Tampereen yliopisto
Tampere University

Juan P. Lagos

TRAINING BASED SEGMENTATION FOR TISSUE EXTRACTION IN WHOLE SLIDE IMAGES

Faculty of Computing and Electrical Engineering
Master of Science Thesis
April 2019

ABSTRACT

JUAN PABLO LAGOS: Training based segmentation for tissue extraction in whole slide images
Tampere University
Master of Science thesis, 66 pages
April 2019
Master's Degree Programme in Information Technology
Major: Audio-Visual Signal Processing
Examiner: Prof. Heikki Huttunen
Keywords: convolutional neural network, histopathology images, semantic segmentation, backpropagation, discrete convolution.

Reducing the time and storage memory required for scanning whole slide images (WSIs) is crucial. In this thesis work we tested and assessed the performance of two popular neural network architectures, namely *DeepLabV3+* and *Unet*. In addition to that, a desktop application used to annotate histopathology images was developed, such application ultimately provided the data needed in order to train the neural networks. Both *DeepLabV3+* and *Unet* accurately separated the regions of interest out of the WSIs, however *DeepLabV3+* outperformed *Unet*, striking a pixel wise accuracy of 96.3%, while *Unet* scored 94.7% in the same metric. Morover *DeepLabV3+* also outscored *Unet* in the IoU metric with values of 0.446 and 0.398 respectively. We showed the effectiveness of using deep neural networks for the case of semantic segmentation in histopathology images, more specifically for extracting tissue areas from WSIs, and how this can be used to improve the performance of WSI scanners.

PREFACE

The research topic of this thesis work was provided by Jilab Inc and supervised by Professor Heikki Huttunen, head of Machine Learning research group at Tampere University. I would like to thank Professor Heikki Huttunen for supporting this research, I truly appreciate his effort and time as well as his will to share his knowledge and expertise in this topic. His help and guidance made things a lot easier during the development of this research, it was a true privilege to learn from him.

I would like to thank Jilab Inc for providing me with all the samples required for this research. They also allowed me to use their facilities and equipment without which this research would have not been possible. I would like to extend my gratitude to Onni Ylinen, Senior Software Engineer at Jilab Inc. He always showed great interest and support during the course of this research. His feedback and brilliant ideas were really useful, he was always there to help and assist whenever I needed his support.

Last but not least, I would like to thank my parents Francisco Lagos and Alba M. Benítez for their enormous effort and sacrifice just so I can dedicate my life to what I feel passion for, they are ultimately the authors of this thesis. To my sister Paula A. Lagos for being an incredible role model for me and my girlfriend Katherine Briceño for her unconditional support.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND SYMBOLS

WSI Whole Slide Image

ANN Artificial Neural Network

CNN Convolutional Neural Network

DCNN Deep Convolutional Neural Network

FCCN Fully Connected Cascade Network

RNN Recurrent Neural Network

ReLU Rectified Linear Unit

BMLP Bridged Multi Layer Perceptron

MLP Multi Layer Perceptron

UI User Interface

UX User Experience

1. INTRODUCTION

Image segmentation is a very relevant and decisive step in image analysis and image classification. Any further procedure involved in image processing will eventually rely heavily on image segmentation and so will the quality of the final results. Image segmentation can be performed by manual procedures but it leads to subjective measures and human based errors. In fact, as stated by K. Fu and J. Mui [1], "the image segmentation problem is basically one of psychophysical perception, and therefore not susceptible to a purely analytical solution".

The image segmentation problem has been largely studied during the last few years and there is ongoing research on it. There exist different segmentation techniques which address this problem and provide efficient methods. The selection of such methods depends strongly on the case of study and its particularities.

Medical image analysis is taking great advantages of the advances and latest breakthroughs of image segmentation as it helps to optimize different kinds of measurements and reduce inaccuracies. Whole slide images (WSI) for example, are digitized conventional glass slides that are used by medical experts worldwide for different applications, such as diagnosis, research or education. There are several applications of image segmentation handling such digitized images. Figure 1.1 shows an example of a typical WSI.

WSIs are produced by utilizing specialized hardware and software. The former refers to a scanner which digitizes glass slides and the latter is used to visualize and analyze the resulting digitized images known as digital slides [2]. Such digital slides are usually high resolution images which may take enormous memory space [3]. This introduces new challenges as scanner manufacturers try to provide devices capable of producing high resolution digital slides in a short span of time.

It is here then, when image segmentation plays a vital role to deal with the problems of storage and time optimization. As it turns out, pre-localizing the tissue from slides can save up to 40% of the time required for scanning and it effectively reduces memory space [3]. Pre-localization of the tissue requires extracting the regions of the

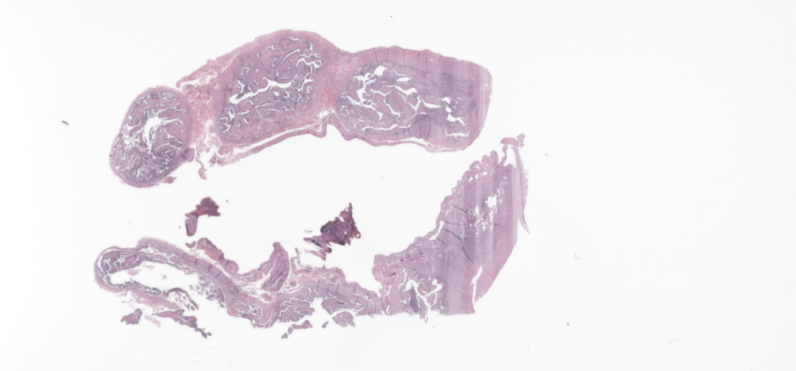
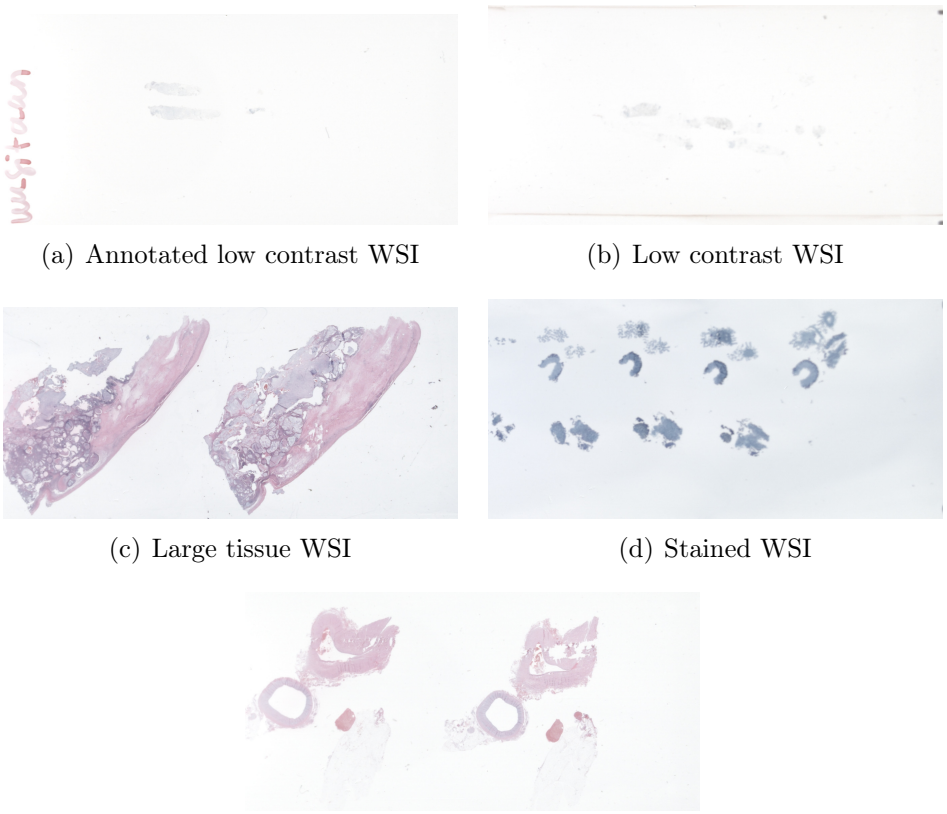


Figure 1.1 *Whole slide image*

slide image which correspond to the tissue and separate them from the background. The outcome of pre-localization is what is also referred to as mask. A mask indicates the regions in the slide image that should be scanned and the ones that should be ignored in order to save time and memory space. Overlaying the mask on the slide image produces an image as shown in Figure 1.2.

Figure 1.2 *Localization of tissue*

Despite of the numerous studies for pre-localizing tissue, it is not yet a trivial problem. Physical glass slides are prone to contain artifacts such as dust particles, air bubbles and abnormalities which can make the segmentation process even more challenging. Moreover, the variability among samples should also be taken into account. Samples usually vary depending on the size of the tissue, staining, color, shape and distribution [3]. Figure 1.3 shows an example of such variability, Figure 1.3(a) and Figure 1.3(b) for instance, present low contrast between the tissue and the background, opposite to Figure 1.3(c) which shows high contrast. There are also samples which have been stained as in the case of Figure1.3(d) where blue staining has been applied on the sample, that not only changes the background in comparison with other samples but also reduces the contrast between the tissue and the



(a) Annotated low contrast WSI

(b) Low contrast WSI

(c) Large tissue WSI

(d) Stained WSI

(e) Variability within a WSI

Figure 1.3 *Variability in WSI samples*

background. Also samples such as the one in Figure 1.3(e) brings an extra challenge as the variability is present within the sample, there are sections of the tissue with high contrast but there are also some slices of tissue with low contrast. All in all, there are samples from which extracting the tissue can be a tough task, even for expert pathologists.

In this thesis work we examined two different deep learning architectures, namely *UNet* and *DeepLabV3+*, and assessed their performance for a company case. The company's name is Jilab Inc, it is a Finnish company located in Tampere which offers a wide range of pathology services for customers around the globe. Jilab is looking forward to implementing a supervised learning method that allows for image segmentation and tissue extraction from WSIs. In adition to that, a desktop application was developed to annotate histopathological images which ultimately provided the data we used to train our algorithms.

In the next section, background information about the problem of image segmentation is presented, followed by Section 3 where the concept of artificial neural networks

is discussed. The models used in this thesis work are described in sections 4 and 5. Thereafter the results are presented and discussed in Section 7. Finally, conclusions are presented in Section 8.

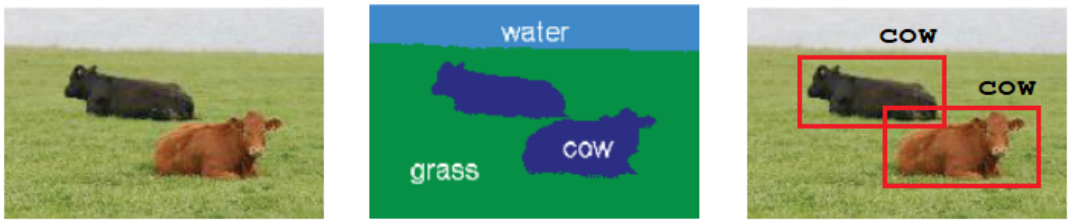
2. IMAGE SEGMENTATION

Image segmentation is a process through which objects and regions of interest are separated from a given image. Objects are often predefined classes which are further on detected and separated from the rest of the content within the image. For instance, an image segmentation task could be separating human faces from a set of images, "human face" defines the class and the process would return a set of coordinates, pixels in this case, within which the objects are located. There could be more than one class to be separated from a single image. For example one could use image segmentation to separate not only human faces but also cars, cats and dogs from the same image. For tasks of this kind, bounding boxes are often used to indicate the location of such objects i.e, human faces. However, bounding boxes are not the focus of this work as we are interested in segmenting accurate contours of objects.

There are two general approaches for identifying and extracting objects from an image, region based segmentation and edge based segmentation [4]. The former identifies all the pixels that belong to a certain object whereas the latter returns only those pixels which define the boundary of the object.

Different methods have been developed in the field of image segmentation throughout the time. There are those methods based on pattern recognition techniques which compute the physical properties of an image in order to identify objects within it. Such properties can be the colors, the intensity of the colors, the gradient and so on, which are then computed and those regions or pixels with similar properties are clustered together and thus identifying objects in the image. The success image segmentation based on pattern recognition methods depends highly in the definition of the group of properties through which objects will be identified. It is however a considerable challenge in those cases where there is a big variability from image to image.

There is a more recent approach, which has shown to be very effective in the problem of image segmentation. It is artificial neural networks (ANN) which are based on distributed nonlinear parallel processing [4]. ANNs have demonstrated to be a very



(a) Input image (b) Pixel wise segmentation (c) Bounding box segmentation

Figure 2.1 *Image segmentation example,* ***Source:*** [5]

powerful tool in image segmentation problems and it is a very active research area nowadays. The methods presented in this thesis are based on them for which this topic will be analyzed further in Section 3. An ANN is a basic mathematical model of the human brain. An ANN is composed at its root level by computational units called neurons which receive one or multiple inputs and produce an output. There is usually a large number of neurons in an ANN which are interconnected forming different architectures in order to fulfill the functionality of the network.

When dealing with image segmentation problems there are some major challenges which are to be taken into account, some of them could be a considerable concern depending on the case of study. In most cases however, it will be very likely to encounter issues related to noise and in some cases computational resources may come up short. Noise can appear as undesired objects in the image samples which may require some prepossessing steps in order to get rid of them or at least diminish their effect, but noise can also be outlier samples which are not representative but they can produce misleading results. In this work, we will deal with different kinds of artifacts such as air bubbles, dust particles, which are present in some of the samples which are often misclassifed and we aim to perform a good segmentation despite of them.

Image segmentation has shown to be effective in various applications ranging from hand text recognition, sign gestures recognition, face recognition, medical images processing, among others. It has reduced the dependency on manual procedures as human errors also decrease.a basic examples of how image segmentation works is depicted in Figure 2.1. Three different classes are defined: cow, grass and water and they are to be segmented or separated from a given input image, Figure 2.1(a) is the input image and Figure 2.1(c) is the output where each pixel in the original image is assigned one of the classes predefined and thus the objects are separated from the original image.

Image segmentation methods are expected to work ideally as good or even better than what is shown in Figure 2.1 and in most cases a large number of sample images is involved, for that reason the algorithms should allow for a considerable level of variability between samples.

3. ARTIFICIAL NEURAL NETWORKS

An ANN is a basic mathematical representation or model of the human brain. It attempts to reproduce one of the most powerful capabilities of the human brain which is learning. The human brain can learn how to do numerous kinds of tasks, such as recognizing faces, driving, painting, etc. Artificial neural networks can learn, or more specifically they can be trained in order to predict and classify with the same accuracy of that of a human brain or even better in some cases.

Computers are far better and faster than humans when solving complex calculations in a very short time, however they are not capable of learning, not at least before machine learning was first introduced. By introducing neural networks we enable computers to learn in a very similar way as we humans do and learn from previous experiences. For example, if a neural network is to classify 10 different types of animals, it is fed with several images, it attempts to classify them one by one and compensates its errors. By repeating this process over and over again, the error of the network can be minimized and consequently its accuracy is maximized.

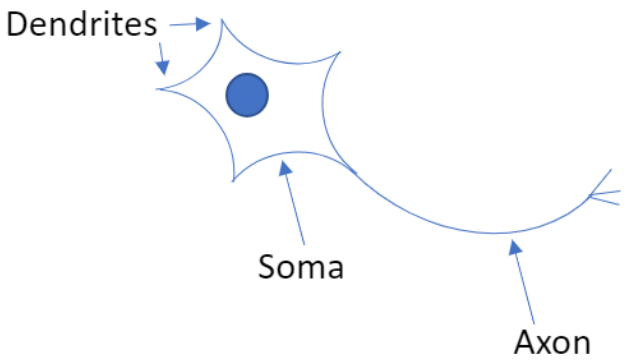
In general, the more samples that are fed to the ANN, the more experience it will gain, the more accurate it will be. Depending on how complex the task is, an ANN may need a large number of samples in order to be trained which brings a downside when using neural networks since it is sometimes very difficult to gather and handle that many samples.

An ANN consists of a large number of computational units referred to as neurons, each one of which receives one or multiple input values and produces one output. Neurons are interconnected and the information flows through such connections across the entire network.

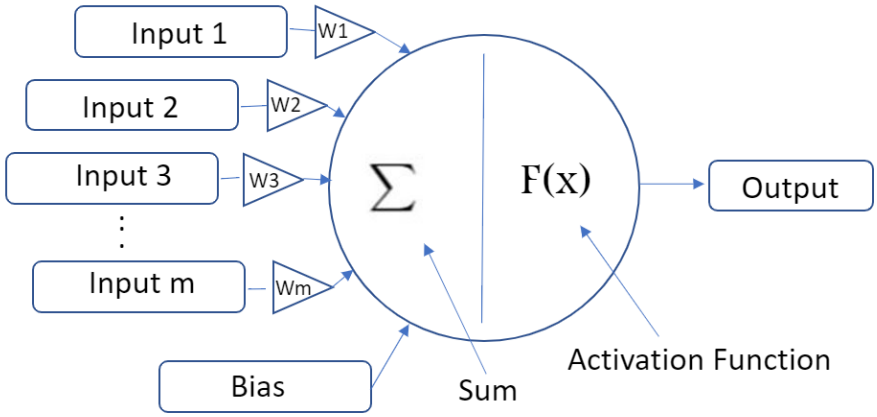
In this section ANNs will be explained more in depth, taking into account the main components that conform an ANN and how it is trained. We will also review some architectures and topologies of artificial networks in Section 3.3.

3.1 Artificial Neuron

Biological neurons are composed of dendrites, one soma and an axon. Dendrites receive the information which is later processed within the soma and then the information is passed on to the axon which is connected to other neurons [6]. Analogically an artificial neuron receives the information in the form of inputs which are then weighted and summed together by arithmetic addition and then passes the result to a transfer function also referred to as activation function [7]. Figure 3.1 shows the representation of a biological (Fig 3.1(a)) and an artificial neuron (Fig 3.1(b)).



(a) Biological neuron



(b) Artificial neuron

Figure 3.1 *Biological and artificial neuron*

Mathematically speaking, each neuron ultimately represents a function that depends on the linear combination of the inputs as shown in eq. 3.1 where $\mathbf{w}$ is the weights, $\mathbf{x}$ is the inputs, b is bias, F is the activation function and the neuron output y is given by

$$y = F(\mathbf{w}^T\mathbf{x} + b) \tag{3.1}$$

where

$$\mathbf{x} = \begin{bmatrix} x_1 & x_2 & \dots & x_n \end{bmatrix}, \tag{3.2}$$

$$\mathbf{w} = \begin{bmatrix} w_1 & w_2 & \dots & w_n \end{bmatrix}. \tag{3.3}$$

The ultimate goal is then finding those values of $\mathbf{w}$ and b for each one of the neurons in the network, in such way that the network is capable of classifying and predicting accurately. The process through which those values are found is called learning and it is by learning that an ANN can gain experience. The weights in a neural network are "learned" as the network is trained and fed with more samples. This is explained in further in Section 3.4.

3.2 Types of Activation Functions

Activation functions allow for a nonlinear behaviour of an ANN and thus it is possible to mimic the type of responses of a human brain. The most common activation functions used in ANNs are nonlinear Sigmoid and ReLU [8]. Using such activation functions allow the response of a neural network, approximate non-linear functions. However, there is a downside on using this kind of function as they can be computationally expensive [9] in contrast with linear functions. Choosing the type of activation functions to be used in the network depends heavily on the specific needs of each case. Activation functions limit the output range of each neuron [10] and ultimately the output range of the network itself.

- **Sigmoid**

  The sigmoid function is not very computationally expensive when used in neural networks trained by backpropagation [10] and it is easily differentiable [11]. Training a neural network will be explained later in this section. The output range of the sigmoid function is $(0, 1)$ for any given real value. The characteristic curve is shown in Figure 3.2. The sigmoid function is also called logistic function and it is defined as in eq. 3.4 [12].

$$sgm(s) = \frac{1}{1 + e^{-s}}. \tag{3.4}$$

- **Hyperbolic Tangent Function**

  The hyperbolic tangent function is also a nonlinear function and ANNs using this function or functions of the same kind, exhibit nonlinear behaviour when there are enough neurons [13]. This function is one of the most commonly used among nonlinear activation functions [14].

  The hyperbolic tangent function, as claimed by M. Bharathi and M. M. Rekha [15], presents a faster response compared to the sigmoid activation function, therefore it is of special interest when the application requires such behaviour. The output range of this function is $(-1, 1)$ for any real value x and it is mathematically defined as the ratio between the hyperbolic sine and the hyperbolic cosine functions (eq. 3.5) which equates the ratio of the half-difference and half-sum of two exponential functions [10]:

$$tanh(x) = \frac{sinh(x)}{cosh(x)} = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}. \tag{3.5}$$

  The characteristic curve of the hyperbolic tangent function is as shown in Figure 3.2.

- **Rectified Linear Unit (ReLU)**

  The rectified linear unit is an unbounded activation function which has become very popular in deep learning [16]. It is of special interest due to its numerical properties [17]. As it is demonstrated by X. Glorot et al. [18], using the rectified linear unit as activation function makes a better representation of the biological neuron and it can produce better results than the hyperbolic tangent function. It is also referred to as hinge activation defined as:

$$g(u) = max(0, u). \tag{3.6}$$

  It simply returns 0 when $u < 0$ and returns linear function with slope equals to 1 otherwise as shown in Figure 3.2.

  Using the ReLU activation function is very advantageous when dealing with the vanishing gradient problem. The vanishing gradient problem is a well known issue in the context of deep neural networks and it occurs when the gradient of the loss function approaches zero as it decreases at every layer during backpropagation in such way that the layers closer to the input do not

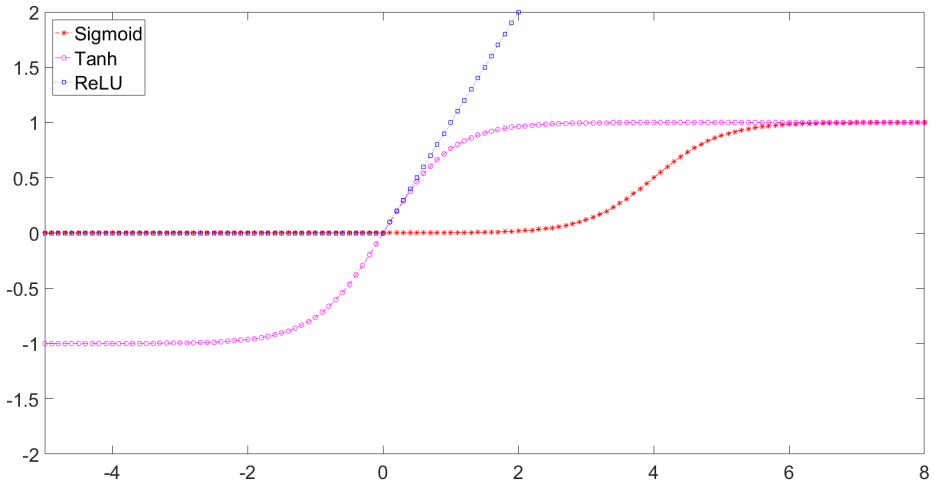


Figure 3.2 *Characteristic curves of activation functions*

learn at all. This is explained in more detail in Section 3.4.2. In the case of the rectified linear function the result is always either 0 when the input is less than 0 or 1 otherwise, and that is how the problem of vanishing problem is avoided [19].

3.3 Architecture of an Artificial Neural Network

An ANN consists of three main layers: input layer, hidden layers and output layer (Fig. 3.3).

- Input Layer: This layer receives the inputs. Such inputs are numerical values representing different types of data such as images, sound tracks and other kinds of signals.
- Hidden Layers: These layers connect the input layer to the output layer. They are responsible for extracting patterns [20]. Hidden layers are connected after one another which means that each layer will extract features based on the previous layer. There can be several hidden layers depending on the application.
- Output Layer: The output layer produces human readable result in contrast with the hidden layers where the output is often not straightforwardly understandable. The output layer is connected to the previous hidden layers and it returns the final prediction and/or classification.

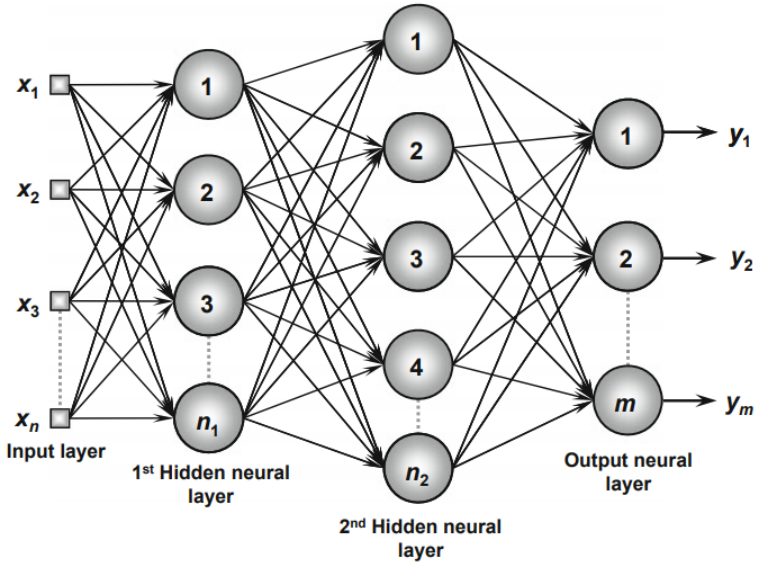


Figure 3.3 *General Architecture of an ANN,* ***Source:*** *[20]*

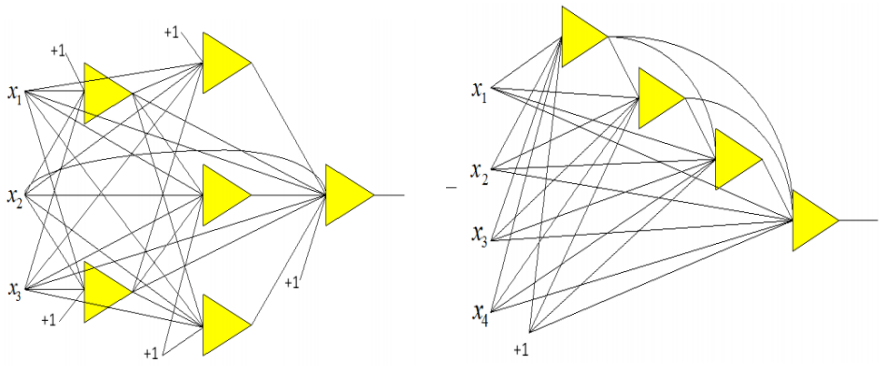
Figure 3.3 depicts an ANN composed by 2 hidden layers, but there can be many of them. Neural networks with multiple hidden layers are usually known as deep neural networks. Deep learning takes advantage of the powerful computational capabilities developed in the last couple of decades in order to process neural networks with several hidden layers.

The figure also shows that all the neurons from each layer are connected to each and everyone of the neurons of the next layer, however that is not always the case and there are scenarios where the layers are not fully connected. Using non-fully connected, also referred to as sparsely-connected neural networks, can bring advantages including improvement in accuracy performance as well as a decrease of the hardware energy consumption, which is usually a consequence of the high complexity of fully connected networks [21].

A. Ardakani et al.[21] proposed an architecture based on a sparsely-connected neural network and achieved results that show up to 84% reduction in energy consumption and up to 90% less in memory usage compared to fully connected neural networks.

Networks where the information flows in one direction, are known as feedforward networks. Feedforward networks can be of the type multilayer perceptron (MLP), like the one shown in Figure 3.3, or they can be bridged multilayer perceptron (BMLP) [22]. BMLP topology allows connection across layers (Fig. 3.4), in contrast with MLP networks where layers are connected one after another. As a special case of BMLP there are fully connected cascade networks FCCN (Fig.3.4(b)) where each

one of the layers consists of one neuron only [23].



(a) Bridged Multilayer Perceptron (b) Fully Connected Cascade Networks

Figure 3.4 *BMLP and FCCN,* ***Source:*** *[23]*

One of the disadvantages when using MLP topologies is the uncertainty on how many hidden layers should be used. Using to many of them will make the network learn patterns that may be irrelevant for the task, on the other hand not using enough hidden layers will cause the network to be inaccurate [24]. In FCCN architectures however, the hidden layers are added one at a time as the network learns [25], which solves the problem brought up by using MLP architectures. Each one of the layers in a FCCN is composed of one neuron, and each neuron is connected to the original inputs and to the layers previously added. They are also connected to a bias input.

There are other networks where information flows in two directions, they are bidirectional networks. In bidirectional networks it is possible for instance, to connect the output layer with their inputs. Such network is referred to as recurrent neural network (RNN) [26]. RNNs depend not only on the original input but also on the network's previous results. RNNs are widely used for example, in adaptive filtering where given a stationary random process the network is capable of predicting future values [27]. Some other applications where RNNs have been used are motion detection, music synthesis and financial forecasting [28]

The possibility of connecting neurons and layers in various ways allow for different architecture types. Certain features and patterns can be detected easier using a type of architecture that suits best. For instance, there are architectures which can be very effective for recognizing sound patterns, while some other architectures are more suitable when addressing problems related to image segmentation.

3.4 Learning Process

The learning process is that through which an ANN gains experience. It allows the network to self-adjust its parameters, also called weights, to compensate its output error. In order to do so, there are three general steps performed during the learning process. First the network is presented pairs of input and output samples, input samples are those connected to the input layer of the network, and the output samples represent the desired output for those specific input samples. Second, the network processes the input samples throughout its layers and computes the error. Third, the network adjusts its parameters taking into account the last error obtained. Those three steps are iterated over and over until the error is minimized and converges if possible [29].

There are two major learning methods: supervised learning and unsupervised learning. Supervised learning comprises those algorithms in which the desired output for a given set of input samples is known beforehand. The algorithm receives training samples and their corresponding correct classification referred to as ground truth. For instance, if the task of a network is to classify single handwritten letters, the learning algorithm receives images (with handwritten letters) as input and it also receives their corresponding classes eg. "a" , "b", "z"... The algorithm then passes the training samples through the layers of the network, compares the result at the output layer with the ground truth and computes the error. Supervised learning is also known as "Learning with a Teacher" [30] which brings up a clear analogy of having a teacher who knows the correct answers and corrects the student until it learns how to complete the task successfully.

In unsupervised learning in contrast, there is no teacher, in fact it is also referred to as "Learning without a teacher". There is lack of ground truth and the data is unlabeled. The task of a neural network based on unsupervised learning is finding a hidden and previously unknown structure in the data [31]. Unsupervised learning is a powerful tool in cases when labeled data is not available in large data sets and consequently it would take a long time to annotate or label all the samples. Some of the applications of unsupervised learning and neural networks are in the areas of computer vision, natural language processing, speech recognition, optimal control and networking [32].

The methods used in this thesis work are based on supervised learning methods. As it was mentioned previously, when training supervised learning based networks, one must provide the input samples and the ground truth which is the desired output for each one of the input samples. Training data as defined by J. Larsen [33] is:

$$\mathcal{T} = \{\mathbf{x}(k), y(k)\}_{k=1}^{N_{train}}, \tag{3.7}$$

where k is the $k-th$ training sample, $\mathbf{x}(k)$ represents the input samples, $y(k)$ is the desired output (ground truth) and N_{train} is the number of training samples.

3.4.1 Calculating the Error and the Loss

Once the input samples are passed through the neural network all the way to the output layer one can obtain the network's output, let us name it $\hat{y}(k)$. The error then is computed by calculating the difference between the ground truth $y(k)$ and $\hat{y}(k)$, for values of k ranging from 1 to N_{train}. The loss can be calculated by definition with any mathematical function which returns a value that is proportional to the difference between $y(k)$ and $\hat{y}(k)$. It is also called cost function. One of the most common loss functions is the Mean Square Error (MSE) defined as:

$$\begin{aligned} E(\mathbf{w}) &= \frac{1}{N_{train}} \sum_{k=1}^{N_{train}} (y(k) - \hat{y}(k))^2 \\ &= \frac{1}{N_{train}} \sum_{k=1}^{N_{train}} (y(k) - f(\mathbf{x}(k), \mathbf{w}))^2, \end{aligned} \tag{3.8}$$

where $\mathbf{w}$ represents the weights of the ANN and $\hat{y}(k)$ is a function that depends on the input samples $\mathbf{x}(k)$ and the weights $\mathbf{w}$ as in eq. 3.9

$$\hat{y}(k) = f(\mathbf{x}(k), \mathbf{w}). \tag{3.9}$$

However, there are other loss functions that have shown to be equally useful and even more efficient for certain tasks. G. Nasr et al. [34] for example, used ANNs to forecast the demand of gasoline from Lebanon during a seven year period between January 1993 to December 1999. During the training of network they used the cross entropy loss function (eq. 3.10) which proved to be faster and it also improved the overall performance of the network compared to the MSE loss function.

$$
\begin{aligned}
E(\mathbf{w}) &= \frac{1}{N_{train}} \sum_{k=1}^{N_{train}} [\hat{y}(k) \ln y(k) + (1 - \hat{y}(k)) \ln (1 - y(k))] \\
&= \frac{1}{N_{train}} \sum_{k=1}^{N_{train}} [f(\mathbf{x}(k), \mathbf{w}) \ln y(k) + (1 - f(\mathbf{x}(k), \mathbf{w}) \ln (1 - y(k))].
\end{aligned} \tag{3.10}
$$

The aim is to minimize the loss function such that it reaches the minimum value possible. The loss function is continuous and differentiable with respect to the weights $\mathbf{w}$. In other words, one must find those optimum values of $\mathbf{w}$ such that $E(\mathbf{w})$ is minimized and the accuracy of the network is maximized. The loss $E(\mathbf{w})$ can contain many local minima and there are no practical methods that guarantee a global minimum [33]. The gradient of the loss function with respect to the weights $\mathbf{w}$ returns the variation of $E(\mathbf{w})$ with respect to $\mathbf{w}$. Therefore when the values of $\mathbf{w}$ are optimal the value of the gradient of the loss function approaches zero. Let $\hat{\mathbf{w}}$ be the optimal values of $\mathbf{w}$, then:

$$
\nabla E(\hat{\mathbf{w}}) = \left.\frac{\partial E(\mathbf{w})}{\partial \mathbf{w}}\right|_{\mathbf{w}=\hat{\mathbf{w}}} = \left[\left.\frac{\partial E(\mathbf{w})}{\partial w_1}\right|_{\mathbf{w}=\hat{\mathbf{w}}}, ..., \left.\frac{\partial E(\mathbf{w})}{\partial w_m}\right|_{\mathbf{w}=\hat{\mathbf{w}}}\right]^T = 0, \tag{3.11}
$$

where m is the number of weights.

3.4.2 Backpropagation

Backpropagation uses the gradient descend in the learning algorithm [35]. ANNs learn through an iterative process in which the error is computed on every training sample and the weights are updated accordingly. Initially the weights $\mathbf{w}$ are initialized with zeroes or pre-initialized values other than zero. In general the weights are updated in accordance with the direction where the loss function has the steepest descend. This method known as steepest descend was proposed by Cauchy in 1847 [36] and it shows that the update of $\mathbf{w}$ is as follows:

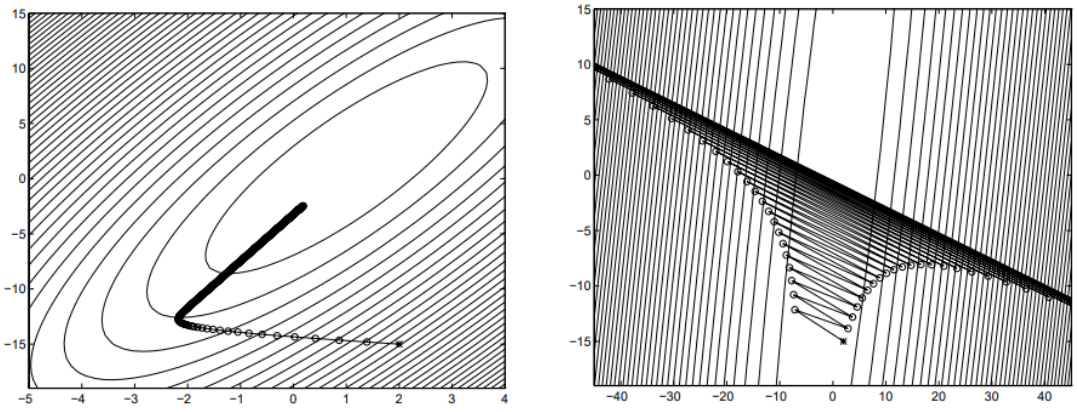
$$
\mathbf{w}_{j+1} = \mathbf{w}_j - \eta \nabla E(\mathbf{w}_j), \tag{3.12}
$$

where $\mathbf{w}_j$ corresponds to the values of the previous iteration, η is the step size and $\nabla E(\mathbf{w}_j)$ is the gradient of the loss function at $\mathbf{w}_j$.

If given a proper step size η, the loss should decrease such that:

$$E(\mathbf{w}_{j+1}) < E(\mathbf{w}_j). \tag{3.13}$$

While eq. 3.12 guarantees an update towards the direction of the steepest descend it does not necesarly guarantee eq. 3.13 as it depends on the value of step size η. It turns out choosing a proper value of η is pivotal and it is not trivial. A value of η that is too small means that it will take too many iterations for the algorithm to minimize $E(\mathbf{w})$, and a value of η that is too large will prevent the algorithm from converging as shown in Figure 3.5 where the loss function is a quadratic function with two weights and its result is plotted on every iteration step. In Figure 3.5(a) for instance, the step size is too small and the loss function converges but it takes many steps to find the valley of the function. Whereas in Figure 3.5(b) the step size is too large and therefore the loss function diverges.



(a) Small Step Size

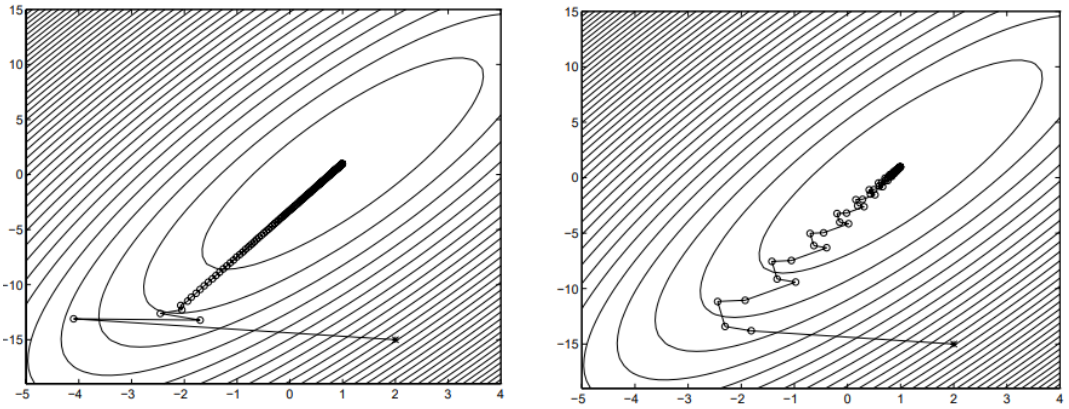
(b) Large Step Size

Figure 3.5 *Step Size,* ***Source:*** *[33]*

Overcoming this challenge leads to a solution in which the step-size is not fixed as in eq. 3.12, instead, it adjusts the step-size as it iterates (eq. 3.14), that is an adaptive step size. That way, it is reasonable to start with a rather large step-size and then make it smaller as the algorithm approximates a minimum.

$$\mathbf{w}_{j+1} = \mathbf{w}_j + \eta_j \Delta \mathbf{w}_j. \tag{3.14}$$

The difference between a fixed step size and an adaptive step size is shown in Figure 3.6. Using an adaptive step size optimizes the number of iterations to make the loss function converge faster (Fig.3.6(b)) as compared with a fixed step size where it takes many more iterations to minimize the loss function (Fig.3.6(a)).



(a) Fixed Step Size (b) Adaptive Step Size

Figure 3.6 *Fixed Step Size vs. Adaptive Step Size,* ***Source:*** *[33]*

Different methods to find an optimal value of η_j have been proposed such as the Cauchy step-size or BB step-size. Dai and Yuan also proposed an alternate minimization gradient method which improves Cauchy's slow converge [36]. It is achieved by minimizing not only the loss function but also the norm of the gradient alternately.

Since the steepest descend is an approximation method, one should decide when to stop iterating. There are different criteria in which one could base on in order to stop iterating. One simple criterion is for example to stop if the change in the loss function from one iteration to another is small enough [33]. This criterion is:

$$\mid E(\mathbf{w}_j) - E(\mathbf{w}_{j+1}) \mid < \tau_{cost}. \tag{3.15}$$

Eq. 3.15 however, does not guarantee that the algorithm is close to a minimum which is determined by 3.11. There is another criterion which deals with that issue [33], it is:

$$|| \nabla E(\mathbf{w}_j) ||_2 < \tau_{grad}. \tag{3.16}$$

This criterion is based not on the loss function as in eq. 3.15 but on its gradient. According to eq. 3.16 the iteration should stop when the euclidean length ($|| \nabla E(\mathbf{w}_j) ||_2$) is smaller than τ_{grad} which is a small constant.

From eq. 3.9 we see that the output of a neural network can be represented as function whose parameters are the input samples and the weights. A general representation of the system of a network such as the one in Figure. 3.3 is:

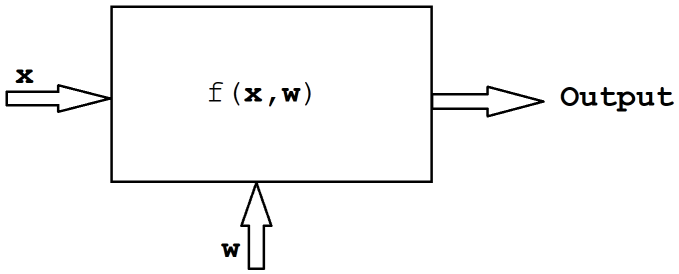


Figure 3.7 *ANN - Black Box*

where $f(\mathbf{x}, \mathbf{w})$ is composed by as many layers there are, so it can be split as follows:

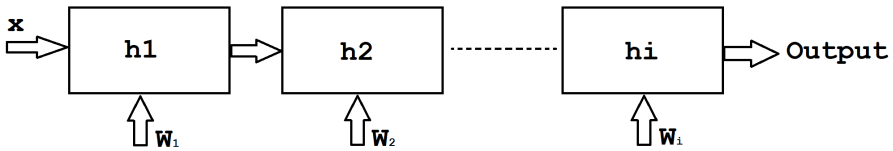


Figure 3.8 *ANN Composition*

where h_1 represents the first layer, h_2 represents the second layer, so on and so forth until the $i-th$ layer. Since $f(\mathbf{x}, \mathbf{w})$ is really a composed function it can be rather difficult to calculate the gradient of the loss function (eq. 3.8) based on it. However, as it was demonstrated by J. Larsen [33], the gradient of the loss function can be calculated efficiently using backpropagation of errors.

Backpropagation calculates the error at the last layer and then propagates it back to the previous layer in order to update the weights $\mathbf{w}$. This method calculates the gradient of the loss function with respect to the weights of each one of the layers and update them accordingly. It can be shown for an ANN that if w_{ij} is the weight connecting the output of the neuron j from the layer *l-1*, and the neuron i from the layer l then:

$$\frac{\partial E(\mathbf{w})}{\partial w_{ij}} = -\frac{1}{N_{train}} \sum_{k=1}^{N_{train}} \delta_i x_j, \tag{3.17}$$

where δ_i is the error calculated for the neuron i to which the weight is connected, and x_j is the result of the linear activation of the neuron j from which the weight is connected [33]. This way, while the error produced at the hidden layers is unknown, the error at the last layer can be easily computed in terms of the network output error $y(k) - \hat{y}(k)$ and the derivative of its activation function as follows:

$$\delta_i(k) = (y(k) - \hat{y}(k))\psi_i'(u_i(k)), \tag{3.18}$$

where $\psi_i(u_i(k))$ is the activation function of the neuron i, and $\psi_i'(u_i(k))$ is its derivative. Then, $u_i(k)$ is defined as:

$$u_i(k) = \sum_{j=1}^{n_H} w_{ij} h_j(k) + w_{i0}, \tag{3.19}$$

where nH is the number of connections from the previous layer, and $h_j(k)$ is the output of the $j-th$ neuron from the previous layer The error for the hidden layers is computed as follows:

$$\delta_j(k) = \delta_i(k) w_{ij} \psi_j'(u_j(k)). \tag{3.20}$$

From there on, it is possible to calculate the error at previous layers straightforwardly following eq. 3.17. The backpropagation algorithm using the gradient descend consists of the following steps:

1. Initialize weights $\mathbf{w}$.
2. Pass every training sample through the network and calculate $\hat{y}(k)$ for each, and the output for each layer.

3. Compute the errors as $error(k) = y(k) - \hat{y}(k)$.
4. Compute the gradients of the loss function using backpropagation (eq. 3.17).
5. Update the weights for each layer using eq. 3.12.
6. Iterate from step 2 until the stopping criteria (eq. 3.16) is fulfilled.

On a neural network with multiple layers however, there appears a problem when using backpropagation. As the error propagates backwards, the gradients with respect to the weights (eq. 3.17) tend to be smaller and smaller and approach zero. Since the update of the weights is proportional to the value of the gradient, the layers at the beginning of the network (closer to the input layer) will learn slower in comparison with the layers placed at the end which learn a lot faster. This is known as the *vanishing gradient problem*.

The vanishing gradient problem happens as a consequence of the derivative terms which are usually less than 1. As it is propagated with successive multiplications, the gradient tends to zero. Also, if the gradient of the activation function returns values larger than 1 then it will cause the opposite effect and the gradient will diverge [37].

Therefore, instead of using traditional activation functions such as *sigmoid* or *tanh* previously studied in Section 3.2, using *ReLU* as activation function mitigates with the problem of the vanishing gradient and it is less computationally expensive to compute [38]. It is explained by the nature of the *ReLU* function, as its gradient is always 1 for values larger than zero and zero otherwise. There are other activation functions which also alleviate the problem of the vanishing gradient such as the exponential linear unit (ELU), the parametric exponential linear unit (PELU) or the scaled exponential linear unit (SELU) [37].

3.5 Convolutional Neural Networks

A convolutional neural network is a specific type of ANN. They are a very powerful tool when it comes to image classification and object recognition tasks. In fact, CNNs have demonstrated to be so effective that sometimes they have outperformed humans in various tasks [39]. C. Szegedy et al. [40] pushed the state of the art for classification and detection when they put together a CNN architecture named "Inception" with which they achieved outstanding results using much less parameters in the network, therefore less consumption of computational resources and yet improving the accuracy.

Aiming at reducing the number of parameters in a CNN is of special interest as it deals with one of the most common limitations of CNNs which is the high demand of computational resources. Convolutional neural networks have proved to be effective for image classification tasks and machine learning problems [41], however computational resources may come up short when dealing with high resolution images, which have made the use GPUs very popular in this field. GPUs paired with optimized implementations of 2D convolutions facilitates the training of deep (several hidden layers) CNNs [42]. Despite of the advantages of using GPUs, this issue is far from being resolved and convolutional networks are often limited by the amount of memory available as well as the speed at which GPUs can operate, consequently causing a long time required for training and classification [42].

CNNs are used to extract patterns and classify images, or image-like data. They also consist of neurons which self-adjust during the learning phase, and just like in an ANN, neurons receive an input, they perform an operation over the input and pass the result to an activation function [43]. However, in comparison with a traditional ANN, a CNN uses spatial information between the pixels of an image by using discrete convolution [44]. Each layer of a CNN will produce an array of feature maps, which are specific features extracted at all locations of the input [45]. For instance, a feature map may represent the sharp edges of an image, another feature map may indicate locations with high contrasts, so on and so forth. Feature maps produced by one layer are then passed on to the next layer which will produce another set of feature maps based on the input.

3.5.1 Discrete convolution

CNNs extract features from images by using discrete convolution between the input and different kernels. In Figure 3.9 for example, two different features were extracted from Figure 3.9(a). The first feature is the result of the convolution between the original image and a kernel which highlights the horizontal edges of the picture (Fig. 3.9(b)). Likewise, a second kernel highlights the vertical edges from the input image when it is convolution-ed with the original picture (Fig. 3.9(c)).

In a convolutional neural network, the first layers (the ones closer to the input layer) usually extract basic patterns and features such as the ones in Figure 3.9. However, further layers extract much more complicated features from the images. As a matter of fact, when going through the results of each one of the layers, the features extracted by a deep convolutional neural network (DCNN) may become not at all unidentifiable by humans, in other words, it may be very hard to explain

why certain network classifies a certain image the way it does just by looking at the features extracted across its layers.

Using a proper number of layers is also very relevant. If the network is too large, it may be extracting features which are actually not relevant for the task, but if there are not enough layers, the network may be missing some features which can be important for the task as those missing features could improve the performance of the network.

(a) Original Image (b) Horizontal edges (c) Vertical edges

Figure 3.9 *Feature extraction,* ***Source:*** [46]

To better understand discrete convolution, let us explain first the mathematical definition. Let f be a function of two variables $f(x, y)$ and g be a function of two variables $g(x, y)$. Then the convolution of f and g is $f * g$:

$$(f * g)(x, y) = \sum_{u=-\infty}^{+\infty} \sum_{v=-\infty}^{+\infty} f(u, v)g(x - u, y - v). \tag{3.21}$$

We can extrapolate from here to define the convolution of matrices since they can also be expressed as a function of x and y as: $A_{xy} = f(x, y)$, where A is a matrix of dimensions $n \times m$ and $B_{xy} = g(x, y)$, where B is a matrix of size $k \times l$. The convolution $f * g$ then produces a matrix C of size $(n + k) \times (m + l)$:

$$c_{xy} = \sum_{u} \sum_{v} a_{uv} b_{x-u,y-v}, \tag{3.22}$$

where u and v hold for all subscripts of a_{uv} and $b_{x-u,y-v}$ [47]. If A is a black and

white image and B is a convolutional kernel, then eq. 3.22 is a typical convolutional operation in a CNN.

According to eq. 3.22 the kernel B is flipped in both axis due to the negative subscripts $-u$ and $-v$. However, flipping the kernels is computationally expensive since it takes longer to feedforward the CNN and backpropagate the error [48]. For that reason, it is usually preferred to use cross-correlation instead of convolution. Cross-correlation is almost the same as convolution except that it does not flip the kernels, therefore same results can be achieved in considerably less time. Cross-correlation is defined as follows:

$$c_{xy} = \sum_u \sum_v a_{uv} b_{x+u,y+v}. \tag{3.23}$$

Note how b subscripts u and v are no longer negative, which indicates there is no need to flip over the kernel B.

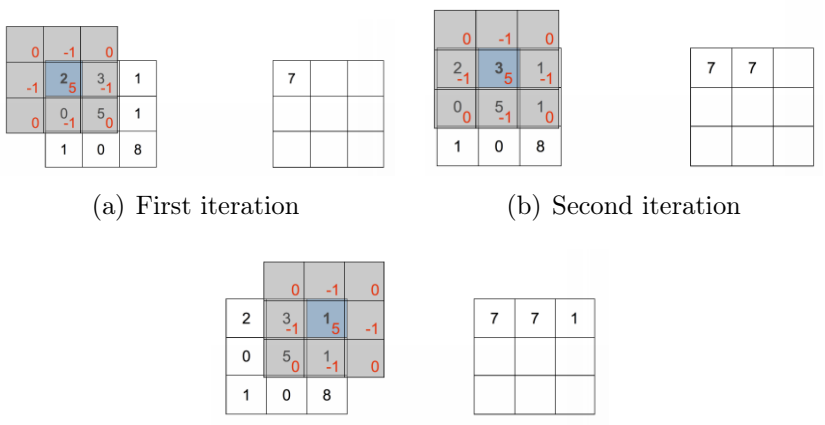
Convolution and cross-correlation operations can be thought of as a sliding window (the kernel) which maps the values of the image and forms a new one, in which certain features are extracted as in Figure 3.10. Let us suppose that our kernel B is a 3×3 matrix and our image is the matrix A of the same size, like this:

$$A = \begin{bmatrix} 2 & 3 & 1 \\ 0 & 5 & 1 \\ 1 & 0 & 8 \end{bmatrix} \qquad B = \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix}. \tag{3.24}$$

We have chosen B to be symmetric for the sake of simplicity just so eq. 3.22 and eq. 3.23 both hold.

The result of $A * B$ in this case equals the result of the cross-correlation R_{AB} as follows:

$$R_{AB} = A * B = \begin{bmatrix} 2 & 3 & 1 \\ 0 & 5 & 1 \\ 1 & 0 & 8 \end{bmatrix} * \begin{bmatrix} 0 & -1 & 0 \\ -1 & 5 & -1 \\ 0 & -1 & 0 \end{bmatrix} = \begin{bmatrix} 7 & 7 & 1 \\ -8 & 21 & -9 \\ 5 & -14 & 39 \end{bmatrix}. \tag{3.25}$$



(a) First iteration (b) Second iteration

(c) Third iteration

Figure 3.10 *Convolution and Cross-correlation,* ***Source:*** *[49]*

The values contained in matrix B are in fact some of the weights $\mathbf{w}$ of a more extend CNN. Those values are self-adjusted during the learning process. The number and size of the kernels will define the number of parameters to train in the network. If a CNN is composed by j number of kernels, and all kernels are of the same size $k \times l$, then the number of parameters which need to be trained is $j \times k \times l$. This holds for 2D images, but if the images are RGB valued, the number of parameters is three times larger.

From eq. 3.21 we can see that the kernel acts as a 2-D sliding window which maps the input sample, we can also see that it slides from $-\infty$ to $+\infty$ in both axis. However, in this case images are only defined within the range of their size and for this reason convolution from $-\infty$ to $+\infty$ is not possible unless the image is padded with zeroes in both axis, only then such convolution would be possible. Furthermore, convolutions with an infinite range are of course not possible using a computer which can only iterate a finite number of times. Convolutions has to be limited to a reasonable range so it can be computed. There are different alternatives: **a)** set the limits of the convolution such that the kernel only maps those values which are previously defined, **b)** set the limits of the convolution in such way that the result keeps the resolution and dimensions of the original input. In that case the input sample may have to be padded with zeroes. Or **c)**, set the limits of the convolution such that the result is a map of higher resolution, for which it would be necessary to pad the input image with zeroes.

There are three basic parameters which defined a convolution operation: the kernel size, the stride and the padding. The kernel size simply defines the dimensions of the convolutional kernel. The stride is the step size the kernel will move across the

input sample. That is the amount of pixels the kernel is translated horizontally and then vertically as it is convolution-ed [50]. Figure 3.11 illustrates a convolution operation between an input sample of size 5×5 and a kernel of size 3×3 with a stride set to 2. A convolution with no stride, or stride set to 1 follows the basic definition of convolution as shown in Figure 3.10. And finally the padding is the amount of rows and columns of zero values added around the input sample.

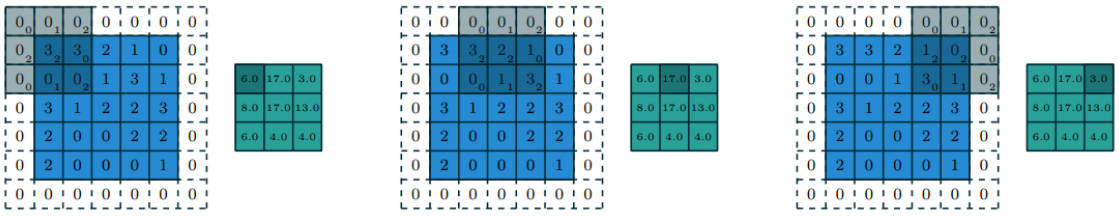


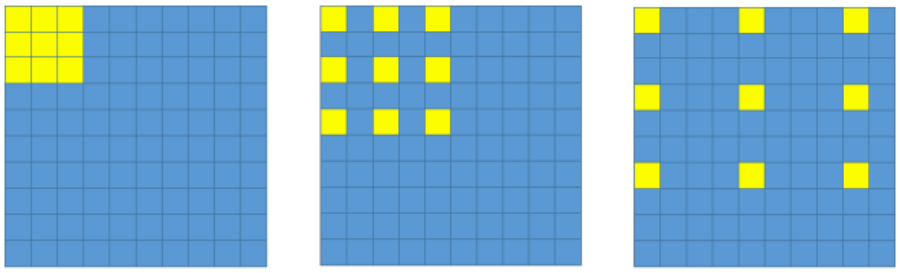
Figure 3.11 *Convolution with stride set to 2,* ***Source:*** *[50]*

As the image is passed through the layers, the convolutional kernels reduce the size of the input sample, therefore some resolution is lost after every convolution. This can become a problem specially with networks composed by many convolutional layers [50].

Let us take a look at the different types of convolutions there are. Besides normal convolution, as the ones explained so far, there is also dilated convolutions and transposed convolutions.

- **Dilated convolutions**: Dilated convolutions can be thought of as "dilated" kernels. That means adding zeroes between the elements of the kernel. That will increase the field of view of the kernel without increasing the parameters. In dilated convolutions a kernel of size $k \times k$ turns into a kernel of $k + (k - 1)(r - 1)$ where r is the dilated stride, thus it allows the extraction of multi-scale contextual information while keeping the resolution [51]. That means that every pixel at the output of the convolution is the result of mapping more pixels from the original input sample while keeping the number of parameters.

  Different convolutions with dilated kernels are shown in Figure 3.12. The squares in yellow belong to the elements of the kernel and the background squares in blue represent the input sample. While Figure 3.12(a) shows a kernel with no dilation, kernels in Figure 3.12(b) and Figure 3.12(c) have a dilation rate of 2 and 3 respectively. A 2-D dilated convolution is defined as in eq. 3.26.



(a) No dilation (b) Dilation rate 2 (c) Dilation rate 3

Figure 3.12 *Dilated Convolutions,* ***Source:*** *[51]*

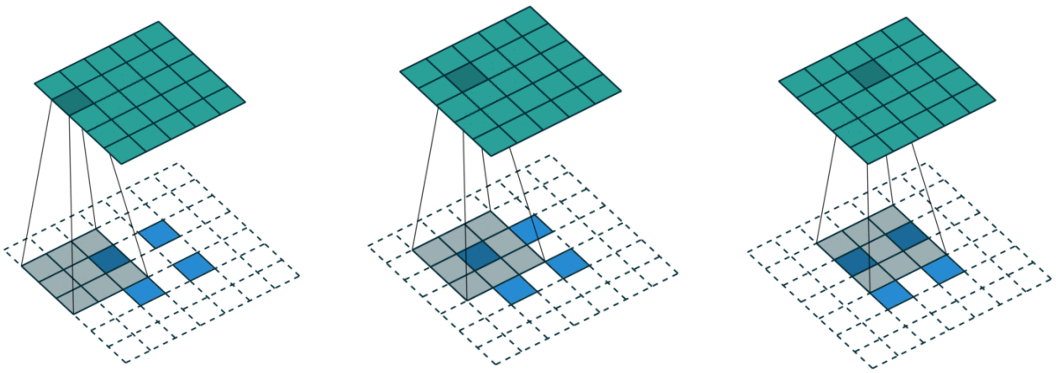
$$c_{xy} = \sum_{u} \sum_{v} a_{uv} b_{x+u\times r, y+v\times r} \tag{3.26}$$

- **Transposed convolutions**: Transposed convolutions generate feature maps with higher resolution out of the input sample. That is, for every pixel in the input sample, multiple pixels are produced at the output as a result of the transposed convolution. In other words it up-samples the input sample. It takes as parameters a dilation rate for the input feature map and the padding which is the number of columns and rows to be added around the input sample. Transposed convolutions are often referred to as inverse convolution or deconvolutions due to the fact that it is possible to recover the spatial resolution lost after performing a regular convolution. However transposed convolution is not the inverse function of convolution in the strict mathematical sense. In fact it does nothing different that normal convolution operations, except for the padding and dilation rate applied to the input feature sample.

  Transposed convolutions are very useful in tasks related to localization, semantic segmentation, super resolution, visualization, visual question answering and recognition [52]. L. Xu et al. [53] presented how transponsed convolutions can be used to deal with artifacts caused by convolution operations. They devised a solution where a separable structure is introduced for deconvolution against artifacts.

  Figure 3.13 shows how transposed convolution is performed, the squares in blue represent the original pixels from the input feature map, the input sample is dilated $\times 2$ and then zero padded. The squares in green represent the output of the transposed convolution. In this case a feature map with spatial resolution of 5×5 was obtained out of an input sample with spatial resolution

of 2×2 using a 3×3 convolutional kernel.



(a) (b) (c)

Figure 3.13 *Transposed Convolutions,* ***Source:*** *[54]*

3.5.2 Layers of a Convolutional Neural Network

There are various types of CNN architectures, however there are three types of layers which are used in most architectures: convolutional layers, pooling layers and fully-connected layers [52], (see Figure 3.14).

- **Convolutional layers**: These layers use convolution kernels in order to extract the features from the input image or feature maps as previously explained in Section 3.5.1. There may be several convolution kernels per layer depending on how many features are intended to be obtained in each. Convolutional layers also perform transposed convolutions, often referred to as de-convolutions or inverse convolutions. Feature maps produced by one layer are usually passed on to a pooling layer, or directly to the next convolutional layer in case there is no intermediate layer. That is why convolutional layers accept as input either the original image or the feature maps produced from the previous layer.

  In order to introduce the capability of the network to learn non-linear features, each one of the values in every feature map, is passed to an activation function like the ones mentioned in Section 3.2. ReLU is a rather common activation function in this context as it has shown to improve the discriminative performance [55]. Tanh is also a common alternative as activation function in CNNs [52].

- **Pooling layers**: The intermediate layers are usually pooling layers. They do not add extra parameters to the CNN since they only perform fixed transformations to the feature maps. Pooling layers reduce the spatial resolution of the input feature map and by doing so they achieve shift-invariance [52], that means that the network is more resilient towards displacements of the input images, which is in most cases a desirable behaviour in a CNN. Pooling layers also make convolutional neural networks more robust to distortions and noise commonly present in the input samples [56].

  Pooling operations replace groups of pixels in the input feature maps by a representative value. Such representative value is usually the maximum value among the group of pixels or the average value. These operations are known as max-pooling and average pooling respectively. Max pooling has proved to make the CNN converge faster as it filters in stronger invariant features and therefore the CNN is capable of dealing with disturbances more accurately as the generalization performance improves [57].

  In the context of CNN, pooling size is usually $(2, 2)$, where each of these numbers are the factors by which the input map is down-scaled in the vertical dimension and the horizontal dimension. Thus, if the pooling size is $(2, 2)$ the original spatial resolution is reduced by half in each dimension.

  J. Nagi et al. [57] achieved an important improvement in the task of hand gestures recognition by putting together a CNN and max-pooling operations (MPCNN) achieving an accuracy of 96% where the task was classifying six different gestures. H. Wu and X. Gu [58] demonstrated the utility of combining pooling layers with dropout operations, referred to as max-pooling dropout. In contrast with a fully-connected networks, non-fully-connected networks implementing dropout randomly disconnect nodes during the training phase which prevents the network from over-fitting.

  Over-fitting is a well known problem in the context of neural networks. It happens when a neural network learns how to classify correctly the training samples and in some cases it may even have an accuracy very close to 100% over the training samples, but then accuracy plummets when the model is tested with samples previously unseen by the network [59]. In that scenario the model is over-fitted to the training data. Using dropout in CNNs on supervised learning has shown to improve the performance in various tasks such as speech recognition, document classification and vision [60].

  According to H. Wu and X. Gu [58], using dropout before max-pooling layers can improve the performance of a CNN as it introduces stochasticity. Max-pooling layers select the highest value from a specific region in the input feature

map, causing that other units in the region are not taken into account at all for further processing. Average-pooling on the other hand, takes all the units into consideration by computing the average between them in a specific region of the input map, thus high activation units may be outshone by many other low activations [58]. That is why stochasticity introduced by max-pooling dropout plays a vital role in improving the CNN performance.

- **Fully-connected layers**: These layers are in charge of making high level abstractions. They are often used as the output layer of the CNN. Each one of the neurons in a fully-connected layer receives connections from all of the activation units from the previous layer, whether it is convolutional layer or a pooling layer. Let us suppose that a given model classifies hand written single digit numbers within the classes 0 to 9, where the last layer is a fully-connected layer and the previous layer outputs a set of feature maps. The last layer would then consist of ten neurons, each one of which receives connections from all the activation units coming from all of the feature maps of the preceding layer. The last layer's output is then an array of probabilities where each element describes the likeliness of the input sample of belonging to one of the classes.

  In some other architectures, fully connected layers are not used as the output layer, they are used instead as feature extraction layers placed before the output layers. Setting up and selecting the layers is tightly related to the classification task of the CNN. In some cases it can be even more accurate not to use fully connected layers for extracting high level features, and instead use features extracted from previous layers to train Random Forest or SVM algorithms [61].

All in all, a general representation of a convolutional neural network is shown in Figure 3.14 where the basic elements mentioned above are put together. The input layer receives the input images, from there on a series of hidden layers are concatenated one after another. The hidden layers compute convolutional operations with kernels and non-linear activation functions in order to produce a stack of feature maps which are then passed through a pooling operation. In Figure 3.14 max-pooling as pooling operation is used, however it can be average-pooling, max-pooling dropout or similar. Then the final feature maps are vectorized and fed to a fully connected layer which finally makes the classification. Depending on the task, the classification can be set to be binary or multi-class classification.

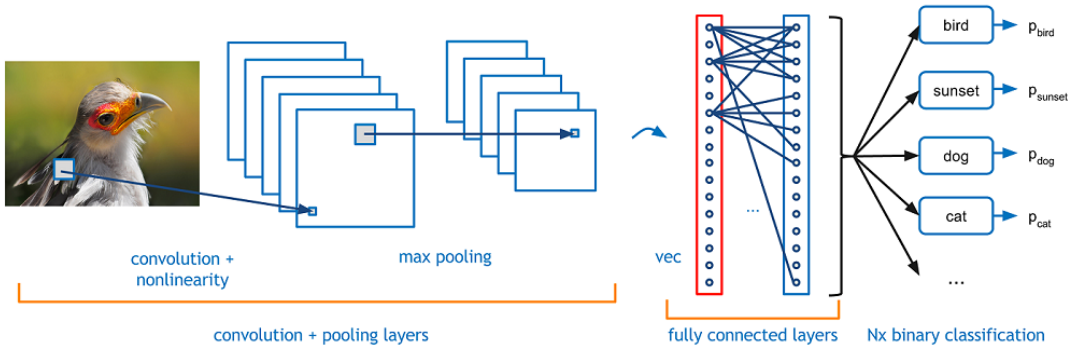


Figure 3.14 *CNN - Basic Architecture,* ***Source:*** *[62]*

3.5.3 Popular Architectures

As convolutional neural networks continue to push the boundaries in machine learning, some particular architectures have become very popular across solutions and they are often used as a baseline for new architectures. Some of the most known architectures in the context of CNN will be described in this section.

- **LeNet-5**: Y. Lecun et al. [63] compared various methods in the task of handwritten character recognition. They observed that multilayer neural networks based on the Gradient-Based Learning technique showed better results compared to all other techniques specially if they are designed to deal with samples containing high variability. That is how they proposed a rather simple architecture called *LeNet-5* which was trained for digits recognition. It consists of seven layers not including the input (Fig. 3.15).

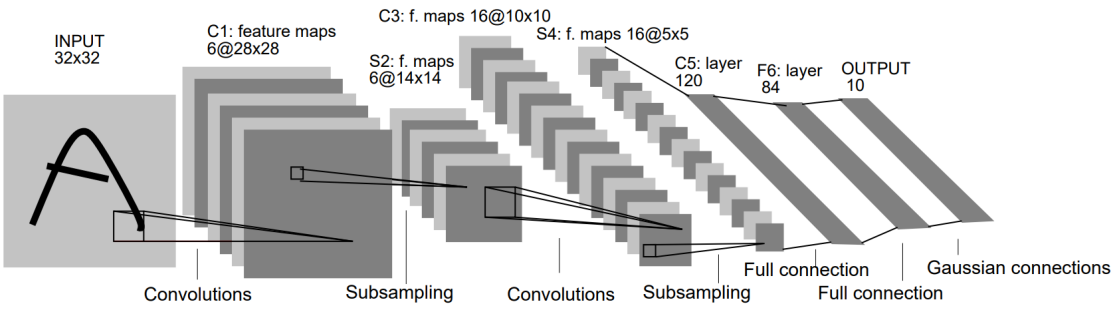


Figure 3.15 *LeNet-5 Architecture,* ***Source:*** *[63]*

The input is an image of size 32×32 and it is followed by a set of convolutional layers and subsampling layers. The first layer connected to the input, shown in the figure as *C1*, is a convolutional layer with 6 feature maps. The kernel

size used in the first layer is 5×5. The size of the feature maps is 28×28. The convolution is computed only for the valid regions of the input image. That means that the input image was not zero padded and therefore the spatial resolution of the feature maps is smaller by four pixels in each dimension. The total number of trainable parameters in the first layer is then *(kernel size + bias input)* $\times$ *number of feature maps* $= (5 \times 5 + 1) \times 6 = 156$.

The subsequent layer *S2* sub-samples the feature maps and reduces the spatial resolution by half. This operation is done by adding the activations of non-overlapping adjacent regions of size 4×4 in the feature maps, then multiplied by a trainable parameter and added to a bias that is also trainable. The result is then passed to a sigmoidal activation function. The number of parameters added in this layer is 12.

Layer *C3* is again a convolutional layer. It maps the feature maps from *S2* into 16 new feature maps. Every feature map in *C3* is not however, connected to every feature map in *S2*. Therefore, the number of connections is reduced and it also causes that every feature map in *C3* is learnt from different feature maps of *S2*, thus breaking the symmetry in the network [63]. *C3* adds 1516 trainable parameters. Layer *S4* sub-samples the feature maps of *C3* performing the same operations as in layer *S2* adding 32 trainable parameters.

Layer *C5* is a convolutional layer which operates with a kernel size of 5×5. Since the size of the feature maps produced by *S4* is also 5×5, *C5* outputs 1×1 feature maps. This layer adds 48120 more trainable parameters.

The next layer *F6* is a fully-connected layer and it consists of 84 units as it was designed by Y. Lecun et al. [63] to represent a stylized image of the corresponding output class. It adds 10164 trainable parameters to the network. And finally the output layer predicts the class, it is fully-connected to *F6* and it consists of one Euclidean Radial Basis Function (RBF) unit per class.

- **AlexNet**: Researchers from the University of Toronto in 2012, trained a convolutional neural network to classify 1.2 million images into 1000 different classes [64]. It consists of five convolutional layers and three fully connected layers. One more layer at the end of the network makes the prediction into the different classes using softmax. Using softmax to make the prediction guarantees that all the values sum up to 1 so the output of the network represents a normalized probability distribution. AlexNet architecture is shown in Figure 3.16.

  The total number of trainable parameters in AlexNet architecture is 60 million. This architecture was used for the ImageNet LSVRC-2010 competition

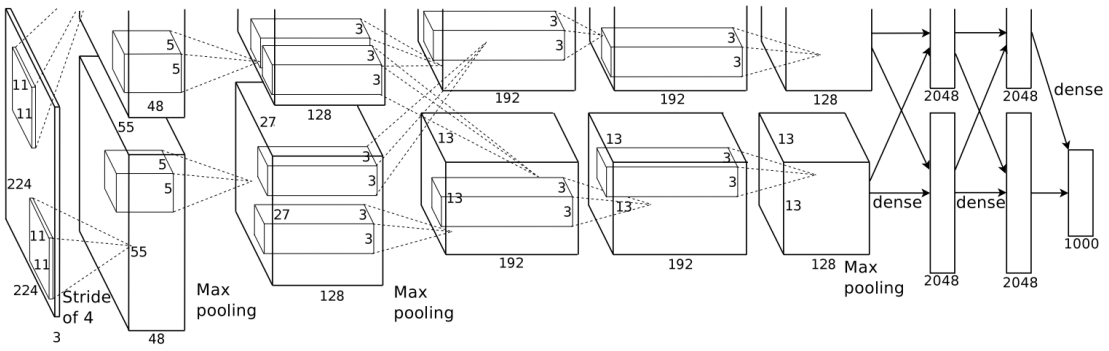


Figure 3.16 *AlexNet Architecture,* ***Source:*** *[63]*

obtaining the first place in the contest. Some of the novel features of the AlexNet are: **a**) Using ReLU as activation function in the hidden layers. Such feature makes the network learn faster than networks using saturated non-linear activation functions such as *tanh*. **b**) Training on multiple GPUs. Two GPUss were used to train the network putting half of the neurons in each. **c**) Local response normalization. AlexNet implements normalization after applying ReLU for some of the layers. This feature reduced the error rate of the network in the ImageNet LSVRC-2010 competition. **d**) Overlapping pooling. Pooling layers do not usually overlap, but in this network using such kind of pooling avoids the network from overfitting and also reduced the network's error rate compared to non-overlapping pooling layers.

- **GoogleLeNet -** ***"Inception"***: This architecture consists of a model in which optimal stages of the network are approximated by readily dense components [40]. This network contains repetitive modules stacked one after another, with max-pooling layers in some of the layers in order to reduce the spatial resolution by half. The module used at the core of this architecture is shown in Figure 3.17

  where 1×1 convolution modules in yellow are used to reduce the dimension. This architecture optimizes the use of computational resources by using the dimensionality reduction characteristic aspect and consequently allows increasing the number of units while keeping the computational complexity from growing rapidly. Another important aspect of this network is the concatenation filters in which convolutions from multiple scales are combined together and that allows the network to process features at various scales simultaneously. Inception achieved a big leap quality wise by using DCNNs with a subtle increase in the computational requirements as compared with smaller architectures.

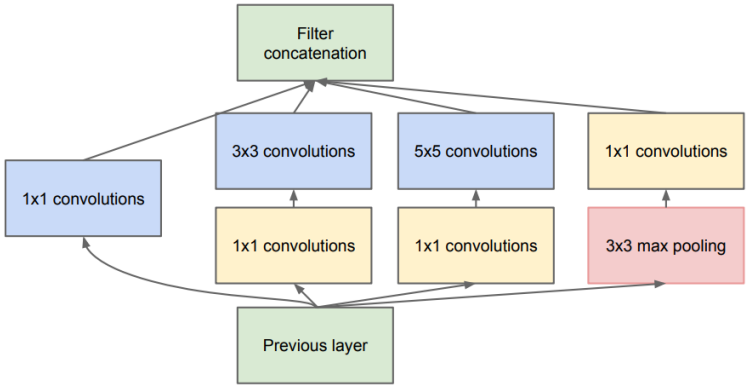


Figure 3.17 *Inception Module,* ***Source:*** *[40]*

The name *Inception* comes from the capability of this network to go deeper and deeper by simply adding pre-made blocks such as the one shown in Figure 3.17. Therefore, it allows to get higher level features straightforwardly. *GooleLeNet* (Fig. 3.18) is a special kind of an *Inception* architecture which consists of 22 layers. In order to deal with the vanishing gradient problem, authors of *GoogleLeNet* added auxiliary classifiers connected to the intermediate layers and added their loss to the the total loss of the network during training. The auxiliary networks are composed of an average pooling layer, followed by a 1×1 convolutional layer, a fully connected layer, a 70% ratio dropout layer and a classifier layer using softmax.

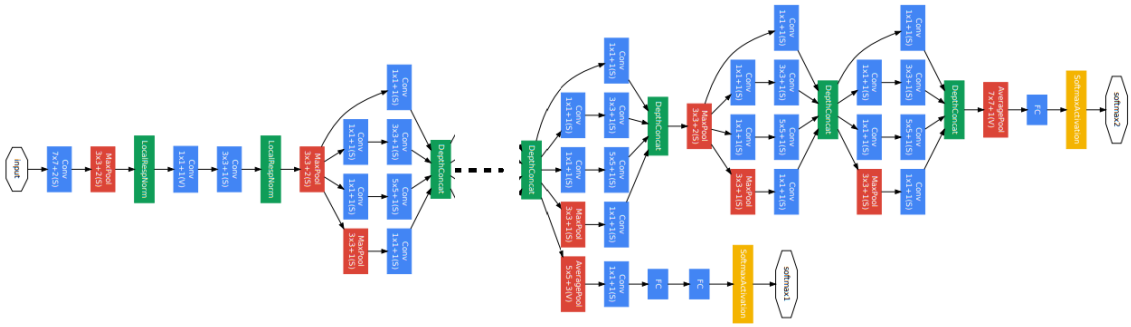
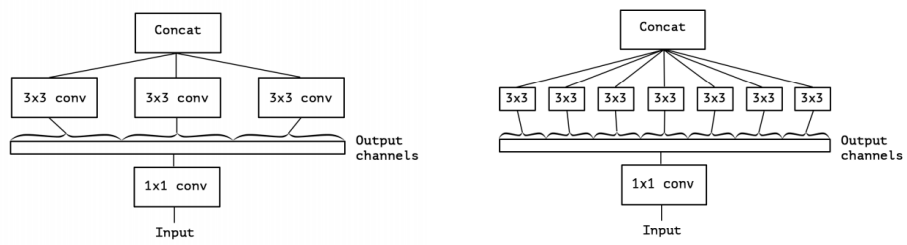


Figure 3.18 *GoogleLeNet Network,* ***Source:*** *[40]*

Figure 3.18 shows GoogleLeNet's architecture where the blocks in blue are convolutional and fully-connected layers, blocks in red are pooling layers, blocks in green are depth concatenation layers, and blocks in yellow represent softmax activations. The dotted line in the middle represents all the repetitive blocks as the one shown in Figure 3.17, which are stacked one after another.

It is very clear at this point, that the *Inception* architecture allows to extract deeper features very easily by simply designing one optimize convolutional block (Fig. 3.17) and using it over and over again depending on the desired depth of the network.

- **Xception - *"Extreme Inception"***: This architecture is a linear stack of depthwise separable convolution layers with residual connections [65]. It is based on an Inception module but taken to "extremes" as shown in Figure 3.19. The xception module is very similar to a depthwise separable convolution. A depthwise separable convolution consists of two steps, the first is called separable convolution and it is simply a spatial convolution that is performed independently over each channel, therefore, it uses as many kernels as there are channels in the input. Note that so far the number of channels has not been incremented. This is followed by step two, which is a pointwise convolution that will be used to increment the number of channels of the output. A pointwise convolution is based on spatial convolutions operated with kernels of size $1 \times 1 \times k$, where k is the number channels of the input. The number of such kernels corresponds to the number of channels desired at the output. All in all, depthwise separable convolutions are much less computationally expensive as compared to a regular spatial convolutions because the total number of multiplications computed during the convolutions is reduced.



(a) Inception module (b) Xception module

Figure 3.19 *Comparison between the Inception module and the Xception module,* ***Source:*** *[65]*

The main differences between an Inception module and an Extreme module are the order of the operations and the non-linearities after the first operation. The former refers to the fact that depthwise separable convolutions perform channel-wise spatial convolutions first and they are then followed by 1×1 convolutions, whereas in the Inception module 1×1 convolutions are performed first. The latter refers to the absence of ReLU non-linearity in depthwise separable convolutions as compared to the Inception module where ReLU non-linearity is performed after both operations, namely 1×1 convolutions and

spatial convolutions [65].

- **VGGNet**: This network was published by K. Simonyan and A. Zisserman [66] and it is aimed to classify large scale images. *VGGNet* is a convolutional neural network which extends the depth to 16 and 19 layers showing a significant improvement as compared to prior architectures.

  VGGNet consists of a stack of convolutional layers which use 3×3 filters with 1 pixel as convolutional stride and the spatial resolution is kept by padding the input of each convolutional layer. Some of those convolutional layers are followed by a 2×2 max-pooling layer with stride 2.

  Then at the end of the network, there is a stack of three fully-connected layers, the last of those three performs a classification into 1000 different classes. All of the convolutional layers use ReLU as non-linear activation function. By slight variations of this general architecture, K. Simonyan and A. Zisserman [66] built different configurations of *VGGNet* which differ mainly in the depth, starting from 11 layers (8 convolutional layers and 3 fully-connected layers) and deeper configurations which use up to 19 layers (16 convolutional layers and 3 fully-connected layers).

  Due to the depth of this network and the number of trainable parameters, it took from two to three weeks to train one model. It can of course be improved by using more powerful computer resources. For the ImageNet Large-Scale Visual Recognition Challenge (ILSVRC), the *VGG* network was trained using an optimized version of the multinominal logistic regression implementing mini-batch gradient descent with momentum and the weights were initialized by training a much shallower version of the network using only some of the first layers and the fully-connected layers at the end, and then keeping the weights obtained to initialize the network before training, thus reducing the training time.

  Despite its simplicity, VGG network demonstrates how much better the results can be just by going deeper and adding more convolutional layers for large scale image classification. Figure 3.20 depicts the basic setup of *VGGNet*.

  The main difference between *VGGNet* and other deep networks used for large scale image classification, is the use of convolutional filters with a rather small receptive field of 3×3 across the whole network. This network achieved the state of the art on the ILSVRC and also demonstrated to be very accurate for various image recognition data sets [66].

- **ResNet**: As computer resources become more powerful, neural networks have explored going deeper and deeper due to the benefits of deep neural networks

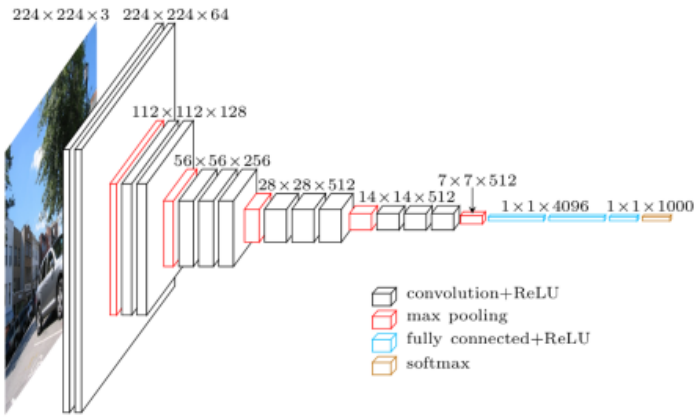


Figure 3.20 *VGG Network,* ***Source:*** *[67]*

such as the VGGNet. Although the vanishing and exploding gradient problem is a latent hazard in deep networks, it has been dealt with the use of normalization layers which allow the network to converge to a final solution.

However, there is still the problem of degradation specially when training deep neural networks. The problem of degradation occurs when the accuracy reaches a maximum value and then it starts to degrade at a high rate and the error increases as more layers are added to the model [68].

The degradation problem was addressed in 2015 by a group of researchers of Microsoft who devised what is known as a *Deep Residual Leaning Network*, referred to as *ResNet* [68]. This network won the first place in the ILSVRC competition in 2015 and also obtained the first place in various recognition task competitions such as the ImageNet localization, ImageNet detection, COCO segmentation and COCO detection.

The hypothesis made by K. He et al. [68] is that it is easier to train a deep network to fit a residual mapping with reference to the input of the previous stacked layers, rather than training the network to fit the original mapping with no reference.

That is, if the original mapping of a stack of layers is $H(x)$, the *RestNet* architecture makes the stack of layers fit another mapping $F(x)$ which is equal to $H(x) - x$. Thus the mapping is recast into $F(x) + x$ as shown in Figure 3.21.

The reference to the input x is achieved through shortcut connections and since the values of x are not modified shortcut connections do not add extra parameters to the network. The modules shown in Figure 3.21 constitutes the building block of the *ResNet* which is stacked multiple times.

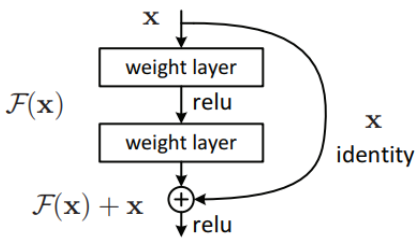


Figure 3.21 *ResNet building block,* ***Source:*** *[68]*

The architecture presented by K. He et al. [68] consists of up to 152 layers. There are two pooling layers, one fully-connected layer which performs the classification into 1000 different classes, and the rest of the network consists of stacked building blocks as the one in Figure 3.21 where each weight layer is a convolutional layer with a receptive field of 3×3. A 34 layer version of the *ResNet* network is shown in Figure 3.22.

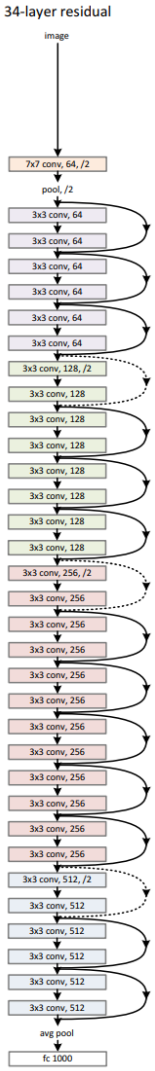


Figure 3.22 *34 Layer ResNet,* ***Source:*** *[68]*

4. U-NET ARCHITECTURE

Medical image processing has always been a very active field as new methods and tools are constantly opening new possibilities for improving the way medical images are processed and the diversification of the types of analysis available. Neural networks is certainly one of the many tools used in medical image processing and they have achieved outstanding results. In the following two sections two different neural network architectures will be explained. These architectures will be then tested in order to evaluate their performance on our own data.

Ronneberger et al. [69] proposed the U-Net architecture and won the ISBI cell tracking challenge in 2015 outperforming prior methods. This architecture consists of two parts: a contracting path and an expansive path.

4.1 Contracting Path

The contracting path is composed by convolutional layers and max-pooling operations. As a result, the contracting path will reduce the dimensionality of the input image and it will be able to classify the image as a whole and localize high order features due to the max-pooling operations. However, localization is lost in the contracting path. In order to be able to perform localization within an image it is necessary to do pixel by pixel classification. Hence, the output of the network should keep the original dimensionality. That is where the expansive path comes into play.

The contracting path consists of convolutional layers concatenated one after another with max pooling operations in between, just like a typical convulational network (see Section 3.5.2). It can be described in terms of a basic block which is repeated all the way to the end of the contracting path. Such block is composed of two convolutional layers with a receptive field of 3×3. The convolutional layers use ReLU as activation function and finally they are followed by a 2×2 max-pooling operation. The number of channels is doubled block after block, so if there are 64 channels in the first block and there are 4 subsequent blocks in the contracting path the total number of channels at the end of the contracting path is 1024.

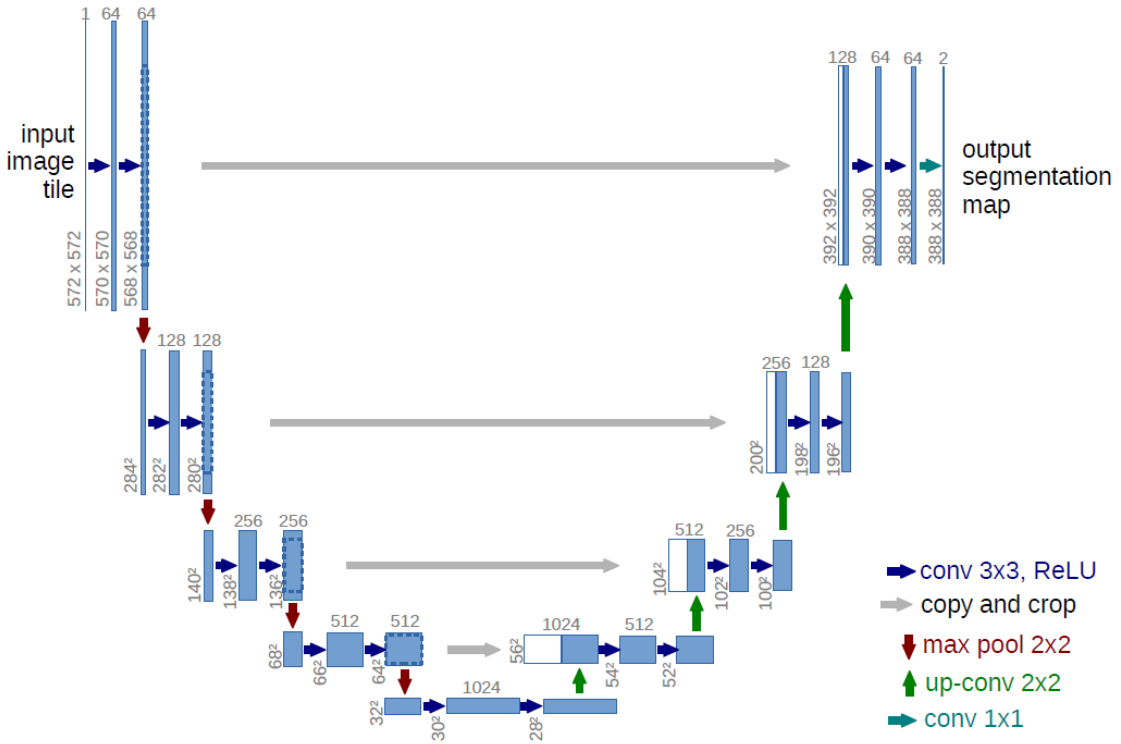


Figure 4.1 *U-Net Architecture,* ***Source:*** *[69]*

4.2 Expansive Path

The expansive path follows the same structure as the contracting path but it uses up-sampling operations instead of max-pooling operations. The expansive path increases the dimensionality allowing the network to localize features even in small patches within the image. Due to the symmetry of the contracting path and the expansive path the architecture resembles a u-shape after which it was named (Fig. 4.1).

The expansive path, copies the structure of the contracting path with slight differences. First, it replaces the max-pooling operations by an up-convolution layer which doubles the spatial dimension of the input in the horizontal and vertical axis while reducing the number of channels by half. Second, every up-convolution layer is followed by a concatenation with the corresponding feature map from the contracting path. Third, the number of channels, in contrast with the contracting path, decrease by half in every block of the expansive path. Finally the output layer consist of a 1×1 convolutional layer which produces one channel per class in the classification task.

If the convolutional layers use only the valid part with no padding added to the

input, there will be some border pixels lost after every convolutional layer, and as a result the spatial resolution at the output of the network will not be the same as in the input. In order to deal with this issue and preserve the spatial resolution, the input images have to be padded before feeding them into the network. The padding strategy is to be defined according to the requirements of the classification task. Otherwise, the input to each convolutional layers can be padded in order to keep the spatial resolution at each step, therefore there is no need for initial padding.

The U-Net has demonstrated to be a good performing alternative for several biomedical segmentation applications and it is specially useful when amount of samples is limited as in most cases in biomedical segmentation [69]. It is then of our interest to use this network architecture and test its performance for our particular case.

5. DEEPLABV3+ AND XCEPTION

DeepLabV3+ is a encoder-decoder structured architecture. Encoder-decoder networks first reduce the spatial resolution, thus extracting higher semantic information, that is what is referred to as the encoder. The decoder on the other hand gradually recovers the spatial resolution and by doing so, it is possible to perform segmentation. For the case of segmentation of histopathological images, it is important to be able to extract those high order features while keeping the spatial resolution. DeepLabV3+ offers the possibility to choose and implement an independent customized network as backbone and connect it to the encoder-decoder structure. This is very useful as it allows for flexibility and such backbone can be any network that suits best for each task. In this section we will review the modules used in DeepLabV3+.

5.1 Atrous Convolution

Atrous convolution is a key operation in DeepLabV3+. It allows to capture multi-scale information by adjusting the field of view of the filters. Atrous convolution is basically a dilated convolution (see Section 3.5.1) except that in this case atrous convolution, by definition, keeps the dimension of the convolutional filters while the resolution of the feature maps is modified as in eq. 5.1:

$$\mathbf{y}[\mathbf{i}] = \sum_{\mathbf{k}} \mathbf{x}[\mathbf{i} + r \cdot \mathbf{k}]\mathbf{w}[\mathbf{k}], \tag{5.1}$$

where $\mathbf{y}[\mathbf{i}]$ is the resulting output feature of the atrous convolution at the location $\mathbf{i}$, in a 2 dimensional space, between the input feature map $\mathbf{x}$ and the convolution filter $\mathbf{w}$, and r is the atrous rate which defines the sampling stride of the input feature map. That is equivalent to a dilated convolution where r is the number of zeroes inserted between two adjacent values of the convolution kernel. In contrast with pooling operation where the spatial resolution is reduced, the output of the

atrous convolution $\mathbf{y}$ will be richer in higher semantic information as the value of r increases, without loss of spatial resolution.

5.2 Encoder

DeepLabV3+ mainly uses atrous convolution inside the encoder to extract the features computed by a DCNN. It consists of parallel atrous convolutions performed with different atrous rates. This is what is known as Atrous Spatial Pyramid Pooling (ASPP). ASPP has shown to be effective and efficient for classifying regions of arbitrary scale [70]. In adition to that, the encoder performs a 1×1 convolution and also one image pooling operation. Thereafter, the results of all the convolutions and the image pooling are concatenated and passed through a 1×1 convolutional layer which will return the output of the encoder as shown in Figure 5.1.

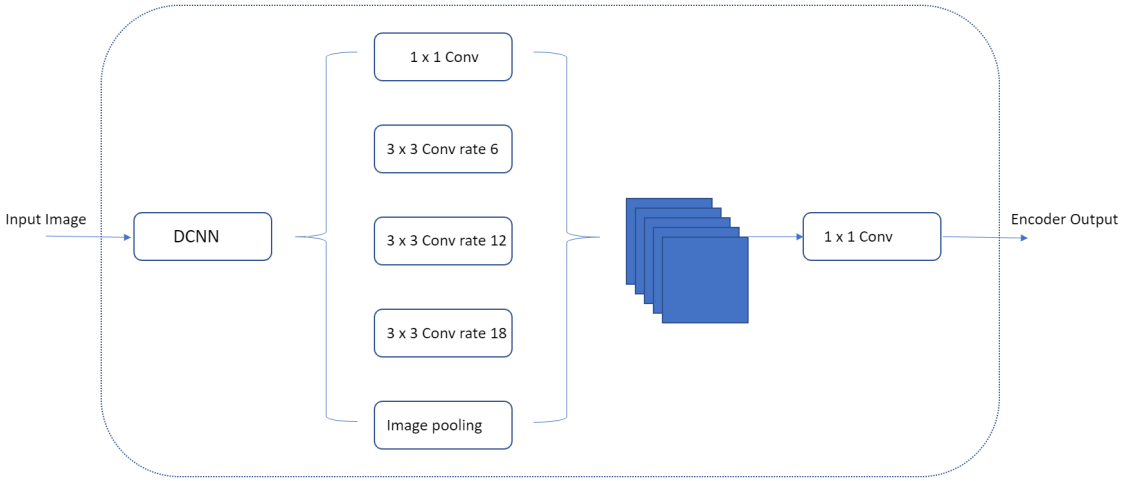


Figure 5.1 *Encoder module of DeepLabV3+*

At this point, it is important to mention that DeepLabV3+ applies atrous depthwise convolution inside the ASPP module, that is one atrous convolution operation performed over each channel independently in the depthwise convolution.

5.3 Decoder

DeepLabV3+ implements a rather simple decoder as shown in Figure 5.2. The decoder should recover the original spatial resolution and it does so by using only few modules. The decoder receives as input the encoder's output and low-level

features from the DCNN in order to compute the decoder's output which is at the same time the network's output and final segmentation.

First in the pipeline, the decoder bilinearly upsamples the encoder's output by $\times 4$ and concatenates the result with the corresponding low-level features from the DCNN which have the same spatial resolution. Note that prior to this concatenation the DCNN low-level features are passed through a 1×1 convolution. That is done in order to reduce the number of channels as they may outnumber the amount of channels from the encoder, and that would diminish the relevance of the encoder.

Thereafter, the decoder performs a 3×3 convolution to refine the features obtained up to that point. Finally it performs a bilinear upsampling operation to fully recover the spatial resolution. For the task of semantic segmentation, the ratio of the input image spatial resolution and the output's spatial resolution, referred to as output stride, is usually 32, meaning that the output image is 32 times smaller. DeepLabV3+ sets the output stride to 16 for denser feature extraction and as demonstrated by Chen et al [71], it provides the best results in terms of accuracy and speed. In order to modify the output stride it may be necessary to modify the last blocks of the network's backbone in such way that the striding is removed [71].

Note that the blocks marked as "DCNN" in Figure 5.1 and Figure 5.2 refer to the same network's backbone.

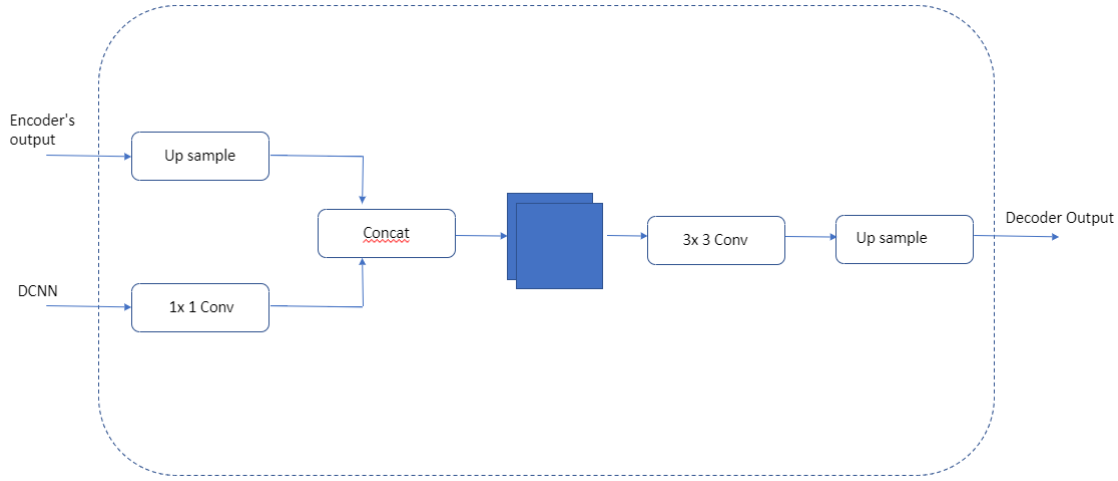


Figure 5.2 *Decoder DeepLabV3+*

As for the network's backbone, shown as "DCNN" in Figure 5.1 and Figure 5.2, Chen et al [71] implemented a slightly modified version of the *Xception* network presented in Section 3.5.3. Namely they made the *Xception* architecture deeper by repeating the middle flow 16 times instead of 8 times. They also replaced all pooling

operations by depthwise separable convolution with striding, and finally they added ReLU activation and batch normalization after every 3×3 depthwise convolution.

6. IMAGE SEGMENTATION IMPLEMENTATION

6.1 Data

All of the laboratory samples were provided by Jilab Inc, they correspond to real laboratory samples taken from different patients whose identities were never disclosed. The samples were 3 inches long by 2 inches wide and they were scanned using a microscope with a 2× microscope objective. The scanner provided low resolution images of the slide samples and also a pre-computed masks. The spatial resolution of the WSIs and masks returned by the scanner is 729 pixels by 1482 pixels. A total of 141 images and 141 pre-computed masks were obtained.

6.2 Annotation

Annotating is a very important step in supervised learning methods. It will ultimately provide the data needed in order to train a model. As the research and breakthroughs in machine learning continue to grow rapidly, the data is still scarce for many tasks. As a matter of fact, data is nowadays one of the most valuable assets for a company and/or researchers [72].

In order to be able to produce data of our own, an annotation desktop application that provides an effective and fast method to annotate WSIs was developed for this thesis project. Jilab Inc holds the copyright and reserves all the rights of this software tool, therefore the source code will not be made open in this document. We will however, review the main user interface (UI) and user experience (UX) aspects and components of the application.

The application was developed using JavasScript based tools, React.js, Redux.js, Electron.js. We used Webpack to bundle the application and Electron Builder to distribute it.

The application consists of two main interfaces: the file loader and the mask editor.

Those two interfaces put together provide a quick annotation tool providing technicians and researchers a user friendly interface without having to make use of a third party tool which will most likely not be tailored for this case.

The file loader allows the user to load up to 200 WSIs and their corresponding masks at a time. When the users first open the application, the loader is the first thing they will see. It will show 200 empty boxes (see Fig. 6.1) which will later contain the WSIs and the masks along with a case ID. The loader also renders a button on the top bar which will prompt the file system for the user to select the folders where the WSIs are stored as shown in Figure 6.2.

Once the the users have selected the folders the interface will render the boxes containing the WSIs and the corresponding masks overlapped on top in such way that the users can easily scroll up and down and review the images (see Fig. 6.3). Notice that up until now the mask editor has not been dealt or tempered with, nonetheless the users already have access to the pre-computed masks and they can asses them in order to decide whether or not they would like to modify them.

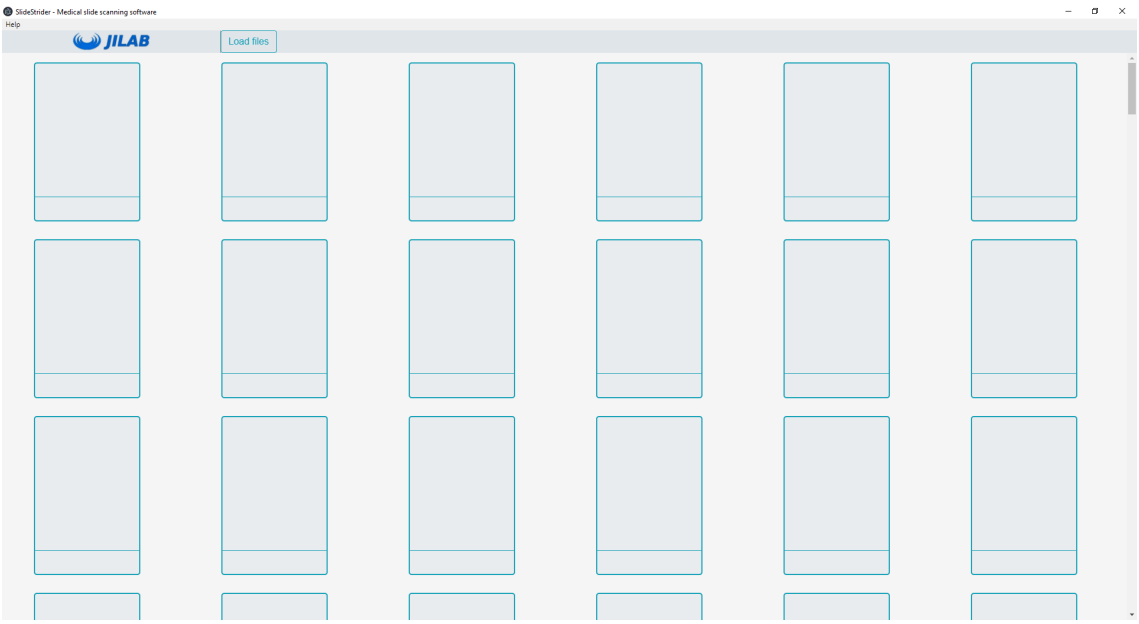


Figure 6.1 *File Loader*

If the user decides to edit a mask, he will have the option to do so by clicking on the top right icon button located on every box. A new window rendering the mask editor will be opened. The editor, as shown in Figure 6.4, is a full size window containing a WSI and its mask overlapped on top. The mask is a black and white image and its opacity has been set to 0.2 in order to be able to show both the mask and the WSI at the same time.

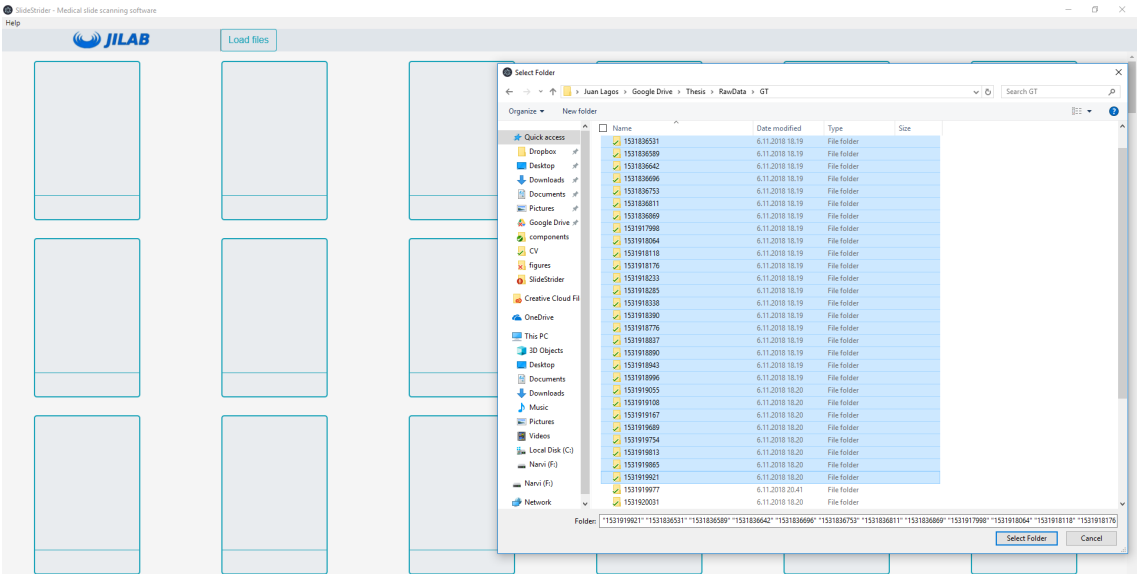


Figure 6.2 *File Loader - folder selection*

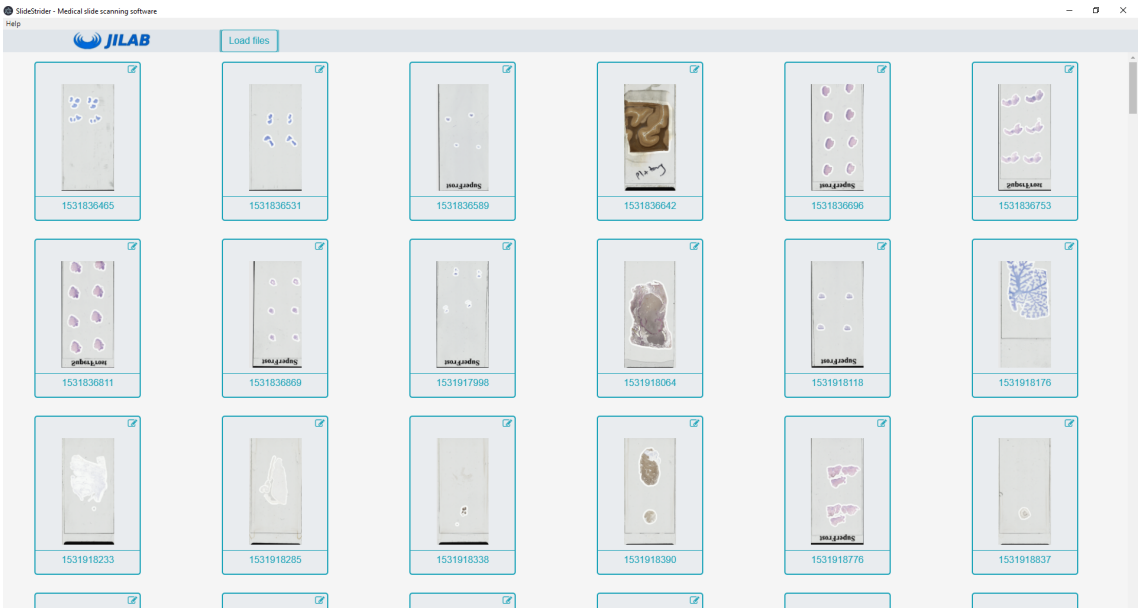


Figure 6.3 *File Loader - WSIs and masks*

The editor consists of four sections, they are highlighted with red boxes in Figure 6.4. The first of them is the left panel, which contains a tools set the users can use to draw on the canvas. Those tools are: a) *Freehand Area*, it allows users to draw freehand areas to include or exclude regions in the mask. By left clicking and holding new regions will be included in the final mask, while right clicking and holding will exclude the region drawn. In order to let the user keep track of the region selected before the drawing is finished, a green closed shape shows the region to be included, and if the region is to be excluded the closed shape is rendered in red color (see

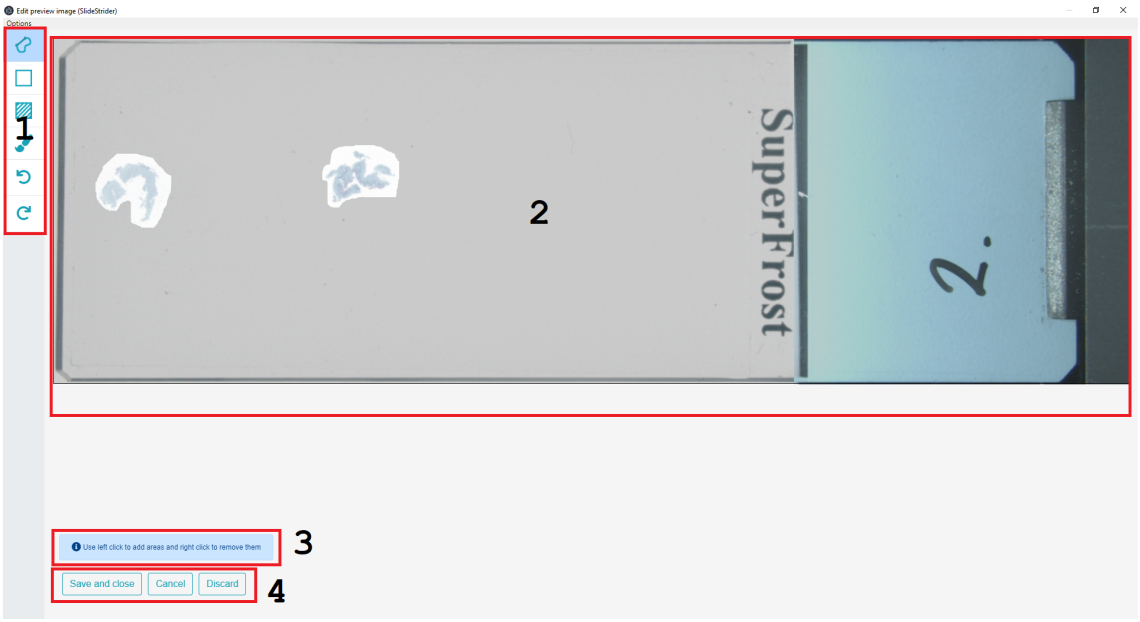


Figure 6.4 *Mask Eidtor*

Fig. 6.5). b) *Fill Area* and *Clear Area*, those two buttons will simply include the entire mask in the case of *Fill Area* or exclude the entire mask in the case of *Clear Area*. c) *Brush*, this tool allows the users to draw with a "brush" instead of drawing areas. This tool is very useful to specify small regions in the mask that can be either included or excluded. Similar to the freehand area tool, users should use the left click to include areas and right click to exclude areas. It is also possible to select a brush size, this option will be shown by hovering on the button (see Figure 6.6). d) *Undo* and *Redo*, those two buttons provide users the possibility to undo action by action and redo them. This feature is very a important usability aspect, as it accounts for potential errors made by users while annotating the masks.

(a) Include freehand area (b) Exclude freehand area

Figure 6.5 *Freehand Area*

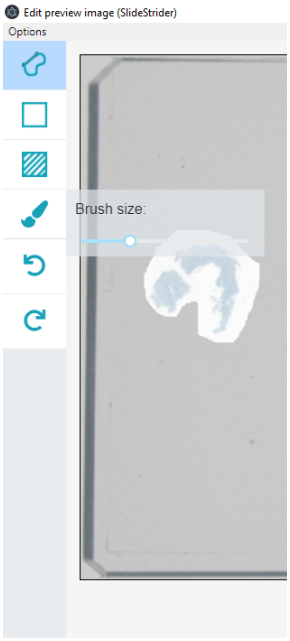


Figure 6.6 *Brush Tool*

The second section of the mask editor is the drawing canvas marked with number 2 in Figure 6.4. It is simply a canvas where users can draw on to include and exclude regions of the mask. The third section marked with number 3 is a simple message that will provide users with instructions about how to use the current tool. Finally the section marked with a number 4 offers three options: *Save and Close*, *Cancel* and *Discard*. *Cancel* and *Discard* have a similar behaviour, they will discard any change made to the mask, however *Discard* does not close the editor while *Cancel* does close the window. The editor also provides the possibility to use hot keys, there are three hot keys available: *Undo*: CTRL + Z, *Redo*: CTRL + Y, *Save*: CTRL + S. *Save* will save the current mask without necessarily closing the editor.

The design of this editor took several versions and prototypes and it will most likely continue under development. It was tested by lab technicians at Jilab Inc who provided feedback and helped to pointing out bugs in the editor. All of their suggestions were taken into account for the last version of the editor as they are the actual users of this application. This editor is currently being used at the premises of Jilab Inc and it has shown to be efficient, useful and user friendly.

6.3 Training

Training was done with a GPU Tesla V100 with 32GB of memory. Keras was used to implement both Unet and DeepLabV3+ from scratch. Keras is a deep learning

API written in python, to create, train and evaluate our models. Keras offers the possibility to add callbacks which are very handy to save time and store the best marking model weights. Namely we used *EarlyStopping*, *ReduceLROnPlateau* and *ModelCheckpoint*.

EarlyStopping stops the training when the loss value does not improve after a given number of epochs, it was set to 20 epochs. *ReduceLROnPlateau* reduces the learning rate once the loss value stops improving, the base learning rate was reduced by a factor of 0.1. *ModelCheckpoint* saves the best marking weights after every epoch.

The loss function used was the binary cross-entropy defined as:

$$C = -\frac{1}{N}\sum_{N} y_i log(\hat{y}_i) + (1 - y_i) log(1 - \hat{y}_i), \tag{6.1}$$

where $\hat{y}_i$ is the predicted probability and y is the target label probability. The index i is either 1 or 2, it corresponds to the two possible classes which in this case are *background* or *tissue*. N is the total number of pixels in the batch.

The time required for training was approximately 30 seconds per epoch in the case of DeepLabV3+ and 47 seconds in the case of UNet.

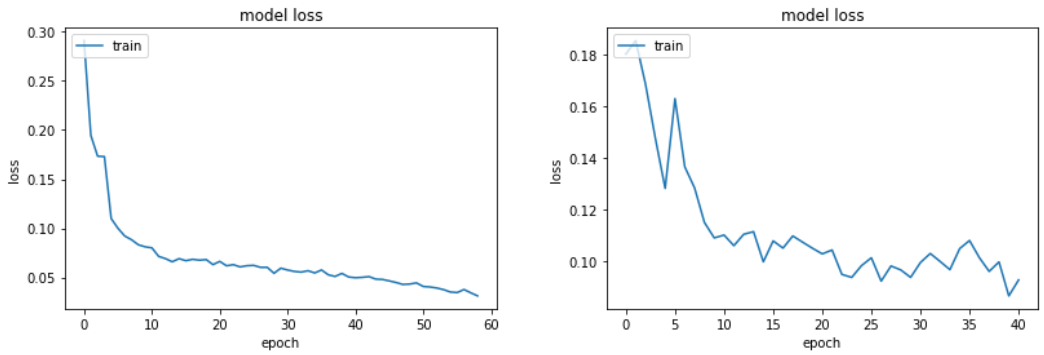
7. RESULTS AND DISCUSSION

In this section we will review the performance of DeepLabV3+ and UNet for the task of semantic segmentation for histopathological images. We will compare the performance of each network as well as the efficiency. Let us start with the value of the loss function and the accuracy for each one of the models:

Model	Loss	Accuracy
UNet	0.13230	0.9471
DeepLabV3+	0.08573	0.9635

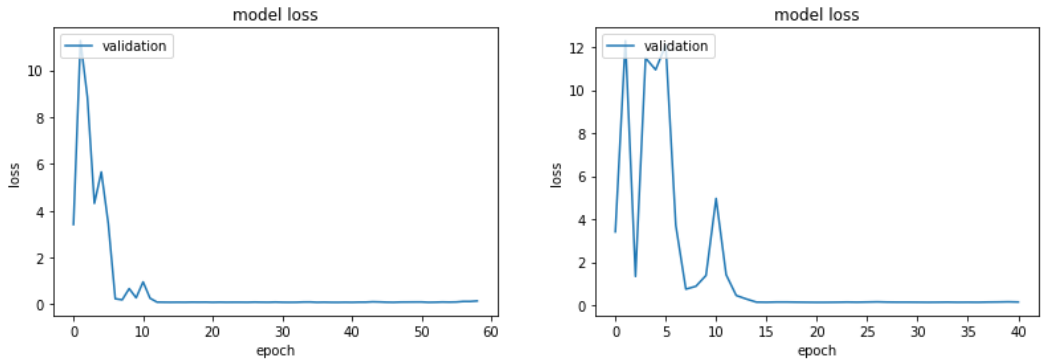
Table 7.1 *Cost and Accuracy - Comparison*

DeepLabV3+ clearly outperforms UNet in these two metrics. Although DeepLabV3+ scores better so far, it is also important to observe the efficiency during training as it may sometimes be preferable to score lower with the trade-off of a much faster training. Figure 7.1 shows the value of the loss for the training data set throughout the epochs for both models, from there we see that DeepLabV3+ decays faster as compared to UNet, however UNet seems to reach its minimum in less epochs. Nonetheless, none of the models seem to reach a global minimum and they tend to improve the loss value, but what is really decisive is the value of the loss for the validation data set which is shown in figure 7.2.



(a) Training History DeepLabV3+ (b) Training History UNet

Figure 7.1 *Loss History - Training Data Set*



(a) Training History DeepLabV3+ (b) Training History UNet

Figure 7.2 *Loss History - Validation Data Set*

For the same hyper-parameters used for the training, DeepLabV3+ outperforms UNet in the metrics analyzed so far. Figure 7.2 again shows how DeepLabV3+ decays faster and reaches a minimum in less epochs, it stabilizes after the 11th epoch whereas UNet does it close to the 15th epoch. Although both models reach the minimum at around the same number of epochs, DeepLabV3+ provides the best loss value as shown in table 7.1.

There were a total of 23 validation samples. Visually the results can be seen in Figure 7.3, where the results for three validation samples are shown. The result for each sample consists of a figure named *specimen* which corresponds to the WSI, a figure named *mask* which corresponds to the ground truth, followed by an image labeled as *probs* which shows a heat map of probabilities where yellow means a high probability of belonging to the class *"tissue"* and purple represents a low probability. In the last column of every sample there is an image labeled as *predicted mask* which is simply the probabilities evaluated for the threshold of 0.5, it is therefore an image of pixels with two different values which are either 1 or 0.

It is however, not clear which model performs better just by performing a visual inspection of the results as they seem to look rather similar. There is a useful metric used for semantic segmentation named *Intersection over Union* (IoU) which performs an assessment on how well the regions of the ground truth match those in the prediction. The IoU is the ratio between the area of intersection and the area of union, calculated for two or more regions. In this particular case, the bigger the area of intersection between the ground truth and the prediction, the smaller the area of union will be. Consequently a higher index of IoU means a better performance. IoU is defined as:

$$IoU = \frac{Area\ of\ Intersection}{Area\ of\ Union}. \tag{7.1}$$

The IoU index was computed for each one of the validation samples, then averaged together to obtaine the results shown in Table 7.2, the results suggest once again that DeepLabV3+ outperforms UNet.

Model	IoU
UNet	0.398
DeepLabV3+	0.446

Table 7.2 *IoU - Comparison*

Validation data set samples were selected in such way that they posed a challenge for the segmentation task. It means that a number of them did not show high contrast between the tissue and background as in the samples shown in Figure 7.3.

It is important to mention that the way samples are annotated will very likely change from person to person and it can potentially bias the results. This difference in the annotations is expected because it is not entirely clear where the boundaries of the tissue is for all the samples. As mentioned previously, the contrast between the tissue and the background can be fuzzy at times, even for human eye. For this thesis work, annotations were made with a certain margin from the tissue area as to guarantee that most of the tissue was included in the final mask. This is because as we deal with patient samples, we would like the medical experts, such as diagnosticians, to have the final saying with little information excluded from the images as it could be of high relevance.

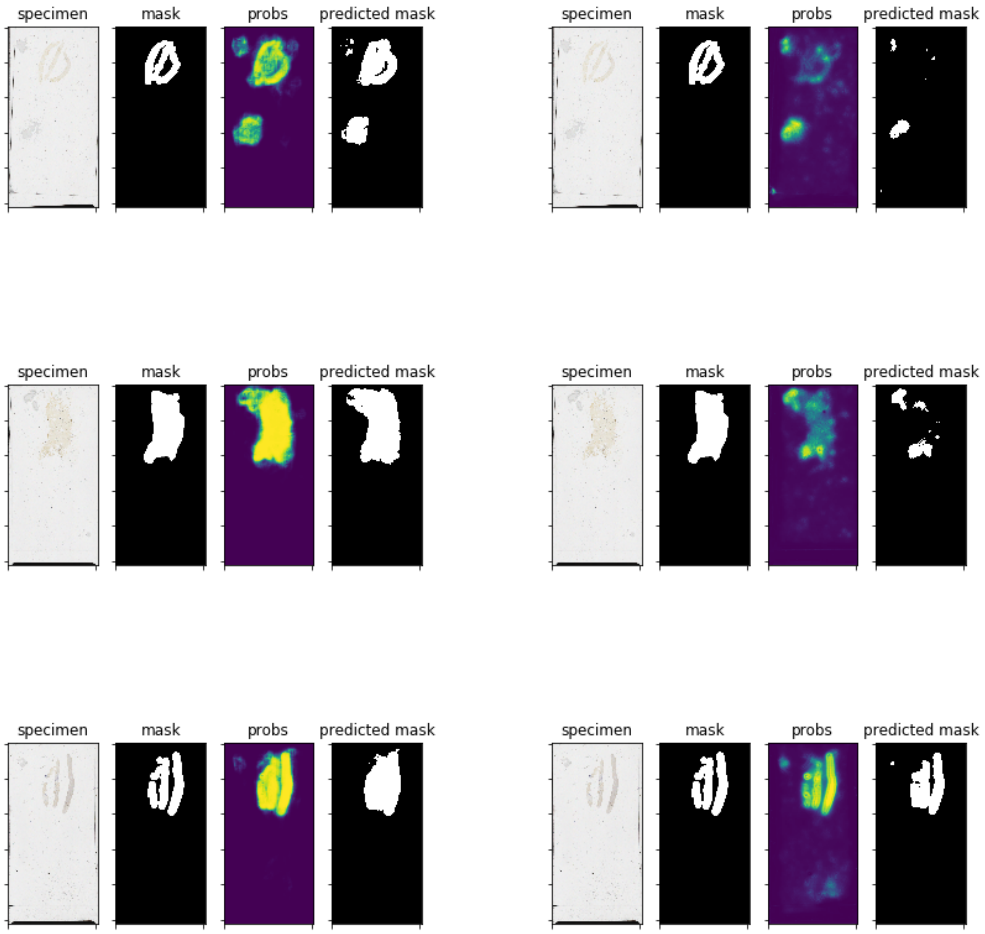


Figure 7.3 *DeepLabV3+ vs. UNet.*

Results for DeepLabV3+ and UNet are shown in the left column and right column respectively.

8. CONCLUSIONS

This thesis presented two feasible solutions for a company case, namely Jilab Inc. In order to reduce the memory and time required in the process of scanning laboratory samples, it was necessary to extract the coordinates where the tissue was located and define regions of interest, thus only those regions of interest can be fully scanned and processed further.

We investigated deep learning and proposed two different models which have previously proved to be effective for the task of semantic segmentation, although it was uncertain how well they would perform for this specific task. Those two models are *UNet* and *DeepLabV3+*. Both models showed acceptable results, however the latter outperformed the former in every metric.

DeeplabV3+ achieves a IoU index of 0.446 whereas UNet scores 0.398 in the same index. DeepLabV3+ also outscores UNet in the loss value striking 0.08573, in contrast UNet scores 0.13230. Similarly the accuracy obtained for with DeepLabv3+ is 0.9635 and 0.9471 with UNet. Visually the results for DeepLabV3+ are consistent and successfully extract the regions of interest from the input images.

In addition to that, a desktop application was developed to annotate WSIs. It is currently being used at the premises of Jilab Inc, it has demonstrated to be easy to use, user friendly, and above all else it provides an efficient tool to obtain data necessary to train supervised training models for this specific task. It also eliminates the dependency on third party tools to annotate images or to obtain annotated samples.

We have provided a feasible and effective solution for the task of semantic segmentation of histopathology images. This thesis work can be further studied and extended to separate objects within the regions of interest already extracted. By effectively extracting the tissue regions of the WSIs, we enabled the possibility to reduce the time required to complete a scan round as well as the memory space required to store the information.

REFERENCES

[1] K. Fu and J. Mui, "A survey on image segmentation," *Pattern Recognition*, vol. 13, no. 1, pp. 3–16, 2000.

[2] A. V. P. Navid Farahani and L. Pantanowitz, "Whole slide imaging in pathology: advantages, limitations, and emerging perspectives," *Pathology and Laboratory Medicine International*, vol. 2015, no. 12, pp. 23–33, June 2015.

[3] R. S. A. Hazem Hiary and V. Chaudhary, "Segmentation and localisation of whole slide images using unsupervised learning," *The Institution of Engineering and Technology*, vol. 7, no. 12, pp. 464–471, July 2013.

[4] I. N. Bankman, "Introduction to segmentation," in *Handbook of Medical Imaging*, ser. Biomedical Engineering, I. N. BANKMAN, Ed. San Diego: Academic Press, 2000, pp. 67 – 68. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780120777907500084

[5] N. M. Zaitoun and M. J. Aqel, "Survey on image segmentation techniques," *Procedia Computer Science*, vol. 65, pp. 797 – 806, 2015, international Conference on Communications, management, and Information technology (ICCMIT'2015). [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050915028574

[6] A. Krenker, J. Bešter, and A. Kos, "Introduction to the artificial neural networks," in *Artificial Neural Networks*, K. Suzuki, Ed. Rijeka: IntechOpen, 2011, ch. 1. [Online]. Available: https://doi.org/10.5772/15751

[7] K. Gurney, *An introduction to neural networks*. UCL Pess, 1997.

[8] D. Pedamonti, "Comparison of non-linear activation functions for deep neural networks on MNIST classification task," *CoRR*, vol. abs/1804.02763, 2018. [Online]. Available: http://arxiv.org/abs/1804.02763

[9] L. C. Velasco, "Performance analysis of different combination of training algorithms and activation functions in predicting the next day electric load," *16th Philippine Computing Science Congress 2016*, 03 2016.

[10] A. Vehbi Olgac and B. Karlik, "Performance analysis of various activation functions in generalized mlp architectures of neural networks," *International Journal of Artificial Intelligence And Expert Systems*, vol. 1, pp. 111–122, 02 2011.

[11] M. Panicker and C. Babu, "Efficient fpga implementation of sigmoid and bipolar sigmoid activation functions for multilayer perceptrons," *IOSR Journal of Engineering*, vol. 02, pp. 1352–1356, 06 2012.

[12] N. Khorrami, M. Alizadeh Pahlavani, and P. Naderi, *A Novel Control Methodology of the GCSC for Damping The SSR Oscillations based on the Neural Networks A Novel Control Methodology of the GCSC for Damping the SSR Oscillations based on the Neural Networks*, 10 2013.

[13] S. Bhattacharyya, U. Maulik, and S. Bandyopadhyay, "Soft computing and its applications," *Fuzzy Sets and Systems - FSS*, vol. 144, pp. 1–30, 01 2011.

[14] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, pp. 436 EP –, May 2015. [Online]. Available: https://doi.org/10.1038/nature14539

[15] M. Bharathi and M. M. Rekha, "A neural network implementation of hyperbolic tangent function using approximation method," *International Journal of Advanced Research in Electronics and Communication Engineering*, vol. 4, 2015.

[16] S. Sonoda and N. Murata, "Neural network with unbounded activation functions is universal approximator," *Applied and Computational Harmonic Analysis*, vol. 43, no. 2, pp. 233 – 268, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1063520315001748

[17] C. O. Magnus Hansson, "Feedforward neural networks with relu activation functions are linear splines," Master's thesis, Lund University, 2017.

[18] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks," in *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, ser. Proceedings of Machine Learning Research, G. Gordon, D. Dunson, and M. DudÃk, Eds., vol. 15. Fort Lauderdale, FL, USA: PMLR, 11–13 Apr 2011, pp. 315–323. [Online]. Available: http://proceedings.mlr.press/v15/glorot11a.html

[19] L. Pauly, H. Peel, S. Luo, D. Hogg, and R. Fuentes, "Deeper networks for pavement crack detection." 34th International Symposium on Automation and Robotics in Construction (ISARC 2017), 07 2017.

[20] I. Nunes da Silva, D. Spatti, R. Flauzino, L. Bartocci Liboni, and S. Reis Alves, *Artificial Neural Network Architectures and Training Processes*, 01 2017, pp. 21–28.

[21] A. Ardakani, C. Condo, and W. J. Gross, "Sparsely-connected neural networks: Towards efficient vlsi implementation of deep neural networks," *CoRR*, vol. abs/1611.01427, 2016.

[22] B. M. Wilamowski, "Neural network architectures and learning algorithms," *IEEE Industrial Electronics Magazine*, vol. 3, no. 4, pp. 56–63, Dec 2009.

[23] X. Wu, "Hybrid learning of feedforward neural networks for regression problems," dissertation, Auburn University, 2014. [Online]. Available: https://etd.auburn.edu/bitstream/handle/10415/4454/dissertation.pdf?sequence=5&isAllowed=y

[24] B. Chandra and P. P. Varghese, "Applications of cascade correlation neural networks for cipher system identification," *International Journal of Computer, Electrical, Automation, Control and Information Engineering*, vol. 1, no. 2, pp. 369 – 372, 2007. [Online]. Available: http://waset.org/Publications?p=2

[25] S. E. Fahlman and C. Lebiere, "The cascade-correlation learning architecture," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. Morgan-Kaufmann, 1990, pp. 524–532. [Online]. Available: http://papers.nips.cc/paper/207-the-cascade-correlation-learning-architecture.pdf

[26] B. Wilamowski, *The industrial electronics handbook*. Boca Raton, FL: CRC Press, 2011.

[27] K.-L. Du and M. Swamy, *Neural Networks and Statistical Learning*. Springer, 12 2014, pp. 337–353.

[28] L. C. Jain, *Recurrent neural networks : design and applications*. Boca Raton, FL: CRC Press, 2000.

[29] R. Rojas, *Neural networks : a systematic introduction*. Berlin New York: Springer-Verlag, 1996.

[30] S. Haykin, *Neural networks and learning machines*. New York: Prentice Hall/Pearson, 2009.

[31] M. Mohammed, M. Badruddin Khan, and E. Bashier, *Machine Learning: Algorithms and Applications*, 07 2016.

[32] M. Usama, J. Qadir, A. Raza, H. Arif, K.-L. Yau, Y. Elkhatib, A. Hussain, and A. Al-Fuqaha, "Unsupervised machine learning for networking: Techniques, applications and research challenges," 09 2017.

[33] J. Larsen, "Introduction to artificial neural networks," *Section for Digital Signal Processing*, p. 52, November 1999. [Online]. Available: http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/2443/pdf/

[34] G. E. Nasr, E. A. Badr, and C. Joun, "Cross entropy error function in neural networks: Forecasting gasoline demand," in *Proceedings of the Fifteenth International Florida Artificial Intelligence Research Society Conference*. AAAI Press, 2002, pp. 381–384. [Online]. Available: http://dl.acm.org/citation.cfm?id=646815.708603

[35] M. Buscema, "Back propagation neural networks," *Substance use & misuse*, vol. 33, pp. 233–70, 02 1998.

[36] Y.-x. Yuan, "Step-sizes for the gradient method," *AMS IP Studies in Advanced Mathematics*, 01 2019.

[37] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation functions: Comparison of trends in practice and research for deep learning," *CoRR*, vol. abs/1811.03378, 2018. [Online]. Available: http://arxiv.org/abs/1811.03378

[38] F. Godin, J. Degrave, J. Dambre, and W. D. Neve, "Dual rectified linear units (drelus): A replacement for tanh activation functions in quasi-recurrent neural networks," *Pattern Recognition Letters*, vol. 116, pp. 8 – 14, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167865518305646

[39] H. Hosseini, B. Xiao, M. Jaiswal, and R. Poovendran, "On the limitation of convolutional neural networks in recognizing negative images," *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp. 352–358, 2017.

[40] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–9, 06 2015.

[41] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*. IEEE, Aug. 2017. [Online]. Available: https://doi.org/10.1109/icengtechnol.2017.8308186

[42] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International*

Conference on Neural Information Processing Systems - Volume 1, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 1097–1105. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999257

[43] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," *ArXiv e-prints*, 11 2015.

[44] D. Stutz, "Understanding convolutional neural networks," *Lehr- und Forschungsgebiet Informatik VIII, Computer Vision*, 2014. [Online]. Available: https://davidstutz.de/wordpress/wp-content/uploads/2014/07/seminar.pdf

[45] Y. Lecun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," *ISCAS 2010 - 2010 IEEE International Symposium on Circuits and Systems: Nano-Bio Circuit Fabrics and Systems*, pp. 253–256, 05 2010.

[46] J. Wu. (2017) Introduction to convolutional neural networks. [Online]. Available: https://pdfs.semanticscholar.org/450c/a19932fcef1ca6d0442cbf52fec38fb9d1e5.pdf

[47] F. Keller. (2010, February) Convolutions and kernels. [Online]. Available: http://www.inf.ed.ac.uk/teaching/courses/cfcs1/lectures/cfcs_l15.pdf

[48] S. S. Liew, M. Khalil-Hani, F. Radzi, and R. Bakhteri, "Gender classification: A convolutional neural network approach," *Turkish Journal of Electrical Engineering and Computer Sciences*, vol. 24, pp. 1248–1264, 03 2016.

[49] S. Chaudhuri. (2010) MS Windows NT kernel description. [Online]. Available: http://graphics.stanford.edu/courses/cs148-10-summer/docs/04_imgproc.pdf

[50] J. Murphy. (2016) An overview of convolutional neural network architectures for deep learning. [Online]. Available: https://pdfs.semanticscholar.org/64db/333bb1b830f937b47d786921af4a6c2b3233.pdf?_ga=2.81959766.137558942.1555925833-414122284.1546717070

[51] Y. Li, X. Zhang, and D. Chen, "Csrnet: Dilated convolutional neural networks for understanding the highly congested scenes," *CoRR*, vol. abs/1802.10062, 2018. [Online]. Available: http://arxiv.org/abs/1802.10062

[52] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, and T. Chen, "Recent advances in convolutional neural networks," *Pattern Recognition*, vol. 77, pp. 354 – 377, 2018. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031320317304120

[53] L. Xu, J. S. J. Ren, C. Liu, and J. Jia, "Deep convolutional neural network for image deconvolution," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'14. Cambridge, MA, USA: MIT Press, 2014, pp. 1790–1798. [Online]. Available: http://dl.acm.org/citation.cfm?id=2968826.2969026

[54] P.-L. Pröve. (2017) An introduction to different types of convolutions in deep learning. [Online]. Available: https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d

[55] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ser. ICML'10. USA: Omnipress, 2010, pp. 807–814. [Online]. Available: http://dl.acm.org/citation.cfm?id=3104322.3104425

[56] S. L. Hijazi, R. Kumar, and C. Rowen. (2015) Using convolutional neural networks for image recognition. [Online]. Available: https://pdfs.semanticscholar.org/bbf7/b5bdc39f9b8849c639c11f4726e36915a0da.pdf?_ga=2.78155988.137558942.1555925833-414122284.1546717070

[57] J. Nagi, F. Ducatelle, G. A. D. Caro, D. Cireşan, U. Meier, A. Giusti, F. Nagi, J. Schmidhuber, and L. M. Gambardella, "Max-pooling convolutional neural networks for vision-based hand gesture recognition," in *2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, Nov 2011, pp. 342–347.

[58] H. Wu and X. Gu, "Max-pooling dropout for regularization of convolutional neural networks," in *Neural Information Processing*. Springer International Publishing, 2015, pp. 46–54. [Online]. Available: https://doi.org/10.1007/978-3-319-26532-2_6

[59] P. Domingos, "A few useful things to know about machine learning," *Commun. ACM*, vol. 55, no. 10, pp. 78–87, Oct. 2012. [Online]. Available: http://doi.acm.org/10.1145/2347736.2347755

[60] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 06 2014.

[61] B. Athiwaratkun and K. Kang, "Feature representation in convolutional neural networks," *CoRR*, vol. abs/1507.02313, 2015. [Online]. Available: http://arxiv.org/abs/1507.02313

[62] A. Deshpande. (2016) A beginner's guide to understanding convolutional neural networks. [Online]. Available: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

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

[64] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 1097–1105. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999257

[65] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017, pp. 1800–1807.

[66] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[67] D. Frossard. (2016) Vgg in tensorflow. [Online]. Available: https://www.cs.toronto.edu/~frossard/post/vgg16/

[68] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016.

[69] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015. [Online]. Available: http://arxiv.org/abs/1505.04597

[70] L. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *CoRR*, vol. abs/1706.05587, 2017. [Online]. Available: http://arxiv.org/abs/1706.05587

[71] L. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," *CoRR*, vol. abs/1802.02611, 2018. [Online]. Available: http://arxiv.org/abs/1802.02611

[72] M. Yousif, "The rise of data capital," *IEEE Cloud Computing*, vol. 2, no. 2, pp. 4–4, Mar 2015.