications are done manually through a microscope which is a both a time consuming and laborious task.

Automatic recognition systems to classify different taxa and species of plankton have been introduced even as early as 1984, when Jeffries, Berman, Poularikas, et al. [10] presented a pattern recognition system by feature extraction on different zooplankton. In 1998, Tang, Kenneth Stewart, Vincent, et al. [11] devised a system to extract features from plankton images using invariant moment feature and Fourier boundary descriptors and trained a small neural network classifier. V, Reguera, González-Gil, et al. [12] published in 2002, DiCANN, a network to automatically categorize 23 different species of dinoflagellates from microscopic images using features extracted by using the Fourier power spectrum and texture density.

Later work include, by Schulze, Tillich, Dandekar, et al. [13] - PlanktonVision, a system using local binary pattern, elliptic Fourier descriptors and the histogram to extract features such as texture, shape, size and pigmentation from plankton, and using a deep neural network with two hidden layers as a classifier. And by Zheng, Wang, Yu, et al. [14], in 2017 using multiple kernel learning to classify plankton from features extracted by using ten different extraction methods.

1.3 Thesis Objective

As of 2020, the Norwegian Petroleum Directorate (NPD) have over 200,000 palynological slides (palyslides) collected from over one thousand well drilling, with fossils ranging from 3 to 370 million years old. The NPD is in the process of digitizing their collection and are interested in the possibility of using image processing or machine learning to aid geologists in their work.

The main objective of this thesis is to explore the possibility of detecting and classifying different species of dinoflagellates from palyslide images. To do this, a system to detect both the position and size of objects from the palyslide images is created by using traditional image processing techniques. By extracting annotated dinoflagellates from palyslide images, a deep convolutional neural network can be trained to classify dinoflagellates from a range of species.

1.4 Thesis Overview

Chapter 2 - Background
In this chapter, relevant background theory is presented.

Chapter 3 - Data and Materials
In this chapter, the dataset, as given from the Norwegian Petroleum Directorate is presented. The procedure of creating a palynological slide is discussed and the final dataset used in the thesis is presented.

Chapter 4 - Proposed Method
In this chapter, the method for detecting and classifying objects, as well as preprocessing the dataset is presented.

Chapter 5 - Experiments and Results
In this chapter, the conducted experiments and their results are presented. First the object detection algorithm and its performance and later tuning and performance evalutation of the deep neural network.

Chapter 6 - Discussion
In this chapter, the challenges and limitations of the proposed method is discussed together with the results of the conducted experiments.

Chapter 7 - Conclusion
In this chapter, a summary and conclusion of the thesis is given, as well as suggestions for future work and potential improvements.

Chapter 2

Background

2.1 Dinoflagellates

As mentioned in the introduction in chapter 1, dinoflagellates (shown in figure 2.1) are a type of marine microplankton. Mostly considered as a type of algae, dinoflagellates are single-celled eukaryotes and comprise a large proportion of the planktonic biomass in both marine and freshwater environment [9].

Some species of dinoflagellates perform a resting stage as a part of their life cycle, transforming into a dinoflagellate cyst (dinocyst) composed of dinosporin. At this stage they are capable of being highly preserved in sedimentary rock [7].

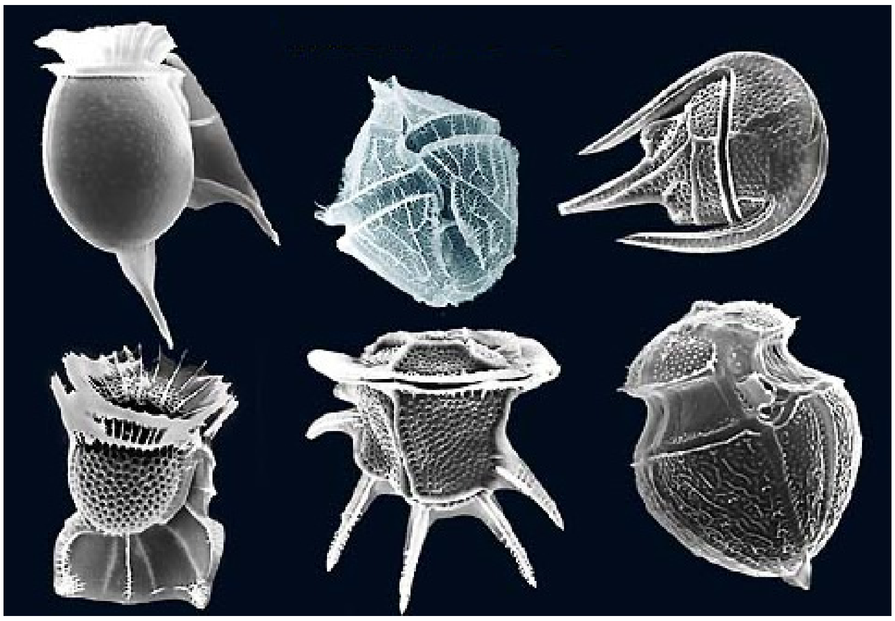


Figure 2.1: Example of different species of dinoflagellates `This figure is reprinted in unaltered form from Wikimedia Commons, File: Dinoflagellates.jpg. Licensed under CC BY 2.0 by the user fickleandfreckled`

2.2 Whole Slide Imaging

Whole slide imaging or digital microscopy refer to scanning of a complete microscopic slide. Usually they are created by stitching many smaller images, creating a single high-resolution image file. As these images are large in resolution, they also contain downsampled versions of the image, allowing for greater overview without having to import the entire image to memory or process downsampling. High-end digital slide scanners are also able to scan images using several focus points, enabling the user to manual focus while viewing the image on a computer [15]–[17].

2.2.1 OpenSlide

As there is no standardized whole slide image format, many vendors create proprietary or use closed undocumented formats, making it difficult to use these images outside the vendors viewer and applications. OpenSlide is an open source C-library (with Python bindings) which support many of these formats, enabling users to work with many different formats without having to depend on vendor-specific software [18], [19].

2.3 Image Processing

2.3.1 Bit Depth and Color Models

A digital image is created by combining smaller picture elements known as pixels. When displayed on a monitor, each pixel in an image is usually represented by red, green, and blue sub-pixels. This is the RGB color model, and each pixel is digitally represented by the value of each red, green, and blue channels. How many different values a single pixel can have, i.e. how many colors can be displayed, is known as the bit depth. With a 24-bit bit depth, each channel can be represented by 256 different values or 8-bit. This gives a total of over 16.7 million different color combinations that can be displayed.

An image with a bit depth of one is known as a binary image, as each pixel can only be represented by one of two values. Gray-scale images have a bit depth of more than one and can allow for significantly more than only two values but are defined with only one color channel.

Processing images based on color can be challenging when using the RGB-model. The way we think of colors is not by their combination of the primary colors, but their hue and saturation. A more intuitive way to represent colors images is therefore the HSV-model.

This model is a direct conversion of the RGB-model, but instead of portraying a pixel from its combination of primary colors, it defines a pixel from its hue, saturation and value as shown in figure 2.2.

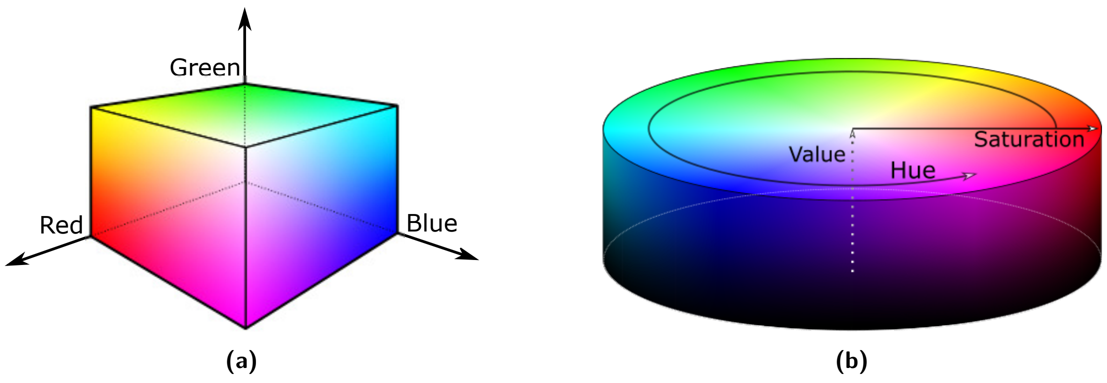


Figure 2.2: Spatial representation of the **(a)** RGB-model and **(b)** HSV-model.

2.3.2 Mathematical Morphology

With mathematical morphology, an image is interpreted as a set of pixels, and common functions from set theory are utilized to manipulate and transform images. Originally it was developed for use on binary images, but its use has later been extended to also include grayscale images as well as continuous functions [20].

The morphological functions have two inputs, the image to be processed and a structuring element[1]. These are then combined using set operators, such as intersection and union.

Unlike linear filters, morphological operators do not use cross-correlation or convolution, but rather the applied set operator together with its structuring element. The structuring element slides over an image, and at each pixel its elements are compared with the set of underlying values. If the sets of elements match the condition defined by the set operator the resulting pixel will be set to a pre-defined value [21].

The structuring element can have different shapes and sizes, as shown in figure 2.3, depending on the desired result. Some morphological function, e.g. the Hit-and-miss transform, uses more advanced structuring elements to detect corners of figures, while the most basic is a 3 by 3 matrix containing only 1's. The structuring element usually has its origin or "anchor" in the center, but some operators may use different origins.

[1]Often (mis)called a kernel, however some believe this should be reserved for convolutional and cross-correlational functions

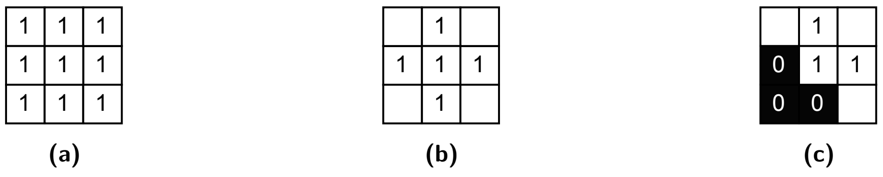


Figure 2.3: Example of structuring elements used by mathematical morphology functions **(a)** and **(b)**: Basic and commonly used structuring elements. **(c)**: Structuring element used by the Hit-and-Miss transform for corner detection

2.3.2.1 Erosion and Dilation

The two base operators within mathematical morphology are called erosion and dilation, usually denoted with $\ominus$ and $\oplus$ respectively. Performed on a binary image, they are equivalent to the boolean functions AND and OR with regards to the structuring element.

The **erosion** of a binary image $\mathcal{X}$ by the structuring element $\mathcal{B}$ is defined as the set of all points z, such that $\mathcal{B}$, translated by z is contained in $\mathcal{X}$, as shown in equation 2.1 [22]

$$\mathcal{X} \ominus \mathcal{B} \triangleq \{z : \mathcal{B}_{+z} \subseteq \mathcal{X}\} = \bigcap_{y \in B} \mathcal{X}_{-y} \tag{2.1}$$

Expanded from binary to gray scale images, this can be implemented such that for a given pixel at position (x, y) in the original (src) image, the resulting eroded pixel $dst(x, y)$ is the *minima* in the region $(x + x', y + y')$ as defined by the structuring element superimposed on the original image, as shown by equation 2.2. In the resulting dst image, equation 2.2 is performed for all pixels in the source image [23], [24].

$$dst(x, y) = \min_{(x',y'):element(x',y') \neq 0} src(x + x', y + y') \tag{2.2}$$

The **dilation** of a binary image $\mathcal{X}$ by structuring element $\mathcal{B}$ is defined as the set of all points z such that the intersection of $\mathcal{B}^s$, the symmetric of $\mathcal{B}$ with respect of the anchor point, translated by z, and $\mathcal{X}$ is non-zero as shown by equation 2.3 [22].

$$\mathcal{X} \oplus \mathcal{B} = \{z : (\mathcal{B}^s)_{+z} \cap \mathcal{X} \neq \emptyset\} = \bigcup_{y \in B} \mathcal{X}_{+y} \tag{2.3}$$

Similarly to erosion, this can be implemented for gray scale images as taking the *maxima* in the set of pixels in the region defined by the structuring element [23], [25].

$$dst(x, y) = \max_{(x',y'):element(x',y') \neq 0} src(x + x', y + y') \tag{2.4}$$

When applied to images, the erosion function removes the outermost layer of a shape and the dilation expands it. Figure 2.4 shows the erosion and dilation functions applied on a small image of an 'A' using the simplest form of structuring element, a 3 by 3 matrix with 1's and its anchor in the center such as shown in figure 2.3a.

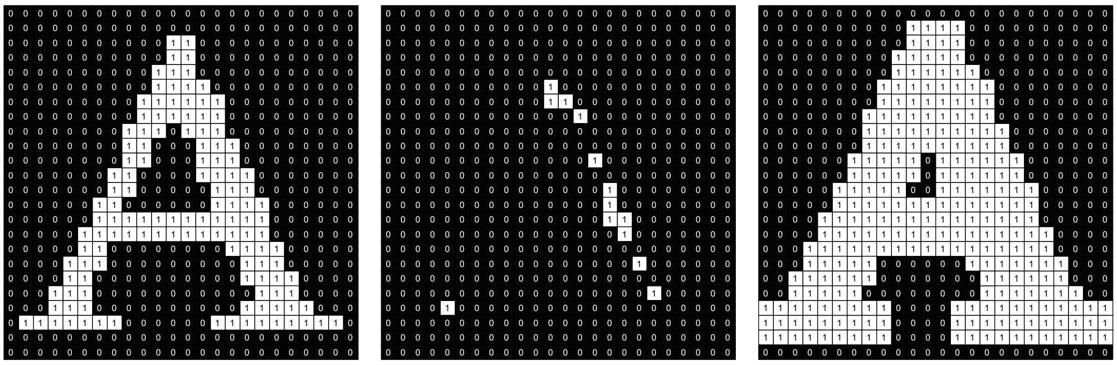

(a) Original image **(b)** Erosion **(c)** Dilation

Figure 2.4: Mathematical morphological basic operations performed with a 3 by 3 kernel composed of only 1's

Erosion and dilation are known as dual functions such that the erosion of the foreground is equivalent to the dilation of the background in an image, as shown by equation 2.5 [22], [26].

$$\mathcal{X} \oplus \mathcal{B} = (\mathcal{X}^C \ominus \mathcal{B}^S)^C \tag{2.5}$$

2.3.2.2 Opening and Closing

Opening and closing are two basic functions for noise reduction using mathematical morphology and are composed of sequential operations of erosion and dilation.

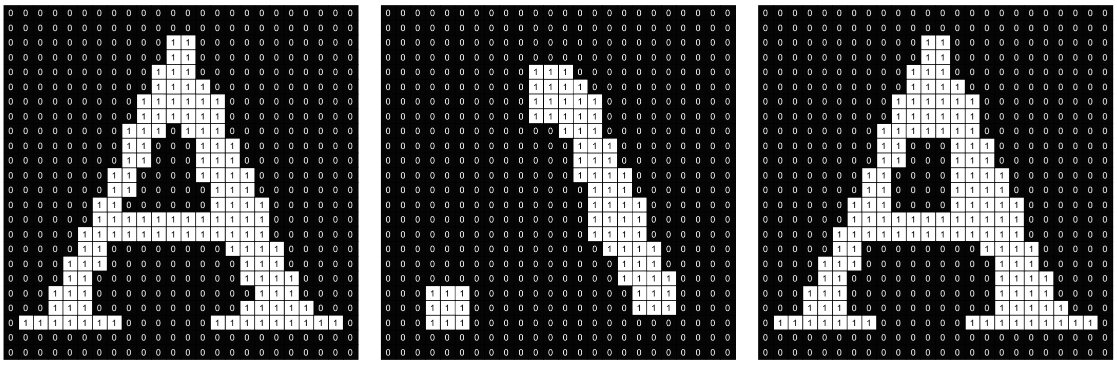
The opening of an image $\mathcal{X}$ by structuring element $\mathcal{B}$ is defined as the dilation of the erosion of the image and is denoted by $\bullet$-symbol as shown in equation 2.6. This sequence of operations can be used to remove small objects or noise in an image while retaining most of its original shape.

$$\mathcal{X} \bullet \mathcal{B} \triangleq (\mathcal{X} \ominus \mathcal{B}) \oplus \mathcal{B} \tag{2.6}$$

Contrariwise to opening, the erosion of the dilation of an image is called the closing of the image and is denoted by the $\circ$-symbol as shown in equation 2.7. Equally opposite, the closing is used to remove small holes in objects while retaining most of the original shape.

$$\mathcal{X} \circ \mathcal{B} \triangleq (\mathcal{X} \oplus \mathcal{B}) \ominus \mathcal{B} \tag{2.7}$$

In figure 2.5, both opening and closing is shown performed on a small image of an 'A'. As the image is relatively small, opening removes parts of the object. Closing on the other hand seems to almost not have any effect on the object. To increase the effect opening and closing have, multiple iterations can be performed.



(a) Original image **(b)** Opening **(c)** Closing

Figure 2.5: Opening and Closing performed on a image using a 3 by 3 structuring element of 1's

2.3.3 Distance Transform

The distance transform calculates the shortest route from the foreground to the background for all pixels in a binary image. The result is called a distance map, an image where each pixel value corresponds to the resulting distance in the original image. Several different metrics can be used to calculate the distance. Most known are the Euclidean, rectilinear and Chebychev distances.

The most accurate distance metric is the Euclidean distance. For a point in the foreground (x_1, y_1), the distance to the closest point in the background (x_2, y_2) is defined as following:

$$D(x_1, y_1)_{euclidian} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \tag{2.8}$$

The Euclidean distance is however the slowest to calculate and may also produce non-integer results.

The rectilinear distance, often called city block or taxicab distance results in counting each pixel when moving in horizontal and vertical directions as shown in equation 2.9. This is the fastest distance metric to calculate, but also the worst approximation to the euclidean distance.

$$D(x_1, y_1)_{rectilinear} = |x_2 - x_1| + |y_2 - y_1| \tag{2.9}$$

The Chebychev distance is similar to the rectilinear distance, but also allows for diagonal movement as shown in equation 2.10. The Chebychev distance is therefore often called the chessboard distance after how the king is allowed to move on a chessboard.

$$D(x_1, y_1)_{chebychev} = max(|x_2 - x_1|, |y_2 - y_1|) \tag{2.10}$$

As both rectilinear and Chebychev are approximations to the Euclidean distance, the resulting distance map are affected by the choice of metric. Figure 2.6 shows how the distance map of a small image of an 'A' by using the different metrics. It can be seen that the euclidean distance results in a smooth map in all directions, but both the rectilinear and Chebychev distances creates artifacts in the distance map.

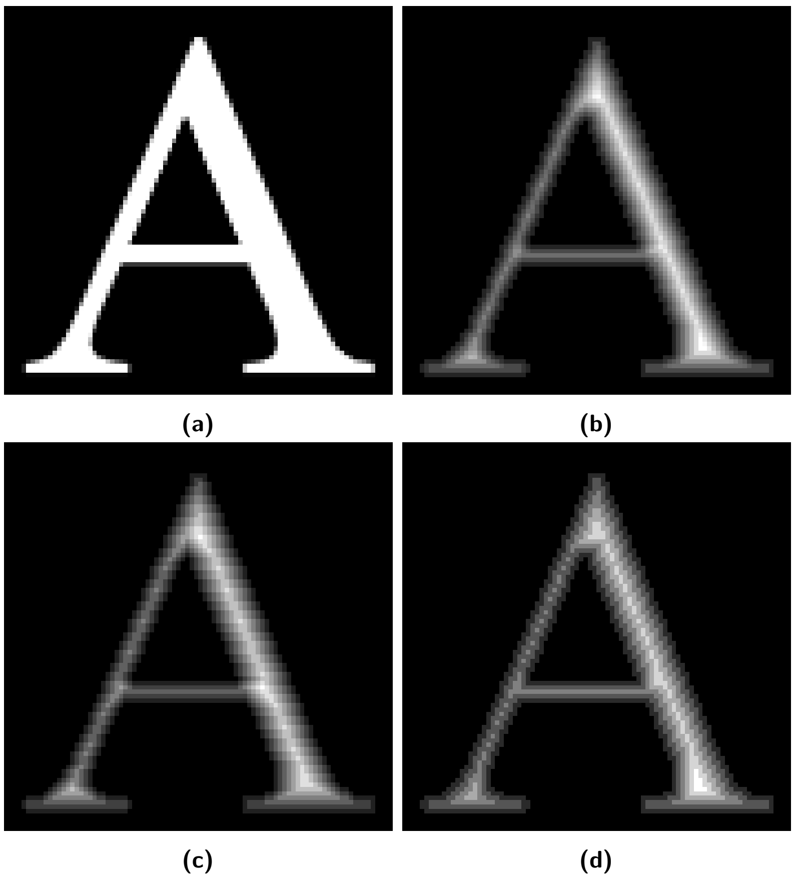


Figure 2.6: Distance map produced using the different methods of calculating the distance of **(a)** the original image. **(b)** The Euclidean distance calculates the true distance, producing a smooth distance map in all directions. **(c)** The rectilinear distance only counts vertical and horizontal movement and can create linear artifacts in the distance map. **(d)** The Chebychev distance also counts diagonal movement and can create diagonal artifacts.

Once the distance metric has been chosen, there are several ways to create the distance map. One way is to perform morphological erosion of the image and count the number of iterations needed before each pixel have been removed. Using a 3 by 3 cross structuring element (such as shown in figure 2.3b) will create a rectilinear distance map, while a 3 by 3 structuring element of one's (as in figure 2.3a) will produce the Chebychev distance map.

Performing morphological erosion for several iterations is a slow and computational intensive task, hence many image processing libraries use the method proposed by Gunilla Borgefors in the paper *Distance transformation in digital images, 1986*[27]. Instead of considering the image as a whole, her method considers smaller neighborhood of pixels and uses a translatory mask of cost-values. Where the distance is the sum of the cost of jumps necessary. Figure 2.7 shows the different masks and the corresponding cost-values are listed in table 2.1. This method only requires two passes to calculate the distance of all foreground pixels in an image[27].

+b	+a	+b
+a	0	+a
+b	+a	+b

(a) 3 by 3 mask

-	+c	-	+c	-
+c	+b	+a	+b	+c
-	+a	0	+a	-
+c	+b	+a	+b	+c
-	+c	-	+c	-

(b) 5 by 5 mask

Figure 2.7: Borgefor's masks for calculating the shortest path to the nearest zero value. The values for a to c are optimized depending on the chosen distance metric and are shown in table 2.1

	3-by-3	5-by-5
Euclidian	$a = 0.955,\ b = 1.3293$	$a = 1,\ b = 1.4,\ c = 2.1969$
Rectilinear	$a = 1,\ b = 2$	Not applicable
Chebychev	$a = 1,\ b = 1$	Not applicable

Table 2.1: Optimized values for different distance metric using the masks in figure 2.7 from Borgefors' method of calculating the distance map. Using a 5 by 5 mask for rectilinear and chebychev distances do not improve the resulting distance map

2.3.4 Segmentation

In image processing, segmentation is the process of separating an image into parts. This can either be to separate similar areas, e.g. of the same color, texture et cetera or to separate multiple objects in the same image.

2.3.4.1 Thresholding

Thresholding an image splits the image into two or more parts defined by the threshold value(s). Binary thresholding contains only one threshold value. For a given pixel in the image ($scr(x, y)$), the thresholded result ($dst(x, y)$) will be set to the maximum value defined by the image's bit depth if the pixel value is above the threshold value, and set to zero if below, as shown in equation 2.11.

$$dst(x, y) = \begin{cases} 2^n - 1 \text{ (for a n-bit image) if } src(x, y) > threshold \\ 0 \text{ else} \end{cases} \tag{2.11}$$

Binary thresholding is the simplest form of thresholding. Multiple defined threshold values is called multilevel thresholding. For color images, one or more threshold value defined for each channel it is called multidimensional thresholding.

To perform the segmentation a threshold value must be selected. In figure 2.8 an example of a grayscale image with its corresponding histogram is shown. In the histogram the two peaks corresponding to the background area and the foreground area can be clearly identified, and a good solution would be to select a threshold value somewhere between 100 and 200.

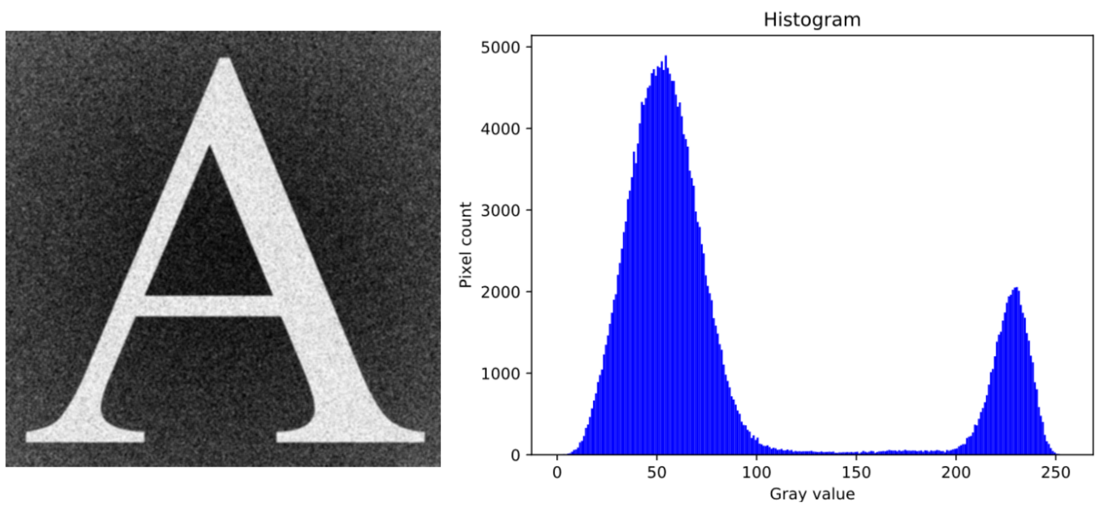


Figure 2.8: Left: A noisy grayscale image. **Right:** The histogram corresponding to the image on the left.

Selecting the threshold value by visually evaluating the histogram does however involve human interaction. In some images it may also not be as obvious what threshold value to choose.

One popular way of automatically selecting a threshold value is the method proposed by Ōtsu Nobuyuki in 1979 [28], later known as Otsu's method. In his proposal he suggest to look at the variance of the histogram and that the optimal thresholding value is the one that maximizes the between-class variance (σ_B^2), or correspondingly minimizes the within-class variance in binary thresholding.

For an image with L gray levels, the number of pixels at each i gray level, normalized to the total number of pixels, is denoted by p_i. For a threshold set at the kth level, the zeroth and first order cumulative moments of the histogram up to the kth level are $\omega(k) = \sum_{i=1}^{k} p_i$ and $\mu(k) = \sum_{i=1}^{k} i \cdot p_i$ respectively and the total mean level $\mu_T = \mu(L) = \sum_{i=1}^{L} i \cdot p_i$. The between-class variance σ_B^2 is then defined as shown in equation 2.12.

$$\sigma_B^2(k) = \frac{[\mu_T\omega(k) - \mu(k)]^2}{\omega(k)[1 - \omega(k)]} \tag{2.12}$$

Using Otsu's method, σ_B^2 is calculated for all k possible threshold values, such that the optimal threshold value k^* is σ_B^2 is maximized, as shown in equation 2.13. Figure 2.9 shows the result of Otsu's method performed on the image from figure 2.8.

$$\sigma_B^2(k^*) = \max_{1 \leq k < L} \sigma_B^2(k) \tag{2.13}$$

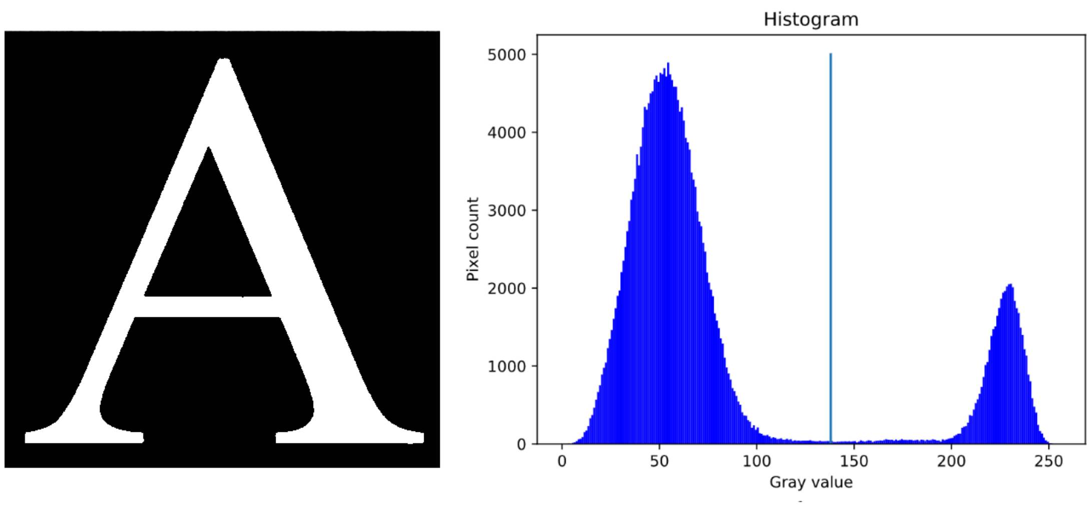


Figure 2.9: Left: Result of thresholding the image in figure 2.8 using Otsu's method. **Right**: The histogram of the original image. The vertical line indicates the optimal threshold value at gray level 138, as found by Otsu's method.

2.3.4.2 Connected Components Labeling

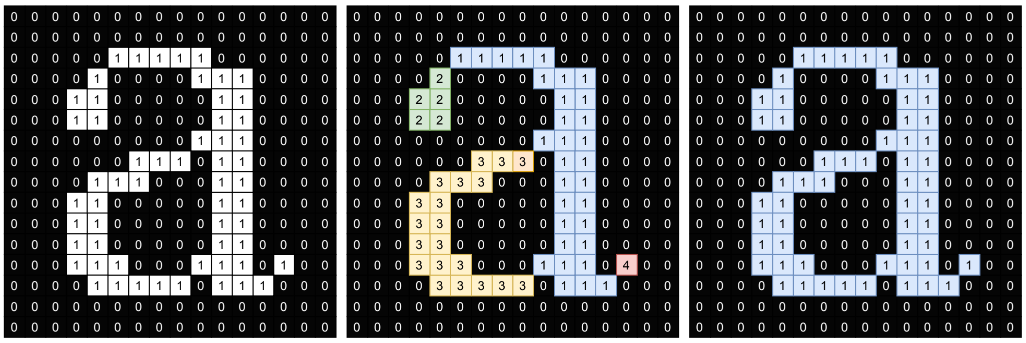
Connected components labeling is a method to identify multiple objects within an image by using pixel connectivity and giving each group of pixels their own unique label.

For a pixel at position (x, y), any pixels within its defined neighborhood is said to be connected, i.e. belongs to the same group, if and only if they have the same value[2]. In binary and grayscale images it is common to define the neighborhood using 4-way or 8-way connectivity.

With 4-way connectivity, the neighborhood of a pixel at (x, y) is the region defined by $(x \pm 1, y)$ and $(x, y \pm 1)$. The neighborhood using 8-way connectivity is the same as 4-way with the addition of $(x \pm 1, y \pm 1)$, i.e. all the surrounding pixels [29].

The whole image is scanned from top to bottom, checking all the pixels. If a pixel corresponding to the foreground is detected, and it is not connected to any known groups, it is assigned a new label. Similarly, if it is connected to a known group, it is assigned the same label as the rest of the group.

Figure 2.10 shows an example of this method performed on a small image. When using 4-way connectivity, only horizontal and vertical connected pixels are regarded as being in the same group, resulting in more groups than one might expect.



(a) Original image **(b)** 4-way connectivity **(c)** 8-way connectivity

Figure 2.10: Connected component labeling using 4-way and 8-way connectivity. With 4-way connectivity, only horizontal and vertically connected pixels are counted as the same group. Using 8-way connectivity all surrounding pixels of the same value is counted as the same group

2.3.4.3 Watershed

Instead of segmenting an image based on pixel connectivity or the histogram, the watershed method views the entire image as a topographical map. In geographical terms,

[2]or for grayscale images, within a range of values

the watershed is the natural separation of two adjacent catchment basins. By letting each pixels value indicate its elevation, the image transforms into a terrain composed by peaks, ridges and valleys, and the watershed method tries to find the basins and watershed-lines that separates them [30][31].

A common analogy to describe the method is to think of the watershed method as a flooding of the terrain created by the image. Start by creating holes through the terrain at all the local minimums and block off the edges. While keeping it flat, slowly and uniformly lower the terrain into a body of water such that the water emerges through the holes. As the water level continues to rise it will fill up the catchment basins, but instead of letting them overflow, create a dam at the points where the water would meet or overflow into another basin. When the terrain is fully flooded only the dams are left emerged from the water, and the regions between them indicate the different segments[32][31].

In practice however, this straightforward method will often result in over segmentation of the image, often to the point where the result rendered useless. This is because every local minimum, even those from noise in the image, creates its own catchment basin and a resulting segment.

One of the more successful method of improving the result is a marker based approach, where the amount of catchment basins are limited to the marked regions, as presented by F.Meyer [32]. His approach greatly reduces over segmentation, but the need for markers is a drawback and cause the method to act as a region growing method rather than automatic segmentation [32][31].

Figure 2.11 shows a marker-based watershed algorithm performed on an image of a handful coffee beans. For every bean a corresponding marker, or seed, is placed before entering the watershed algorithm. The result is then overlaid the original image. In F. Mayer's method, which is implemented in the Python image processing library OpenCV, each marker must have their own label, such as created by connected component labeling. Since all the starting points have labels, the marker does not need to be continuous or even whole; if the confluence of flood comes from a minimum with the same label, they are simply combined. When the algorithm is finished, the resulting segments will all have the same labels as their corresponding markers.

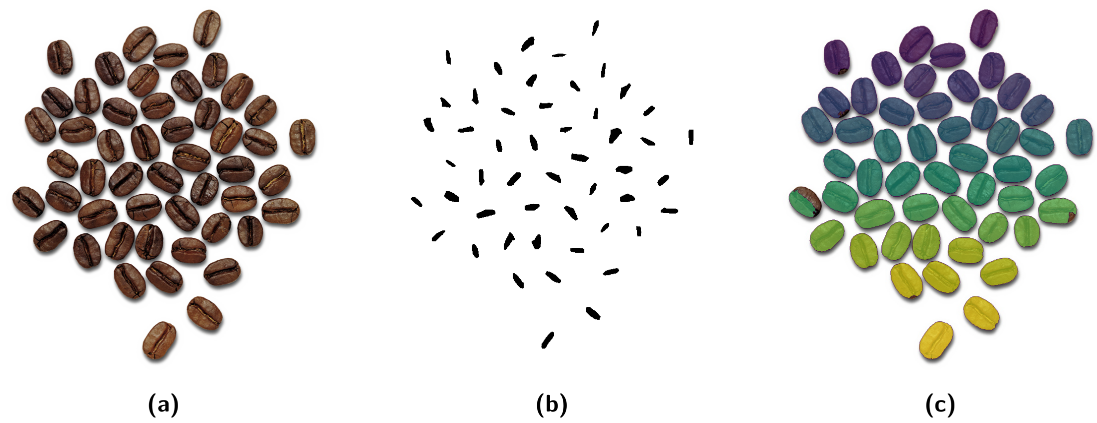


Figure 2.11: Example of a marker-based watershed transform performed on an image. **(a)** The original image. **(b)** Markers, or starting points for the watershed algorithm. **(c)** Result from the watershed algorithm, overayed on the original image. `Image taken from pngimg.com, reproduced in unaltered form (a) and altered form (c). Licensed under CC BY-NC 4.0`

2.4 Neural Networks

The brain is one of the most intriguing organs of the human body, and many have tried to understand its functions and operations. The notion that the brain is the center of human intelligence and consciousness can even be traced back to the age of Hippocrates in the fifth century [33]. It would however take many years until the field of neuroscience would make significant progress. In his publication *Histologica du Systéms Nerveux de l' Homme et des Vertèbres* from 1911, Santiago Ramón y Cajal would identify and describe that the brain and nervous system is composed of one particular type of cell, later known as the neuron [34].

An individual neuron receives chemical inputs in through receptors on the dendrites of the cell. Depending on what chemical received, it can either excite or inhibit excitation of the neuron, and the sum of all the dendritic inputs will decide if the neuron will fire an electrical impulse or not. If an electrical impulse is sent, it will travel along the axon of the cell and transmitted to the next neuron, gland or muscle via neurotransmitters at the end of the axon [35]. Figure 2.12 shows an annotated sketch of the neuron with its dendrites and axon. The contact points to and from neurons are called synapses, and each neuron contains between 10,000 and 150,000 of them [36].

In 1943, Warren McCulloch and Walter Pitts presented their mathematical model of a biological neuron. Their findings showed that emulating a neuron, weighting the synaptic links, one could create a structure that can compute any mathematical function, given enough neurons and synaptic links between them [37]. In 1958, Frank Rosenblatt invented the perceptron, and the perceptron algorithm. Unlike the McCulloch and Pitts model, the perceptron uses weighted inputs that would be adjusted using the perceptron algorithm, and after several iterations is trained to classify two classes by a linear separation. The perceptron and other types of artificial neurons are the basic building blocks of a neural network and combining several layers of these between the input and output layer are known as a deep neural network [34].

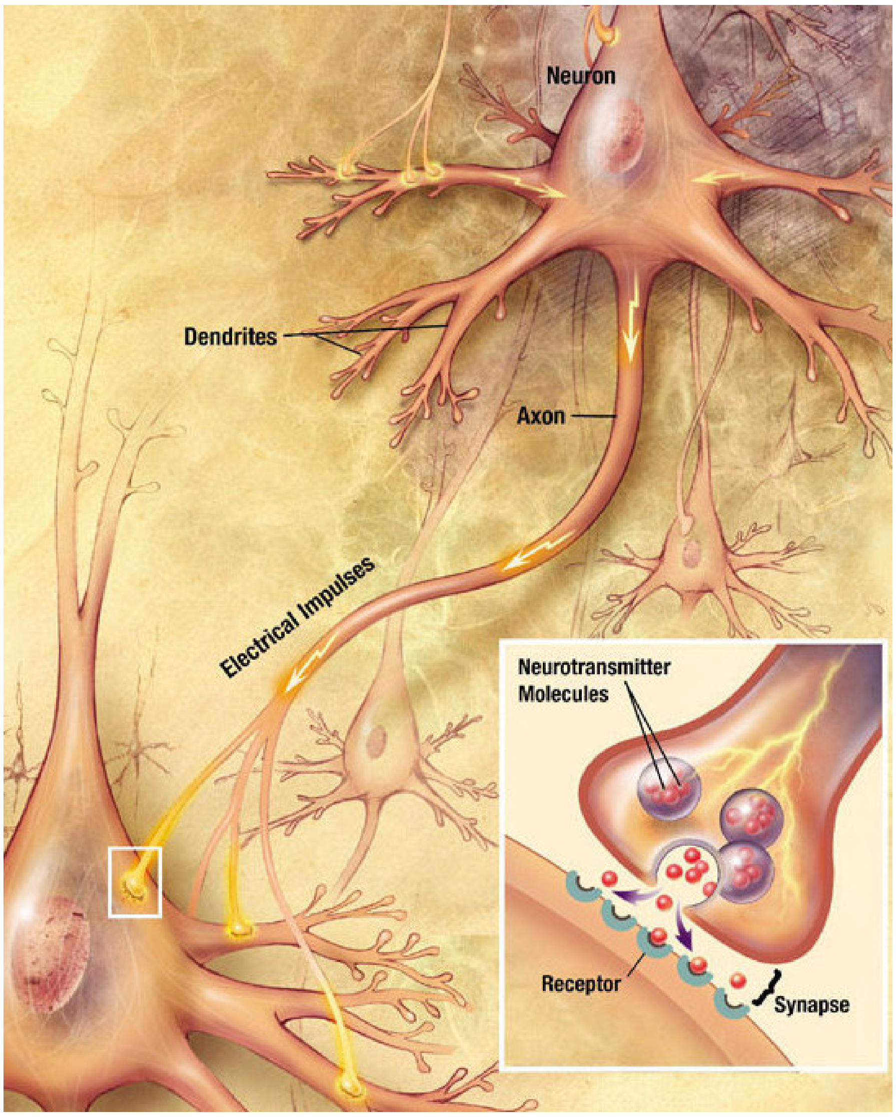


Figure 2.12: A biological neuron is composed by inputs (dendrites) and an output (axon). The dendrites receive inputs from other axons connected to the cell and will fire an electrical output if enough charge is created. The charge is transformed to a biochemical signal in the synapses at the end of the axon.
`This figure is reprinted in unaltered form from Wikimedia Commons. File: Chemical_synapse_schema_cropped.jpg. Created by US National Institutes of Health, National Institute on Aging. Released in the public domain.`

2.4.1 Artificial Neural Networks

An artificial neuron is a mathematical model that is constructed to mimic a simplified biological neuron. Each input of the artificial neuron is weighted and summed before entering an activation function, and if conditions are met, the output is activated. How the output responds and what conditions are required are dependent on what type of activation function used. Frank Rosenblatt's perceptron, for example, used a simple step-function (Heaviside function) as its activation function; if the sum of all the weighted inputs are above zero, the perceptron outputs a one, and below, a zero. It is important to note that in order to create a nonlinear output the activation function itself must also be nonlinear[3], even when combining several neurons in a network. This non-linearity is part of what makes a neural network able to generalize and outperform traditional machine learning algorithms and other statistical functions, such as linear regression [38].

Figure 2.13 shows an illustration of an artificial neuron. The inputs $[x_0, \ldots, x_n]^T$ are weighted by $[w_0, \ldots, w_n]^T$. A bias can be added to give preference the output. In many cases the bias is regarded as being x_0, equal to 1, and having a weight w_0 equal to the bias weight b. Given this definition, the output y of the artificial neuron is as shown in equation 2.14, where f is the neurons activation function.

$$y = f\left(\sum_{i=0}^{n} w_i x_i\right) = f(\boldsymbol{w}^T \cdot \boldsymbol{x}) \tag{2.14}$$

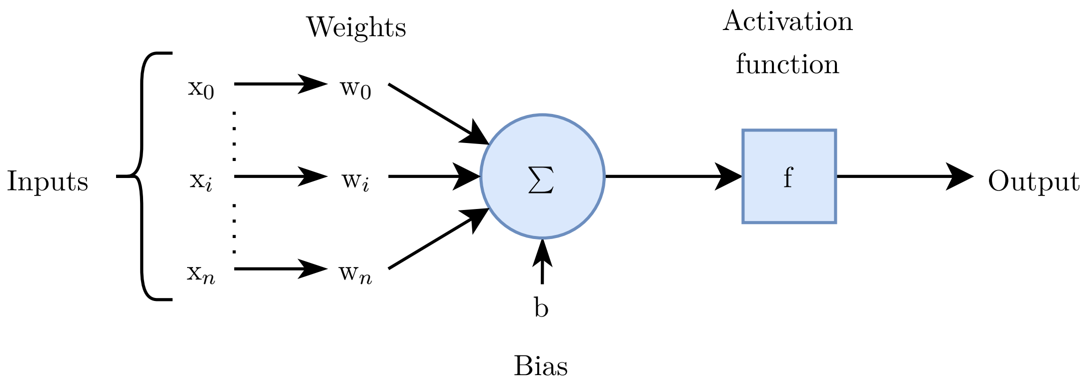


Figure 2.13: Model of an artificial neuron. The weighted inputs are summed before entering an activation function. The type of activation function will dictate the outputs response of a given input. Note that in many cases a neuron is drawn as a single circle, without explicitly drawing the weights and activation function.

In order create an artificial neural network able perform advanced classification tasks it is often necessary to increase the complexity of the network. This can be done by adding

[3]More information on activation functions will be discussed later, in section 2.4.2

more neurons and layers of neurons in the network. The layers in between the input and output layer are called hidden layers as their individual outputs does not necessarily represent the final prediction. A network composed of multiple hidden layers are known as deep neural networks. Figure 2.14 shows a type of deep neural network known as a multilayer perceptron or a deep feedforward[4] neural network.

In many cases, the main goal of an artificial network is to output a prediction for a given input, e.g. what class an object belongs to given its features, or even what is the best current move in a game of Go[39]. This means that the network itself is often regarded as a "black box", and the exact mathematical function it approximates is not necessarily of significant value for the user. The most basic way of training an artificial neural network is by using what known as supervised learning, training using only data that has a corresponding ground truth label. The features of the training data is entered to the network and the output is entered to a loss function. The loss functions task is then to compare the predicted value with the ground truth label, giving a metric of how well the model performed. This can then be used to determine how the weights in the network should be adjusted, for example by using its derivative as in the gradient descent method, and the test is performed again until the results are adequate.

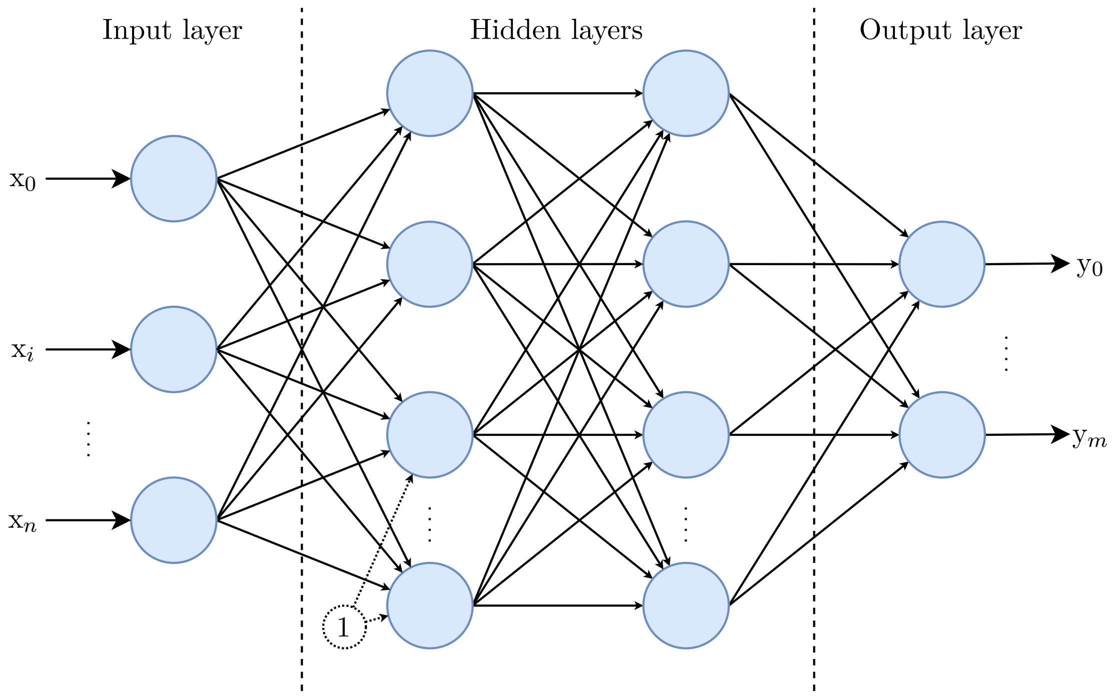


Figure 2.14: A deep neural network is a network structure containing multiple layers of neurons and is often used for more complex classification tasks. The depicted network is often called a multilayer perceptron or deep feedforward network

[4]It is called a feedforward network because there is no feedback-loop in the network. A network with feedback-loops is called a recurrent neural network

2.4.2 Activation Function

In a regular artificial neuron, the sum (x) of the weighted inputs are entered to the activation function, and the corresponding output (y) depends on the neurons activation function. As mentioned in the introduction of section 2.4, Frank Rosenblatt's perceptron used a step-function as its activation function, meaning the output of the neuron is strictly binary. This function is however at a disadvantage, as many effective training algorithms, such as the gradient descent, use the derivative of the activation function to calculate the new set of weights.

There are many different activation functions commonly used in neural networks. Research shows that some functions perform better than other in certain types of networks and with certain data types, although the performance gain is often marginal [40] [41].

2.4.2.1 Sigmoid

The Sigmoid function, as shown by equation 2.15 and in figure 2.15, has been a commonly used activation function for neural networks. Since its output is between zero and one, it can also be used at the output layer such that its output represents a probability. It does however have the drawback of being unstable when used in the early layers, as the gradient of the Sigmoid function have a tendency to either vanish or explode [40].

$$y = \frac{1}{1 - e^{-x}} \tag{2.15}$$

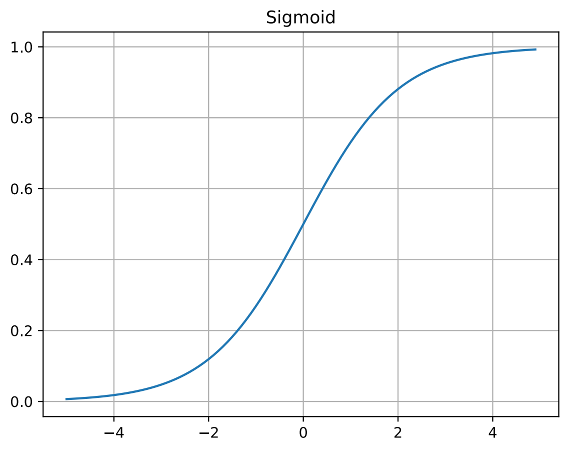


Figure 2.15: Plot of the Sigmoid function. A common activation function used in artificial neural networks

2.4.2.2 Rectified Linear Unit (ReLU)

In later years the ReLU function, equation 2.16 and figure 2.16, has been more popular for use in hidden layers of a neural network. Especially compared to the Sigmoid function it converges using fewer iteration, such that the training of the network is much faster [41].

Like the Sigmoid function, ReLU also has its drawbacks. As the ReLU function outputs zero for all negative input values, the weights of a neuron can be changed such that a situation occurs where the neuron outputs zero for all inputs, meaning it will never activate on any data at all and weights will not be updated further. This phenomenon is known as the dying ReLU problem. Because of this, there exists many similar functions to the ReLU that tries to mitigate this issue and improve performance, such as the Exponential Linear Unit (ELU), Scaled ELU (SELU), Leaky ReLU, Parametric ReLU, Gaussian Error Linear Unit (GELU), Swish and many more [41].

$$y = \begin{cases} x, & \text{if } x > 0 \\ 0, & \text{if } x <= 0 \end{cases} \tag{2.16}$$

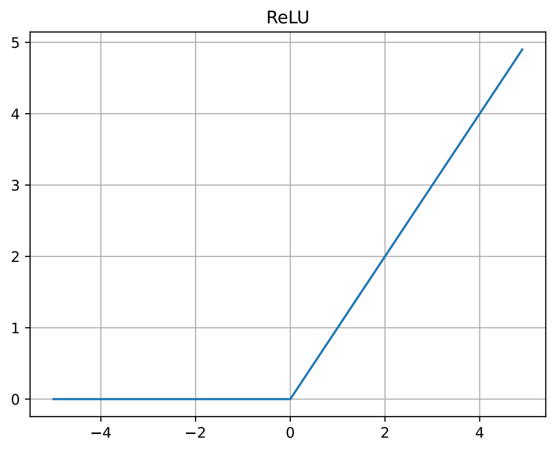


Figure 2.16: Plot of the ReLU function

2.4.2.3 SoftMax

The softmax function, shown in equation 2.17, is commonly used as the activation function in the output layer of a neural network for multiple class classification. It calculates the distributed probability, such that it outputs the probability of each outcome, j, over all C possible outcomes [38].

$$y_j = \frac{e_j^x}{\sum_{c=1}^{C} e_c^x}, \text{ for } j = 1, \ldots, C \tag{2.17}$$

2.4.3 Convolutional Neural Networks

Unlike traditional deep neural networks, such as the multilayer perceptron, a convolutional neural network (CNN) is a specialized network for processing multidimensional data. Instead of a feature vector, $[x_0, \ldots, x_n]$, and weights $[w_0, \ldots, w_n]$, a CNN's input is a multidimensional feature array which is convolved with one or more filter kernels. Furthermore, this implies that only the kernel's coefficients are changed during the training process, which applies to the entire layer. In the subject of deep learning, vectors and arrays are commonly referred to as tensors [38].

The multidimensional property of a convolutional neural network means its effective at preserving the spatial correlation of features, which proves useful when classifying images. This was shown by Le Cun et al in 1989, when their convolutional network for classifying images of handwritten digits obtained a low five percent error rate using a test set of over two thousand images [42]. Their method of creating the network has by many been regarded as the de facto standard of such networks and been the inspiration to much of the later development.

The progress of image recognition and classification did however stagnate in the 90's due to the lack of processing power. Especially in real-time applications, requiring fully parallel computational operations, often meant designing proprietary hardware such as FPGA-processors to be able to operate on a useful level. In 2004, Oh and Jung [43] proved that the computations in a neural network could be implemented on graphical processing units (GPU) on regular computer hardware, greatly increasing the computational speed.

GPU-accelerated computations are not limited to the field of deep learning and have greatly increased the possibilities within many research areas. This has also led to the release of specialized hardware for computational purpose, such as the Nvidia Tesla series. The Tesla cards processing unit also contains specialized processing cores named tensor cores, made specifically for the many matrix operations used in deep learning [44].

As the name of the network applies, a CNN relies on the convolution operator. Each convolutional layer can typically be divided in three stages: the convolution stage, detector stage and pooling layer. In the first stage, the input to the network is convoluted with one or more matrices of weights called filter kernels. The kernel slides across the input tensor, and a 2D-convolution is performed at each index. The result of this operation is entered to the layers activation function in the detector stage, creating what is known as a feature map at the output[5]. The last stage, the pooling layer, is an optional step that downsamples the image before entering the next layer. The pooling layer is further explained in section 2.4.4.

The convolutional layers of a CNN are often said to be the feature extractor in the network, such that the entire network can be split into two parts, the feature extractor and the classifier. The classifier is made up by fully connected (dense) layers, like a common feedforward network, with a flattened output from the last pooling layer as the input [38].

Figure 2.17 shows an illustration of the structure to a CNN network, where the convolutional layers are a part of the feature extraction, and dense layers in the classification. Each layer in the feature extraction is made up by a convolution, detection, and pooling step.

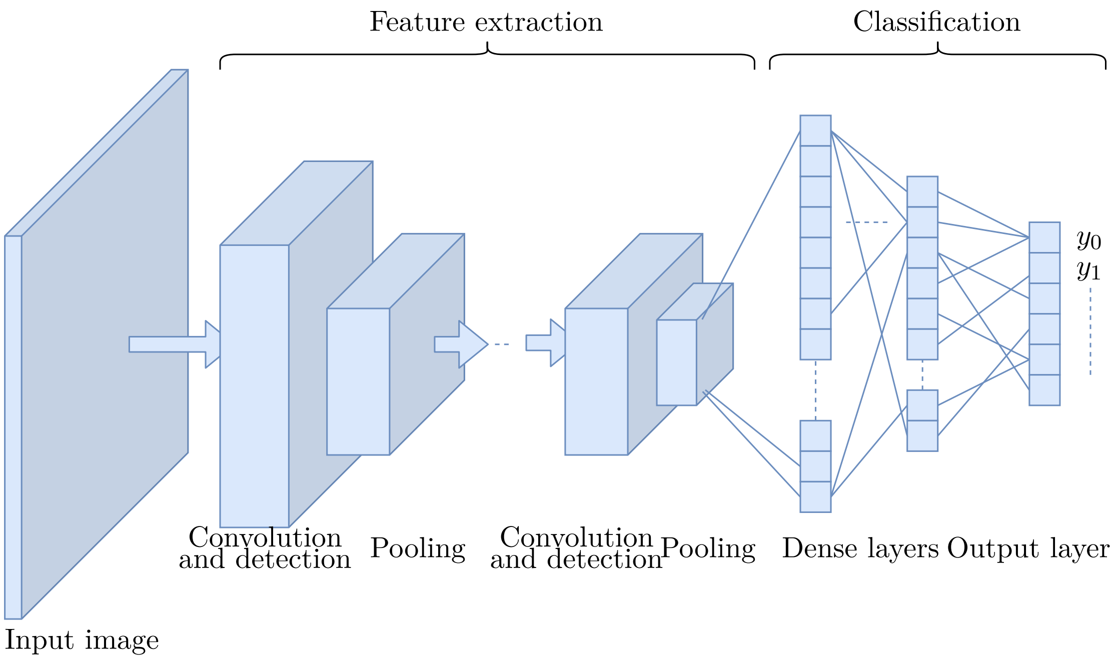


Figure 2.17: The two steps of a convolutional neural network: feature extraction and classification. An input image is first put through convolutional layers where the input is convoluted with a set of filter kernels before entering an activation function in the detection stage. Lastly the output from the final pooling layer is flattened and goes through the classifier

2.4.3.1 Convolution Operator

In a convolutional layer a 2-dimensional discrete convolution is performed on the input (I) by the filter kernel (K), such that the resulting image (S) contains the result of the

[5]Depending on the input tensor and the network layer, there may be more than one kernel, creating multiple feature maps at the output of the layer

inputs convolution at each index (i, j), as shown by equation 2.18[6]. The convolutional operator is usually denoted by the $*$-symbol.

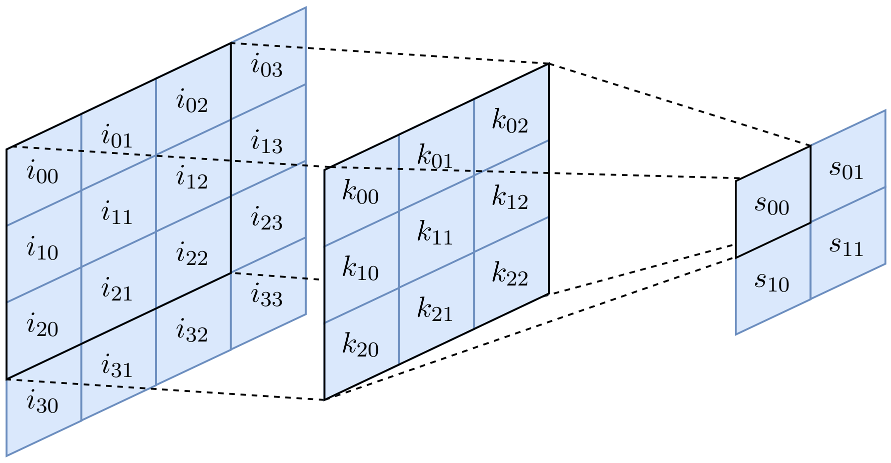
$$\begin{aligned} S(i,j) = (I * K)(s,j) &= \sum_m \sum_n I(m,n)K(i-m, j-n) \\ &= \sum_m \sum_n I(i-m, j-n)K(m,n) \end{aligned} \tag{2.18}$$

This means that each pixel of the resulting image, is the result of the sum of all the pixels within a region defined by the size of the kernel multiplied by a flipped version of the kernel (or image as the convolution is commutative).

For simplicity, many neural network libraries implement this without flipping, meaning the convolution operation have been replaced by cross-correlation, as shown by equation 2.19, all though they may call it convolution. This does however not affect the result of the network; after training, the resulting kernel weights will be the same in both cases, only flipped [38].

$$S(i,j) = (I * K)(s,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{2.19}$$

This process is illustrated in figure 2.18, the resulting pixel $s_{00} = i_{00}k_{00} + i_{01}k_{01} + \cdots + i_{22}k_{22}$. For s_{01}, the kernel is moved one step to the right. This procedure is repeated until all the pixels have been covered. The region of which a single result is representing, is called its receptive field, i.e. for s_{00} the receptive field is the region defined by the nine pixels in the top left corner of the image.



i_{00} i_{01} i_{02} i_{03}
i_{10} i_{11} i_{12} i_{13}
i_{20} i_{21} i_{22} i_{23}
i_{30} i_{31} i_{32} i_{33}
k_{00} k_{01} k_{02}
k_{10} k_{11} k_{12}
k_{20} k_{21} k_{22}
s_{00} s_{01}
s_{10} s_{11}

Figure 2.18: The cross-correlation of the image (left) by the filter kernel (middle) and the result (right). Each element in the result is calculated using the formula from 2.19

[6]Here the commutative property of the convolution is also shown

As one may notice in the example above and in figure 2.18, the resulting image is smaller than the input. To keep the output dimension of the image same as the input, a common solution is to use zero-padding. By expanding the input, creating a boarder of zeros, the original dimensions are kept. If one wants to reduce the output dimensions even further, such to reduce the amount of computations, i.e. convolutions performed, the kernel can be moved more than one step at the time. This parameter is called the stride of the kernel. In the example above, the stride is set to one and the kernel moves one step between each calculation.

2.4.4 Pooling

The pooling layer is a step often used in convolutional neural networks, placed after the detection stage in each layer, as mentioned in section 2.4.3. Figure 2.19 shows pooling applied to a small 4 by 4 feature map by using a 2 by 2 window and two different methods. By average pooling, the mean value of each window is extracted, and in max pooling the maximum value.

By reducing the shape of the feature map within the network, pooling makes it able to extract translation invariant features, reducing the chance of over training the network. Pooling also reduces the memory and computational requirements [38].

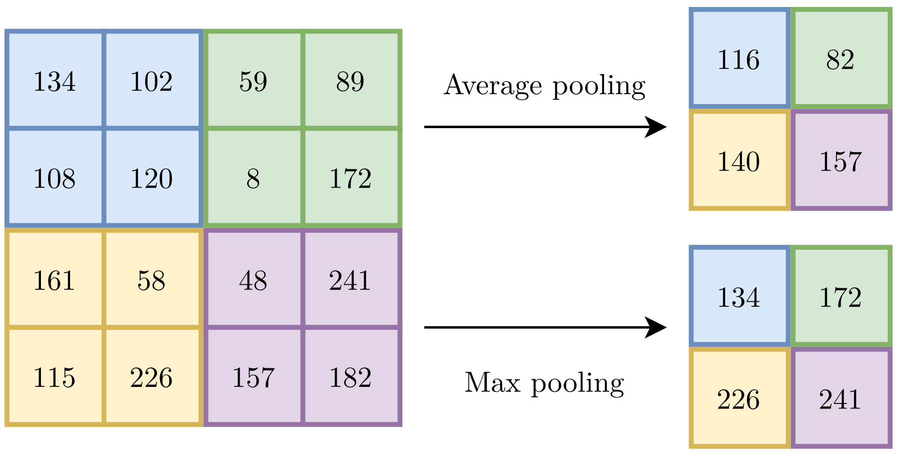


Figure 2.19: Pooling is a method of downsampling the feature map. This makes the network invariant of small translations, reduces the chance of over training, as well as reducing memory and computational requirements

2.4.5 Loss Function

When creating an artificial neural network, the goal is to approximate a function that produces a desired output depending on the input given. In order to generate the best possible results, the networks run through an iterative optimization process commonly referred to as training.

To assess the performance of the network, the result is entered to a criterion function or objective function. When the objective is to minimize the criterion function it is commonly referred to as a loss function or cost function. Depending on the network structure, type of data, and desired output from the network, many different loss functions exists and they play a crucial role in training and performance evaluation of the network.

2.4.5.1 Categorical Cross-Entropy Loss Function

For neural networks created to classify between multiple classes, using the SoftMax activation function in its output layer, the most common loss function is the categorical cross-entropy loss function.

Categorical cross-entropy (CE) is defined as the sum of a ground truth label (t) multiplied with the logarithm of the output prediction from the network ($\hat{y}$) over all classes (C) as shown in equation 2.20.

In multi-class classification it is common to use a one-hot encoded truth vector, such that the only non-zero value of t is the correct class [38].

$$CE = -\sum_{i=1}^{C} t_i log(\hat{y}_i) \tag{2.20}$$

2.4.6 Evaluation Metrics

As the loss function is optimized during the training process it gives a metric for the performance of the network. These results can however be intricate to interpret and difficult to compare between different network structures and models.

2.4.6.1 Accuracy

The accuracy metric, shown in equation 2.21, is often used together with loss during the training process, giving a better representation and making it easier to conceptualize the networks performance gain for each iteration. A common training strategy is to use the training data together with validation data, i.e. data only used to check the performance; If the accuracy of the training data is high while the accuracy of the validation data is low, the network might be over-fitted.

$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Number of predictions}} \tag{2.21}$$

Most metrics are calculated by comparing correctly identified samples with incorrect identified samples. In a binary classification scenario, the correct classifications are often referred to as true positives (TP) and true negatives (TN), and the incorrect classifications are called false positives (FP) and false negatives (FN). Using this notation, the accuracy is calculated as shown in equation 2.22.

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{2.22}$$

2.4.6.2 Precision, Recall and F1-score

Precision and recall, shown in equation 2.23 and 2.24 respectively, can both provide additional information as to how well the network performs. These metrics are especially useful when dealing with multiple classes and imbalanced training data. As the accuracy metric views all the classifications as a whole, both precision and recall are class dependent metrices and can shed light on any bias in the network.

Precision is the fraction of those correctly classified of a particular class divided by all samples that were predicted to be the same class. In other words, that of all the samples the network predicts to be a particular class, how many were correctly identified. This provides an indication of how well the network is to identify each individual class, but it does not take account for cases that were not correctly identified.

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}} \tag{2.23}$$

Recall is the number of correctly identified of a class divided by the total number of samples of that class. Recall is sometimes referred to as the sensitivity or true positive rate. This is particular useful in some instances, for example in a medical trial where it could be favorable to detect some false positives instead of missing a diagnosis.

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}} \tag{2.24}$$

The F1-score is what is known as the harmonic mean between precision and recall, and is shown in equation 2.25. This takes both the precision and recall of a network in account. A F1-score of one indicates perfect precision and recall.

$$F_1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.25}$$

2.4.6.3 Confusion Matrix

The confusion matrix is a valuable tool to visualize and gain insight to a network's performance on a per class level. An example of a confusion matrix is shown in figure 2.20. The vertical axis indicates the true class label of the samples, and the horizontal axis the model's prediction of all the samples. This means that all correctly predicted samples are gathered in the diagonal of the matrix. The confusion matrix makes it trivial to identify incorrect classification and how the inaccuracy of the model affects the predicted values.

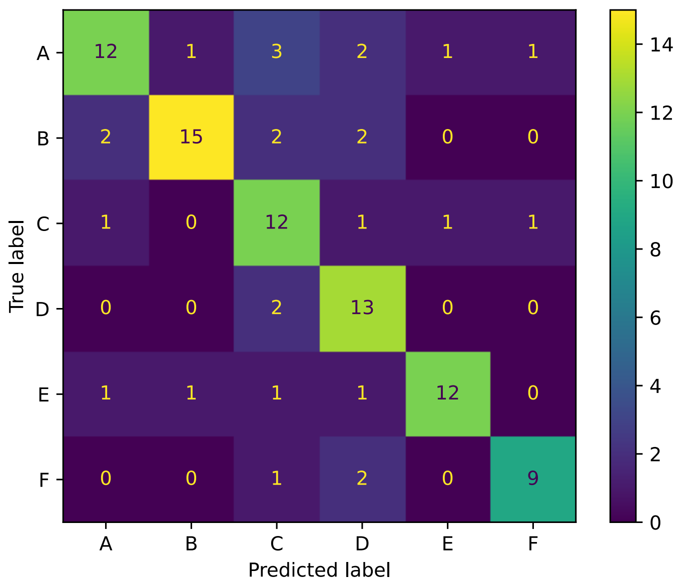


Figure 2.20: Example of a confusion matrix created using the Scikit Python library. Comparing the true label and predicted label all correctly identified predictions are represented in the diagonal line from top left to bottom right

2.4.7 Transfer Learning

Transfer learning is the concept of using a pretrained artificial neural network for a different task than it was originally trained for. The general notion is that a deep neural network trained using a large enough data set will be able to extract general features that also applies to similar types of data.

As mentioned in section 2.4.3, a convolutional neural network can be regarded as having two parts, the feature extractor and the classifier. When using transfer learning in practice, a pretrained network or backbone can be used as a general feature extractor by replacing the classifier with new fully connected layers.

Training a deep neural network is often a demanding task, both in computational power and time. By freezing all the layers in the feature extractor, only the new classifier is trained. This vastly reduces the amount of trainable weights in the network while still being able to yield satisfactory results with a small dataset. If the new dataset is large, the model can be fine-tuned by only freezing some (or even none) of the layers in the feature extractor.

Within the field of image recognition, a common benchmark for performance is the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [45], where it is competed to get the best performance using a very large dataset. Many of the entries to this challenge are readily available to use for transfer learning, for example in the neural network package Keras for python. Keras' API even lists the available models after their performance in the ILSVRC [46].

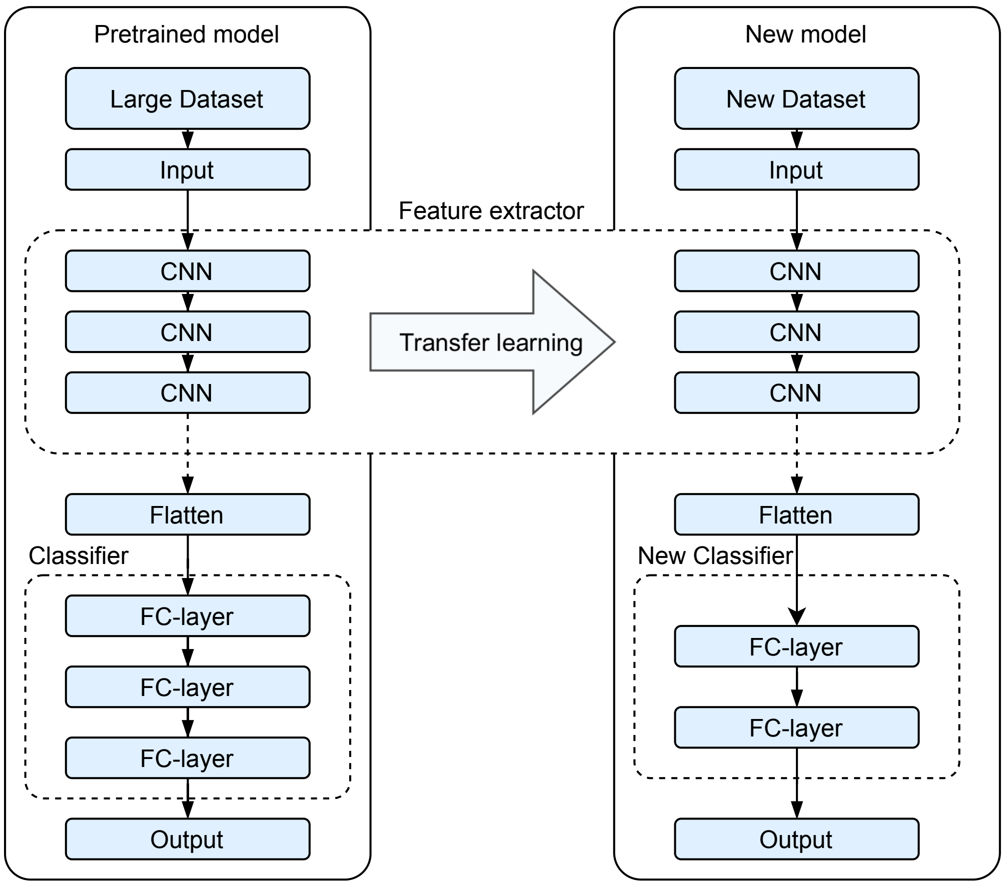


Figure 2.21: Transfer learning is the concept of using knowledge from a pretrained network for a new task

Chapter 3

Data and Materials

3.1 Palynological Slide Images

As the project of digitizing the catalogue of samples at the Norwegian Petroleum Directorate is still in its starting phase, the data set made available for this thesis was somewhat limited. Seven palynological slides were made available for this thesis. Six of the slides were made from core samples taken from the Ekofisk oilfield, and the last one from Johan Sverdrup. Both these oilfields are located in the North sea, off the coast of Norway, as shown in figure 3.1. Especially the Ekofisk field has a great historical value; it was the first oil field discovered in the North sea, in 1969 by the Phillips Petroleum Company (now ConocoPhillips co.), and marked the beginning of what we now call the Norwegian oil adventure [47].

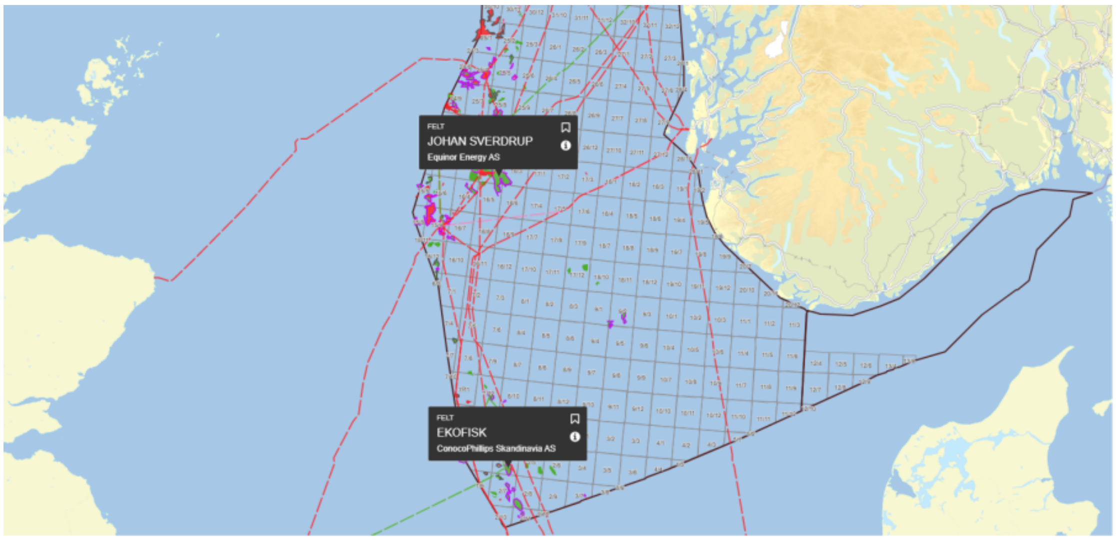


Figure 3.1: Map showing the location of the oilfields Ekofisk and Johan Sverdrup outside the coast of Norway where the dataset were taken from. `Map reproduced in altered form from norskpetroleum.no. Map licensed under CC BY 4.0 and contains data from the Norwegian Petrolium Directorate licensed under NLOD`

Of the six samples from the Ekofisk field, five were taken from a well named 2/4 C-11 [48] and one from 2/7 14 [49], at depths ranging from 3,064 to 3,249 meters below the deck of the platform. The last sample from the Johan Sverdup field was taken from a well named 16/3-2 [50] at a depth of 1,998.8 meters.

To create the palyslide, a small sample is taken from a larger core sample. The position of the sample is measured, such that the depth can accurately be determined and labeled. The sample is grounded to dust before it is dissolved in an acid, usually nitric, hydrochloric, or hydrofluoric acid. The type of acid and how long the process takes is dependent on the amount and type of residue in the sample. The sample is washed with tap water before one or two drops are placed on a microscopic slide and let to dry overnight. A coverslip is glued on top of the sample with resin and a label is placed to note where the sample is taken from and the procedure used to create the sample.

The samples are digitized using a pathological scanner from 3D Histech, the Pannoramic 1000. This model can hold up to 1000 microscopical slides and have an optical resolution of 0.25 μm per pixel. It is claimed to be the fastest whole slide scanner on the marked, able to scan up to 100 slides per hour [15]. As the dinoflagellates are not perfectly flat, each slide is scanned multiple times at different focus levels. This allow a user to manually focus the view in the software, or each focus level can be stacked such that only the parts in focus are kept in the final image. This stacking process creates a bigger depth of field and allows an entire dinoflagellate to be in focus instead of just parts of it.

Due to the file size of a slide image and the practicality of a single focus plane, only stacked slide images have been used in this thesis. Stacking merges each layer such that the size of each slide is reduced to about three gigabytes. The digital resolution of a slide image is approximately 184,000 by 96,000 pixels, covering a region of 46 by 24 millimeters. At this resolution, one square millimeter of palyslide is represented by over 16 megapixels.

Figure 3.2 shows a macro image of a palyslide as well as the slide's label. The label shows that this sample is taken from a depth of 3,070.2 meters, from well 2/4-C-11. "Ø 1/4'" indicates that the sample has been dissolved in a 40 percent nitric acid solution for 15 seconds.

(a) (b)

Figure 3.2: **(a)** Macro image of a scanned palyslide. **(b)** The palyslide's label. The label shows that this sample is taken at a depth of 3070.2 meters from well 2/4-C-11. "Ø 1/4'" indicates that this sample have been dissolved in a 40 percent nitric acid solution for 15 seconds

3.2 Labeling Data

Each slide can contain anything from a few hundred to over a several thousand dinoflagellates and fragments of dinoflagellates as well as inertinite and other types of material not removed by the acid preparation. The washing procedure may also introduce crystal structures that originate from the tap water.

As there are many similar species of dinoflagellates and it can be difficult to differentiate between them, the data were mostly labeled by Robert Williams at the Norwegian Petroleum Directorate. The annotations were made by using the 3DHistec's CaseViewer software. Only whole, or mostly whole dinoflagellates were annotated. Fragments, crystals, inertinite and other types of residuals were ignored in the annotation process.

3.3 Dataset

From the seven slide images, 530 dinoflagellates were annotated from 21 different species, as listed in table 3.1[1]

Species Name	Number of samples
Senoniasphera Inornata	173
Fibrocysta Axilis	121
Palaeoperidinium Pyrophorum	117
Spongodinium Delitiense	65
Cribroperidinium "Prominoseptatum"	19
Spongodinium Delitiense (operculum)	13
Dingodinium Tuberculosum	2
Dingodinium Tuberosum	2
Gonyaulacysta Jurassica	2
Sentusidinium Pilosum	2
Systematophora Areolata	2
Tubotuberella Apatela	2
Acanthaulax Venusta	2
Thalassiphora Pelagica	1
Sirmiodinium Grossii	1
Chytroeisphaeridia Cerastes	1
Chytroperidinium Sp.	1
Danea Californica	1
Enoscrinium Galeritum Reticulatum	1
Leptodinium Mirabile	1
Scriniodinium Inritibile	1

Table 3.1: Complete list of samples

3.3.0.1 Dataset Split

A customary practice when training neural networks is to split the dataset into three: training, validation, and testing. Both training and validation data is used during the training process and the test data is used afterwards to evaluate the performance of the network. Since this would mean that many of the classes could not be verified or tested, all the classes with less than 20 samples were collected in a single class named "Other

[1]A list of all annotations sorted by slide and example image of all different species can be found on table B.1 and in Appendix B

Dinoflagellates". The data were split with 60 percent in D_{train}, and 20 percent in D_{val} and D_{test}.

Species Name	D_{train}	D_{val}	D_{test}	Total
Senoniasphera Inornata	103	34	36	173
Fibrocysta Axialis	72	24	25	121
Palaeoperidinium Pyrophorum	70	23	24	117
Spongodinium Delitiense	39	13	13	65
Other Dinoflagellates	32	10	12	54
Total	316	104	110	530

Table 3.2: The D-dataset. Classes containing less than 20 samples were put together to a single class, "Other Dinoflagellates"

Chapter 4

Proposed Method

This chapter will explain the methods and process used to process, detect, and identify dinoflagellates in palynological slides using traditional image processing techniques together with deep neural networks.

The general idea of this thesis can be divided into three different parts, as illustrated by figure 4.1. Part 1 covers generating and exporting the training data from the palynological slides, Part 2 will cover object detection and lastly, Part 3 covers objects identification using convolutional neural networks. All the parts are however closely linked, as both the training data and detected objects will need be in the same format for the neural network to be able to classify the objects.

To view the palyslide images both CaseViewer and QuPath was used. All parts were created using Python 3.7, apart from a script written in Groovy to import annotations to QuPath. The code written for the proposed method is embedded in this file and explained in appendix A.

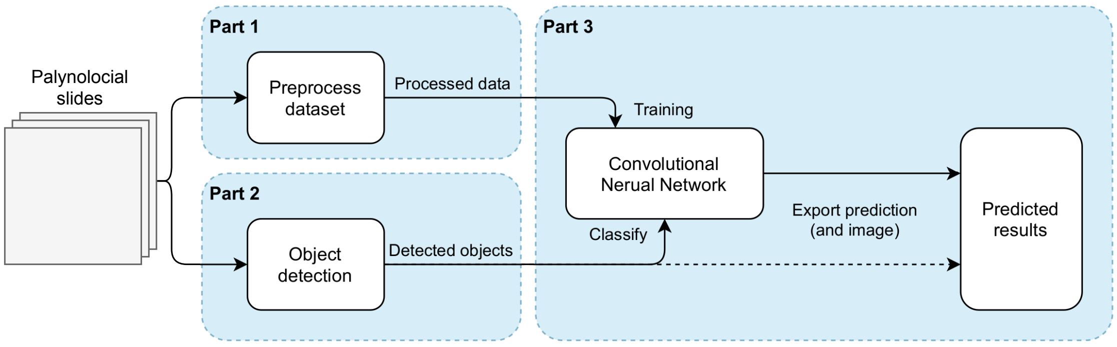


Figure 4.1: Overview over the process pipeline. Part 1 covers generating and exporting the data used to train the neural network. Part 2 is the object detection by using traditional image processing. Part 3 covers the setup and use of a convolutional neutral network to classify the dinoflagellates

4.1 Preprocess Dataset - Part 1

The dataset delivered from the Norwegian Petrol Directorate contained annotated palynological slide images in the mirax (.mrxs) format. This is an industrial and relative closed format, consisting of one .mrxs file as well as a folder containing the actual image split into different .dat-files. The folder also contains one configuration file (.ini) containing metadata. This means that there is no single file where the annotations can be easily extracted. The Python library OpenSlide have methods to read and extract regions from the image files but is not able to extract or read annotations from the slide image. Luckily, the 3DHistec's Slide converter is able to extract the annotations in a conversion process and export the annotations to an XML-file, such that each slide will have an associated XML-file containing all annotations.

Annotations exported from CaseViewer are defined as general polygons using $n+1$ points, where n is the number of line segments defined by the polygon. The annotations were mostly done using squares, such that the location of an annotation consisted of five sets of (x,y) pixel coordinates.

After parsing, each annotation extracted is defined by five parameters:

- x_0 - horizontal position of the top left corner in pixel coordinates
- y_0 - vertical position of the top left corner in pixel coordinates
- w - width of object in pixels
- h - height of object in pixels
- name - Extracted name of object from annotation, i.e. object class

By inspecting the shape and size of the different objects, a standardized region size in the lowest layer of 512 by 512 pixels was chosen. This gave a good overview, about 127 by 127 micrometer, for all of the objects in the dataset without including too much background as shown in figure 4.2. In practice the file size and resolution were a bit large, so by using the layer property of the mirax format the second layer was chosen. This reduces the resolution to 256 by 256, while still covering the same physical region.

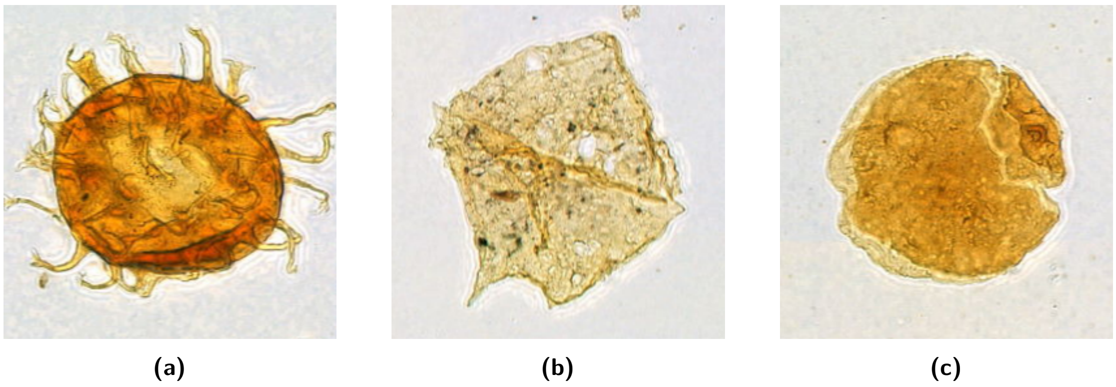


Figure 4.2: Example of the extracted regions of 127 by 127 µm containing different species of dinoflagellates: (a) Fibrocysta Axialis, (b) Palaeoperidinium Pyrophorum, and (c) Senoniasphaera Inornata

To expand the annotated region to a standardized shape, equation 4.1 was used. x_0 and y_0 are the original top left corner of the object, w and h are the width and height of the object, and w' and h' are the width and height of the new region. This gives x'_0 and y'_0 corresponding to the new top left pixel with the object in the center.

$$\begin{aligned} x'_0 &= x_0 + \frac{w}{2} - \frac{w'}{2} \\ y'_0 &= y_0 + \frac{h}{2} - \frac{h'}{2} \end{aligned} \tag{4.1}$$

All the images were exported and saved to individual folders named by the species name, as well as a file containing the filenames, originating slide filename, species name, position and shape.

4.1.1 Implementation

The methods used in this section was composed of a mixture of pre-made and self-made function. Table 4.1 shows what packages were used and what was self-made. To parse the annotation file, the minidom package within python was used. To export the dataset; pillow was used to save images and numpy was used to save a list of all the objects.

Method	Python XML-parser (minidom)	Numpy	Pillow	Self Made
Parse Annotation File	x			x
Expand Object Window				x
Export Dataset		x	x	x

Table 4.1: Packages and self-made functions that were used in section 4.1

4.2 Object Detection - Part 2

Object detection is Part 2 as illustrated in figure 4.1. In figure 4.3, this process is shown in greater detail. Each slide image is preprocessed, before a block-wise object detection algorithm is performed.

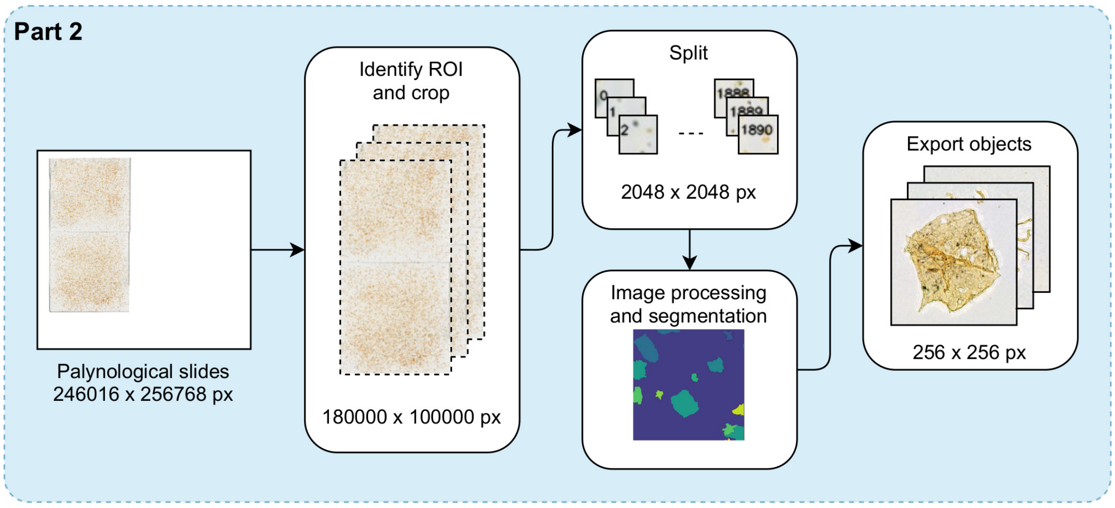


Figure 4.3: Detailed procedure for Part 2. Each slide image is cropped and split into blocks. A block-wise object detection algorithm is performed and the objects exported

4.2.1 Palyslide Image Preprocessing

The whole slide image as it is created by the scanner is too large to process directly. Without any preprocessing, the scanned image is in the order of 250,000 by 250,000 pixels, even though the palyslide itself cover a region of about 184,000 by 96,000 pixels. In figure 4.4, the whole slide image is shown in its entirety (downsampled version).

Figure 4.4: Raw image file from the 3DHistec's P1000 scanner as read by OpenSlide in Python

In section 3.1 it was mentioned that the process of creating the slide, the dinoflagellates are diluted in a water solution before they are placed on a microscope slide and dried overnight. This process causes a higher concentration of dinoflagellates in the center of the coverslip and can just be seen in figure 4.4. This means that some of the edge can be removed without removing too many dinoflagellates. By removing the outer edge, the transition region from microscopic slide to coverslip is removed.

As mentioned in section 4.1, a single dinoflagellate will maximum occupy a region of about 512 by 512 pixels in the highest resolution layer. If a downsampled version of the whole slide image was to be used in object detection, e.g. layer 4, where the slide image is "only" 11,250 by 5,625 pixels, the size of the objects as well as artifacts caused by downsampling will make it to be improbable to distinguish between multiple objects. Especially for objects in close proximity to each other.

In order to preserve as much detail as possible, but still be able to perform object detection, the whole slide image is split into a grid, where each block covers a 512 by 512 micrometer region, i.e. 2048 by 2048 pixels at the highest resolution.

To extract the palyslide region in the image, algorithm 1 was used. The result this algorithm returns the position, width and height of the image region in pixel coordinates corresponding to the level of which the image was processed in. As this is a quite simple method, checking all pixels in the image, it should be self-explanatory that this is not possible to run at the highest resolution. By using algorithm 2, the pixel coordinates returned from algorithm 1 can be converted to any layer as for each layer down, the resolution is halved.

Algorithm 1: Finding the region of the slide (RoS) in the raw slide image

```
Input: Downsampled raw slide image
Output: Position (x0, y0), and shape (width, height) of the RoS
gray_image ← Convert input to grayscale
RoS ← where gray_image > 0 AND gray_image < 255
x0, y0 ← min(RoS)
width, height ← max(RoS) - (x0, y0)
Return (x0, y0), (width, height)
```

Algorithm 2: Level based point translation

```
Input: Point to translate (point), Level to translate from (from_level), Level to
       translate to (to_level)
Output: Translated point (translated_point)
translated_point ← point^(from_level - to_level)
Return translated_point
```

The grid is calculated such that the maximum number of whole blocks are placed within the width and height, centered in the image at the highest resolution level. To further remove the edge, i.e. the transition region from the microscope slide to the coverslip, the outer frame of blocks can be removed. As exporting each block as image files would occupy unnecessary disk space, only the block number, row and column index, position, and shape is saved.

4.2.2 Segmentation and Object Detection

As discussed in section 2.3.4 Segmentation, in order to detect and differentiate between objects in an image there exists a wide verity of methods. Amongst the discussed methods is the watershed method (in 2.3.4.3) which will be used here in its marker-based form. The markers will be created by using thresholding and morphological functions and are inspired by tutorials made available by the creators of OpenCV [51][52].

The first step in the process is to remove the background, such that a resulting binary image only contains objects in the foreground. A palyslide will often have many different types of dinoflagellates together with other residual material. This creates a wide range of shapes, sizes and colors in the image and can make removing the background a non-trivial problem. Figure 4.5 shows a block image and the result of thresholding it using Otsu's method (as discussed in section 2.3.4.1) directly. As shown, this also removes many of the objects in the process and would be sub-optimal for use in object detection.

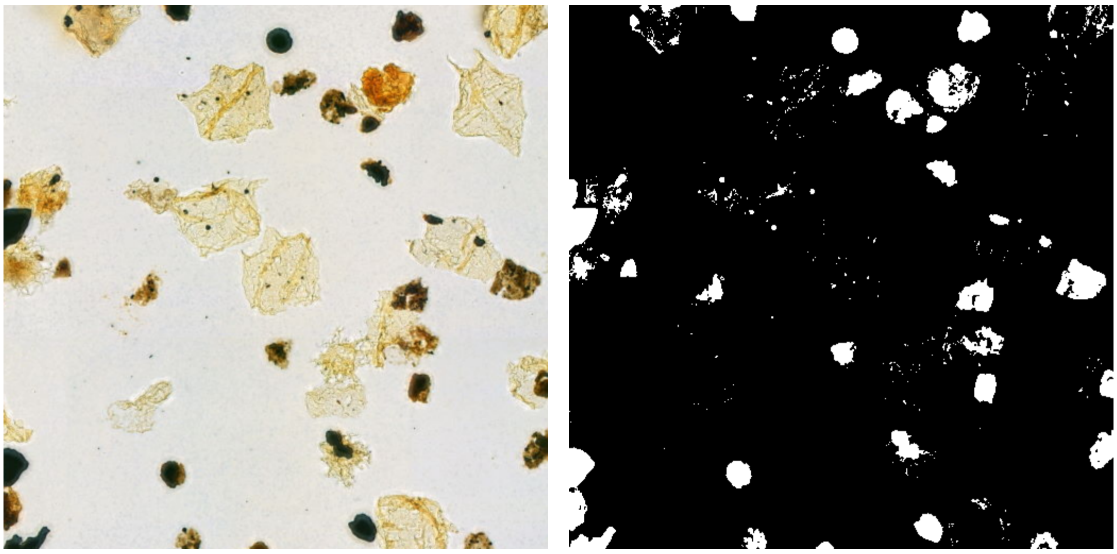


Figure 4.5: **(a)** A single block from the grid of the palyslide image. **(b)** The block thresholded by using Otsu's method. The first step in detecting objects is to remove the background of the image. As seen in **(b)**, Otsu's method is too aggressive and removes much of the foreground

By converting the image to the HSV format, the background can much easier be identified. Figure 4.6 shows each of the channels of the image in the HSV format, represented as grayscale images. Since the background is unsaturated the objects can easily be identified in the saturation channel. By subtracting an inverted thresholded version of the saturation channel from the original image the background could be removed.

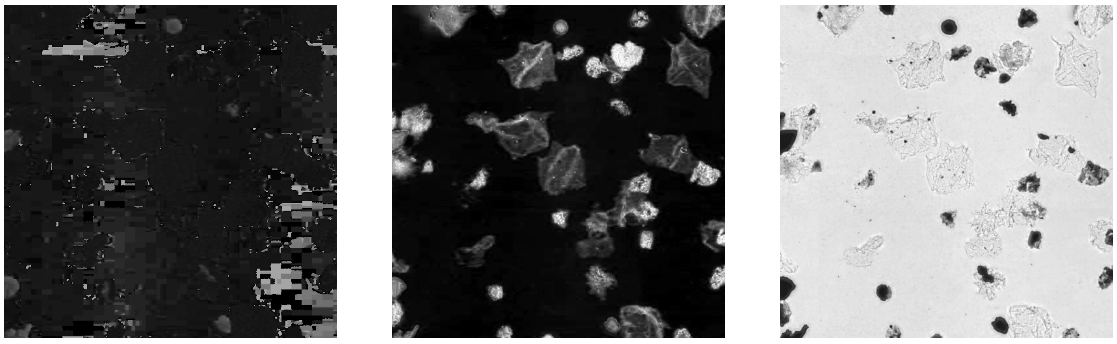


Figure 4.6: Each of the color channels in the image converted to the HSV format. **(a)** Hue, **(b)** Saturation, and **(c)** Value

As the image contains both very dark and very bright objects, the morphological and thresholding functions will most likely remove or over represent certain objects the image. To detect as many objects as possible the object detection is performed in two steps. First detect and identify the position of all the darker objects in the image using parameters that is suited to identify most objects. Afterwards, these are removed from the image, such that the rest can be detected using more sensitive parameters.

Figure 4.7 shows the process used to perform the object detection for each block in the palyslide image. Instead of removing the background from the original image, both the saturation and value channel are used directly. The hue channel does not contain any useful information about the location and shape of objects in the image and is therefore not used. Darker objects are found in step one, by inverting and thresholding the value channel, performing morphological transformation and calculating the distance map of the objects in the image. The distance map is thresholded to separate objects that are in close proximity or touching, and the resulting binary image contains the seed markers for each dark object. All the markers are labeled using connected component labeling and further used as the markers in the watershed algorithm.

In the step two, the brighter objects are found. In this step, both the saturation and inverted value channel are used and the darker objects, already detected, are removed. Darker objects are expanded using morphological dilation to ensure that the entire object is removed. Afterwards the second step follows the same procedure as in step one.

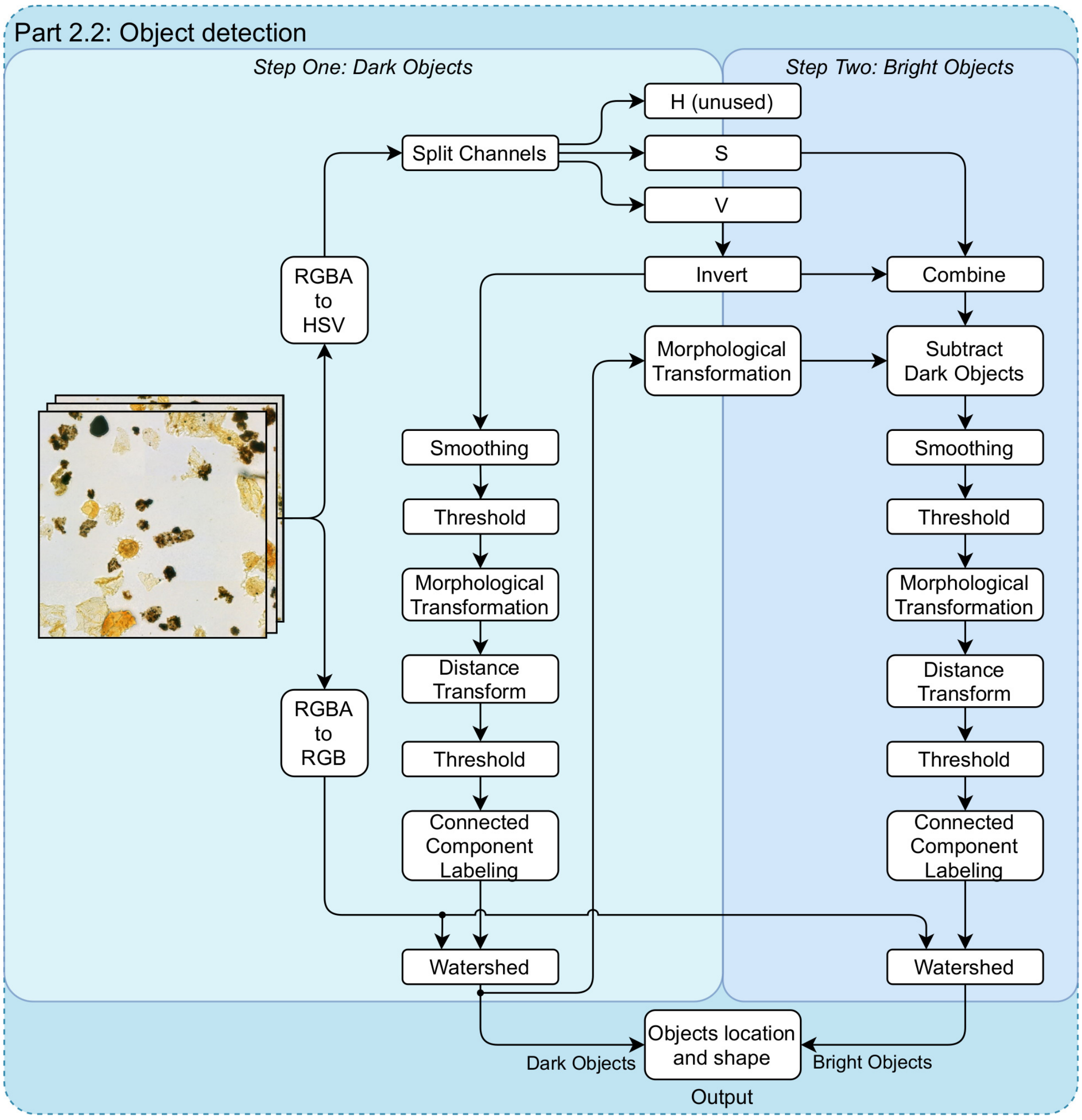


Figure 4.7: The process of the object detection method. Dark objects are first detected in step one, using parameters to detect most objects. Light objects are detected by using more sensitive parameters on the image in step two, where the dark objects are first removed

Figure 4.8 show the result of the proposed object detection algorithm performed on a single block. Each object region has its own label, here represented in different colors, such that it is possible to locate the position and shape of each object.

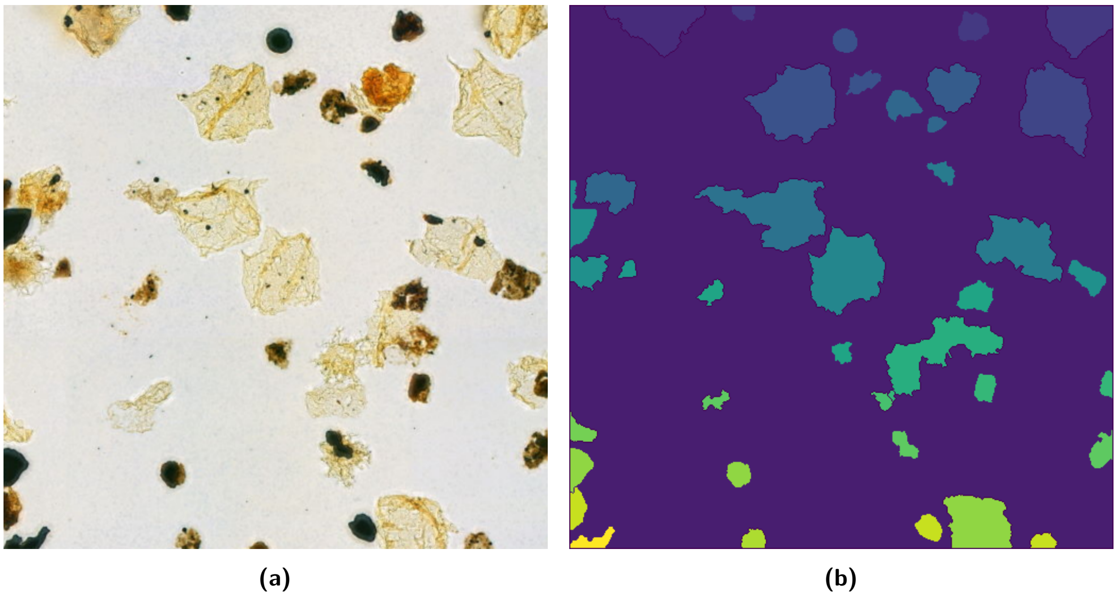


Figure 4.8: **(a)** Original image. **(b)** Resulting markers from the proposed object detection algorithm. Each object has its own label, here shown as different colors.

To excerpt the objects position and shape from the result of the watershed methods, algorithm 3 is used. This generates an object list in which each object gets a position defined by the position of its top left pixel, as well as the width and height of the object.

Algorithm 3: Object list from markers

```
Input: Output from the watershed algorithm (label_image)
Output: List with the position and size of all objects from the input (object_list)
Initialize empty object_list
for val in range(each unique value in label_image) do
    Initialize empty temporary_list
    forall pixels in label_image do
        if pixel == val then
            i, j ← pixel position
            add i, j to temporary_list
    end
    x0, y0 ← min(temporary_list)
    width, height ← max(temporary_list) - x0, y0
    add x0, y0, width, height to object_list
end
Return object_list
```

The absolute position of the object is calculated by adding the object position to the block's position. Depending on what resolution level the object detection method was run at, algorithm 2 can be used to transform the pixel coordinates to the correct level.

Before exporting, all objects smaller than 10 pixels in width or height were removed, as well as smaller objects contained within the bounding box of larger ones. A standard region was defined in section 4.1. To define the new window region, equation 4.1 was used and added for each object in the object list.

The final object list contains all the objects discovered in all the blocks. Numerated and containing all the objects position, widths and heights as defined by the highest resolution level of the image, as well as the position and size of a window with the object in the center.

4.2.3 Implementation

Table 4.2 shows the distribution of ready-made and self-made methods used in section 4.2.

Method	OpenCV	Numpy	Self Made
Find Edge	x		x
Point Level Transformation			x
Generate Grid		x	x
Convert Color Format	x		
Invert image			x
Smoothing	x		
Morphological Transformation	x		
Distance Transform	x		
Threshold	x		x
Connected Component Labeling	x		
Combine Image	x		
Subtract Image	x		
Watershed	x		
Extract Object Position			x
Object Size Threshold			x
Expand Object Window			x
Export Detected Objects		x	

Table 4.2: Packages and self-made functions that were used in section 4.2

4.3 Object Classification - Part 3

Object classification is marked as Part 3 in figure 4.1. This section will cover the general setup of the neural network as well as training and classification of the detected objects.

4.3.1 Transfer Learning

Since the dataset have quite few images, the decision to use transfer learning was made early on. Transfer learning makes, as discussed in section 2.4.7, a neural network able to perform better than training from scratch if the dataset is sparse.

To import the backbone network and create the new classifier, Keras [46] was used with the TensorFlow [53] back-end as this allows for easy implementation with human-readable code. Once a new model has been created it was trained in the procedure shown in figure 4.9. The dataset have been split to three parts, D_{train}, D_{val} and D_{test}, such that during training only D_{train} and D_{val} are used. D_{test} was exclusively used for performance evaluation and not used in training of the network.

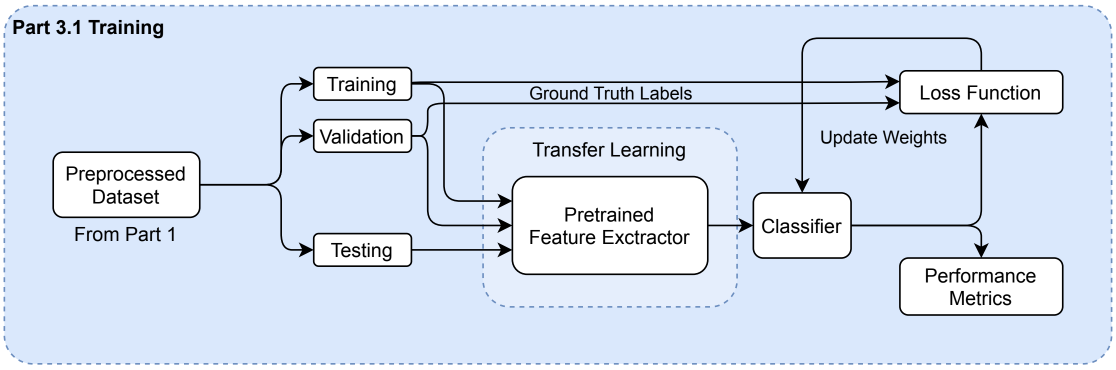


Figure 4.9: The network is trained by using the training and validation portion and evaluated using the test portion of the dataset

Once the model was trained to a satisfactory performance level it was saved such that it could be used to classify the objects detected by the object detection algorithm as shown in figure 4.10.

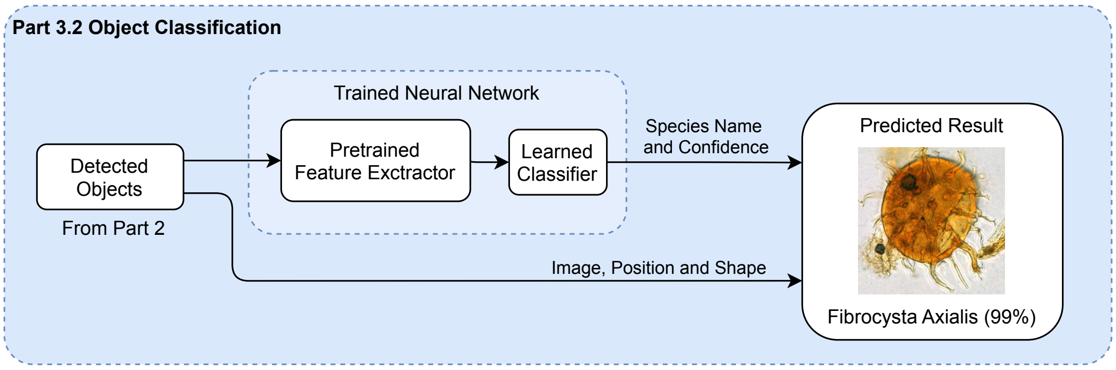

Part 3.2 Object Classification
Detected Objects
From Part 2
Trained Neural Network
Pretrained Feature Exctractor
Learned Classifier
Species Name and Confidence
Image, Position and Shape
Predicted Result
Fibrocysta Axialis (99%)

Figure 4.10: Detected objects from Part 2 can be entered to the trained neural network for classification

A separate script was made to be able to import the region as well as predicted class to QuPath for verification.

4.3.2 Implementation

Table 4.3 shows the distribution of ready-made and self-made methods used in section 4.3.

Method	Keras	SciKit	OpenCV	Pillow	Numpy	Self Made
Import Dataset Path and Class Labels					x	x
Import and Preprocess Dataset	x					x
Load Model Backbone	x					
Create New Classifier	x					x
Training, Loss Function and Updating Weights	x					
Performance Metrics	x	x				
Export and Import Saved Model	x					
Export Identified Objects			x	x	x	x
Import Annotations to QuPath						x

Table 4.3: Packages and self-made functions that were used in section 4.3

Chapter 5

Experiments and Results

5.1 Object Detection Performance Evaluation

The method for detecting objects using the proposed object detection method was described in section 4.2. The whole slide image is split into a grid before object detection is performed on each block. In principle, the method should detect all the objects in the image, such that all objects can be run through the neural network for classification.

In the lower left corner of the slide image from well 2/7-14, a total of 648 individual objects were marked using QuPath. The object detection algorithm should in principle detect all objects, not only dinoflagellates, so all objects within the area were marked.

5.1.1 Method

The slide image was split into a grid, as described in section 4.2, and a section of blocks that contained the markers were extracted. A region of 54 blocks, containing 464 of the 648 marked objects were chosen. The process of manually marking objects led some blocks not being fully marked and the extracted region was chosen such that all blocks contained only marked objects.

Figure 5.1 shows the extracted blocks from the slide image. Each block is represented by its number in the total amount of blocks (the whole image contained a total of 4320 blocks) and each object is marked by a white dot. Each block has a resolution of 2048 by 2048 pixels, which gives the total region 12288 by 18432 pixels or approximately 3.1 by 4.6 millimeter of the palynological slide.

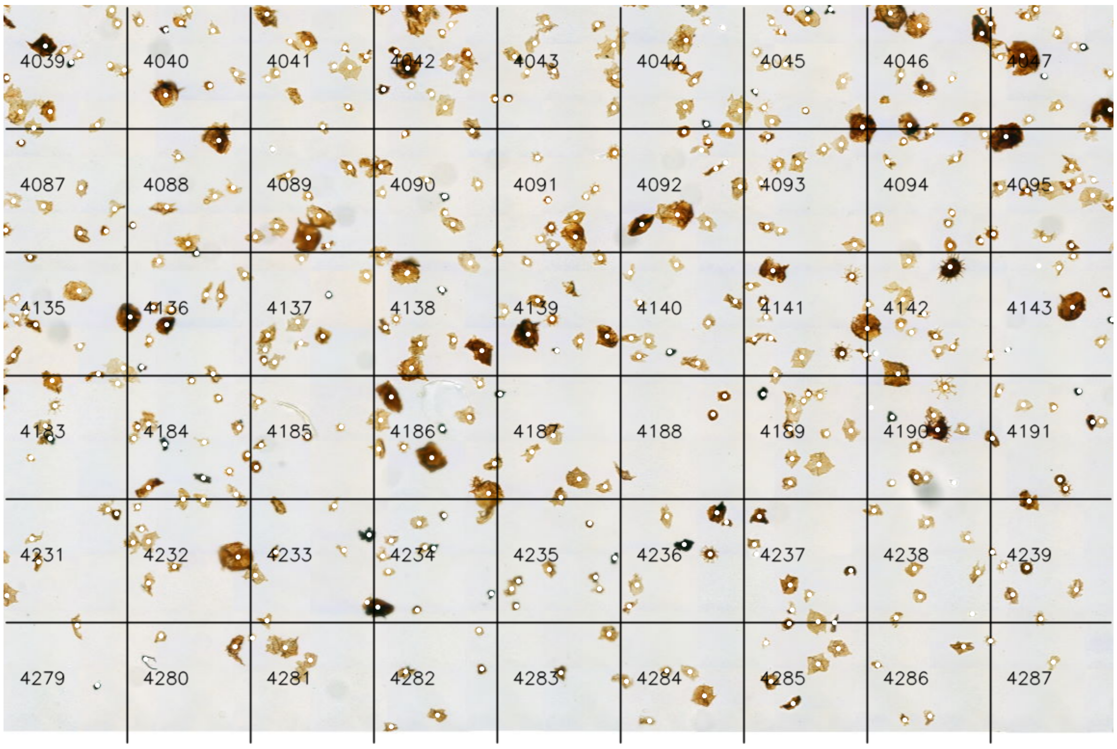


Figure 5.1: To assess the performance of the object detection algorithm a region consisting of 54 blocks from slide image 2/7-14 was tested. A total of 464 object are contained in the image, represented by a white dot

5.1.2 Results

Each block was run through the object detection algorithm producing a list of detected objects. Each detected object is represented by its position, width and height, creating a bounding box around the object. For the purpose of testing the performance of this algorithm, an object was said to be detected if the bounding box contained a marker.

- Objects detected (TP): 433 of 464
- Objects detected only once: 429
- Objects detected more than one: 4
- Undetected objects (FN): 31
- Detections without marker (FP): 269
- Total detections: 676

$$\text{Accuracy} = \frac{\text{TP}}{\text{TP+FP+TN+FN}} = \frac{433}{433 + 269 + 0 + 31} = 59.1\% \qquad (5.1)$$

$$\text{Precision} = \frac{\text{TP}}{\text{TP+FP}} = \frac{433}{433 + 269} = 61.7\% \tag{5.2}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP+FN}} = \frac{433}{433 + 31} = 93.3\% \tag{5.3}$$

Figure 5.2 shows the all the blocks, manually marked objects in white and the bounding box for each object detected in black.

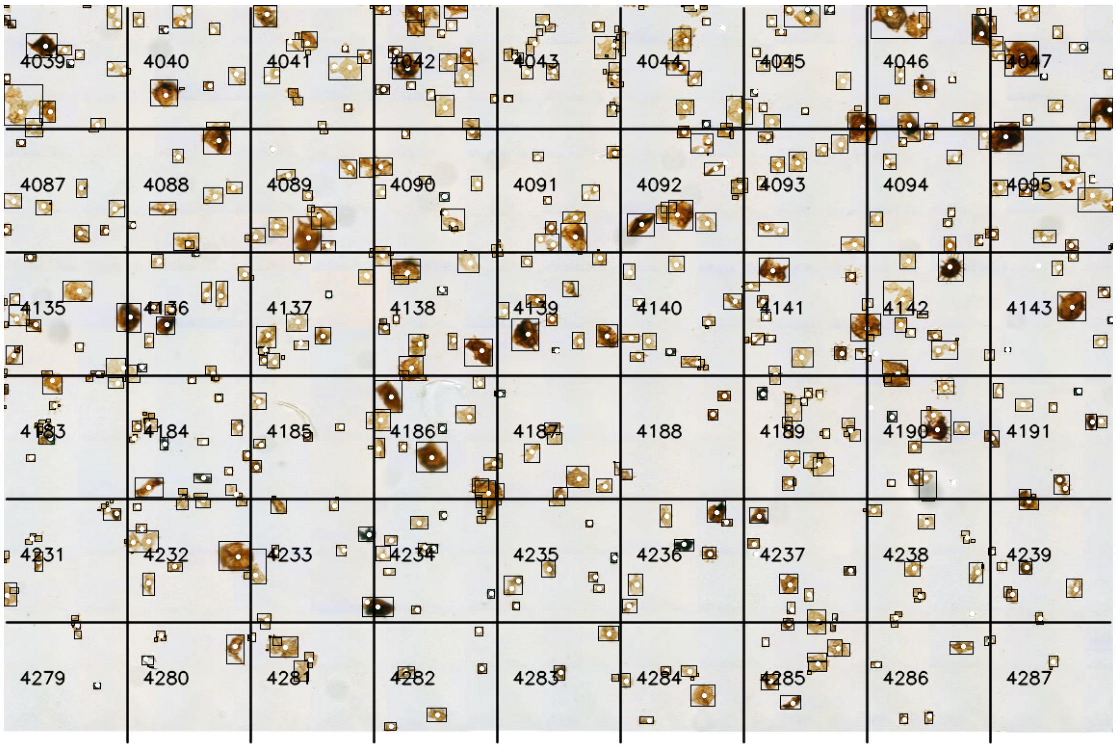


Figure 5.2: Black squares indicates the position and shape of the objects found by the object detection algorithm. Manually annotated objects are marked as white dots

5.1.3 Conclusion

The focus of the object detection method was to be able to detect all the objects within each block of the image. This process does however produce some issues; if an object is on the line between two or more blocks it can and often will be detected multiple times. In figure 5.2, the corner between blocks 4045, 4046, 4093 and 4094 a single object have been detected four times, one for each block. As the method is trying to detect as many objects as possible in two steps, the number of false positives shows that a single object can be detected several times, this can be seen in the lower right corner of block 4186.

These issues can however be removed in post processing. Within object detection neural networks, a frequent practice is to use non-maximum suppression. If two or more objects identified to the same class overlap or is adjacent within a certain threshold, only the object with the highest confidence is kept.

By having some oversegmentation, there is however a higher chance of separating actual adjacent objects. In the results, only four objects were placed within the same bounding box as another one. The most important metric for this algorithm will therefore be the recall. It would be preferred to remove false positives in post-processing than to miss an object all together. Only 6 percent of the markers were missed by the algorithm.

5.2 Transfer Learning - Backbone Network Comparison

5.2.0.1 Backbone Networks

Different models with different network structure will most likely extract different types of features. The purpose of this experiment is to see what network structure would produce the best initial result given the same premise. This does however not reflect the total performance possible to obtain in a given network, but rather what feature extractor produce the most separable results given images of dinoflagellates and default parameters.

A selection of eight different network structures were selected and are shown in table 5.1, all of which performed above 90 percent in the top-5 accuracy of the ImageNet validation dataset. The table also shows the total amount of parameters in each network, as well as the number of parameters that have been trained using the ImageNet dataset. None of which were trained during this experiment but were included to give a sense of how large each network is.

	Top-1	Top-5	Parameters	Trainable parameters in feature extractor
InceptionV3	0.779	0.937	23,851,784	21,768,352
MobileNetV2	0.713	0.901	3,538,984	3,206,976
NASNetLarge	0.825	0.960	88,949,818	84,720,150
InceptionResNetV2	0.803	0.953	55,873,736	54,276,192
ResNet152	0.766	0.931	60,419,944	58,219,520
ResNet152V2	0.780	0.942	60,380,648	58,187,904
VGG16	0.713	0.901	138,357,544	14,714,688
VGG19	0.713	0.900	143,667,240	20,024,384

Table 5.1: A selection of the pre-trained networks available with Keras in Python. Top-1 and Top-5 refers to the networks' accuracy on the ImageNet validation set [46]

5.2.0.2 Network Parameters and Dataset

All the networks were imported as frozen, without the classifying layers and loaded using the pretrained ImageNet weights. The classifier used in all the tests consisted of three fully connected layers of 512, 256 and 128 neurons respectively using the ReLU activation function, structured as shown in figure 5.3. The output layer consisted of five neurons using the SoftMax activation function. The Adam-optimizer was used in the training process with default settings, with categorical cross-entropy as the loss function.

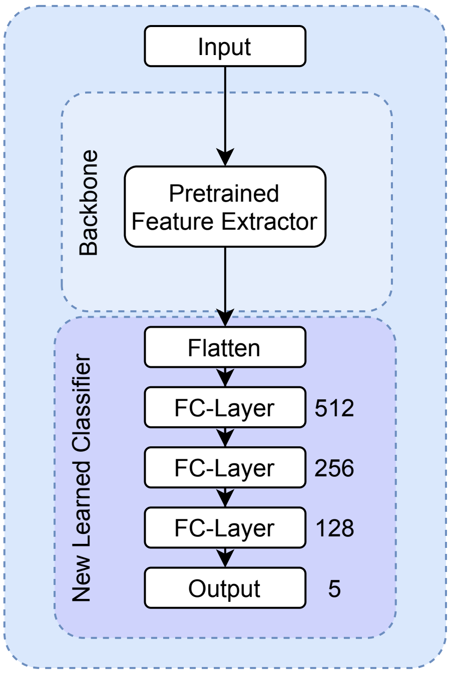


Figure 5.3: Network structure used in the experiment consisted of a pretrained feature extractor from a base model with four new fully connected layers

All networks were set to use the same input size of 224 by 224, and individual pre-process functions, as provided by Keras, were applied to all images before entering the network. They were trained in a total of 20 epochs using a batch size of 32 with D_{train} and validated on D_{val}, with the dataset as described in section 3.3.

5.2.0.3 Results

Figure 5.4 and 5.5 shows the shows the accuracy and F1-score obtained by the networks on the test set. Most of the networks performed surprisingly well, seven out of eight obtained an accuracy of above 80 percent. Both VGG16 and VGG19 obtained an accuracy of 95 percent, well above the other six networks, and performed similarly in both recall and precision. The training graphs also showed a significant low bias over all the networks, but only VGG16 and VGG19 had a low variance as well. Training accuracy and loss graphs as well as plot over recall and precision is available in appendix C.

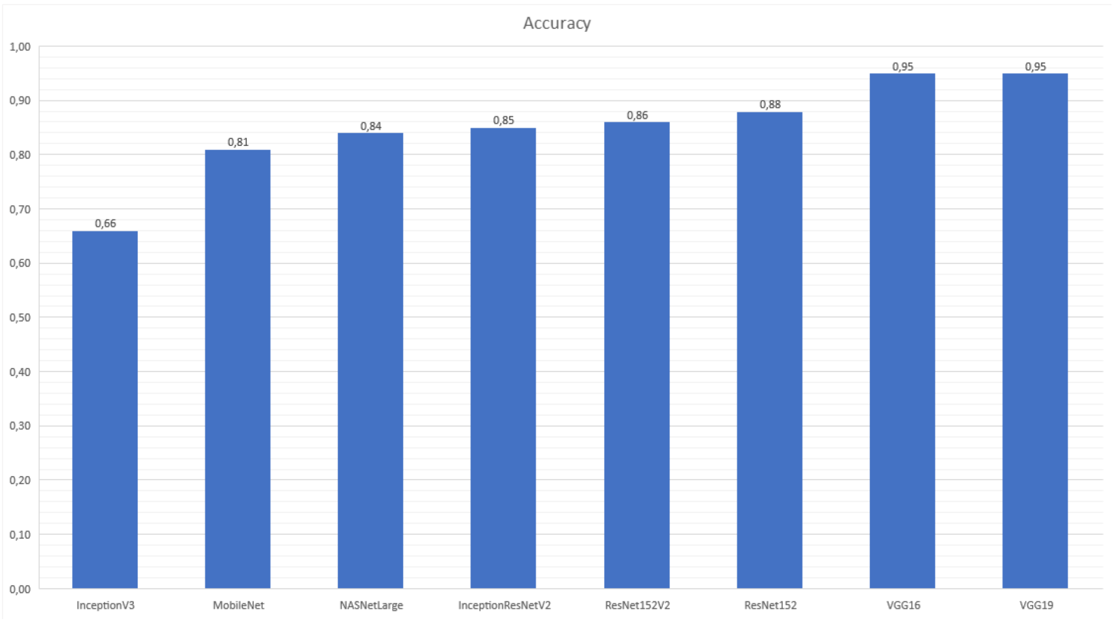
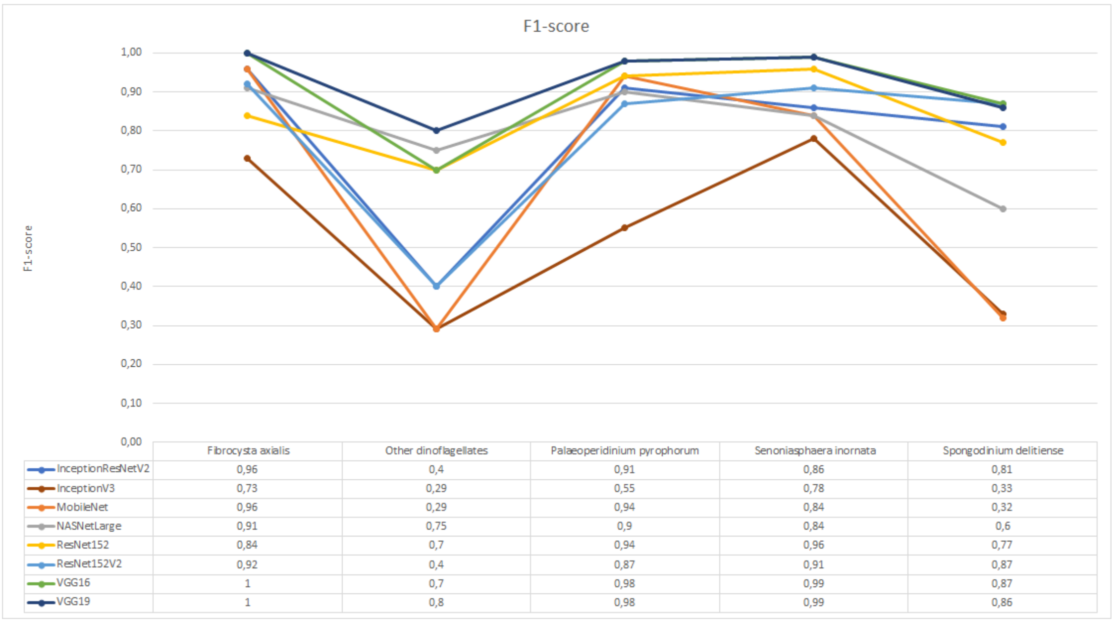


Figure 5.4: Accuracy obtained on D_{test} using different base networks



	Fibrocysta axialis	Other dinoflagellates	Palaeoperidinium pyrophorum	Senoniasphaera inornata	Spongodinium delitiense
InceptionResNetV2	0,96	0,4	0,91	0,86	0,81
InceptionV3	0,73	0,29	0,55	0,78	0,33
MobileNet	0,96	0,29	0,94	0,84	0,32
NASNetLarge	0,91	0,75	0,9	0,84	0,6
ResNet152	0,84	0,7	0,94	0,96	0,77
ResNet152V2	0,92	0,4	0,87	0,91	0,87
VGG16	1	0,7	0,98	0,99	0,87
VGG19	1	0,8	0,98	0,99	0,86

Figure 5.5: F1-score obtained on the different classes using different base networks

5.2.0.4 Conclusion

With the same premise both VGG16 and VGG19 outperformed the other deep neural networks that was tested. While the results might differ if using other settings and hyper-parameters, the point of the test was to determine what network would be the best backbone for further development of a neural network for classifying dinoflagellates.

VGG16 and VGG19 are both regular convolutional neural networks in the sense that they only consist of convolutional, pooling and fully connected layers. Not counting pooling layers, VGG16 have 16 layers and VGG19 have 19 layers, between the input and the output layer [54]. As the difference between the networks were marginal, VGG16 was chosen to be used further. VGG16 have fewer layers and parameters which might reduce the memory and hardware requirements during both training and the later image classification process.

5.3 Object Classification Performance

The VGG16 network that was chosen obtained a 95 percent accuracy on the test set during the selection process of the base network. This is already a high accuracy, but might not necessary represent the best possible combination of hyperparameters for the optimal result.

5.3.1 Classifier Hyperparameter-Tuning

Deciding the hyperparameters in a neural network can be a time-consuming task. Many parameters can and often are correlated such that changing one will affect the others. This task will often require changing one or more parameters and retrain the network multiple times.

There are several tactics in which to try to find the optimal set of parameters. The most straightforward method is the grid search method, in which all combination in within a range are tried. This makes it also one of the more time-consuming method as the network needs to be retrained for all combinations. Another, closely related to the grid method, is the random search method, in which a randomized set in a range of hyperparameters are tested.

As a combination of these method, a randomized portion of the grid search was used to evaluate the performance impact of different hyperparameters in the classifier. Three sets of hyperparameter ranges were used, where a randomized small amount, a total of 75 of the possible combinations were tried and evaluated as shown in table 5.2.

Run	1	2	3
Batch Size	16, 32	16, 32	32
Epochs	20-100	100	20
Dropout	0-0.5	0-0.5	0.3-0.6
Learning Rate	$0.1 \cdot 10^{-6}$ - $1.8 \cdot 10^{-3}$	$0.1 \cdot 10^{-6}$ - $1.8 \cdot 10^{-3}$	$0.6 \cdot 10^{-6}$ - $1.0 \cdot 10^{-3}$
FC-Layer 1	16-128	128-1024	256-512
FC-Layer 2	16-128	128-1024	256-512
FC-Layer 3	16-128	N/A	256-512
Activation Functions	ELU, SeLU, ReLU	ELU, SeLU, ReLU	ReLU
Permutations	15	48	12

Table 5.2: Three runs of randomized grid search for selecting the classifier's hyperparameters

To automatically perform the training using different hyperparameters the Python package Talos [55] was used together with Keras [46] and Tensorflow [53]. The input layer size was increased to 256 by 256 by 3 as to avoid downsampling the images. The Adam optimizer was used with the categorical cross-entropy loss function as before.

5.3.1.1 Result

For each run, the performance were listed according to the accuracy obtained on the D_{val} portion of the dataset. Tables 5.3, 5.4 and 5.5 shows the top 3 performance from each run.

Epochs	Batch Size	Dropout	FC-1	FC-2	FC-3	Activation Function	Learning Rate	Validation Loss	Validation Accuracy
20	8	0.1	128	128	128	ReLU	0.0005	0.6328	0.9712
20	32	0.1	128	128	32	ReLU	0.001	1.4315	0.9519
100	8	0.1	128	32	32	ReLU	0.001	2.2653	0.9519

Table 5.3: Top 3 from grid search run one

Epochs	Batch Size	Dropout	FC-1	FC-2	FC-3	Activation Function	Learning Rate	Validation Loss	Validation Accuracy
100	32	0	256	512	N/A	ReLU	0.00124	0.23	0.99
100	16	0.1	256	512	N/A	SeLU	0.000806	0.37	0.99
100	32	0	256	256	N/A	SeLU	0.001801	0.34	0.98

Table 5.4: Top 3 from grid search run two

Epochs	Batch Size	Dropout	FC-1	FC-2	FC-3	Activation Function	Learning Rate	Validation Loss	Validation Accuracy
20	16	0.4	512	256	256	ReLU	0.00108	2.8988	0.9615
20	32	0.4	256	256	256	ReLU	0.00072	1.2662	0.9615
20	32	0.6	256	512	256	ReLU	0.00084	0.5408	00.952

Table 5.5: Top 3 from grid search run three

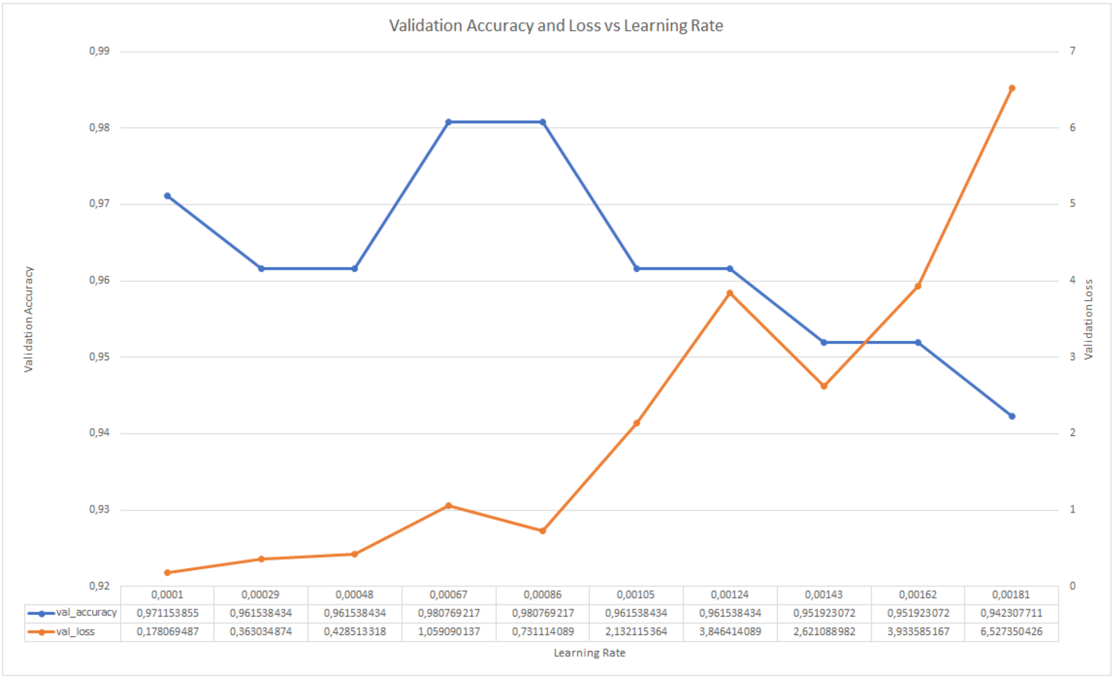
5.3.1.2 Conclusion

From the tables above it can be hard to see what hyperparameter have the most impact and especially since not all combinations were tried. It can however be seen that the highest accuracy and lowest loss were found using only two fully connected layers, and the ReLU activation function performed best in all three runs.

5.3.2 Learning Rate

As the grid search experiment above did not test all possible combinations of hyperparameters it was decided to perform a similar search using the best performing configuration, two fully connected layers using the ReLU activation function with 256 and 512 neurons, and only change the learning rate.

A selection of ten different learning rates in the range from 0.0001 to 0.00181 were tested. In figure 5.6 it can be seen that the learning rate affects both the accuracy and the loss of the network.



	0,0001	0,00029	0,00048	0,00067	0,00086	0,00105	0,00124	0,00143	0,00162	0,00181
val_accuracy	0,971153855	0,961538434	0,961538434	0,980769217	0,980769217	0,961538434	0,961538434	0,951923072	0,951923072	0,942307711
val_loss	0,178069487	0,363034874	0,428513318	1,059090137	0,731114089	2,132115364	3,846414089	2,621088982	3,933585167	6,527350426

Figure 5.6: Result of accuracy and loss by different learning rates

From this it was concluded that the optimal learning rate is in the region around 0.00086 as this produced the highest accuracy while maintaining a low loss of the validation data.

5.3.3 Data Augmentation

As the dataset is sparse a technique to improve performance is to use data augmentation. Using Keras this can be applied to any dataset on-the-fly using the `ImageDataGenerator`.

In table 5.6 the settings used during the final training run is shown and their effect is shown in figure 5.7.

Rotation Range	180
Width Shift Range	0.001
Height Shift Range	0.001
Horizontal Flip	Yes
Vertical Flip	Yes
Zoom Range	0.9 - 1.2
Channel Shift Range	40.0

Table 5.6: Data augmentation parameters

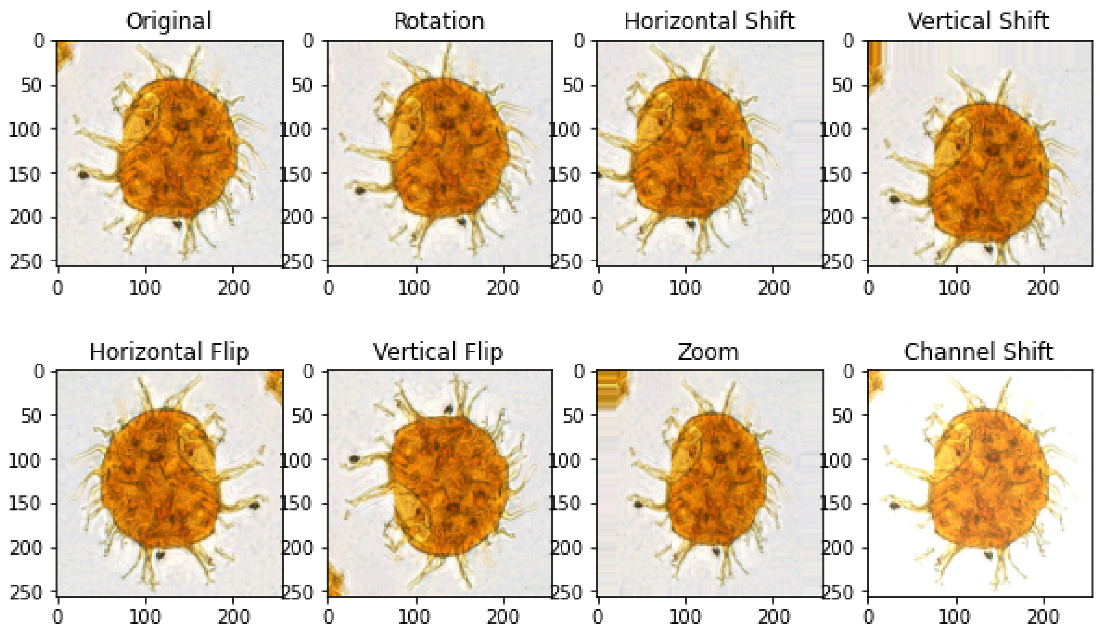


Figure 5.7: Image augmented using the `ImageDataGenerator` from Keras

5.3.4 Classification Result

Using the hyperparameters that were discovered in section 5.3.1 and 5.3.2 together with augmented data resulted in an accuracy of 99 percent on D_{test}, with only one image miss-classified, as shown by the confusion matrix in figure 5.8. Table 5.7 shows the classification report as generated by SciKit.

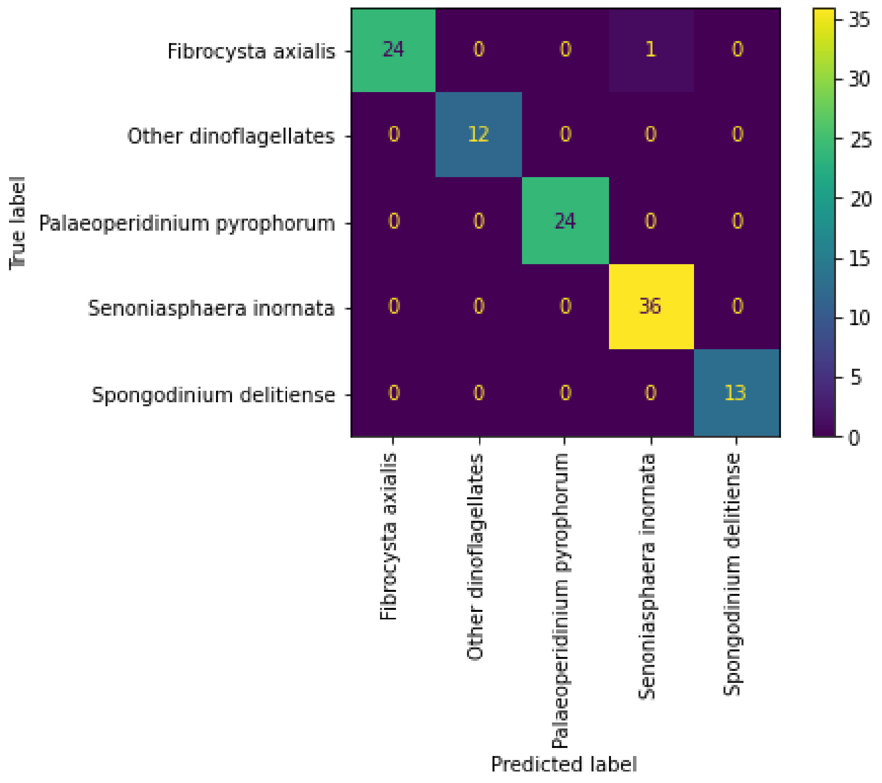


Figure 5.8: Confusion matrix showing the classification results of D_{test}

	Precision	Recall	F1-Score	Support
Fibrocysta Axialis	1.00	0.96	0.98	25
Other Dinoflagellates	1.00	1.00	1.00	12
Palaeoperidinium Pyrophorum	1.00	1.00	1.00	24
Senoniasphaera Inornata	0.97	1.00	0.99	36
Spongodinium Delitiense	1.00	1.00	1.00	13
Accuracy			0.99	110
Macro Avg	0.99	0.99	0.99	110
Weighted Avg	0.99	0.99	0.99	110

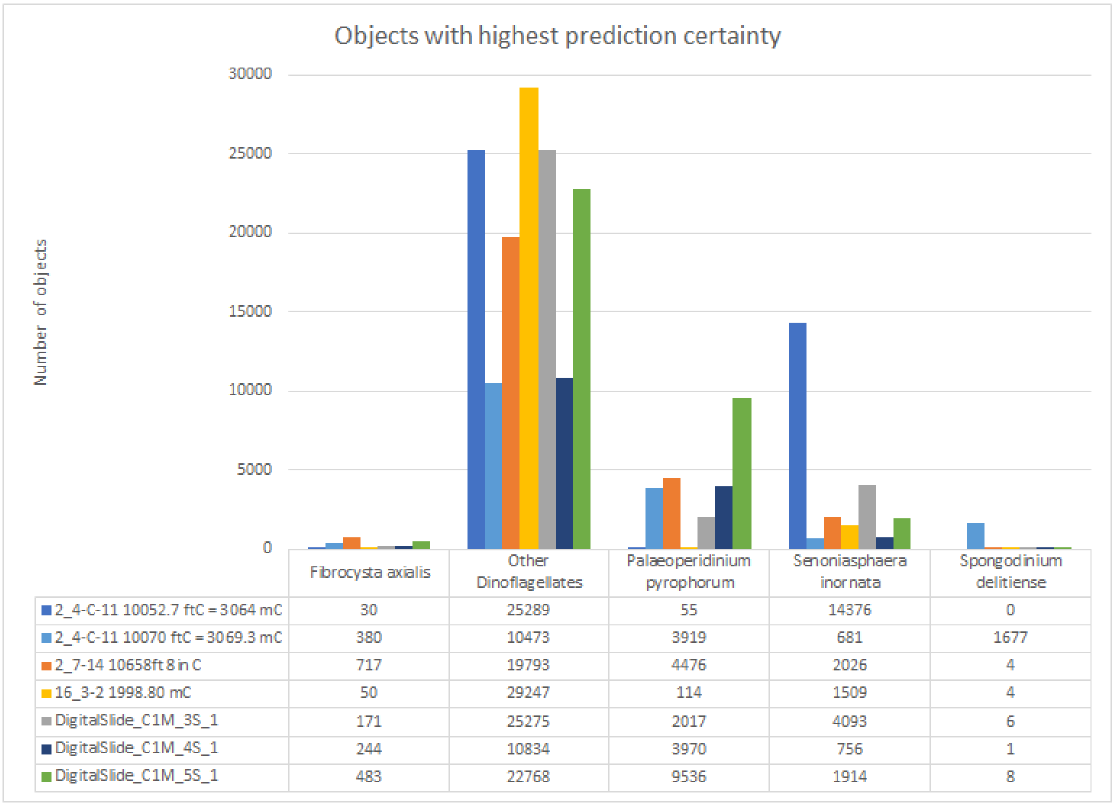
Table 5.7: Classification report of D_{test}

5.4 Palynological Slide Object Detection and Classification

As a final test both the object detection algorithm and object classifier were run on all the slides. Table 5.8 lists the number of objects found in each slide and figure 5.9 shows the distribution of the classified objects.

SlideName	Num of Objects	Time	Objects/sec
2_4-C-11 10052.7 ftC = 3064 mC	53335	32 min 47 sec	27.11
2_4-C-11 10070 ftC = 3069.3 mC	25080	15 min 54 sec	26.27
2_7-14 10658ft 8in C	39379	23 min 17 sec	26.75
16_3-2 1998.80 mC	34292	20 min 48 sec	27.47
DigitalSlide_C1M_3S_1	42976	27 min 24 sec	26.13
DigitalSlide_C1M_4S_1	20941	12 min 26 sec	28.06
DigitalSlide_C1M_5S_1	45748	28 min 36 sec	26.66

Table 5.8: Detected objects from the slides



	Fibrocysta axialis	Other Dinoflagellates	Palaeoperidinium pyrophorum	Senoniasphaera inornata	Spongodinium delitiense
🟦 2_4-C-11 10052.7 ftC = 3064 mC	30	25289	55	14376	0
🟦 2_4-C-11 10070 ftC = 3069.3 mC	380	10473	3919	681	1677
🟧 2_7-14 10658ft 8in C	717	19793	4476	2026	4
🟨 16_3-2 1998.80 mC	50	29247	114	1509	4
⬜ DigitalSlide_C1M_3S_1	171	25275	2017	4093	6
🟦 DigitalSlide_C1M_4S_1	244	10834	3970	756	1
🟩 DigitalSlide_C1M_5S_1	483	22768	9536	1914	8

Figure 5.9: Distribution of the classified objects

Chapter 6

Discussion

6.1 Palynological Slides and Dataset

The dataset consisted of seven palynological slides that had been scanned and contained a total of 530 annotations from 21 different species. 15 of the species had only one or two annotations, meaning they could not be used as individual classes during training. The two species Cribroperidinium "Prominoseptatum" and Spongodinium Delitiense (operculum) had 19 and 13 annotations, respectively. Only three classes had above one hundred annotations, and none above two hundred.

As mentioned in chapter 3, classes with a low number of annotations were combined into a single class named "Other dinoflagellates". This was done not only as an attempt to include more data during training, but also to increase the variance of the dataset. Having a dataset that only consist of similar images can be prone to overfitting.

Initially, the dataset contained 223 annotations made an expert at the Norwegian Petroleum Directorate. In an attempt to increase the dataset further, it was found that Senoniasphera Inornata, Paleoperidinium Pyrophorum, and Fibrocysta Axialis were distinctive enough to be clearly identified in slides where they had previously confirmed annotations. 111, 92, and 104 annotations were added to their respective class, bringing the total number of annotations to 530. There are many different types of dinoflagellates and many of them can be hard to differentiate for the untrained eye. The different degree of deterioration of the fossils also makes a complete annotation an almost impossible task.

As the annotations were all made using 3DHistech's CaseViewer this meant they were embedded in the mirax file and had no way to export annotations directly. Currently, the only way to export the annotations is to convert the file using 3DHistech's Slide

Converter. Each file had to be "converted" from MRXS to MRXS in which an option could be made to export annotation metadata together with a copy of the entire slide image.

6.1.1 Object Detection

6.1.1.1 Preprocessing

The resolution of the palynological slide images are extremely large. At a full resolution of 250,000 by 250,000 pixels, they are much too large to fit in a normal computer's memory. Downsampling or using a lower resolution layer such that the image could be processed would cause the dinoflagellates to become too small to differentiate. Because of this it was decided to split the image into parts. Using a grid pattern to partial process large images is a straightforward technique that was thought to be able to perform adequate.

The size of each block was chosen to be 2048 by 2048 at the highest resolution level, and thought to be a good balance between the number of blocks and cover area. At this size, each block contained approximately 20 to 50 objects, at a resolution high enough to get good separation of the objects in each block. It is possible that the larger block size would yield better results as larger blocks means fewer edges objects can be split between. The computational demand of many blocks versus large blocks was also not considered and could have an impact on the final computational speed.

6.1.1.2 Detection Algorithm

The object detection algorithm was produced using traditional image preprocessing techniques with the well-established and efficient library OpenCV. By converting the image to the HSV-format the background could easily be removed regardless of the other objects in the image and was a process that worked well. All the objects in the image could then in principle be detected directly by using connected component labeling, but this would create the same label for adjacent or gatherings of objects. To separate the objects, the distance map of the image was calculated and thresholded to create markers for the watershed algorithm. This would first separate objects into smaller markers and after, expand their region to cover the object. This procedure worked for gatherings and adjacent darker objects but would cause brighter objects to disappear. Because of this, the solution was to run the algorithm twice. By first running the algorithm once, the detected objects could be removed and make way for detecting the remaining brighter objects.

6.1.1.3 Object Detection Performance Experiment

As mentioned in the experiment of section 5.1, there are two current issues with the algorithm, both regarding over segmentation. By splitting the image into a grid, objects located on the edge of a block may be detected up to four times. Large objects with strange color combinations, e.g. black spots, would sometimes be split to multiple objects by the watershed algorithm. The former could be mitigated by having an overlapping grid and only detect objects within a frame. For the latter, fine tuning the algorithm seemed to only help marginally, and it was concluded that it was better to detect some objects twice and remove them if they are irrelevant in post-processing rather than trying to further tune the algorithm.

6.1.2 Convolutional Neural Network

As mentioned in section 1.2, previous work seemed to revolve around traditional machine learning techniques, extracting features such as shape, size, and color, and creating a support vector machine or small neural network to classify different types of plankton or dinoflagellates. In later years, the use of convolutional neural networks have skyrocketed and proved many times that it can produce high performing classifiers, all though it can be dependent on the amount of training data available. Many high performing convolutional neural networks have been trained using datasets containing hundreds of thousands (if not million) images.

6.1.2.1 Transfer Learning

When dealing with sparse dataset, transfer learning seems to be key and as can be seen from the results in section 5.2 and 5.3 it is possible to obtain well above 90 percent accuracy even before fine-tuning.

6.1.3 Final Results

Both the object detection algorithm and classification performed good in their individual performance test. With a detection rate of 94 percent for all objects, and correctly classifying 99 percent of the data in D_{test}.

The small amount of data in the dataset is however greatly limiting in the final evaluation of the results. The dataset have a low variance of within-class data, i.e. individual classes within D_{train}, D_{val} and D_{test} look similar and could even be from the same slide image.

The optimal solution would be to separate the data such that the split in dataset were all taken from different slide images and taken from a range of deterioration states. In addition, each class should have a similar amount of data. Using datasets which have uneven class distribution can cause imbalanced classification results.

Evaluating the performance on a whole palynological slide image would however be incredibly difficult. As listed in section 5.4, the amount of detected objects range from 20,000 to over 50,000 objects. These numbers are of course including any over-segmented objects, such that the actual number would be lower, but is still daunting if one was to annotate all objects. A final test could be performed on a smaller region in an image, but would require an expert to verify or to provide a fully annotated region only used for testing.

Chapter 7

Conclusion

7.1 Conclusion

The objective of this thesis was to explore the possibility of detecting and classifying dinoflagellates from images created by scanning palynological slides from the Norwegian Petroleum Directorate. The proposed method consists of detecting objects within the palynological slide images by using traditional image processing techniques and classifying them using a convolutional neural network.

From each slide image a grid was created such that each block contained 2048 by 2048 pixels. Using traditional image processing such as thresholding, mathematical morphology, distance transform and watershed on each block, a detection rate of 93.3 percent of the objects was obtained when tested on a large region with 464 marked objects within a palynological slide image.

For the convolutional neural network, the dataset consisted of a total of 530 annotations divided amongst 21 different species. Because of the imbalanced dataset, several species were collected into a single class. Despite the sparse dataset, using transfer learning with a VGG-16 [54] deep convolutional network trained on the ImageNet [56] dataset as the backbone, an accuracy of 0.99 was obtained as well as 100 percent precision and recall on four of five classes on the test portion of the dataset.

7.2 Future Work

Further improvement on the object detection algorithm can be done. The issue of over-segmenting objects can be improved by further fine-tuning the algorithm and implementing non-maximum suppression on classified objects. For objects near and across several blocks in the grid array this can be improved by using larger blocks or overlapping grids.

Creating a program to aid professionals annotate objects would be a great method of improving both the quality and quantity of data in the dataset. By gathering more data, more classes can be introduced, and a more generalized system can be made.

There is also the possibility of using other deep neural network object detection algorithms, such as YOLO [57] and the Fast Region-based Convolutional Network method (Fast-RCNN) [58].

List of Figures

List of Tables

Bibliography

[1] *The government's revenues*. Norwegianpetroleum.no. Library Catalog: www.norskpetroleum.no. URL: https://www.norskpetroleum.no/en/economy/governments-revenues/ (visited on 06/26/2020).

[2] *Resource accounts for the Norwegian shelf*. Norwegianpetroleum.no. Library Catalog: www.norskpetroleum.no. URL: https://www.norskpetroleum.no/en/petroleum-resources/resource-accounts/ (visited on 06/26/2020).

[3] *Seismic surveys*. Norwegianpetroleum.no. Library Catalog: www.norskpetroleum.no. URL: https://www.norskpetroleum.no/en/exploration/seismic-surveys/ (visited on 06/26/2020).

[4] Nils H. Lundberg and Nils Gundersen. *petroleumsleting*. In: *Store norske leksikon*. May 17, 2020. URL: http://snl.no/petroleumsleting (visited on 06/26/2020).

[5] *WILLIAM SMITH BIOGRAPHY | William Smith's Maps - Interactive*. Library Catalog: www.strata-smith.com. URL: http://www.strata-smith.com/?page_id=279 (visited on 06/26/2020).

[6] Birgitte Ferré Hjortkj. "Et geologisk arbejdsredskab". In: *GEOLOGI - Nyt fra Geus* 1 (1996), p. 12. ISSN: 1396-2353. URL: https://www.geus.dk/media/16739/nr-1-1996.pdf (visited on 06/25/2020).

[7] *Dinoflagellates*. The Palynological Society. Library Catalog: palynology.org. Jan. 3, 2015. URL: https://palynology.org/dinoflagellates/ (visited on 06/25/2020).

[8] *Dinosporin*. In: *Wikipedia*. Page Version ID: 909770383. Aug. 7, 2019. URL: https://en.wikipedia.org/w/index.php?title=Dinosporin&oldid=909770383 (visited on 06/25/2020).

[9] *Dinocyst*. In: *Wikipedia*. Page Version ID: 953210230. Apr. 26, 2020. URL: https://en.wikipedia.org/w/index.php?title=Dinocyst&oldid=953210230 (visited on 06/25/2020).

[10] H. P. Jeffries, M. S. Berman, A. D. Poularikas, et al. “Automated sizing, counting and identification of zooplankton by pattern recognition”. In: *Marine Biology* 78.3 (Feb. 1, 1984), pp. 329–334. ISSN: 1432-1793. DOI: 10.1007/BF00393019. URL: https://doi.org/10.1007/BF00393019 (visited on 06/28/2020).

[11] Xiaoou Tang, W. Kenneth Stewart, Luc Vincent, et al. “Automatic Plankton Image Recognition”. In: *Artificial Intelligence for Biology and Agriculture*. Ed. by S. Panigrahi and K. C. Ting. Dordrecht: Springer Netherlands, 1998, pp. 177–199. ISBN: 978-94-010-6120-9 978-94-011-5048-4. DOI: 10.1007/978-94-011-5048-4_9. URL: http://link.springer.com/10.1007/978-94-011-5048-4_9 (visited on 06/28/2020).

[12] Herry V, Beatriz Reguera, Sonsoles González-Gil, et al. “Dinoflagellate categorisation by artificial neural Network (DICANN)”. In: *Sea Technology* 43 (Dec. 1, 2002), pp. 39–46.

[13] Katja Schulze, Ulrich M. Tillich, Thomas Dandekar, et al. “PlanktoVision - an automated analysis system for the identification of phytoplankton”. In: *BMC Bioinformatics; London* 14 (2013). Num Pages: 115 Place: London, United Kingdom, London Publisher: BioMed Central, p. 115. DOI: http://dx.doi.org.ezproxy.uis.no/10.1186/1471-2105-14-115. URL: http://search.proquest.com/docview/1347648600/abstract/2EA99269E0E34CFDPQ/1 (visited on 06/25/2020).

[14] Haiyong Zheng, Ruchen Wang, Zhibin Yu, et al. “Automatic plankton image classification combining multiple view features via multiple kernel learning”. In: *BMC Bioinformatics* 18.16 (Dec. 28, 2017), p. 570. ISSN: 1471-2105. DOI: 10.1186/s12859-017-1954-8. URL: https://doi.org/10.1186/s12859-017-1954-8 (visited on 06/25/2020).

[15] *Pannoramic 1000*. 3DHISTECH Ltd. Library Catalog: www.3dhistech.com. URL: https://www.3dhistech.com/products-and-software/hardware/pannoramic-digital-slide-scanners/pannoramic-1000/ (visited on 06/08/2020).

[16] *Whole Slide Imaging | MBF Bioscience*. URL: https://www.mbfbioscience.com/whole-slide-imaging (visited on 06/27/2020).

[17] *Petroleum exploration*. 3DHISTECH Ltd. Library Catalog: www.3dhistech.com. URL: https://www.3dhistech.com/solutions/petroleum-exploration/ (visited on 06/28/2020).

[18] *OpenSlide*. URL: https://openslide.org/ (visited on 03/16/2020).

[19] *OpenSlide Python — OpenSlide Python 1.1.1 documentation*. URL: https://openslide.org/api/python/ (visited on 03/16/2020).

[20] *Mathematical morphology*. In: *Wikipedia*. Page Version ID: 937600704. Jan. 26, 2020. URL: https://en.wikipedia.org/w/index.php?title=Mathematical_morphology&oldid=937600704 (visited on 03/20/2020).

[21] *Morphology*. URL: http://homepages.inf.ed.ac.uk/rbf/HIPR2/morops.htm (visited on 03/20/2020).

[22] Alan C. Bovik, Bovik, and Alan C Bovik. *The Essential Guide to Image Processing*. San Diego, UNITED STATES: Elsevier Science & Technology, 2009. ISBN: 978-0-08-092251-5. URL: http://ebookcentral.proquest.com/lib/uisbib/detail.action?docID=452947 (visited on 03/24/2020).

[23] *OpenCV: Image Filtering*. URL: https://docs.opencv.org/trunk/d4/d86/group__imgproc__filter.html (visited on 03/23/2020).

[24] *Morphology - Erosion*. URL: http://homepages.inf.ed.ac.uk/rbf/HIPR2/erode.htm (visited on 04/24/2020).

[25] *Morphology - Dilation*. URL: http://homepages.inf.ed.ac.uk/rbf/HIPR2/dilate.htm (visited on 07/02/2020).

[26] *Glossary - Mathematical Morphology*. URL: https://homepages.inf.ed.ac.uk/rbf/HIPR2/matmorph.htm (visited on 03/18/2020).

[27] Gunilla Borgefors. "Distance transformations in digital images". In: *Computer Vision, Graphics, and Image Processing* 34.3 (June 1986), pp. 344–371. ISSN: 0734189X. DOI: 10.1016/S0734-189X(86)80047-0. URL: https://linkinghub.elsevier.com/retrieve/pii/S0734189X86800470 (visited on 03/26/2020).

[28] Nobuyuki Otsu. "A Threshold Selection Method from Gray-Level Histograms". In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1 (Jan. 1979), pp. 62–66. ISSN: 0018-9472, 2168-2909. DOI: 10.1109/TSMC.1979.4310076. URL: http://ieeexplore.ieee.org/document/4310076/ (visited on 04/05/2020).

[29] Robert Fisher, Simon Perkins, Ashley Walker, et al. *Image Analysis - Connected Components Labeling*. 2003. URL: http://homepages.inf.ed.ac.uk/rbf/HIPR2/label.htm (visited on 05/22/2020).

[30] *The Watershed Transformation page*. URL: http://www.cmm.mines-paristech.fr/~beucher/wtshed.html (visited on 05/25/2020).

[31] Rafael C. Gonzalez and Richard E. Woods. *Digital image processing*. 3rd ed. Upper Saddle River, N.J: Prentice Hall, 2008. 954 pp. ISBN: 978-0-13-168728-8.

[32] F. Meyer. "Color image segmentation". In: *1992 International Conference on Image Processing and its Applications*. 1992 International Conference on Image Processing and its Applications. Apr. 1992, pp. 303–306. ISBN: 0-85296-543-5.

[33] T. Breitenfeld, M. J. Jurasic, and D. Breitenfeld. “Hippocrates: the forefather of neurology”. In: *Neurological Sciences* 35.9 (Sept. 2014), pp. 1349–1352. ISSN: 1590-1874, 1590-3478. DOI: 10.1007/s10072-014-1869-3. URL: http://link.springer.com/10.1007/s10072-014-1869-3 (visited on 05/02/2020).

[34] Sergios Theodoridis. *Machine learning: a Bayesian and optimization perspective.* OCLC: 910913108. Amsterdam Boston Heidelberg London New York Oxford Paris San Diego San Francisco Singapore Sydney Tokyo: Elsevier, AP, 2015. 1050 pp. ISBN: 978-0-12-801522-3.

[35] *What is a neuron?* Library Catalog: qbi.uq.edu.au. Nov. 22, 2016. URL: https://qbi.uq.edu.au/brain/brain-anatomy/what-neuron (visited on 05/02/2020).

[36] *nervecelle – Store medisinske leksikon.* URL: https://sml.snl.no/nervecelle (visited on 05/02/2020).

[37] Warren S. McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The Bulletin of Mathematical Biophysics* 5.4 (Dec. 1943), pp. 115–133. ISSN: 0007-4985, 1522-9602. DOI: 10.1007/BF02478259. URL: http://link.springer.com/10.1007/BF02478259 (visited on 05/03/2020).

[38] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning.* MIT Press, 2016. URL: https://deeplearning.org.

[39] *AlphaGo: The story so far.* Deepmind. Library Catalog: deepmind.com. URL: /research/case-studies/alphago-the-story-so-far (visited on 05/29/2020).

[40] Ian H. Witten, Eibe Frank, Mark A. Hall, et al. “Chapter 10 - Deep learning”. In: *Data Mining (Fourth Edition).* Ed. by Ian H. Witten, Eibe Frank, Mark A. Hall, et al. Morgan Kaufmann, Jan. 1, 2017, pp. 417–466. ISBN: 978-0-12-804291-5. DOI: 10.1016/B978-0-12-804291-5.00010-6. URL: http://www.sciencedirect.com/science/article/pii/B9780128042915000106 (visited on 06/12/2020).

[41] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. “Searching for Activation Functions”. In: *arXiv:1710.05941 [cs]* (Oct. 27, 2017). arXiv: 1710.05941. URL: http://arxiv.org/abs/1710.05941 (visited on 06/12/2020).

[42] Y. Le Cun, L.D. Jackel, B. Boser, et al. “Handwritten digit recognition: applications of neural network chips and automatic learning”. In: *IEEE Communications Magazine* 27.11 (Nov. 1989), pp. 41–46. ISSN: 0163-6804. DOI: 10.1109/35.41400. URL: http://ieeexplore.ieee.org/document/41400/ (visited on 05/10/2020).

[43] Kyoung-Su Oh and Keechul Jung. “GPU implementation of neural networks”. In: *Pattern Recognition* 37.6 (June 2004), pp. 1311–1314. ISSN: 00313203. DOI: 10.1016/j.patcog.2004.01.013. URL: https://linkinghub.elsevier.com/retrieve/pii/S0031320304000524 (visited on 05/10/2020).

[44] *Tensor Cores in NVIDIA Volta GPU Architecture*. NVIDIA. Library Catalog: www.nvidia.com. URL: https://www.nvidia.com/en-us/data-center/tensorcore/ (visited on 05/10/2020).

[45] Olga Russakovsky, Jia Deng, Hao Su, et al. "ImageNet Large Scale Visual Recognition Challenge". In: *arXiv:1409.0575 [cs]* (Jan. 29, 2015). arXiv: 1409.0575. URL: http://arxiv.org/abs/1409.0575 (visited on 06/17/2020).

[46] Keras Team. *Keras documentation: Keras Applications*. Library Catalog: keras.io. URL: https://keras.io/api/applications/ (visited on 06/20/2020).

[47] Norsk Oljemuseum. *Ekofisk*. In: *Store norske leksikon*. Oct. 29, 2019. URL: http://snl.no/Ekofisk (visited on 05/29/2020).

[48] *Wellbore - Factpages - NPD: 2/4-C-11*. URL: https://factpages.npd.no/en/wellbore/pageview/development/all/945 (visited on 06/08/2020).

[49] *Wellbore - Factpages - NPD: 2/7-14*. URL: https://factpages.npd.no/en/wellbore/pageview/exploration/all/116 (visited on 06/08/2020).

[50] *Wellbore - Factpages - NPD: 16/3-2*. URL: https://factpages.npd.no/en/wellbore/pageview/exploration/all/334 (visited on 06/08/2020).

[51] *OpenCV: Image Segmentation with Distance Transform and Watershed Algorithm*. URL: https://docs.opencv.org/3.4.9/d2/dbd/tutorial_distance_transform.html (visited on 06/05/2020).

[52] *OpenCV: Image Segmentation with Watershed Algorithm*. URL: https://docs.opencv.org/master/d3/db4/tutorial_py_watershed.html (visited on 06/05/2020).

[53] *TensorFlow*. TensorFlow. Library Catalog: www.tensorflow.org. URL: https://www.tensorflow.org/ (visited on 06/28/2020).

[54] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *arXiv:1409.1556 [cs]* (Apr. 10, 2015). arXiv: 1409.1556. URL: http://arxiv.org/abs/1409.1556 (visited on 06/16/2020).

[55] *autonomio/talos*. original-date: 2018-05-04T20:36:41Z. July 3, 2020. URL: https://github.com/autonomio/talos (visited on 07/05/2020).

[56] *ImageNet*. URL: http://www.image-net.org/ (visited on 06/17/2020).

[57] Joseph Redmon, Santosh Divvala, Ross Girshick, et al. "You Only Look Once: Unified, Real-Time Object Detection". In: (June 8, 2015). URL: https://arxiv.org/abs/1506.02640v5 (visited on 07/08/2020).

[58] Ross Girshick. "Fast R-CNN". In: (Apr. 30, 2015). URL: https://arxiv.org/abs/1504.08083v2 (visited on 07/08/2020).

Appendix A

Python Code

Embedded in this file is both the Python code for extracting annotations, train the neural network, perform object detection, classification, and a Groovy script for importing the new annotations in QuPath.

A.0.1 `1_export_annotations.py`

This script is made to export images based on their placement read from the image's annotation file in the XML-format. All annotations will be exported in the center of a 256 by 256 pixel image from resolution level one in the palynological slide images sorted in folders by their annotation's label. In addition, a separate text file is exported containing the image file name, source image file name, annotation label, position and shape of all the exported images.

A.0.2 `2_train_vgg16.py`

This script contains the code to load the pretrained VGG16 network, download the pretrained weights and add the custom classifier. After training the accuracy and loss per epoch is plotted as well as a confusion matrix and classification report from the test set. The best model is automatically exported and saved as a hdf5-file.

A.0.3 `3_object_deteciton.py`

This script contains the object detection algorithm. For all palyslide images in a folder, the object detection algorithm is run and exports a text-file containing the position and shape for all objects.

A.0.4 `4_object_classification.py`

This script contains the code for using the model trained from `2_Learning_VGG16_train.py` and the exported object location file from `3_Locate_objects_in_image.py`. Classified objects are exported in a text file with the position, shape, size, predicted class and prediction confidence.

A.0.5 `5_quPath_import_annotation.groovy`

This script contains code to import annotations to QuPath (v.0.1.2). After opening a slide image go to Automate -> Show script editor or press Ctrl+`{`. Enter the path to the file exported from `4_Identify_objects.py` and press run to import annotations.

Note: QuPath will crash if too many objects are imported. Limit the number of imported objects to <1000.

A.0.6 `imfun.py`

This file contains helper functions used in the various scripts as explained below.

A.0.6.1 expand_region(pos, shape, new_shape)

For a region with `pos = tuple(x0, y0)` of the top left corner, and `shape = tuple(width, height)` this function returns the new position corresponding to a `new_shape = tuple(new_width, new_height)` without moving the region.

A.0.6.2 p2level(point, from_level, to_level)

As OpenSlide/Whole slide images contain multiple resolution level where each level is half the size of the previous one, this function transforms the position of a `point = tuple(x, y)` from level `int(from_level)` to level `int(to_level)`.

A.0.6.3 tuple_sum(tup1, tup2)

This function returns the sum of two tuples

A.0.6.4 tuple_subtract(tup1, tup2)

This function returns the difference between `tuple(tup1)` and `tuple(tup2)`

A.0.6.5 find_edge(image)

Used to detect the region of interest in images with a white and black border such as raw mirax images. Note, this is not usable on full resolution whole slide images, use on a resolution level from 3 and higher (lower resolution). Returns `x0, y0, width`, and `height`, i.e. the position and shape.

A.0.6.6 generate_grid(openslide_image)

Uses `find_edge` and `p2level` and calculates a grid based on the selected `block_size` (default `tuple(2048, 2048)` in level 0 resolution). Returns a list containing `[[index, column_number, row_number, (x0, y0), (width, height)],...]`.

A.0.6.7 segment_position(markers, block_position, image_level)

Returns a list of the position `tuple(x0, y0)` and shape `tuple(width, height)` of markers created by the object detection algorithm. By entering `block_position = tuple(block_x0, block_y0)` and `int(image_level)` the absolute position and shape is calculated in the highest resolution level (level 0).

Appendix B

Dinoflagellates from the palyslides

Filename	Species name	Annotations
2_4-C-11 10052.7 ftC = 3064 mC	Senoniasphera inornata	173
2_4-C-11 10070 ftc = 3069.3 mC	Fibrocysta Axialis	41
	Palaeoperidinium Pyrophorum	22
	Spongodinium delitiense	65
	Songodinium delitiense (operculum)	13
2_7-14 10658 ft 8 in C	Cribroperidinium "prominoseptatum"	19
	Danea californica	1
	Fibrocysta Axialis	5
	Palaeoperidinium pyrophorum	3
	Thalassiphora pelagica	1
16_3-2 1998.80 mC	Acanthaulax venusta	2
	Chytroeisphaeridia cerastes	1
	Chytroperidinium sp.	1
	Dingodinium tuberculosum	2
	Dingodinium tuberosum	2
	Endoscrinium galeritum reticulatum	1
	Gonyaulacysta jurassica	2
	Leptodinium mirabile	1
	Scriniodinium inritibile	1
	Sentusidinium pilosum	2
	Sirmiodinium grossii	1
	Systematophora areolata	2
	Tubotuberella apatela	2
DigitalSlide_C1M_3S_1	–	–
DigitalSlide_C1M_3S_1	Palaeoperidinium pyrophorum	7
DigitalSlide_C1M_5S_1	Fibrocysta Axialis	75
	Palaeoperidinium pyrophorum	85

Table B.1: A total of 530 dinoflagellates from 21 different species were annotated from the seven palyslides

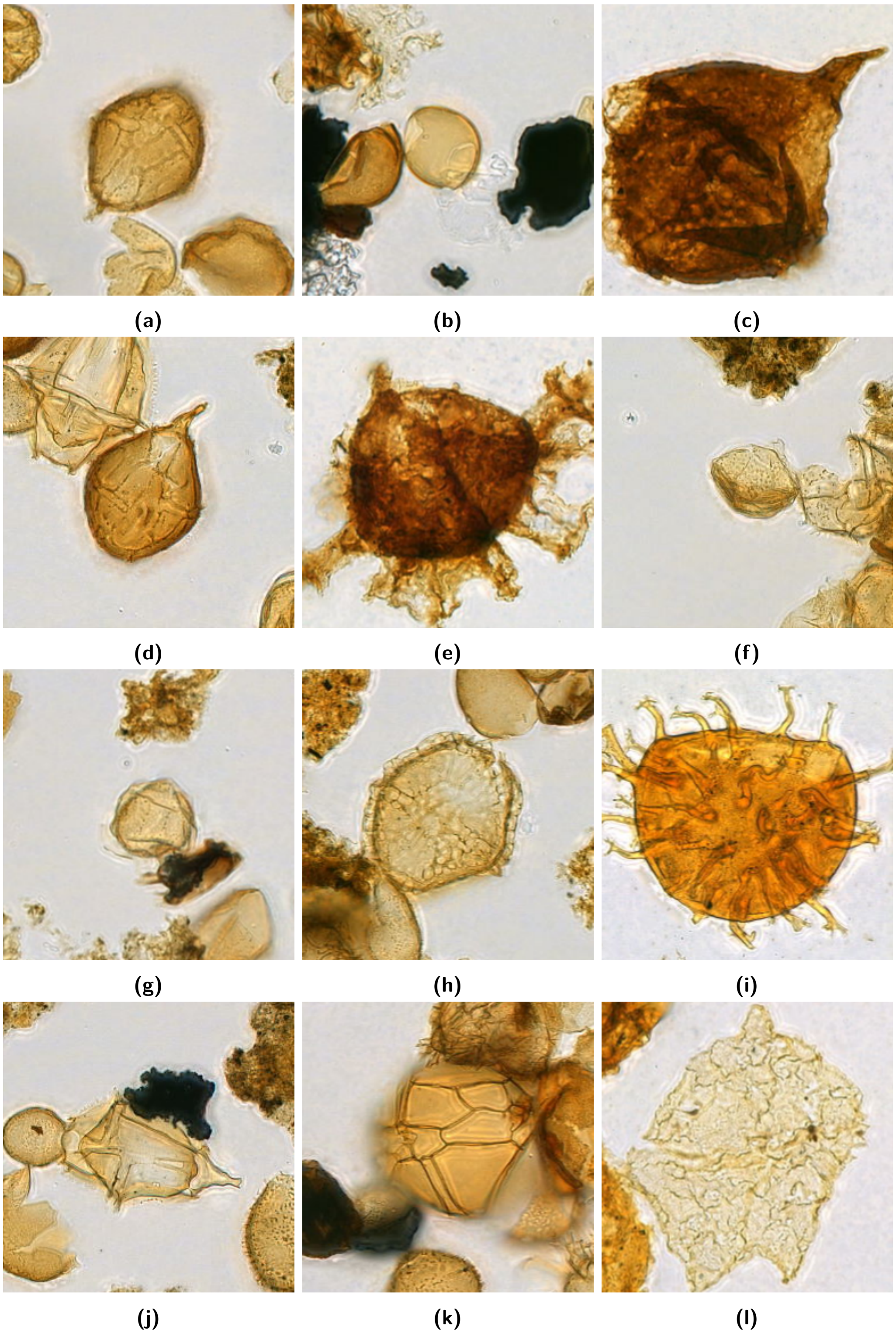
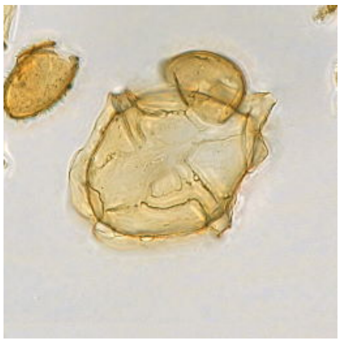
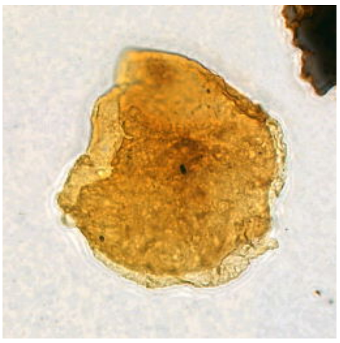
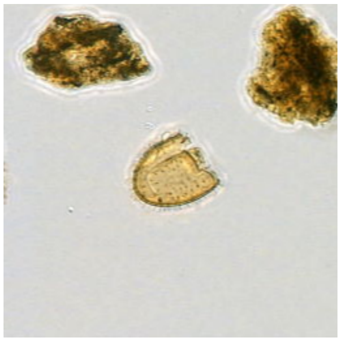
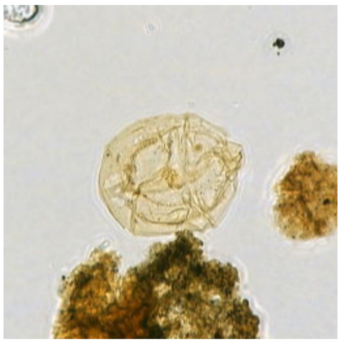


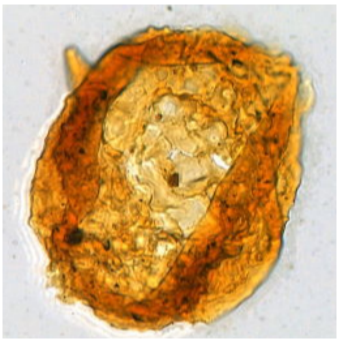
Figure B.1: Image showing the different species of dinoflagellates in the dataset (1/2) **(a)** Acanthaulax Venusta, **(b)** Chytroeisphaeridia Cerastes, **(c)** Cribroperidinium Prominoseptatum, **(d)** Cribroperidinium sp., **(e)** Danea Californica, **(f)** Dingodinium Tuberculosum, **(g)** Dingodinium Tuberosum, **(h)** Endoscrinium Galeritum Reticulatum, **(i)** Fibrocysta Axialis, **(j)** Gonyaulacysta Jurassica, **(k)** Leptodinium Mirabile, **(l)** Palaeoperidinium Pyrophorum

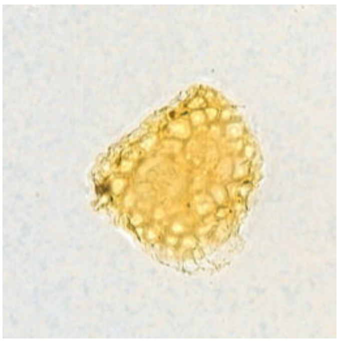

(a)

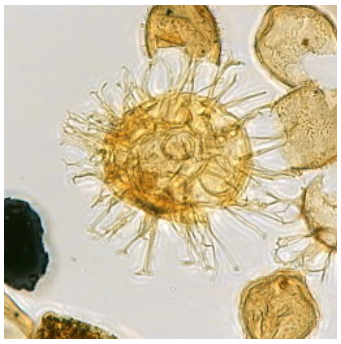

(b)

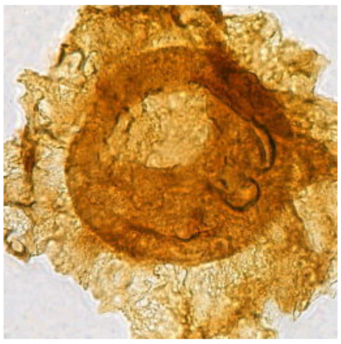

(c)


(d)


(e)


(f)


(g)


(h)

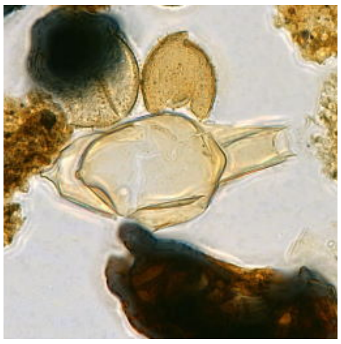

(i)

Figure B.2: Image showing the different species of dinoflagellates in the dataset (2/2) **(a)** Scriniodinium Inritibile, **(b)** Senoniasphaera Inornata, **(c)** Sentusidinium Pilosum, **(d)** Sirmiodinium Grossii, **(e)** Spongodinium Delitiense, **(f)** Spongodinium Delitiense (operculum),**(g)** Systematophora Areolata, **(h)** Thalassiphora Pelagica, **(i)** Tubotuberella Apatela

Appendix C

CNN Transfer learning test

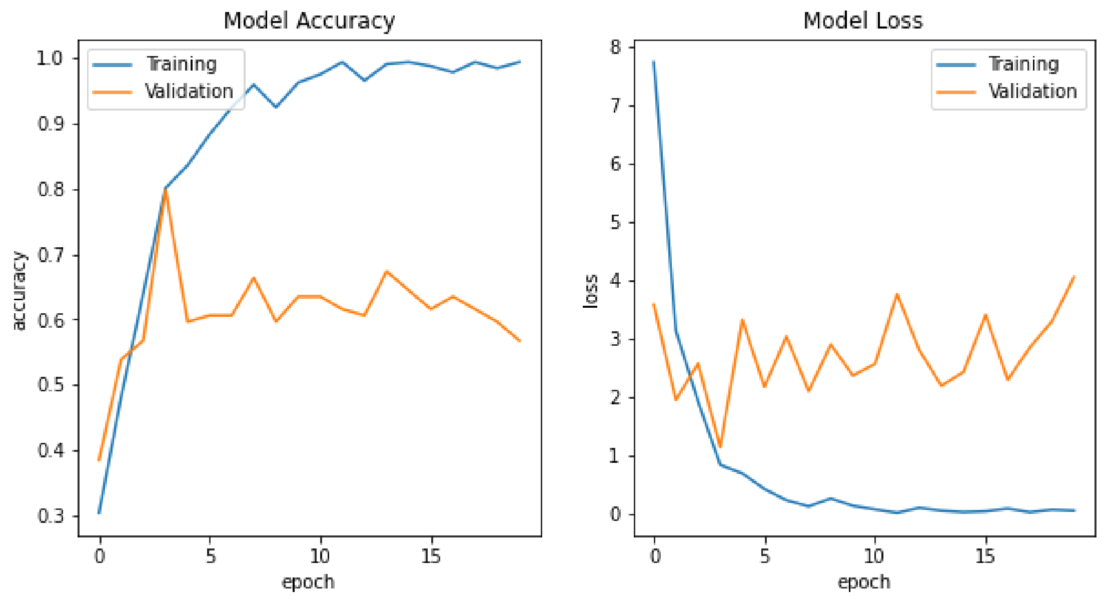


Figure C.1: InceptionV3 training accuracy and loss

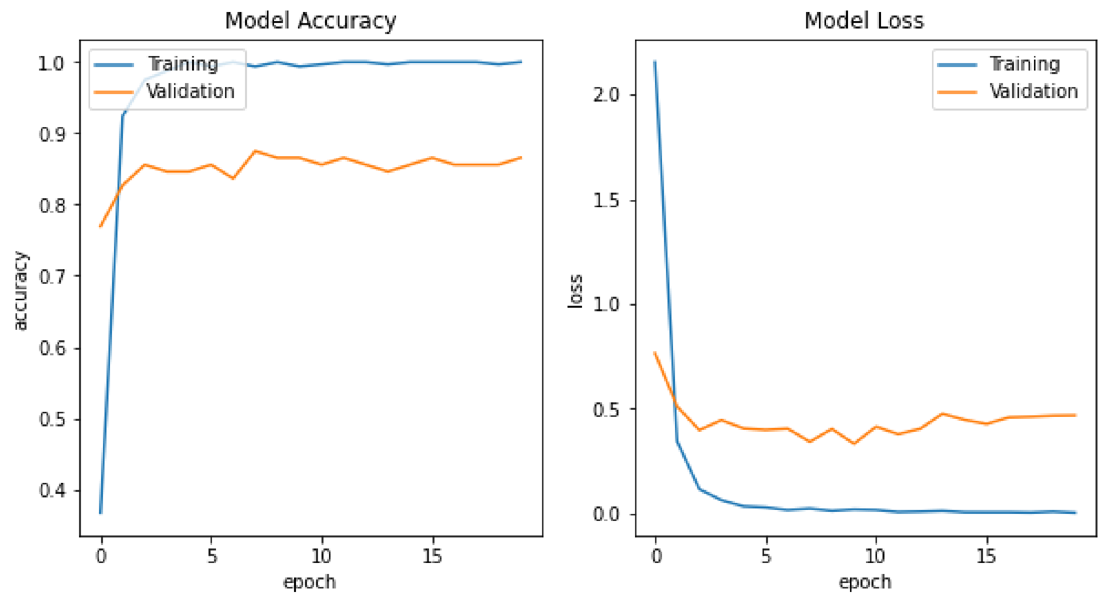


Figure C.2: MobileNetV2 training accuracy and loss

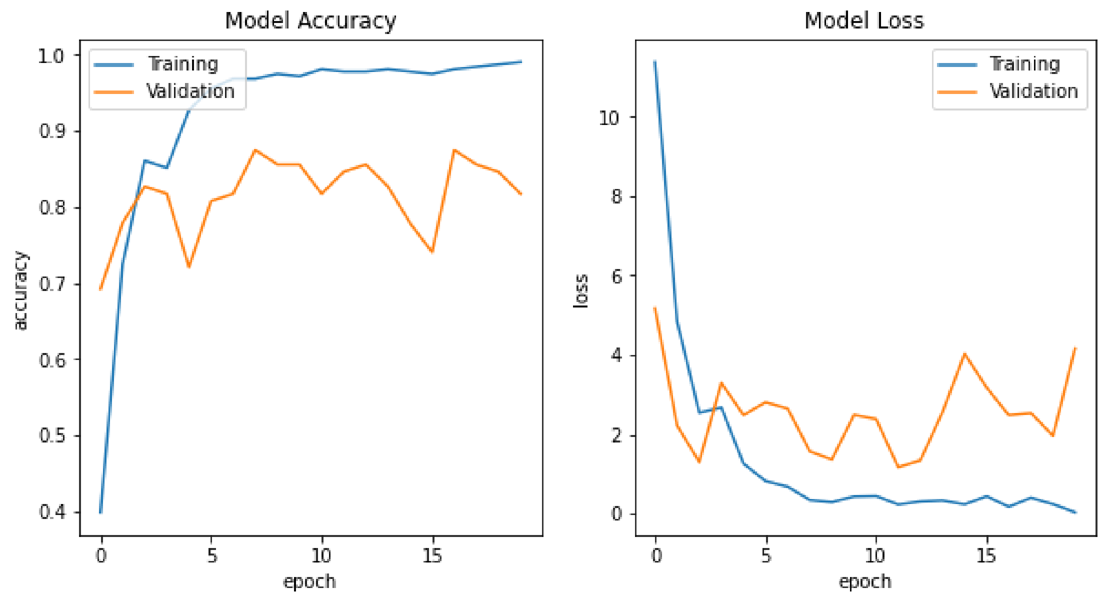


Figure C.3: NASNetLarge training accuracy and loss

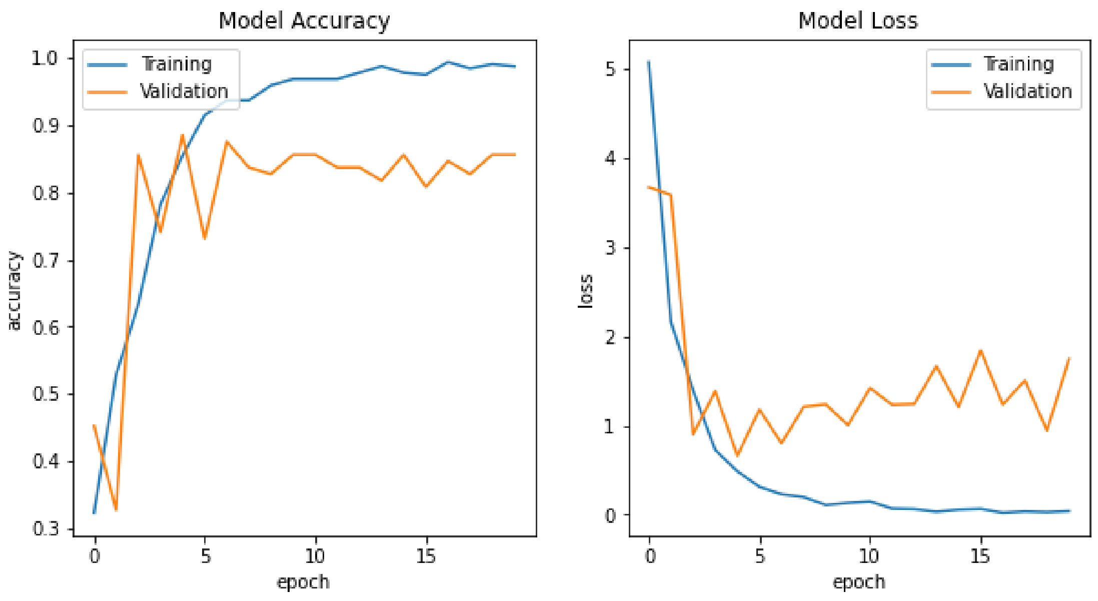


Figure C.4: InceptionResNetV2 training accuracy and loss

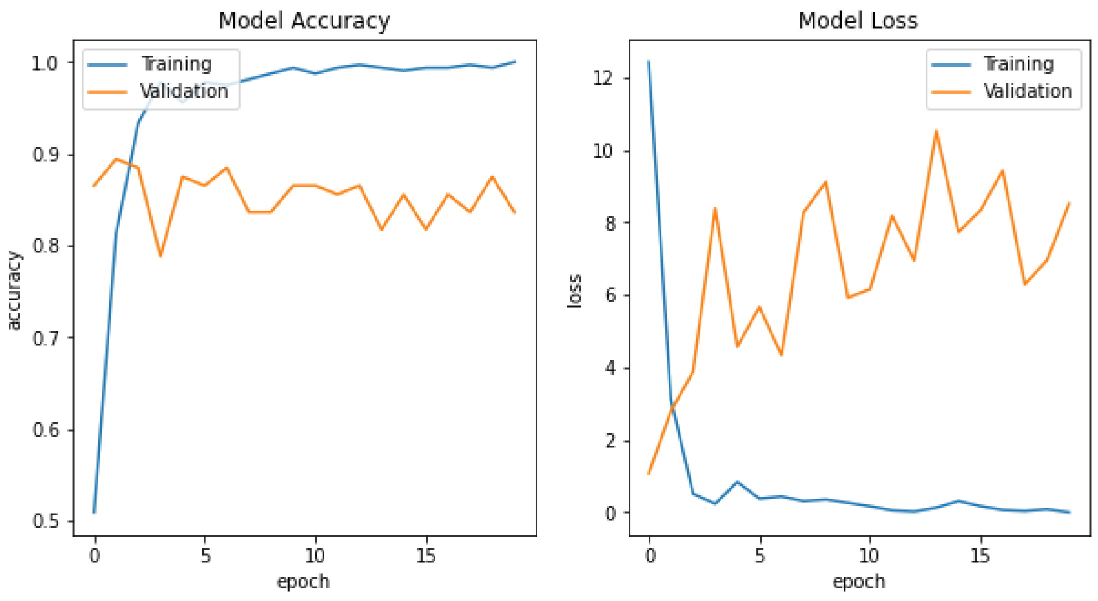


Figure C.5: ResNet152 training accuracy and loss

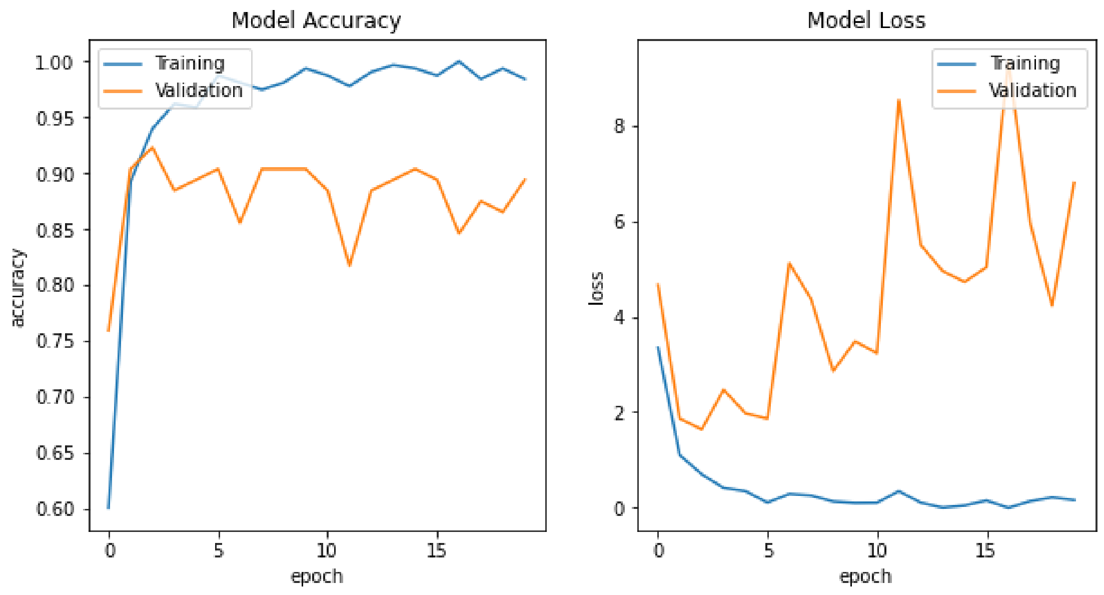


Figure C.6: ResNet152V2 training accuracy and loss

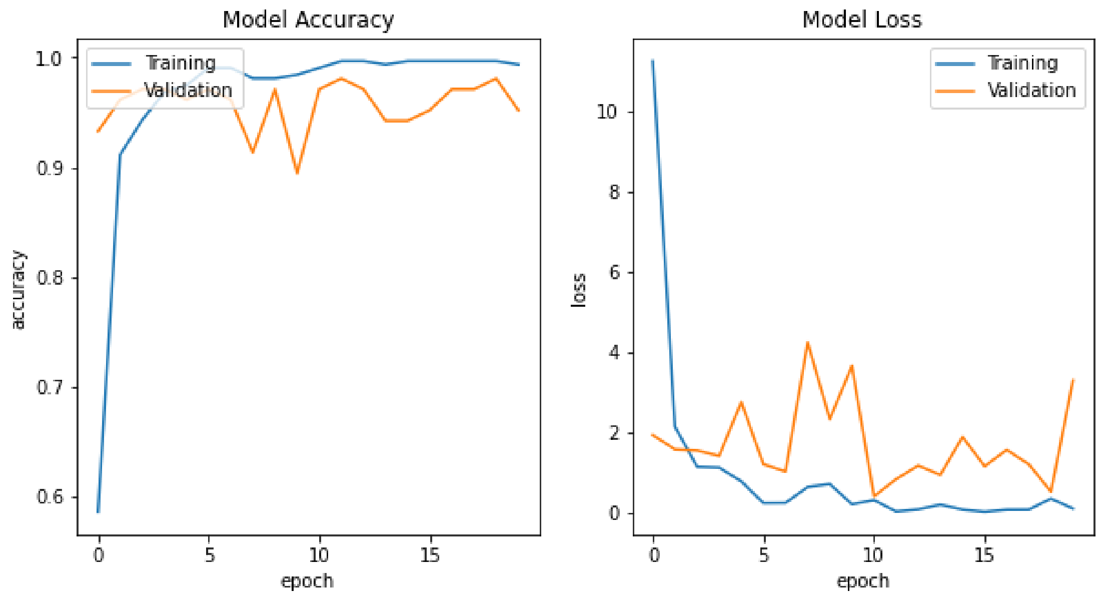


Figure C.7: VGG16 training accuracy and loss

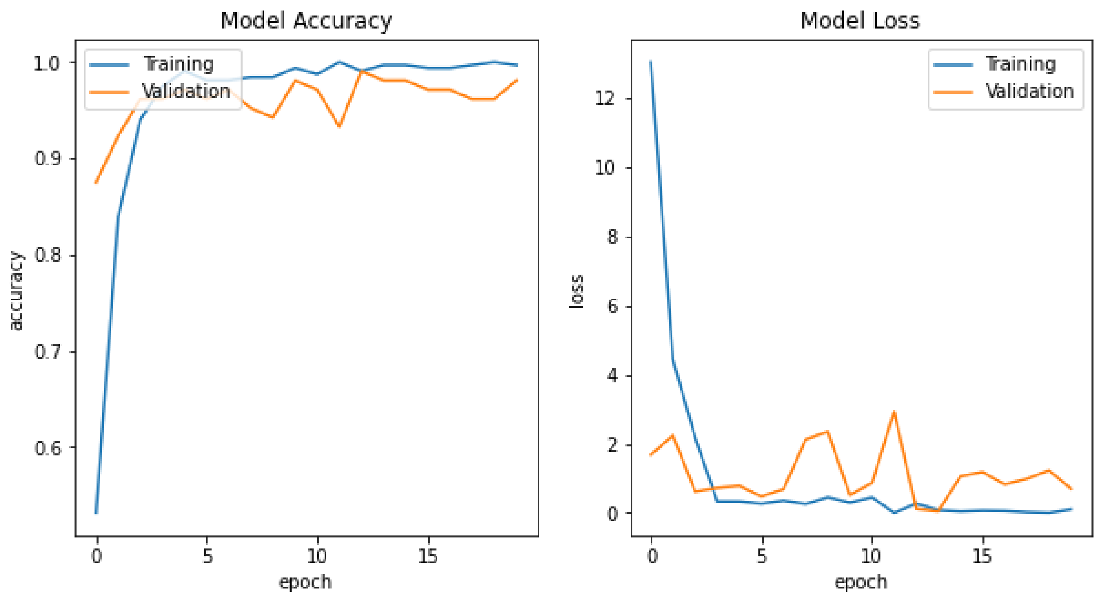


Figure C.8: VGG19 training accuracy and loss

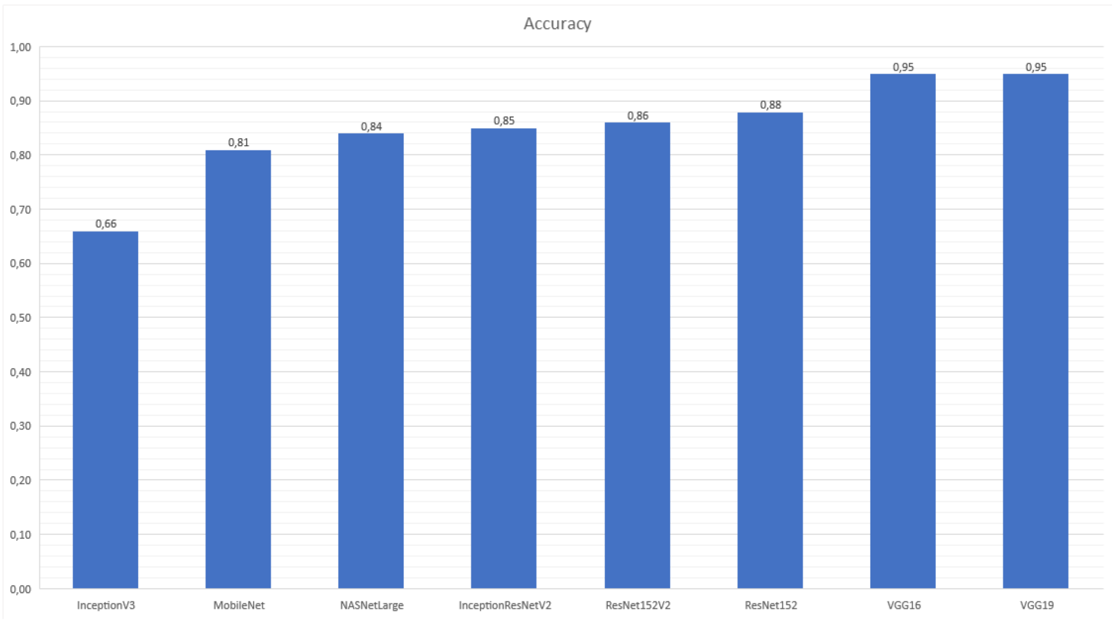
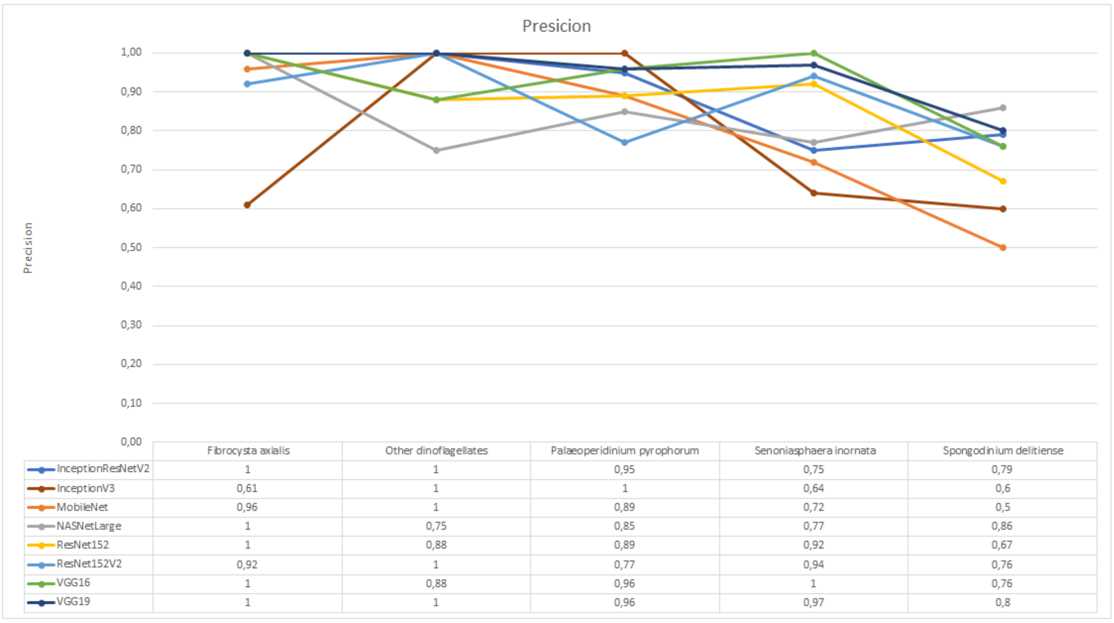
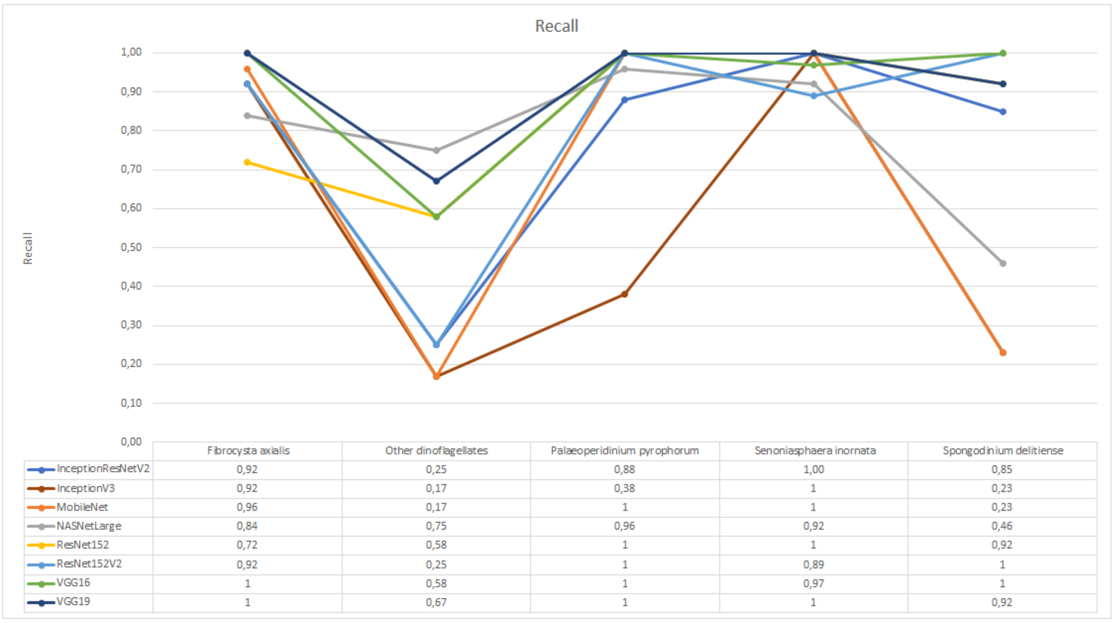


Figure C.9: Comparison of the accuracy of different models using default settings



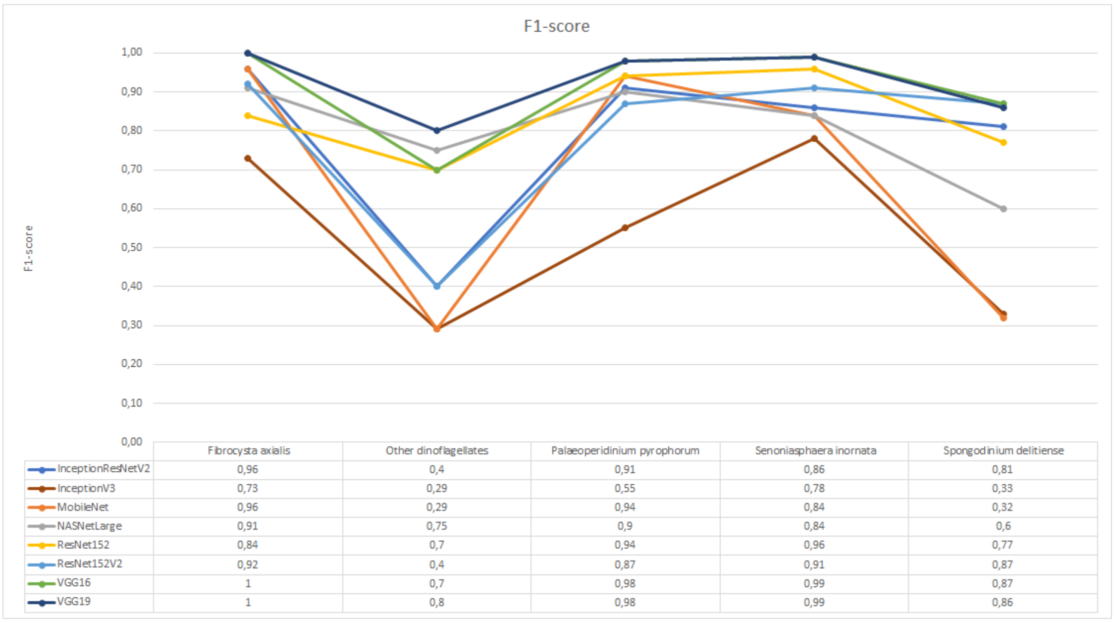
	Fibrocysta axialis	Other dinoflagellates	Palaeoperidinium pyrophorum	Senoniasphaera inornata	Spongodinium delitiense
InceptionResNetV2	1	1	0,95	0,75	0,79
InceptionV3	0,61	1	1	0,64	0,6
MobileNet	0,96	1	0,89	0,72	0,5
NASNetLarge	1	0,75	0,85	0,77	0,86
ResNet152	1	0,88	0,89	0,92	0,67
ResNet152V2	0,92	1	0,77	0,94	0,76
VGG16	1	0,88	0,96	1	0,76
VGG19	1	1	0,96	0,97	0,8

Figure C.10: Comparison of the precision of different models using default settings



	Fibrocysta axialis	Other dinoflagellates	Palaeoperidinium pyrophorum	Senoniasphaera inornata	Spongodinium delitiense
InceptionResNetV2	0,92	0,25	0,88	1,00	0,85
InceptionV3	0,92	0,17	0,38	1	0,23
MobileNet	0,96	0,17	1	1	0,23
NASNetLarge	0,84	0,75	0,96	0,92	0,46
ResNet152	0,72	0,58	1	1	0,92
ResNet152V2	0,92	0,25	1	0,89	1
VGG16	1	0,58	1	0,97	1
VGG19	1	0,67	1	1	0,92

Figure C.11: Comparison of the recall of different models using default settings



	Fibrocysta axialis	Other dinoflagellates	Palaeoperidinium pyrophorum	Senoniasphaera inornata	Spongodinium delitiense
InceptionResNetV2	0,96	0,4	0,91	0,86	0,81
InceptionV3	0,73	0,29	0,55	0,78	0,33
MobileNet	0,96	0,29	0,94	0,84	0,32
NASNetLarge	0,91	0,75	0,9	0,84	0,6
ResNet152	0,84	0,7	0,94	0,96	0,77
ResNet152V2	0,92	0,4	0,87	0,91	0,87
VGG16	1	0,7	0,98	0,99	0,87
VGG19	1	0,8	0,98	0,99	0,86

Figure C.12: Comparison of the F1-score of different models using default settings