UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

CNN for segmentation and classification of pollen grains

Dissertation presented by
Maxime JAHANSHAHI

for obtaining the Master's degree in
Biomedical Engineering

Supervisors
Christophe DE VLEESCHOUWER

Readers
Victor JOOS DE TER BEERST, Anne-Laure JACQUEMART

Academic year 2019-2020

Abstract

The automatisation of classifying pollen grains is an important issue that could save a lot of resources and time in a ton of domains. From helping the healthcare industry cope with allergy related problems, to being able to map and analyse whole ecosystems, the opportunities of applications are countless.

The problem of classifying pollen grain is addressed in two major steps, with the help of CNN's. At first we implement a segmentation CNN, enabling us to identify grain pollen pixels in an image, whatever the type of pollen. This tool is used to automatically segment pollen grains in images where only one type of pollen is present. The second step consists in using those segmented images to train a pixel-wise classification CNN, aiming at recognising the kind of pollen in images depicting mixtures of several pollen types. Both CNNs are based on the well-known U-Net architecture. This second instrument can be used to detect the percentage of pollen grains from the species present in the image. The method used in this thesis is particular, in the sense that, in final, our classification CNN is trained with images of pure pollen grains, without requiring the manual segmentation of images including multiple species mixed together.

The implemented U-net model is able to perfectly discern any pollen grain pixels from background pixels. Overall, the classification algorithm has a really good classification success rate, and is almost always able to detect the majority of the present species in an image.

Key words. CNN, pollen segmentation, pollen classification, U-net, pollen grains, data augmentation, convolutional neural network.

To my friends and family.

Acknowledgments

I sincerely want to thank the PhD students and my thesis supervisor with whom I had the chance to work with.

Special thanks to Jean Léger, who helped me by providing a solid base for the U-net architecture, and helped me detect some very annoying bugs in the code more than once.

The same goes for Victor Joos de ter Beerst, who provided me with a insight for comprehending Cytomine and how to import data from it. I am also very grateful for my supervisor, Christophe de Vleeschouwer, who helped me structure and considerably improve this document.

Lastly, many thanks to my friends and family who supported me during these, sometimes very difficult, months.

Tervuren, August 2020

Contents

Abbreviations

CNN	Convolutional Neural Network
MCC	Matthews Correlation Coefficient
MLE	Maximum Likelihood Estimation
MSE	Mean Squared Error
NN	(Artificial) Neural Network
ReLU	Rectified Linear Unit
SVM	Support-Vector Machine

Introduction

Palynology is a very peculiar field of study within the realm of ecology. Dealing with pollen and spores of plant species, it may be one of science's most underrecognised branches. It may look like a very niche and narrow biological field, but is has uses in a myriad of other environments. It is used paleobotany and melissopalynology, where palynology can reveal lots of information about past and current local ecosystems. Melissopalynology is a field that studies honey and pollen to gain biological and geographical knowledge about the local flora. Another field of interest tied to palynology is hydrology, where pollen, spores or specific sediments can help determine above or underground water systems. A last example is the use of pollen samples in forensic science to help criminal investigations, by bringing a unique kind of evidence to the table.

The automatization of the process of classifying pollen grains is an important issue that would save a ton of time and resources. Going from being beneficial in the healthcare industry for allergy-related systems to mapping entire ecosystems, pollen grain classification can be useful in hundreds of domains.

Only a few decades ago, biologists still used light microscopes to identify and quantify pollen grains. After huge technological advances in computer science, imaging techniques and imaging hardware, the complete automation of classifying pollen with almost no human interaction is becoming achievable. Most classification models are composed of a crucial component to be able to differentiate one type of constituent from another: *discriminant features*. Discriminant features represent the specific attributes and characteristics of the object of interest. These characteristics will be used by the algorithm to create a mathematical model which will be able to discern one object of interest from the other. A big problem with palynology is that different types of pollen are *really* similar[1]. This makes the algorithms very sensitive to parameter variation. The design and selection of these kind of features are a time-consuming task. Recently, deep learning algorithms have emerged. These, more complex kind of algorithms, are able to learn these features by themselves, without humans intervening, thus saving them from a tedious and time-consuming task.

A lot of authors have tried to solve the problem of pollen classification, based on the manual extraction of features. A notorious paper [2] from 1996 published a brief summary of the state of the art until then, and more importantly the demands and needs of palynology to elevate the field to a higher level, thus making it a more powerful and useful tool. These needs can be summarised in distinct categories. First of all, the need for more data, more pollen grains per sample and more data-sets. Secondly, the need for faster analysis. To put an order of magnitude on it, in 1995 it took a trained researcher between 2-10 hrs to analyse a sample. Finally, another improvement field would be resolution. Fine-resolution palynology could for example discover climate shifts.

As of today, manual extraction of discriminant features remains a used technique. We can divide this kind of analysis into several subcategories. [3]

- Morphological methods: Here visual features like colour, shape or symmetry are measured. Treloar & coll.[4] tried to work on a solution based on visual features like grain perimeter, roundness and area, then applying a Fisher discriminant. They classified 12 different pollen types from a Polynesian island, with a classification rate up to 95%. With SVM's as classifier,[5] used morphological details of the pollen grains contour as discriminative features. They achieved 93.8% $\pm$ 1.43 of success while classifying 47 tropical honey plants.

- Texture-based methods: These approaches make use of the characteristics of the pollen grain surface as discriminative feature. As an example, [6] presents a system that detects and classifies pollen grains based on a combination of shape and texture. An Urticaceae data-set, which are very similar family, are used for performance evaluation. They achieved 89% success. Alternatively, based on grey-level co-occurrence matrices and neighbourhood grey level dependence statistics, [7] were able to classify 5 different pollen-types with more ore less 75% classification success.

An alternative to the manual extraction of the features would be to let an algorithm learn these by itself by training it to fit a data-set. This can be done through deep learning. Deep learning is a subset of machine learning, inspired by the human brain. It uses a hierarchical structure with a series of layers to create neural networks. The more layers the network has, the more complex the model can get. While the lower layers will identify simple features such as curves or straight lines, the higher levels will be able to extract more complex characteristics. The hierarchical nature of the structure allows it to analyse data non-linearly, in contrary to traditional algorithms. A downside to these neural networks is that they need a consequent amount of training data and computing power.

Whether it is via manual or automatic extraction of features, when comparing results of different studies, we have to bear in mind that the degree of success will be hugely influenced by plenty of parameters. Not only the similarity between the pollen species but also the quality of the images or the amount of input images will greatly increase the accuracy of the model.

Existing state-of-the-art articles and research on pollen grain classification conventionally rely on data-sets constituted of scans composed of multiple species of pollen. In this work, we present a solution to the pixel-wise identification and classification of pollen grains by automatic extraction of the discriminant features, using a convolutional neural network. The specificity of this thesis compared to other existing work, and thus its contribution to the state of the art, lies in the data-set. Indeed, typically the problem is addressed by training a convolutional network with images containing a number of different species of pollen grains. This method has in reality many drawbacks, as it involves the quite time-consuming task of annotating each scan for the training data. Not only is it tedious, but it also demands a certain level of expertise, as the person annotating must also know how to differentiate the grains by sight.

On the contrary, in this thesis we are dealing with a data-set composed of scans with pure pollen. In other words, every image in our training set consists of only one species of pollen. This strategy has quite a few advantages. In the first place, this technique need a lot less manual annotation, so it is less time-consuming compared to classic approaches. Secondly, the fact we are able to train the data with images of pure pollen, makes the algorithm modular. This means that at any moment, we can just add a few annotated scans of a chosen species, launch the algorithm, and the model will be able to recognise the new pollen species. Whereas conversely, we would have to first obtain and then annotate images of mixed pollen species, which would take a lot more time and effort.

We will use these advantages cited above to attempt to create a powerful and efficient

classification tool. The strategy utilised to address this classification problem is the following: Having manually annotated a few dozen images of pure pollen grains, we will first construct a pixel-wise segmentation CNN to facilitate the annotation of future supplementary data. The second step will be to implement the pixel-wise, multi-label classification algorithm. We will do this by basing our architecture on the well-known and efficient U-net network. By training this network with our pure pollen images, we will then get a model that will be able to predict mixed images. To get a clearer picture of the strategy described precedently, a bloc-diagram in figure 1 illustrates how the problem is tackled.

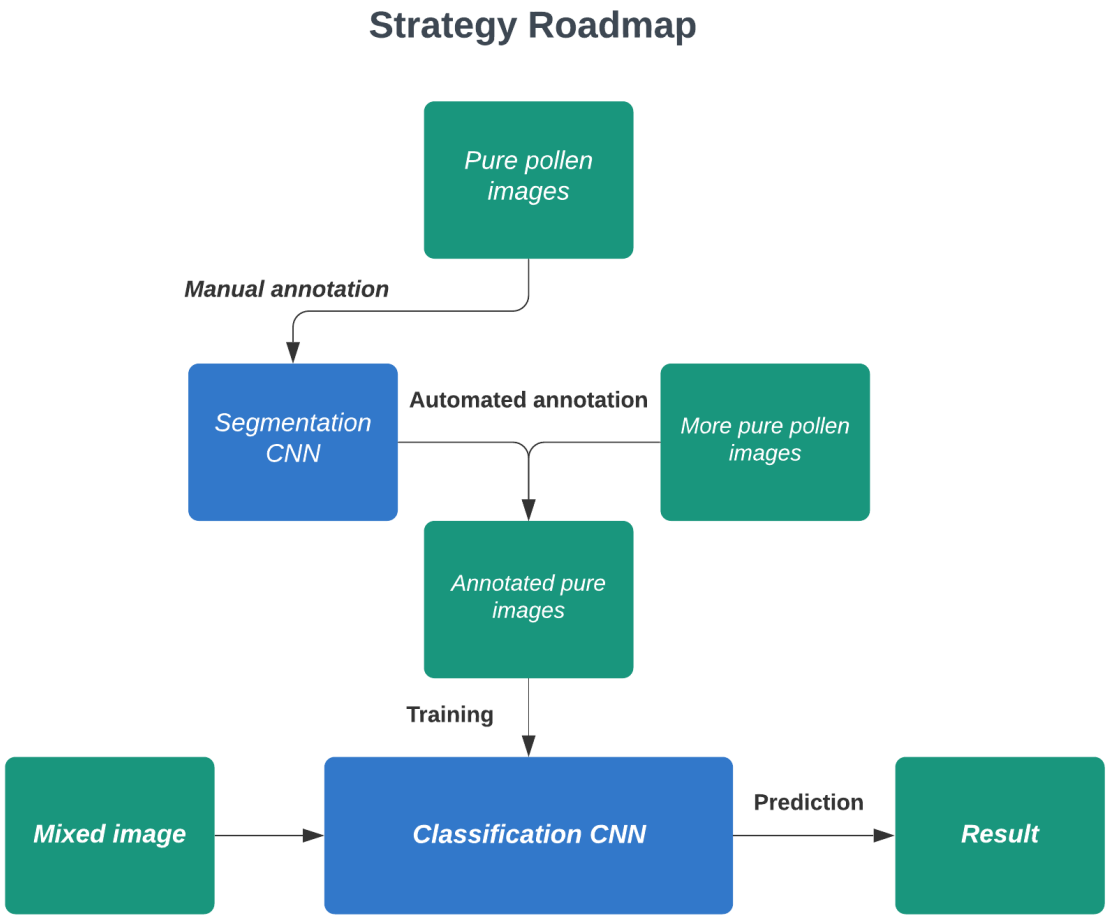


Figure 1: An illustration of the strategy used to address the problem presented in this thesis

This report is structured in the following way: In chapter 1, we start by going over the basics of neural networks. We then more specifically describe the functioning of convolutional networks, and some important concepts related to the subject. Then, in chapter 2, the characteristics of the data-set are detailed. Further, in chapter 3, we go over the specific architecture used for our neural network, explaining its main components, advantages and drawbacks. In this same chapter, we will detail the design of our segmentation CNN, followed by the workings of the pixel-wise classification CNN. Lastly, in the final chapter (Ch4), we go over the results of our work. First by explaining the way we will be evaluating our method, then reviewing and discussing our findings in the last section.

Following the discipline of reproducible research, the source code and data files required to reproduce the experimental results of this master's thesis can be downloaded from `https://bitbucket.org/maxjahan/polclas/src/master/`.

Chapter 1

Background

In this first chapter Section 1.1 introduce Neural networks, while section 1.2 will concisely describe the basic concepts underlying their training. In section 1.3, we will delve deeper into the concept of a specific type of neural network, specifically suited for image processing: the convolutional neural network.

1.1 Neural networks

The task of classifying pollen grains could be learned by any individual trained with basic biologic knowledge. This skill of observing an image and being able to categorise its content into semantically meaningful objects, is an aptitude a human learns since (s)he is a child. The first time a child sees a dog, he can't differentiate it from a cat. But with enough data, after having seen thousands of each species, any capable human can differentiate them from a hundred meters away. The human brain is constantly creating new connections between neurons, associating context elements, learning patterns and thus expanding our perception of the environment.

It is exactly this kind of learning that is reproduced in neural networks. These algorithms, inspired from the human brain, are designed to recognise patterns. They are able to learn relationships between specific characteristics, make generalisations, inferences, and even reveal relations that would be hidden to the human eye. This is why neural networks are such a powerful tool for image classification.

The human brain consists of hundreds of millions of neurons, connected via synapses. When a sufficient amount of synapses connected to a certain neuron are activated, this neuron will fire and send an outgoing signal via output synapses. The way these millions of neurons and synapses are connected shape the way we think and act.

In neural networks, the neurons are replaced by artificial neurons, and they are connected with each-other by weighted synapses. The activation of the artificial neuron will depend on:

- The weights used to combine the neuron inputs. These weights will change as the neural network will learn.
- An activation function, this can be a simple threshold for example.

The neurons are organised in layers, and each neuron from an arbitrary layer can be influenced

by the neurons of the previous layer by taking input from it. In figure 1.1 we can see how one neuron is influenced by neurons of a previous layer. The point of such an architecture, composed of a series of layers consisting of multiple neurons, is to help us cluster and classify data. The neural network will approximate a function $f(x) = y$ for a given input x, and output y. With the help of training data, the function will learn to fit the training set. The training of the network is done by minimising a loss function(see section 1.2.3) by adjusting the weights of the neurons. After being trained, the network will approximate a function that will be able to predict an output y' for some given data x'.

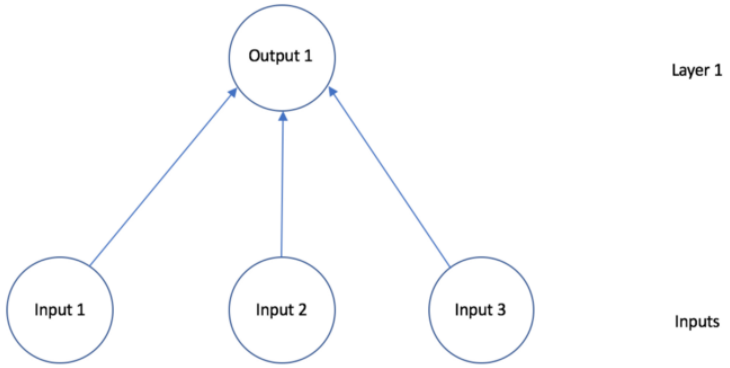


Figure 1.1: Illustration of the relations between artificial neurons in a neural network. We can see that the 3 lowest neurons, constituting the input layer, are giving an input to the upper neuron .

An example of a small neural network constituted of only 2 layers is depicted in figure 1.1. As the number of layers grow, the complexity of the model also grows. In fact, more than 3 layers is enough to qualify as a "deep" learning algorithm. In deep learning, the number of layers can even go up to a thousand. When talking about layers in a neural network, and modelling its structure, we can define a few different types of layers. The following different kinds of layers are depicted in figure 1.2:

- **The input layer** is the first layer, providing the input data to the network.
- **The output layer** is the last layer, giving us the predicted result.
- **The hidden layers** are all the layers between the first and the last one. Every layer can apply a function to the previous layer. As the layers increase, so does the level of abstraction. For example, in facial recognition , the first hidden layers could be used to identify a nose or an eye, while the last hidden layers will be able to classify a whole face. This hierarchy in features is what enables the neural network to analyse very large and high-dimensional data sets. The job of the hidden layers is to gradually transform the input data into a feature the output layer can use.

Fully Connected layer In neural networks, when all the inputs from one layer are connected to every node of the next layer, we call this fully connected layers. Following this definition, the network depicted in figure 1.2 is fully connected.

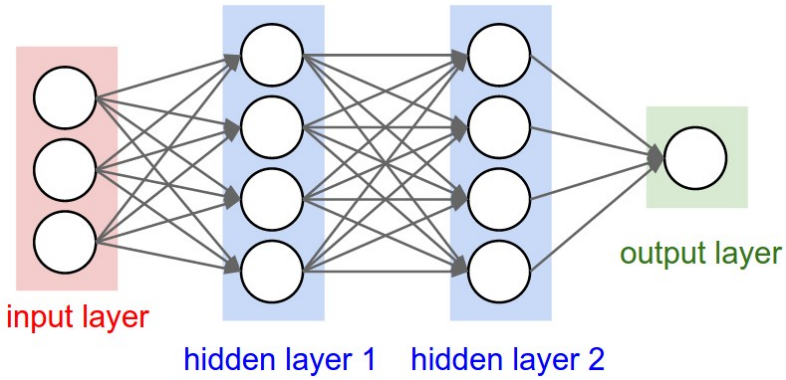


Figure 1.2: Illustration of the different types of layers in a NN. This network is fully connected. The red neurons represent the input layer, the blue neurons form the hidden layers, while the green neuron constitutes the output layer.[8]

1.2 Key-concepts of a neural network

To summarise, neural networks are composed of several layers of nodes, that will influence each other based on the weights given to each of these nodes. To train a neural network, an optimization function will compute the error between the network's prediction and the ground-truth. Based on that error, it will recompute and adjust the weights using an iterative gradient-descent algorithm. With good parameters, after a certain amount of iterations(or epochs, in machine learning slang), the error is minimised and, hopefully, the model becomes able to generalise unseen data. An example of a potential real-life application of a NN to predict lung cancer is depicted in figure 1.3.

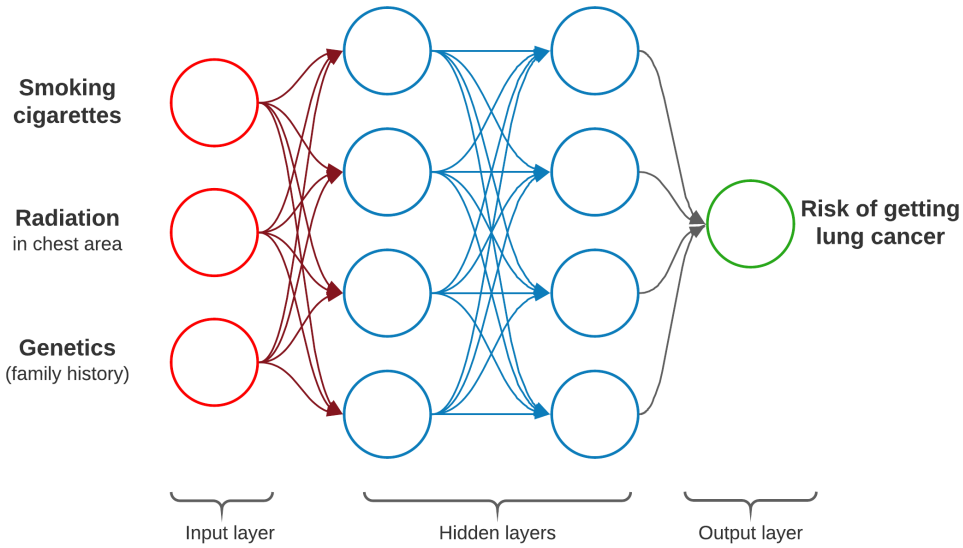


Figure 1.3: Potential application of a neural network. The goal is to predict the probability of a random person getting lung cancer, given his/her smoking habits, the endured chest radiation and his/her genetic predispositions. This network example could be extended by adding a lot more risk factors, making it more precise in it's prediction

1.2.1 Gradient-descent

The term used for the most popular and commonly used optimization function is "gradient descent". This is the function that will adjust the weights on the nodes to optimize the result based on the error. Let us now briefly explain the way gradient descent works. We can comprehend this more easily by comparing and picturing it as a mathematician hiking in a mountainous region.

Initially, all weight coefficients are randomly assigned. We can imagine this by parachuting the little man on a random location on the region. Then, based on a certain loss/cost function, the algorithm will calculate the parameter variation that results in the highest decrease in error. This calculation will repeat itself until we arrive at a local minimum. Let's now get back to our little guy. So, the man has a very short sight and can thus only see the next step he could take around him. He will always choose the steepest path (downhill) he can take. After enough steps taken, he will slowly get to the lowest point in the region. Once he realizes that taking steps doesn't get him much lower, and he is stuck in the same little spot for a while, he stops walking. This would be the convergence of the algorithm.

Taking figure 1.4, we could also illustrate this by imagining a ball that is placed on the top of the mountain. It will slowly roll down the mountain and eventually end up in one of the lower, blue areas, a local minimum. This also demonstrates that, depending on the initial placement of the ball, it could end up in a different place. This is also one of the biggest drawbacks of this algorithm : the possibility of getting stuck on a local minimum.

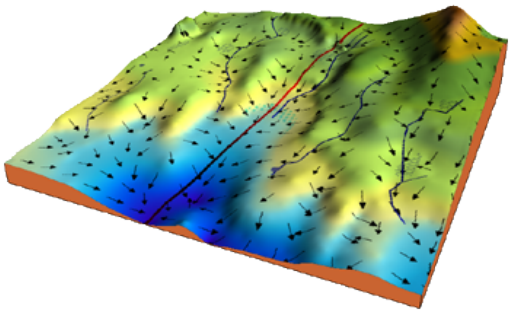


Figure 1.4: Illustration of the gradient descent method [9]

1.2.2 Learning rate

The learning rate is another important parameter that will impact the efficiency of a deep learning algorithm. It influences the rate at which the weights are updated per iteration. So, a high learning rate will adapt the weights quicker, thus requiring less iterations of the algorithm, but could overshoot the solution and converge to a sub-optimal solution. At the other end, a learning rate that is too small could extend the number of iterations to an unreasonable amount. Moreover, it has been shown experimentally that, for the same amount of epochs, small learning rates lead to poorer generalisation capabilities [10]. The better choice is almost always to pick an adaptive learning rate. This means the learning rate will vary over the training process. This can be implemented easily by halving the value of the learning rate every X amount of epochs. An even more complex approach could be taken, like the one Tom Schaul et al.(2013) presented in their method with a learning rate able to decrease or increase during the learning process, making it suitable for non-stationary problems [11].

Going back to our analogy of a mathematician walking in the mountains, we can compare the learning rate to the size of his steps. A high learning rate would mean he takes huge steps, descending the mountains very quickly, but adding the risk of overshooting the lowest point of the valley and thus missing the optimal weight choice for the network. A very tiny step-size would mean the mathematician would take days, or even months to get to a minimum. This would result in the man getting out of supplies and having to end his journey before being able to finish it.

1.2.3 Loss functions

When training a neural network using the gradient descent optimization algorithm, it has to compute the error for each iteration. This evidently calls the need for an error function, also called loss function. The choice of this loss function will influence the efficiency of the algorithm. There are many possibilities here and choosing the right loss function can be challenging.[12]

A very popular way of choosing this function is using the concept of Maximum Likelihood Estimation, or MLE. MLE is a probabilistic framework that can be used for finding the best statistical estimation for model parameters, based on training data. [13] The loss function under such a framework, will compare how close the predicted distribution of parameters are to the distribution of the parameters in the training set. One important property of MLE functions is consistency. This means that, with increasing data, the parameters will converge to their true data. In deep-learning, a very popular loss function for binary or multi-class classification problems is Cross-Entropy. This is a measure coming from the field of information theory. It is used to measure the difference between two probabilistic distributions. The most popular loss function, for regression problems, is the Mean squared error or MSE function. As the name indicates, it computes the average of the squared differences between the predicted and true values. The results will thus always be positive. Due to the square, small differences will be less notable, while big mistakes in the model's prediction will be punished harsher and results in big errors.

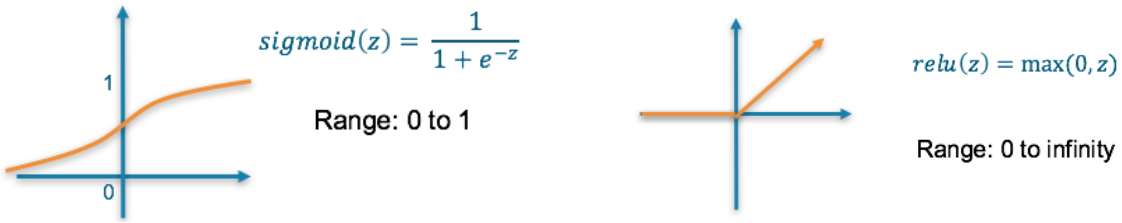
1.2.4 Activation functions

The activation function also has a major role to play in the efficiency of a neural network. Basically, the activation function is an mathematical equation, determining the output of a neuron or node. It determines if the neuron "fires" or not. It acts like a mathematical gate between the input, and the next layer. Almost all modern neural networks now use non-linear activation function, enabling it to learn complex data, and make precise predictions. A few of the most popular functions are depicted in figure 1.5. The sigmoid function (fig. 1.5a) is often used, as it is quite simple to compute, and output values are bound and thus don't get too big. A drawback of the sigmoid function and other classic activation functions is that it suffers from the vanishing gradient problem. In short, when using gradient descent with such activation functions in deep neural networks, the error will decrease abruptly with every layer because of the derivative of the chosen function. [14][15] [16]

The Rectified Linear Unit or ReLU is an alternative that (figure 1.5b) will output the input directly if positive, and will otherwise output zero. Because of it ability to cope with the vanishing gradient problem, and its great performances, it is the default activation function for many NN.

A last and important activation function is the *Softmax*. Like the sigmoid, the softmax is a function based on logistic regression. The softmax turns the scores into a normalised, probability

distribution. This means that sum of the scores for all classes is equal to 1. Thus, this kind of activation function is useful when the output classes are mutually exclusive(there can be only one right answer) [17].



(a) Sigmoid Activation function

(b) ReLU activation function

Figure 1.5: Several popular activation functions [15]

1.3 Convolutional Neural networks

In the field of visual imagery, we often use a special kind of neural network, specifically adapted to the analysis and processing of image data-sets. They are called Convolutional Neural Networks (CNNs). The same way neural networks are based on the biological structure of the human brain, CNNs are modeled after the visual cortex. The first layers of neurons each have a spatial expertise, meaning that every neuron responds to stimuli in a specific area, the receptive field of a neuron. This enables the network to detect patterns, like edges or lines. With a growing number of layers, these patterns increase in complexity, and can allow a CNN to distinguish a tree from a car, or even a man from a woman. But what exactly makes a Convolutional neural network different from a classic, multi-layer network? [18][19][20][21]

1.3.1 Convolutional layer

Well, the answer is in the name, they have convolutional layers. In these layers, we want to extract information from the input image by using different filters. Just like any (hidden) layer, these kinds of layers take an input, and then transform that input into an output. The transformation in a convolutional layer is done by a convolution operation. Mathematically speaking, the operations used is actually a cross-correlation, or a sliding dot-product. But for some reason the deep-learning community likes calling it a convolution. A convolution operation uses two sets of information and merges it into one. The first set of information is the input image itself, while the second is a feature detector, also called a filter (fig.1.6).

The convolution operation is performed by *sliding* the filter over the input, and each time executing an element-wise matrix multiplication for all elements of the matrix, and summing those up. In figure 1.7, we can see the first step of a convolution operation for the given image. Next, we will shift the filter one spot to the right, and compute the output for that product. We continue this until the filter has sled over the whole image, and the outputted feature map is fully computed. There can and will often be multiple filters, meaning this whole process will have to be repeated for each filter.

The actual values of the filters are learned by the neural network itself, although we will have to manually select the number of filters used and the filter size. The values for the filters will determine what kind of features are selected, the number of filters will influence the number of features selected, and the filter size will determine the size of the receptive field.

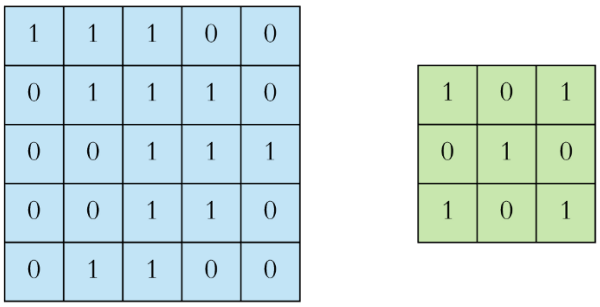


Figure 1.6: Illustration of the input image on the left, and the feature map/filter on the right.[22]

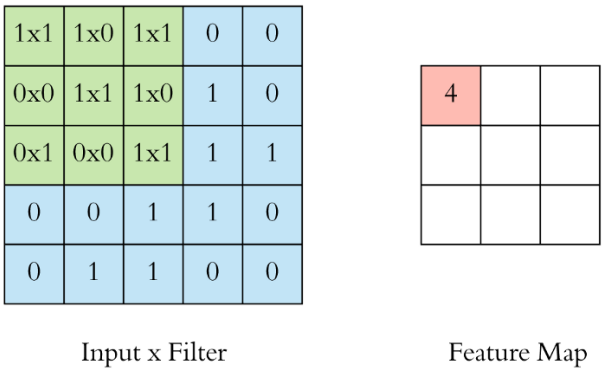


Figure 1.7: Illustration of a convolution operation for the first pixel. On the left, we see the superposition of the filter with the input. The output, on the right, is computed by making a pixel-wise multiplication and summing all the resulting values up. [22]

1.3.2 Pooling

To reduce overfitting and to shorten computing and training time, we often use pooling to reduce the size of the matrix. The pooling operation down-samples both in width and height, keeping the depth dimension intact. There are different types of pooling possible, but the most popular one is maxpooling. Like the convolution operation, it slides the window over the input. For each pooling window, it takes the maximum value and outputs it. A visualisation of a maxpooling operation can be seen in figure 1.8

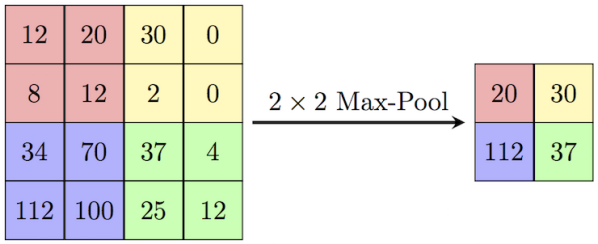


Figure 1.8: In this illustration of a maxpool operation with window 2x2, we see that the operation consists in taking the maximum value for each window. Looking at the red square in the output, we see that the resulting value 20 is the highest value from all red squares in the input. We not that the dimensions of the matrix are halved.[22]

1.3.3 CNN architecture

Now that we have described all the basic building blocks of a convolutional network, let us explain the typical architecture of aforementioned, with figure 1.9. Starting with the input image, we perform a convolution with a certain number of filters. Secondly, we max-pool the output of the convolution, reducing the number of parameters to optimize. These first two steps can be repeated as much as desired, to increase the feature detection and its receptive field. Lastly, the result is flattened into a fully connected layer, and finally output into a class using an activation function.

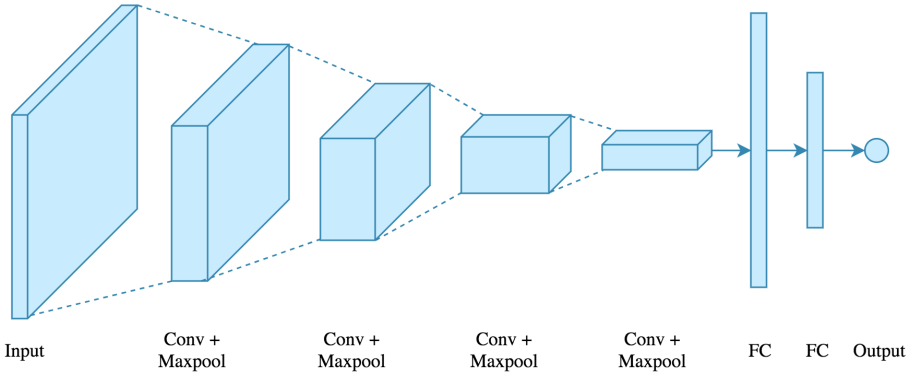


Figure 1.9: Typical architecture of a CNN used for image classification[23]

1.3.4 Conclusion

To summarise, artificial neural networks are a set of algorithms inspired by the human brain, that are constructed with as goal to cluster large data-sets or to classify data. They are made of a series of layers composed of neurons. The number of neurons can go from a few hundred to even millions of nodes that are very densely interconnected. Most of them of organised in a "feed-forward" way, meaning that each neuron gets a weighted input, only from precedent layers. To be able to classify data, a neural network has to be trained. This is done by minimising a loss function by adjusting the neurons weights as to fit the data as well as possible.

CNN's are a variant of NN that are very popularly used in image processing. By means of convolution operations, pooling layers and fully connected layers, they are designed to automatically learn spatial hierarchies of features. Which is why they are such a good fit for image processing, and for our pollen classification problem.

Chapter 2

Data-set characteristics

In this chapter, we paint a clearer picture of the data-set we will work on. Section 2.1 describes how the image data-set was given to us, the specific format of the images and their dimensions. We also detail how we reduced the data-set to a compute-friendly level, and how we converted it to a usable format. Finally, section 2.2 explains the reasons why we decided to apply data augmentation techniques to the data-set, and introduce the kinds of data augmentation that have been applied.

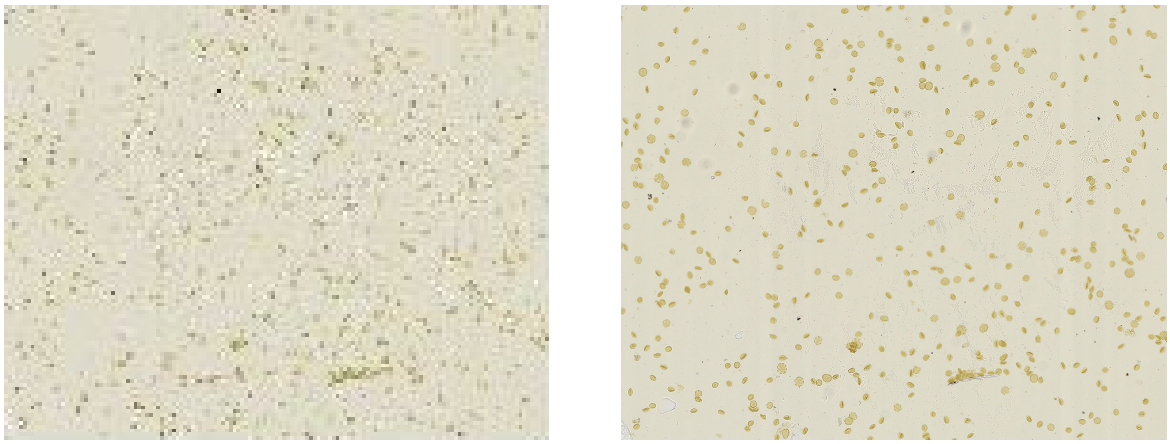
2.1 Image Format

The raw data-set consists of about 120 image files uploaded to Cytomine [24], an online, open-source web platform that allows people to collaboratively annotate and analyse large images. Cytomine is also able to semi-automatically process large data-sets with machine learning algorithms. The image files in the data-set are encoded as ".SCN" files. This file-type is quite peculiar and abstruse, as it is almost exclusively used when dealing with specific microscopes. The extension denotes a file type resulting from Leica virtual microscopy imaging devices. These files are containers for multiple "BigTIFF" image scans, which is already a more popular format. Leica-microscope's ouput files can only be opened with a few specific scientific image-processing software. In the context of this thesis, we opted for ImageJ [25]. ImageJ is an open-source image processing tool designed to handle with large, multidimensional data. It is extremely extensible and has thousand different plugins. These plugins makes ImageJ able to read and manipulate a wide range of image formats. One specific plugin, Bio-Formats [26], enables the reading and manipulation of microscopy data, including .SCN files.

With the help of ImageJ and the Bio-Formats plugin, we can open and visualize the data-set. Each .SCN file from the raw data contains multiple versions of the same microscope scan of a pollen type, every time in a different resolution (see figure 2.1), up to more than 5 different images per pollen scan. To get the best results possible, we chose to work with one of the highest resolutions available in the scans, which is 1 pixel for 0.5 micron, or 1 micron equals 2 pixels.

Then, for each .SCN file, we extracted the pollen scan in the desired resolution, and converted that into a .PNG image format. To have a uniform image set, and to limit computing time, we also decided to crop each image to a fixed dimension size : 4000x4000 pixels, or 2000 x 2000 microns. This gives us an image per species containing on average between 100 and 200 pollen grains. This format and uniformity in the data will facilitate the manipulation of the data-set.

Later, these images will be used to generate the training data-set. A desired amount of images of size 800 by 800 will be randomly cropped, together with their respective masks, and form the training set.

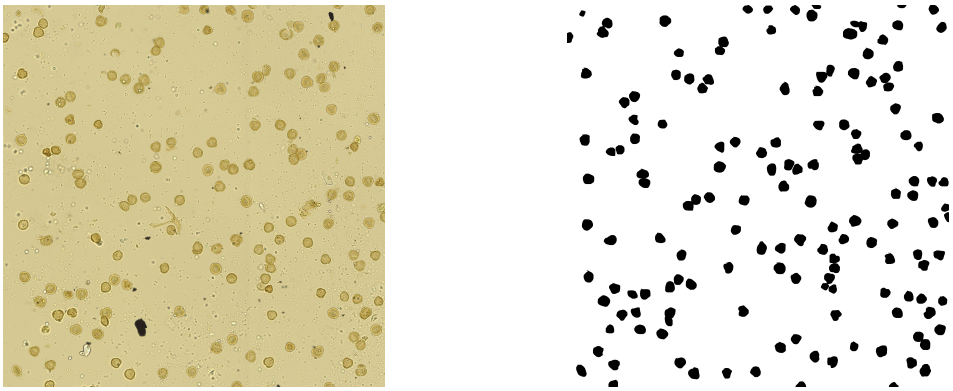


(a) resolution : 143x114 pixels (b) resolution : 2288x1824 pixels

Figure 2.1: The same image with different pixel resolutions. The pollen type is Salvia verticillata

In the context of this work, we want as much different pollen species as possible to make the pollen-identifier as powerful as possible. This means that we aim to have the most diverse data-set possible, in term of pollen species. The data-set consists of hundred of scans, with sometimes multiple files concerning the same pollen type. To avoid redundancy, and improve efficiency, we decided to take only one series of scans per pollen-species into account. The remaining scans were removed from the data-set.

Masks In order to supervise the training of our CNN, we need to get masks locating the pollen grain pixels for every image we want to include in our data-set. These masks are used to discern if a pixel is pollen or background, and will serve as ground truth for our network. With the help of ImageJ and a few java macros, we were able to manually annotate the pollen microscopy scans, and create masks for every different species of pollen. We did this by manually selecting the pollen grains one by one with imageJ's freehand selection tool, then combining all these selections together to finally save it as a binary mask. An example of a pollen species (Crambe maritima) scan and its corresponding mask can be seen in figure 2.2



(a) (b)

Figure 2.2: A sample of Crambe maritima pollen and its corresponding annotated mask

2.2 Image Augmentation

CNN performance improves a lot with the amount of available data. Therefore, in order to improve our neural network, without drastically increasing the amount of data to annotate, we can apply data augmentation to the existing data-set. Data augmentation are a set of techniques that allow to create new training data, from an already existing data-set. Image data augmentation is perhaps one of the easier data-augmentation types to comprehend and visualise. It consist of various transformations of the existing images, such as rotations, scale modification or brightness changes to create synthetic data. [27][28]

Considering our data-set, consisting entirely of pollen grains, it makes total sense of using data-augmentation to strengthen our neural network. Indeed, pollen cells from different species can be very similar. Therefore, we need a lot of data to make our model as robust as possible, enabling it to differentiate and classify different species.

The following data augmentation techniques were applied to our pollen data-set:

Geometric transformations

- Distortion : A geometric distortion is applied to the image
- Rotation : The image is rotated to a maximum of 45 degrees clockwise or counter-clockwise
- Scaling : The image is scaled inward, or outward, with a maximum of a 10% scaling factor.
- Flip : The image has a probability of 1 in 3 of being flipped horizontally
- Transpose : The image has a probability of 1 in 3 of being transposed

Colour space augmentations

- Brightness: The brightness of the image is slightly altered with a magnitude of maximum 10%
- Hue: There is a probability that the hue of the image is slightly changed
- Saturation : There is a 50% probability that the saturation of the image is changed with a magnitude of maximum 10%

2.3 Conclusion

To summarise this chapter, we started from a data-set containing very large files, formatted in the unpopular .SCN format, a unusable file type when working in Python. With imageJ, we were able to extract usable images from the .scn files and convert them into PNG files. We subsequently annotated these images to create a binary mask for each species. These images are then augmented and thereupon, a predetermined amount of images with fixed dimensions will be cropped from them. The combination of these augmented, cropped images and their corresponding masks will form the training set.

Chapter 3

Method

In this chapter, the core method investigated in this master's thesis will be explained. First, Section 3.1 will go over the choice of architecture for the CNN, explaining in detail why this specific structure was chosen and how it works. Then, in section 3.2 and 3.3 we will explain how we went from a theoretic model, to a practical, working, tool. Section 3.2 will describe how we created the pollen segmentator, a tool that is able to recognise pollen grain pixels from background pixels. Then, section 3.3 will get at the heart of the matter, explaining how we created a neural network able to distinguish, at a pixel level, a pollen species from another. In both of these last sections, we will explain the important implementation choices we made while constructing the networks.

3.1 The U-net architecture

The U-net is a convolutional network, specifically designed for biological image segmentation. It was imagined and developed in 2015 by Olaf Ronneberger, Phillip Fischer, and Thomas Brox at the University of Freiburg, Germany [29]. As of today, it is considered as one of the standard and most efficient CNN architectures for image segmentation, i.e. pixel-wise labeling. [30].

First and foremost, why is a U-net architecture a good choice for segmenting and classifying pollen grains? What are the advantages and drawbacks of such an architecture compared to a classic CNN, composed of a sequence of layers at the same resolution than the native image?

In the first place, in deep learning, it is widely known that we often need huge amounts of data to make a model robust. And sometimes, we can not afford that. Like in this case, collecting and annotating multiple thousand of pollen grains per species is not doable. Fortunately, this architecture accounts for exactly that. U-net is very effective even with a limited data-set. The use of lots of data augmentation on the available training data, ensures that the networks learn invariance to such deformations[31].

Firstly, U-net architecture, by considering multiple scales, allows to combines localised information with contextual information. Note that, since the prediction is a pixel-wise class label, sub-sampled features have to be upsampled to feed the network output. This leads to a hourglass architecture, also named encoder-decoder architecture.

Secondly, the U-Net involves skip connections, to directly transfer the high-resolution features that are computed from the input to the layer of same resolution in the upsampling path leading

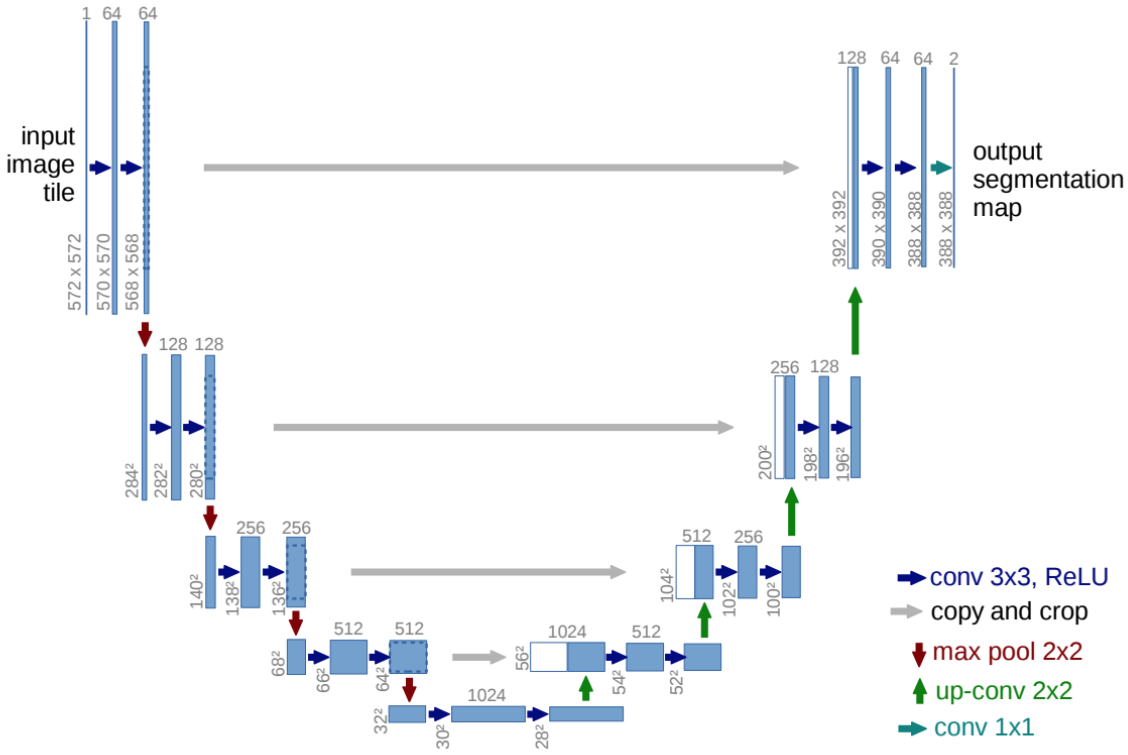


Figure 3.1: U-net architecture. The blue boxes represent multi-channel feature maps. The number of channels is denoted on top of the box. The size of the array is denoted in the left lower corner of the image. The white boxes represent copied feature maps. The arrows denote the different operations [29]

to the network output. Those skip connections help the network to merge local and contextual information, but also favors the back-propagation of gradients. [32].

A general outline of a U-net can be seen in figure 3.1. At first glance from the sketch, we can understand that the *"U"* in U-net comes from it shape. In the structure of the network, we can discern two major pathways. The first one being the contraction path, which is composed of classic convolutions and max pools. The second path is the expansion path, composed of transposed convolutions. It is used for up-sampling and enables to possibility of getting an output with the same dimensions as the input, granting precise localisation to the CNN.

3.1.1 Contraction path

The contraction, or down sampling path, is composed of a repetition of convolutions followed by a ReLU and a max pooling operation. On the figure above, we see that the convolutions are 3x3 and unpadded(meaning that the border pixels are lost). We can see on the illustration that after each layer, the number of features(indicated on top of the blue squares) is doubled. This down sampling also increases the complexity of the features. In conclusion, this path is used to capture the context with the help of a compact feature map[30].

3.1.2 Expansion path

The up-sampling path or expansion path, consists of the up-sampling of the feature map via three major steps that are repeated. Firstly, an up-convolution(transposed convolutions), which expands the size of the image. The second operation is a concatenation of the output from the last operation with the corresponding image from the contraction path. The third operation is 2 regular convolutions. The second operation combines localised, and contextual information. This is what makes the U-net so special.

3.2 Universal pollen segmentator

This section describes the methodology used to implement the U-net algorithm we reported in the previous section to create a universal pollen segmentation tool. This tool labels background pixels with zero, and pollen pixel with one, whatever the pollen species.

The first step to differentiate pollen grains from each other, is being able to discern them from everything else. Pollen grains are often visually very similar, and we can exploit this feature and make it an advantage in our search for a powerful segmentator. This tool will be assigning either the pollen label to a pixel, or in the other case, the background label. Thus, we are trying to develop a tool capable of binary pixel segmentation.

We programmed the entirety of the U-net network in `Python` [33], a very popular, high-level programming language. The model was implemented with the help of Keras [34], an open-source neural network library. Keras facilitates the design and implementation of NN by being user-friendly, modular and highly extensible. A more detailed choice of this particular library can be found in appendix A. In the following paragraphs, we will go over some important implementation choices we took for the parameters of our segmentation algorithm.

In machine learning, when developing a CNN, we have to choose activation functions corresponding with the output we want. In this case, we want to predict a binary outcome : pollen or background. A good choice for an activation function for this kind of prediction would be a sigmoid. As a reminder, a sigmoid will assign for all pixels a value between 0 and 1 for each class, depending on how confident the model is with the pixel being in the class. From that result, the algorithm will just choose the highest value, and that will be the predicted class for that pixel.

Another primordial choice for the construction of the segmentator is the choice of the loss function. This is the function whose minimisation will lead the CNN to predict what is expected. The gradient descent optimization algorithm used to train the network will be based on the gradient of the loss function with respect to the network parameters. In practice, those gradients are computed using the chain rule, through the back-propagation algorithm.[35] We chose to use the Binary Cross Entropy function. The mathematical equation for the cross-entropy is depicted in figure 3.1. It's powerful and efficient, cross entropy loss will measure the performance of a classifier whose output is a probability between zero and one. The cross entropy loss will rise increase as the predictions for the labels are different. Because there is a logarithm in the function, the loss increases very rapidly when the predicted probability is high. This results in the harsh penalisation of confident predictions that are not correct.

$$-\sum_{c=1}^{M} y_{o,c} \log(p_{o,c}) \tag{3.1}$$

In this equation:

- The M denotes the number of classes
- y is a binary indicator (0 or 1) if class label c is the correct classification for observation o
- p is the predicted probability that observation o is of class c

Finally, a last important hyper-parameter is the number of layers in the network. The number of layers in a image-processing CNN is actually an important factor. It influences the size of its receptive field. Remember that the receptive field in a CNN is the region of the input space that affects a particular node of the network. A neuron will thus be able to detect a feature such as an edge or a line, if the object is smaller than the receptive field.

When working with a U-net architecture, the size of the receptive field will double with every layer due to the max-pooling operations. Additionally, the bottleneck(deepest part in the U-net) also doubles the receptive fields . When inspecting various different species of pollen, we observed that the largest pollen grains have a maximum of around 30 pixels. We can now calculate the minimum amount of layers needed to be able to detect the grains: Taking 4 layers, we arrive at a receptive field of $2^{4+1} = 32$ pixels(The "+1" is for the bottleneck). To be safe but not taking an unreasonable and excessive amount of layers, we chose to select a number of 5 total layers.

Now we have designed and implemented an instrument capable of discerning pollen grains from background or other impurities in an image, we are one step away from our final goal. Thanks to this tool, we can now very easily identify pollen pixels in new data. Keep in mind that the quality of the segmentator will depend on the amount of data is supplied to the training algorithm. Thanks to the similar nature of pollen grains, and assuming that we supplied the segmentator with a substantial amount of data, we can now identify and annotate any pollen type.

3.3 Pollen classification

As a reminder, the whole point of the segmentator is to be able to annotate data for our pixel-wise pollen classification algorithm. Due to the special nature of our image data-set, composed only of images of *pure* pollen grains, meaning that only one species of pollen is present in each image, , it makes the segmentator very handy. Indeed, any identified pollen pixels on the image, will all be of the known species, associated to the image. That means we win a lot of tedious, time-consuming, inspection and annotation time. Because as said before, annotating images of mixed pollen species manually demands a high level of expertise and significant human effort.

Now the gathering of training data for our classification algorithm is automatised, the last big step in our work is the implementation of the classification algorithm itself. Just like the segmentation tool, the pollen classification tool was implemented in `Python` with the help of some external libraries such as `Keras`. As in the previous section, we will now detail some of the significant choices we took for the classification U-net algorithm.

We explained before that one must choose the right output layer activation function when conceiving a convolutional network. In this case, we want to predict a single label, from multiple classes. In other words, we hope to assign the correct species from every species the model has knowledge of. A good activation function for this kind of problem is the *Softmax*. This will result

in outputting a numerical value between 0 and 1, for each class, the sum of all outputs equalling to 1. The result can be seen as a probability distribution. Consequently, the chosen pollen class will be the one with the highest value. After the softmax we add an array, setting the highest value of the softmax to one for that class, and zero for the others, effectively one-hot encoding our predictions.

Additionally, the chosen error or loss function will be almost the same as in the segmentation algorithm. In combination with the softmax, we will be using a categorical *Cross-entropy* function to optimise the algorithm. The cross-entropy will optimise the model by comparing the predicted probability distribution with the true distribution.

In conclusion, we constructed a multi-layered convolutional network, capable of multi-label pollen classification. The model was built following the U-net architecture, and is trained with a data-set consisting of images of pure pollen grains. The results of our model applied to our data-set, will be described in the next chapter.

Chapter 4

Applications

In this final chapter, we illustrate the application of our implemented U-net classification model on our data-set. First of all, section 4.1 describes the way generate the testing set our model. Then, we detail the choice of hyper-parameters for the classification model. Section 4.2 graphically illustrate and discusses the predicted results of our computed model on the testing set.

4.1 Validation method

To be able to test our model, there are two very major things we need to acquire. Primarily, we need a testing set , and secondly we need to compile the model. Let us first start with the less complex one, being the former.

4.1.1 Testing set

Even though our data-set is composed of images of pure pollen, one of the goals of the model is the capacity of differentiating types of pollen grains when they are mixed together. This is why the need of images composed of multiple types of pollen is so important. Unfortunately, due to time constraints, we were not able to acquire mixed images that corresponded with the species of our data-set of pure pollen images. Thankfully, we were able to synthesise mixed images by ourselves by tinkering with the pure images. We composed these mixed images by arbitrarily picking a pollen species from the data-set, and then cropping a random, non background part of that image. To be able to have to most uncropped pollen grains on the images, we chose to create 2 by 2 matrices of different images. An example of such an synthesised mixed image can be seen in figure 4.1.

The drawbacks of these kind of synthesised images compared to real mixed images, is that sometimes we can get awkward, unnatural crops that will hinder the identification of the right pollen species. A second drawback is that we get non-uniform background that will again complicate the prediction. But this problem can thankfully be partially removed by adding a few composite images to the training data, to teach the model to ignore the often sudden contrast changes at the borders of the regions. A third problem is that, by randomly selecting species, we could get an imbalance in the size of the classes.

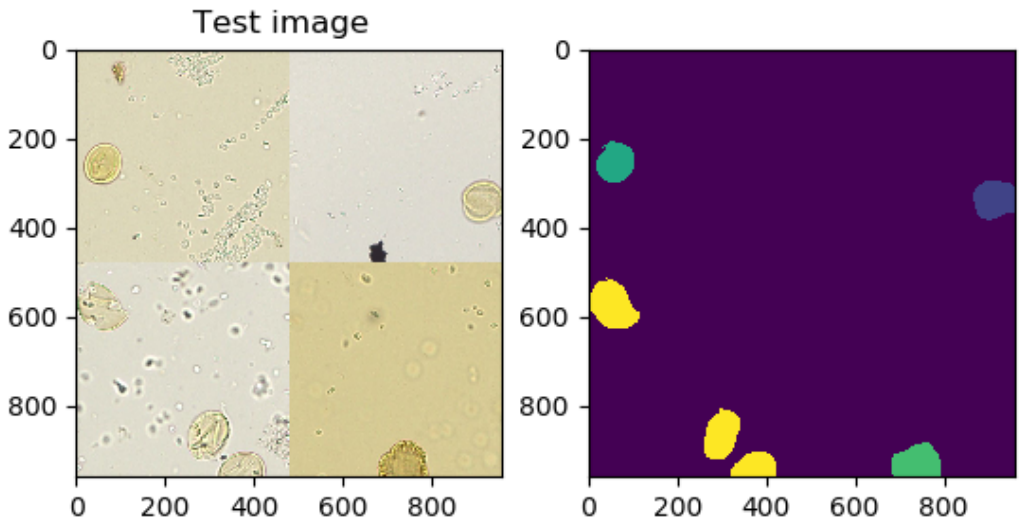


Figure 4.1: A synthesised image of multiple different pollen species(left), with it corresponding mask(right). From top to bottom and left to right : Arabidopsis thaliana, Arrabis hirsuta, Genista pilosa, Crambe maritima

4.1.2 Model

When compiling a model, even with a given architecture and data-set, there are a dozen of hyper-parameters that will influence the capabilities of the model. In these following paragraphs, we go over the exact values of the ones we chose for the validation of our proof-of-work.

Classes We chose to train our model with a total of **11 different species** of pollen grains. From the data that was at our disposal, we tried to select species that were visually as diverse as possible.

Training, Validation and Testing sets The compiled U-net classification model was fit on 170 images, composed of 99 pure pollen images, and 60 synthesised mixed images. It is fine-tuned with a validation data-set, composed of 33 images that are - following the ratio of the training set - either mixed or pure images. Finally, the testing set is composed of 30 synthesised mixed images. All images have the same dimensions : 800 by 800 pixels. This ratio split of sets gets us at **70% / 15% / 15%**, which is a classic ratio split in machine learning.

Learning rate The learning rate is perhaps one of the most important, if not the most important parameter to configure correctly for a model. A correct learning rate can only be guessed through experiencing with the model. An even better way to improve the time to convergence, is to decrease the learning rate as the model advances into the algorithm : an adaptive learning rate. We found out that starting with a **learning rate of** $2e^{-4}$ was a good way to rapidly arrive at convergence. We also chose a decaying factor of 0.5, and a step size of 20. In other words, our learning rate is halved every 20 steps.

Epochs and Batches The number of epochs, corresponding to the number of times the algorithm goes trough the data to update the weights, was set to **100 epochs**. This choice is based on the observation of the loss curves. By plotting the loss with respect to the number of epochs, we can see that it has converged after a hundred epochs. We compiled the model while working on 2 different batches, to ensure the convergence of the model was not a lucky draw/

Layers As stated before, to ensure that our model has a large enough receptive field to recognise pollen grains from other background elements, we constructed a U-net composed of **5 layers**.

Activation and Loss functions For our multi-label pollen classification network, we chose to opt for a **SoftMax** activation function, combined with a **categorical cross-entropy** loss function.

4.1.3 Test results measurement method

In order to be able to quantitatively quantify our results, we need a way gauge the success(or failure) of our model. We have to keep in mind that because the images were manually annotated, we can not be certain that the annotated mask are the "real" truth. The error is human, and that means that only comparing the pixel values directly wont necessarily give us the exact performance of the network. Thus is why we will also judge based on visual inspection. But even with visual proof, we need quantitative proof that our model works. To achieve this, we will measure a few different statistics.

To validate the classification capabilities of the network, we will also compare the ratio of the predicted pollen species in the test images, with the ratios of the masks. We inspect the ratios to compare if the most present pollen types in the prediction are the same as the one in the masks. This is a somewhat similar approach as the Jaccard coefficient described below, but it is simpler and a quicker approach.

Jaccard similarity coefficient

The Jaccard similarity index, or coefficient, measures the similarity between two sets of data. It is a very popular measuring statistic, as it is efficient and very simple to understand. The value range goes from 0, a really bad classification, to 1, a perfect one. The coefficient is formally defined as the size of the intersection of the two sets, divided by the size of the union of the two sets. A mathematical representation of the Jaccard coefficient is presented in figure 4.2.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

Figure 4.2: A mathematical representation of the Jaccard similarity coefficient

Matthews correlation coefficient

We will finally measure our classifier with one last instrument. The Matthews correlation coefficient(MCC), or phi coefficient is a popular and reliable statistic that measures the quality of a binary or multi-class classifier. Unlike the Jaccard index, it takes true and false positives and negatives. It handles unbalanced data-sets better, and is thus a better measure when the classes differ a lot in size. The formula for the computation of this statistic can be found in equation 4.1. The MCC has a range going from -1 to 1, with -1 corresponding with a really bad classifier, and 1 indicating a completely correct one. We will use this coefficient to compare the predicted images with their ground truth.

$$MCC = \frac{TP * TN - FP * FN}{\sqrt{(TP + FP) * (TP + FN) * (TN + FP) * (TN + FN)}}$$

4.1: A mathematical representation of the Matthew correlation coefficient. TP = True positive, TN = True negative, FP = False positive, FN = False Negative

Summary To conclude, in addition to visual inspection, we will judge our model's performance by calculating and analysing the jaccard similarity coefficients for each class. This will give us a measure of the similarities between the true value and the prediction set. Subsequently, we will measure the capabilities of our classification algorithm with the Matthews correlation coefficient, which is a bit more reliable as it takes a few more elements into account than the jaccard index,like true and false positives.

4.2 Results

4.2.1 Loss and accuracy rate curves

To ensure our algorithm has converged, we must first inspect the loss curves. The curves we constructed depict the cross-entropy loss of the model, with respect to the number of epochs. Ideally, the value of the loss would drop as close as possible to 0, and converge. When comparing the validation loss with the training loss, we prefer the former being larger than the latter. Conversely, this would mean that the model is overfitted.

In figure 4.3 we can see a graph of the categorical cross entropy loss in function of the number of epochs. The loss has converged to a very reasonable value of 0.0082 that lies very close to 0. We can thus assume with relatively great confidence that this model has converged.

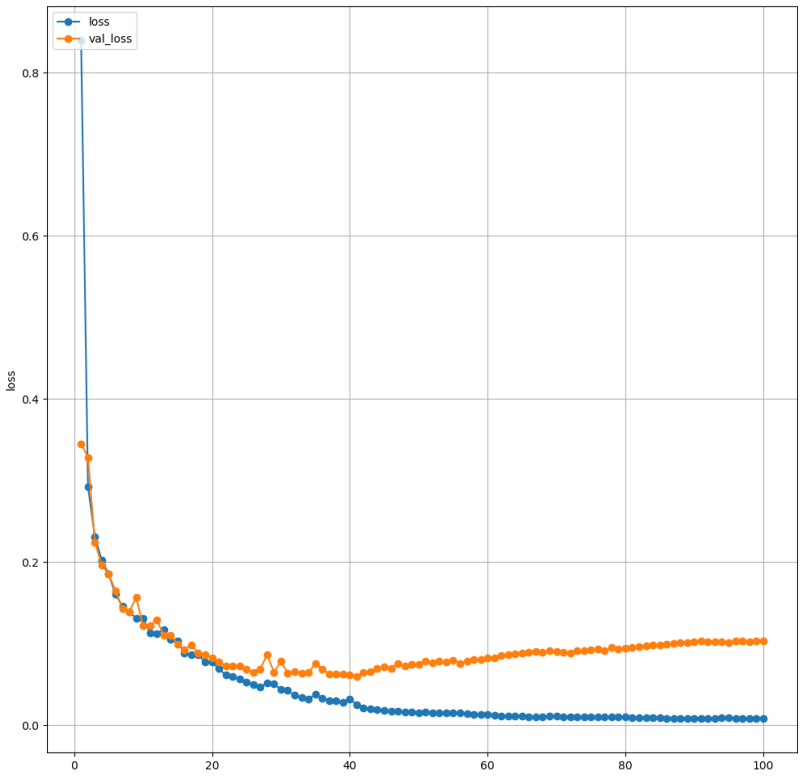


Figure 4.3: The value of the cross-entropy loss of the model in function of the number of iterations of the algorithm (Batch 1 of 2) The loss converges to a value of 0.0082, while the validation loss converges to 0.103

Another way of gauging the convergence and performance of the network would be looking at the accuracy of the CNN, it measures how accurate your model's prediction is compared to the true data. It computes the amount of pixels correctly classified. The accuracy of our model in function of the epochs is depicted in figure 4.4. We can observe that the accuracy converges to 1, again confirming the model's convergence.

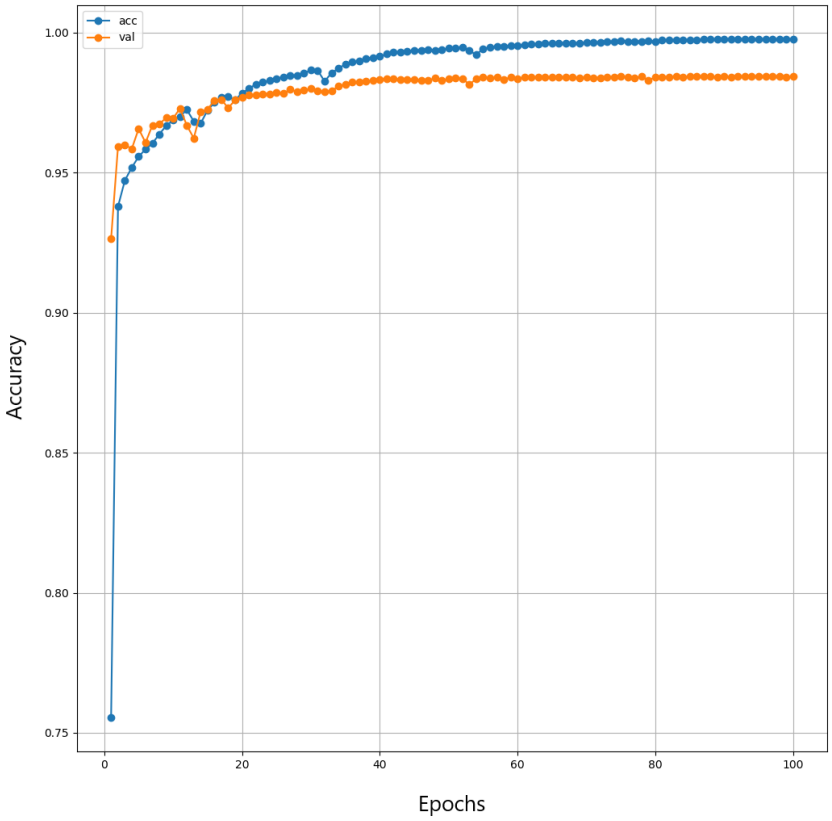


Figure 4.4: The value of the accuracy of the model in function of the number of iterations of the algorithm (Batch 1 of 2)

4.2.2 Discussion of the results based on visual analysis

We will now analyse the results by inspecting visually some of the predictions. As a reminder, our predicted results from our softmax are then transformed into one-hot encoding for visibility reasons. Thus, a prediction can be illustrated by a series of images, one for each class. Having 11 classes, we must look at the 11 different images to judge the correctness of the model. To be visually more appealing, we won't be putting the 11 images and the 11 corresponding masks on the screen. We will be only be showing the images for the classes where more than 5% of the identified pollen was detected. If there are less than 4 image with more than 5% predicted, we will show the 4 images with the most pollen identified.

We also must not forget that, because the test images are randomly generated by the cropping of pure images, there is reasonable probability that the test-set is unbalanced. We delve deeper into this phenomenon by computing the total number of pixels per class. The result can be seen in table 4.1. We can see that indeed, there is far less pollen of class 1,3 and 8 than the other classes. A percentage-wise comparison of the present classes is depicted in table 4.2.

Let us take a random image [4.5] and discuss the prediction made by our model. Additional prediction images will be made available in appendix B.2. Looking at the background class in figure 4.5, we can see that all pollen grains are pretty well segmented. The categorising of the

	Background	Class 1	Class 2	Class 3	Class 4	Class 5
Number of pixels	$1.948 * 10^7$	$1.9 * 10^4$	1.5710^7	$6.41 * 10^3$	$1.03 * 10^5$	$2.64 * 10^5$
	Class 6	Class 7	Class 8	Class 9	Class 10	Class 11
Number of pixels	$1.4 * 10^5$	$3.47 * 10^5$	$5.29 * 10^4$	$1.93 * 10^5$	$3.96 * 10^5$	$1.04 * 10^5$

Table 4.1: This figure depicts the total number of pixels per class in the testing image-set

	Background	Class 1	Class 2	Class 3	Class 4	Class 5
Percentage of pixels		1.15%	0.95%	0.39%	6.27%	16%
	Class 6	Class 7	Class 8	Class 9	Class 10	Class 11
Percentage of pixels	8.5%	21.16%	3.24%	11.78%	24.1%	6.3%

Table 4.2: A percentage-wise comparison of the number of pixels in a class compared to the total amount of pixels in the testing image-set.(without background).

pollen grain pixels into classes is quite successful too. There are a few pixels identified as class one instead of class ten, but as a whole, this image's classification has been rather upstanding.

Most present species prediction The U-net classification model does a good job at gauging the ratio of present pollen species in an image. Of the 33 predicted mixed images, the model was correct 31 times about the most represented pollen species. It predicted the two most present species 28 out of the 33 images, and guessed the 3 most present pollen types right about 80% of the time. Most errors made can be justified by the fact that they are often pollen types that were cropped by the transformation of multiple images into one.

4.2.3 Jaccard similarity coefficient

We calculated the average Jaccard coefficient for all classes on the 33 test images. The values for every coefficient are report in table 4.2.3. From these results, we can deduce that the classification is a success. With an average of 55,7% similarity, the model is quite capable. We can see that, as expected, the coefficient is extremely high for the background class. This means that the model is especially successful with the differentiation of background pixels in respect to pollen pixels. The coefficient is a lot lower for some specific classes. However with this small(33 images), testing set composed of random pollen species, we still have to remain careful by looking at these numbers, because some classes may be underrepresented. Indeed, with closer inspection, we observe the classes with a low coefficient correspond to the ones that are pixel-wise less represented in the test images.

	Background	Class 1	Class 2	Class 3	Class 4	Class 5
Jaccard Coefficient	0.9833	0.242	0.41	0.12	0.495	0.47
	Class 6	Class 7	Class 8	Class 9	Class 10	Class 11
Jaccard Coefficient	0.62	0.797	0.29	0.642	0.67	0.76

Table 4.3: This table depicts the Jaccard coefficient values for all 12 classes(background + 11 pollen species). The coefficient compares the prediction image set with the ground truth.

Matthews correlation coefficient

The average Matthews correlation coefficient was computed for each pollen species and for the background. As observed in table 4.4, the MCC's for some classes are not very high, and as for the Jaccard index, these values correspond to the same, underrepresented classes. But if we look at the most present species, the MCC are pretty close to 1, and correspond thus to a good prediction.

	Background	Class 1	Class 2	Class 3	Class 4	Class 5
MCC	0.892	0.283	0.507	0.253	0.608	0.587
	Class 6	Class 7	Class 8	Class 9	Class 10	Class 11
MCC	0.704	0.877	0.421	0.755	0.75	0.852

Table 4.4: This table depicts the Matthews correlation coefficient values for all 12 classes(background + 11 pollen species). The corresponding pollen species for each class can be found in appendix B

From these results, we can conclude that when there are enough pollen pixels in a given image, the model is able to classify that grain of pollen quite well. Problems arise when only a small or a half cropped piece of pollen grain is present on the image. The model has then a much harder time identifying those pixels.

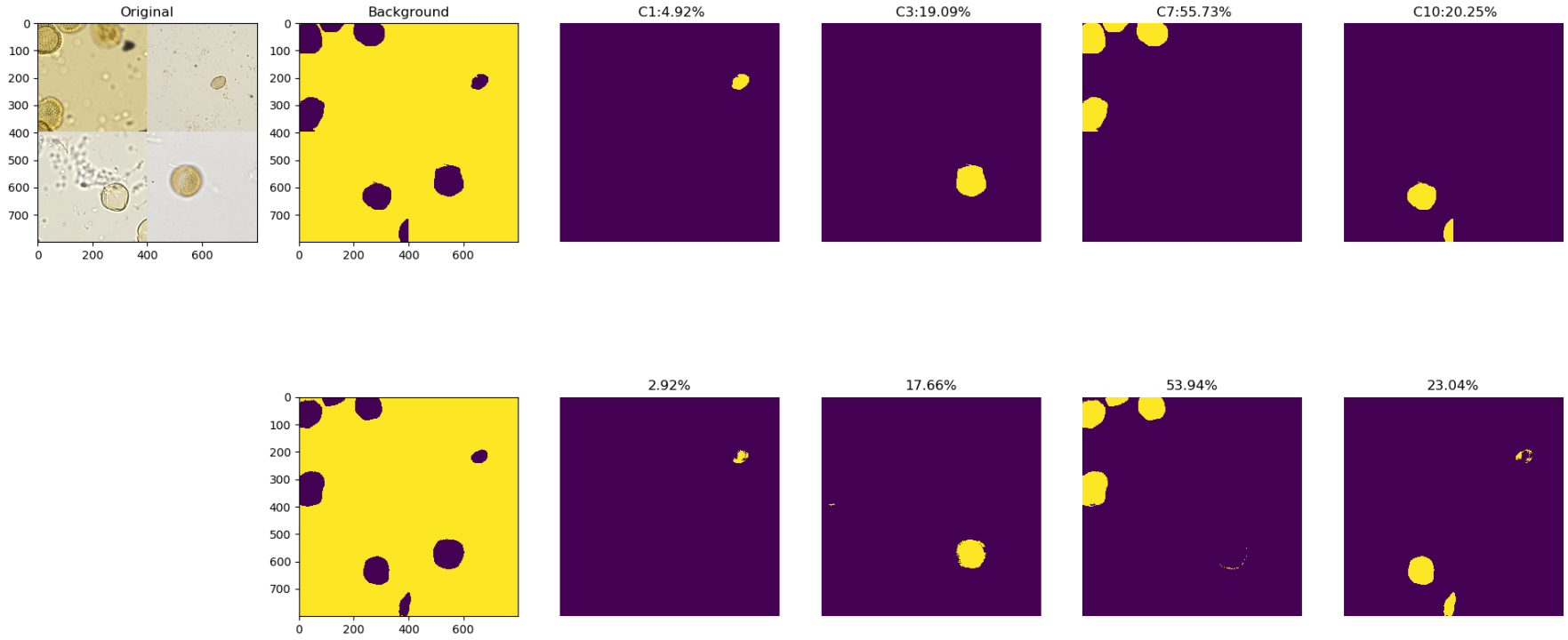


Figure 4.5: A visualisation of the prediction for a random mixed image. The upper row consists of the ground truth(masks), the lower row are the predictions by the model. Except the first image depicting the original image, we can see from left to right: The background class, followed by class 1, 3, 7 and 10. The species corresponding to each class can be found in Annex B.1

Conclusion

In this master's thesis, we tried to find a way to improve the automation of pixel-wise pollen grain classification, by training it with a data-set of pure images(only composed of one species of pollen). The advantages of addressing the problem this way, is that it requires a lot less time-consuming annotation time and inspection. As we know, manual multi-class annotation demands a *lot* of human effort and a certain degree of expertise.

We addressed the problem in the following way: first we implemented a Universal pollen segmentator: a tool that identifies pollen grains by labelling background pixels with zero, and pollen pixels with one. This gives us an instrument which allows us to add more pollen species to the data-set without taking hours to annotate it. Secondly, we implemented a classification CNN, with as design the notorious U-net architecture. We consequently trained our pixel-wise classification algorithm with augmented the data-set composed of pure pollen images.

The results of our classification algorithm, evaluated based on the jaccard similarity coefficient and the Matthews correlation coefficient, suggest our model has a pretty good classification level, but only for the well represented species. The model was trained with limited data, and the testing images were unnatural, synthesised images constructed from several pure images. Moreover, the size of the classes were unbalanced. This means that with more data and applied to real mixed images of pollen, the resulting predictions would probably be even better.

To improve this pollen classification strategy, we could in the first place create the same model at a much larger scale. It would be interesting to see the results of the same model trained with hundreds if not thousands of different pollen species. One could imagine that the segmentation tool would perform even better than it does in this work. An eventual problem that could arise, one we even encountered while testing the CNN, is that all the data corresponding to a particular pollen species has to be uniform. If for a given species, half of the images are taken with one microscope, and the other half with another type of microscope that does not have the same acquisition technique, there is a possibility that the classifier will struggle to learn that species.

Palynology, the study of pollen grains, is a field with a myriad of significant applications in various sectors like the healthcare industry, botany or even forensic science. A big part of what palynology encompasses as a science, is the identification and localisation of the origin of the pollen grains. This is why an efficient strategy in the automatisation of the classification of these grains is such an important issue. This work has explored some potential solutions with the hope of facilitating the development of future achievements in this field.

Appendix A

Implementation

A.1 PyTorch or Keras?

When constructing a CNN, we must choose a neural network library to implement our chosen architecture. There are two major options, each with their advantages and drawbacks.

Advantages of Keras

- **Architecture** Keras has a much more simpler architecture than PyTorch. It it far more concise and easier to read than PyTorch.
- **High level API** Compared to PyTorch, Keras is syntactically far more simple and easy to code with. Its is very user-friendly and this limits the very time-consuming programming.
- **Backend** Backends for Keras include:TensorFlow, Theano and Microsoft CNTK backends.

Advantages of PyTorch

- **High level API** PyTorch offers a low-level programming, being more flexible for more complex and custom usage.
- **More powerful** PyTorch has a higher speed and is more powerful at computing operations than Keras. Which is why PyTorch is more useful for large data-sets

Considering the reasons above, realising we don't work with a very large data-set, and we are quite novice in the implementation of neural networks, we chose Keras as our NN library.

Appendix B

Prediction images

B.1 Species corresponding to each class

- Class 1: Arabidopsis Thaliana
- Class 2: Arrabis Hirsuta
- Class 3: Brassica Napus
- Class 4: Capsella bursa-pastoris
- Class 5: Colutae arborescens
- Class 6: Coronilla Varia
- Class 7: Crambe Maritima
- Class 8: Cytisus scoparius
- Class 9: Galeopsis angustifolia
- Class 10: Genista Pilosa
- Class 11: glechoma Hederacea

B.2 More examples of predictions of synthesised images

Figure B.1: A visualisation of the prediction for a random mixed image.The species corresponding to each class can be found in Annex B.1

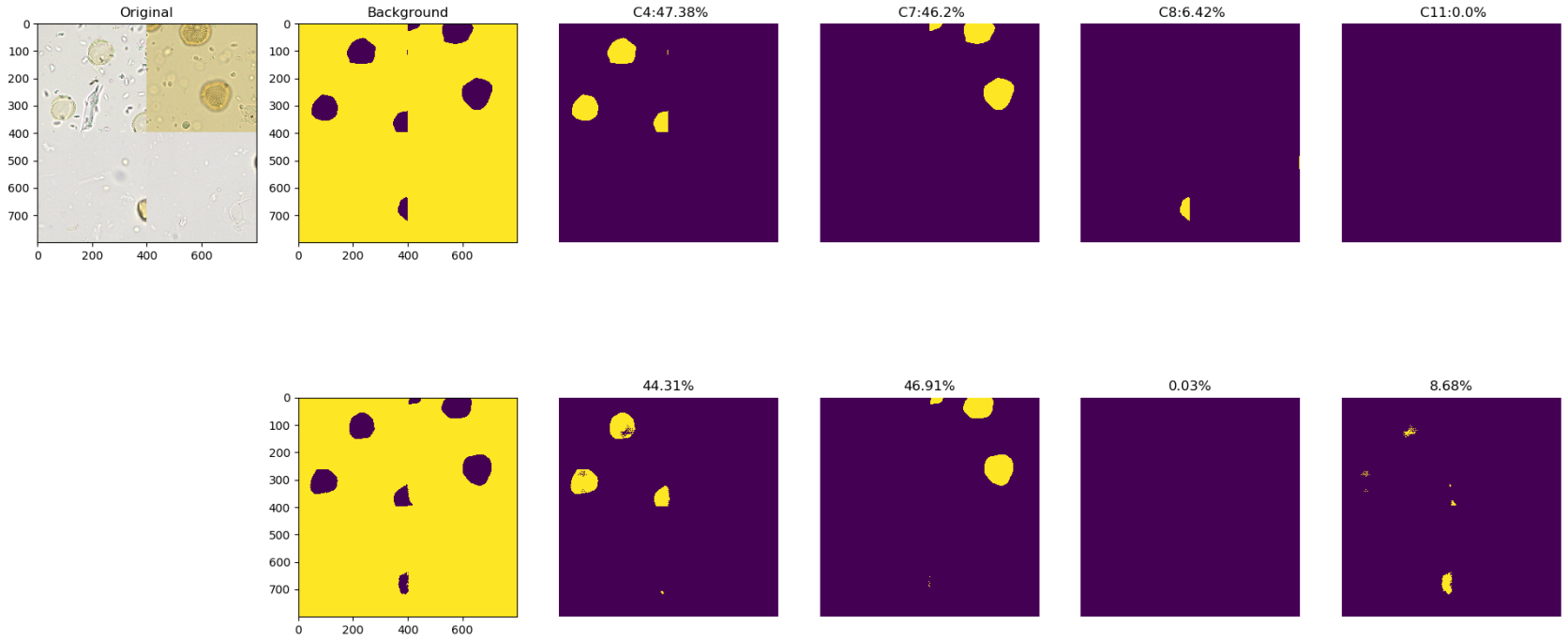


Figure B.2: A visualisation of the prediction for a random mixed image.The species corresponding to each class can be found in Annex B.1

Bibliography

[1] K. A. Holt and K. D. Bennett, "Principles and methods for automated palynology," *New Phytologist*, vol. 203, no. 3, pp. 735–742.

[2] E. Stillman and J. Flenley, "The needs and prospects for automation in palynology," *Quaternary Science Reviews*, vol. 15, no. 1, pp. 1 – 5, 1996.

[3] V. Sevillano and J. L. Aznarte, "Improving classification of pollen grain images of the polen23e dataset through three different applications of deep learning convolutional neural networks," *PLOS ONE*, vol. 13, pp. 1–18, 09 2018.

[4] W. J. Treloar, G. E. Taylor, and J. R. Flenley, "Towards automation of palynology 1: analysis of pollen shape and ornamentation using simple geometric measures, derived from scanning electron microscope images," *Journal of Quaternary Science*, vol. 19, no. 8, pp. 745–754, 2004.

[5] C. M. Travieso, J. C. Briceño, J. R. Ticay-Rivas, and J. B. Alonso, "Pollen classification based on contour features," in *2011 15th IEEE International Conference on Intelligent Engineering Systems*, pp. 17–21, 2011.

[6] M. Rodriguez-Damian, E. Cernadas, A. Formella, M. Fernandez-Delgado, and Pilar De Sa-Otero, "Automatic detection and classification of grains of pollen based on shape and texture," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 36, no. 4, pp. 531–542, 2006.

[7] M. Fernández-Delgado, P. Carrión, E. Cernadas, and J. Galvez, "Improved classification of pollen texture images using svm and mlp," *3rd IASTED International Conference on Visualization, Imaging and Image Processing (VIIP2003)*, vol. 2, 01 2003.

[8] S. University, "Cs231n: Convolutional neural networks for visual recognition." http://cs231n.stanford.edu/, 2015-2020.

[9] ADALTA, "Surfer 18." Available at `https://www.adalta.it/Pages/-GoldenSoftware-Surfer-010.asp`. Retrieved on july 2020.

[10] Y. Li, C. Wei, and T. Ma, "Towards explaining the regularization effect of initial large learning rate in training neural networks," in *Advances in Neural Information Processing Systems 32* (H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, eds.), pp. 11674–11685, Curran Associates, Inc., 2019.

[11] T. Schaul, S. Zhang, and Y. LeCun, "No more pesky learning rates," in *Proceedings of the 30th International Conference on Machine Learning* (S. Dasgupta and D. McAllester, eds.), vol. 28 of *Proceedings of Machine Learning Research*, (Atlanta, Georgia, USA), pp. 343–351, PMLR, 17–19 Jun 2013.

[12] J. Brownlee, "Loss and loss functions for training deep learning neural networks." Available at `https://machinelearningmastery.com/`

loss-and-loss-functions-for-training-deep-learning-neural-networks/, October 23, 2019. Retrieved on june 2020.

[13] C. M. Bishop, *Neural Networks for Pattern Recognition*. USA: Oxford University Press, Inc., 1995.

[14] R. Draelos, "Loss and loss functions for training deep learning neural networks." Available at https://glassboxmedicine.com/2019/05/26/classification-sigmoid-vs-softmax/, May 26, 2019. Retrieved on june 2020.

[15] S. Ronaghan, "Deep learning: Which loss and activation functions should i use?." Available at https://towardsdatascience.com/deep-learning-which-loss-and-activation-functions-should-i-use-ac02f1c56aa8, July 26, 2018. Retrieved on june 2020.

[16] D. Gupta, "Fundamentals of deep learning – activation functions and when to use them?." Available at https://www.analyticsvidhya.com/blog/2020/01/fundamentals-deep-learning-activation-functions-when-to-use-them/, Jan 30, 2020. Retrieved on july 2020.

[17] Uniqtech, "Deep learning: Which loss and activation functions should i use?." Available at https://medium.com/data-science-bootcamp/understand-the-softmax-function-in-minutes-f3a59641e86d, Jan 30, 2018. Retrieved on july 2020.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[19] A. Deshpande, "A beginner's guide to understanding convolutional neural networks." Available at https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/, July 20, 2016. Retrieved on july 2020.

[20] M. Glossary, "Gradient descent." Available at https://ml-cheatsheet.readthedocs.io/en/latest/gradient_descent.html. Retrieved on july 2020.

[21] DeepLizard, "Convolutional neural networks (cnns) explained." Available at https://deeplizard.com/learn/video/YRhxdVk_sIs. Retrieved on july 2020.

[22] A. Dertat, "Applied deep learning - part 4: Convolutional neural networks." Available at https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2, Nov 8, 2017. Retrieved on july 2020.

[23] V. Varshney, "Making an image classifier using cnns and pytorch." https://medium.com/datadriveninvestor/making-an-image-classifier-using-cnns-and-pytorch-17518905c85b, February 15, 2020.

[24] "Cytomine." Available at https://cytomine.coop/, 2018. Retrieved on july 2020.

[25] "Image-j." Available at https://imagej.nih.gov/ij/index.html, 2018. Retrieved on april 2019.

[26] U. of Dundee & Open Microscopy Environment, "Bio-formats." Available at https://www.openmicroscopy.org/bio-formats/#:~:text=Bio%2DFormats%20is%20a%20software,data%20using%20standardized%2C%20open%20formats., 2005-2020. Retrieved on april 2020.

[27] J. Brownlee, "Image augmentation for deep learning with keras." Available at https://machinelearningmastery.com/image-augmentation-deep-learning-keras/, June 29, 2016. Retrieved on april 2020.

[28] R. Khandelwal, "Data augmentation techniques in python." https://towardsdatascience.com/data-augmentation-techniques-in-python-f216ef5eed69, Dec 11, 2019.

[29] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," 2015.

[30] J. Zhang, "Unet — line by line explanation." Available at https://towardsdatascience.com/unet-line-by-line-explanation-9b191c76baf5, 2019. Retrieved on april 2020.

[31] A. Kizrak, "Deep learning for image segmentation: U-net architecture." Available at https://heartbeat.fritz.ai/deep-learning-for-image-segmentation-u-net-architecture-ff17f6e4c1cf, Sep 6, 2019. Retrieved on may 2020.

[32] A. Mohan, "An overview on u-net architecture." Available at https://medium.com/datadriveninvestor/an-overview-on-u-net-architecture-d6caabf7caa4, Nov 26, 2019. Retrieved on may 2020.

[33] "Python." Available at https://www.python.org/, 1991-2020. Retrieved on july 2020.

[34] "Keras." Available at https://keras.io/, Release date : March 27, 2015. Retrieved on july 2020.

[35] "Deep learning," *Nature Cell Biology*, vol. 521, pp. 436–444, May 2015.

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**