UNIVERSITÀ
DEGLI STUDI
FIRENZE

MASTER DEGREE COURSE IN COMPUTER ENGINEERING

Deep Convolutional Neural Networks for Egyptian Hieroglyphs Classification

---

Author:
Marco LOSCHIAVO

Supervisors:
Prof. Fabrizio ARGENTI
Prof. Alessandro PIVA
Prof. Andrea BARUCCI
Prof. Massimiliano FRANCI

May 28, 2021

Abstract of thesis entitled

Deep Convolutional Neural Networks for Egyptian Hieroglyphs Classification

Submitted by

Marco LOSCHIAVO

for the degree of Master Degree

at The University of Florence

in May, 2021

Convolutional neural networks (CNN or ConvNet) are network architectures for deep learning suitable for image processing that encountered enormous success in recent years. In this thesis, we are interested in the performance of CNNs to classify ancient Egyptian hieroglyphs. A new CNN is proposed architecture and compared to three state-of-the-art CNNs, namely Resnet50, InceptionV3 and Xception.

Two different datasets were used, one that contains 4310 grayscale photos of dimentions 50x75 taken from the book The Pyramid of Unas by Alexandre Piankoff, and one containing 1310 hieroglyphs in the form of sculptures and paintings by Prof. Massimiliano Franci. The images contained in these datasets have been preprocessed and merged into a single dataset. Before the images are fed into the CNN, the dataset have been divided into train, validation and test sets. Image augmentation was applied to the training set in order to improve the effectiveness and generalization of the network.

The results show an excellent ability of our network in recognizing the hieroglyph classes. The model achieves an accuracy of over 97%, better than the other state-of-the-art CNNs. Furthermore, this architecture guarantees a shorter image prediction time and training time.

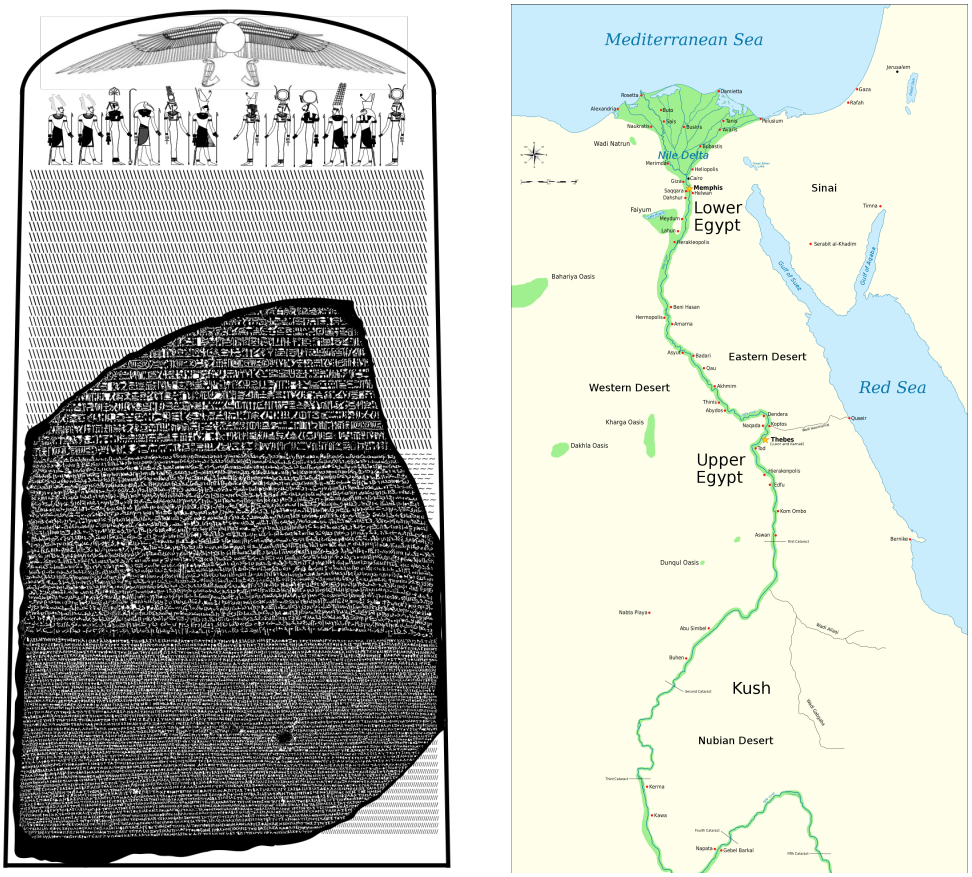
Contents

1

Introduction

The ancient Egyptian hieroglyphic script was one of the writing systems used by ancient Egyptians to represent their language. It have always been a mysterious writing system as their meaning was completely lost in the 4th century CE because during the Ptolemaic (332-30 BCE) and the Roman Period (30 BCE-395 CE) in Egypt, Greek and Roman culture became increasingly influential and the Coptic became the first alphabetic script used in the Egyptian language.

For many years hieroglyphs were not understood at all. In 1799 members of Napoleon Bonaparte's Egyptian Campaign discovered The Rosetta Stone in Egypt, a decree issued in 197 BC by Ptolemy V. It was an ancient Egyptian stele, divided into three text registers, with the lower right corner and most of the upper register broken. The stone was engraved with three inscriptions: hieroglyphs in the upper register, Greek at the bottom and demotic (derived from the ancient Egyptian hieroglyphic script, but used for writing on documents). It was hoped that the Egyptian text could be deciphered through its Greek translation, especially in combination with the presence of the Coptic language, the last phase of the Egyptian language.



(a) **(b)**

Figure 1.1: (a) One possible reconstruction of the original stele of the Rosetta stone. (b) Map of ancient Egypt

This discovery allowed researchers to investigate the hieroglyphs, but it wasn't until 1822 when Thomas Young and Jean-Francois Champollion discovered that these hieroglyphs didn't represent a word, but each hieroglyph represented a sound and multiple hieroglyphs form a word. Their work had a revolutionary impact on the deciphering of ancient Egyptian hieroglyphs. Over time some researchers corrected and refined their understanding of Egyptian and, starting from 1850, it was possible to fully translate the ancient Egyptian texts. The ability to understand hieroglyphs has uncovered much of the history, customs and culture of Egypt ancient past.

It is these ancient hieroglyphs that tell the tales of the otherwise long forgotten Pharaoh. Their achievements and victories on the battlefields

have all been stored within these hieroglyphs.

Since the deciphering of the ancient Egyptian hieroglyphs, much of their culture has been uncovered. By now, most of the ancient texts have been translated, but this does not put an end to the mystic air revolving around the ancient Egyptian culture. In fact, many movies and games only fuel the thought of Egypts magical powers and mummies. To this day only a few people are capable of reading the ancient Egyptian hieroglyphs, but with the aid of current technology it is possible to provide others, like tourists, the ability to uncover the mysteries of ancient texts themselves. This can be made in the form of an App that allows tourists to receive the translation and other background information of a given text, simply by taking a picture.

1.1 Motivation

Convolution Neural Networks are specialized Neural Networks, which are well suited for the processing of images. The CNNs are not a new idea, for example in 1989 have LeCun et al. [19] a CNN used to recognize handwritten digits in mails, but the hype around CNNs has started 2012 after the incredible winning of Krizhevsky et al. [16] on the ImageNet challenge [24] for classification of images into 1,000 categories. The CNN solution of Krizhevsky et al. had a top-5 error-rate of 15.3% compared to the second place with 26.17%. Since this, the error rate is further decreased with other CNN designs and further research on the training of Neural Networks. CNNs have not only reached good results on ImageNet, but they are also successfully applied on different tasks on images, like the recognition of numbers in street view images [12] or in traffic sign classification [5] and also in not image tasks, like natural language processing [6].

CNNs or rather Neural Networks (NN) are very interesting machine learning methods. They have a layered structure, and every layer could learn an abstract representation of the input. This ability is called Feature Learning or known as Representation Learning [18]. Feature Learning is a method, that allows a machine to learn from raw data. The machine

takes the raw data as input and learns automatically the features necessary to solve the machine learning problem. For example, in image classification the raw data is an image, which is represented by an array of pixels. These arrays are fed to the CNN, and the CNN learns useful features from these images to solve the machine learning problem. In the first layer, the CNN could be learn to detect edges, in the next layer the arrangements of the edges and so on [18]. More generally formulated, in the lower layers are basic concepts learned, and with every further layer, die concepts are combined to higher concepts.

In traditional image classification, there must be extracted useful features from the image, which are then used on machine learning algorithms like an SVM [8]. These handcrafted features must be carefully chosen, otherwise, the classification has a bad performance.

This problem of carefully chosen features is not present in NN, but they have other difficulties. One of these is the right choice of hyperparameters and of the architecture, witch could have direct influence on the performance.

1.2 Goal of this thesis

The aim of this thesis is to implement a classifier for ancient Egyptian hieroglyphic using a Convolutional Neural Network. There is not a single CNN architecture that works well on all problems, so it may be necessary to develop a new one. In this thesis, we develop our CNN architecture that can be applied to our task. We had the goal, that CNN worked well on task, and that training and prediction on it were fast, because we only had a limited resource of time and computing power. To compare the performance of our developed architecture, we compare the performance on three other CNN architectures, which are Inception-v3 [26], ResNet-50 [13] and Xception [3].

1.3 Thesis Structure

In the next chapter we will briefly go through theory. First Artificial Neural Networks is explained, leading to Convolutional Neural Networks and Separable Convolution that is the foundation of this project.
The chapter 3 describes the datasets used and explain the preprocessing pipeline for the preparation of the image, before it is applied to the CNN.
In chapter 4 we will show the part relating to the classification problem. We present our CNN architecture, and others CNN architectures that will be used for comparisons. Furthermore, we will describe the transfer learning approach applied to these.
In chapter 5 the results and evaluations of the networks are reported . Two variants of this architecture were compared, one containing residual blocks and one not. In addition, the effects of data augmentation were evaluated, and the results of stratified k-fold cross validation are reported.
In the last chapter is reported a conclusion of this thesis.

2

Machine Learning, Neural network and CNN

Artificial neural networks and, especially, convolutional neural networks are the main topic of this thesis. So, in this chapter we start with a short introduction to the basics of machine learning and continue with an introduction of artificial neural networks and convolutional neural networks. This Section is only a short introduction, and the information are taken from the chapters for Neural Networks from the books of Goodfellow et al. [11], and Michael A. Nielsen [20], if not other cited.

2.1 Machine Learning

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it to learn for themselves.

A practical view of a machine learning system is depicted in Figure 2.1. The process is split into two phases. In the first phase, the machine learning algorithm is used to learn from the training data, and the second phase is the prediction. The training data could be labeled images of

Egyptian hieroglyphics with the task to predict their meaning. The machine learning algorithm learns a model on these data and this model can then be used to predict unseen images of the same task. This prediction is used in the second phase and applies only the learned model. In our example, the learned model get images of a hieroglyphic and predicts the meaning.

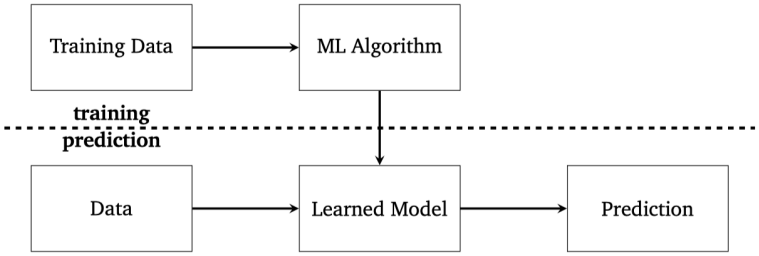


Figure 2.1: Workflow of a machine learning problem. The data will be used to train a model with a machine learning algorithm. Then, in the prediction phase, the learned model can be used to generate a prediction.

2.1.1 Learning Problems

In machine learning it can be distinguished between different learning problems. The main learning problems are:

Supervised Learning : in supervised learning, the machine learning algorithm gets a labeled training set. The training set consists of several examples, and every example is a pair of input data and a labeled output data . Starting from this set of examples, the program is guided to describe a model able to predict the correct output.

At this point the prediction model must be tested with another known dataset independent from the training set. Only when the testing phase is satisfactory the algorithm can be considered reliable for use on unknown data. Therefore, given a supervised problem and the data type, learning steps are:

- *Algorithm selection*: The first step is to choose the supervised algorithm to use. Every method has different strength and

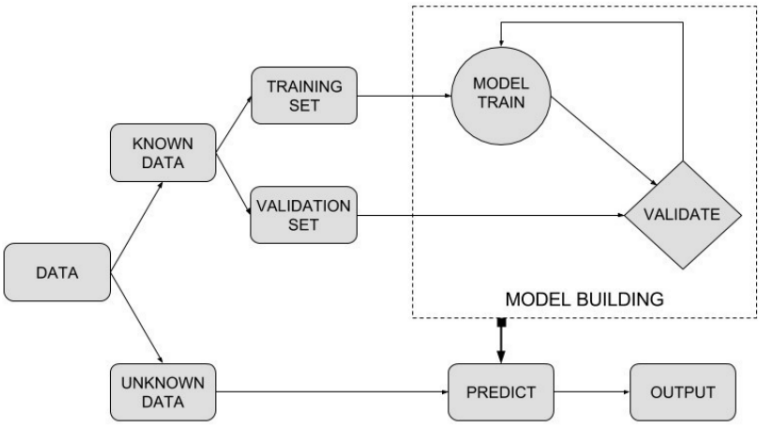


Figure 2.2: Supervised Learning flowchart

weak point. The choice depends on the particular problem and on the kind and amount of available data. Some of these algorithms are: Support Vector Machine (SVM), Decision Tree, Artificial Neural Network and Deep Learning. In this work the focus will be on Deep Learning, extension of ANN.

- *Training*: The training phase is probably the most important one, as the final performances depend on the predictive model built. A known dataset is selected; must be as more representative of the problem as possible. Using dataset not general enough can lead to overfitting and to bad performances. This set, the training set, must provide an output (label) for each listed input. The algorithm is trained with the selected dataset. The aim of this phase is trying to build a model able to fit the data provided, that is predict the correct output for each input provided as best as possible.

- *Testing*: The testing phase is important to test the performances achieved by the prediction model built in the previous phase. Another known dataset, called test set, is prepared. The dataset must provide, as the training set, reliable input and output for each example. An important property of this set is that it should be as independent as possible from the training one. The previously trained algorithm is here used to predict the

input data of the test set. Only the input are used and the output are predicted by the algorithm and stored. The fundamental difference from the train step is that, in this one, the output label are not used to improve the prediction capabilities of the model, but only to evaluate its performances. The predicted outputs are validated using the known outputs. The performance are hence evaluated and analysed. If they are satisfactory it is possible to go to the final step, otherwise the algorithm or the training phase must be reviewed with different precautions or parameters.

- *Model Deployment*: Once the algorithm is trained and validated, it is possible to use it as an automatic system to solve the original problem on new data.

The goal is to find a general function or rule that maps the input to the desired output label. Furthermore, the mapping should be general so that unseen data is also correct mapped.

The supervised learning could be further split in classification and in regressions learning problems. *Classification* means, that the input of an example is divided into two or more classes. The goal is to find a mapping of the input to this classes, even if the the input is an unknown example. For example consider the spam-classification of emails. The classes are "spam" or "not spam". The machine learning algorithm should learn from examples, with known labels, a function or rule, which can associate an email to one of these two labels.

As *regression* we refer to the learning of a continuous value. Or rather, it should be learned a function, that represents the label value in dependency of the input. This could be, for example, the price of a house; the input in this case , could be the size of the house and the size of the property.

Unsupervised Learning: in unsupervised learning, the machine learning process an unlabeled training set. The training set has only examples, but no labels. The machine learning algorithm should find

a structure in the data. This could be done with clustering methods. This means, that examples should be grouped, whit similar properties together.

Reinforcement Learning: in reinforcement learning the machine learning algorithm interact with an environment and must reach a certain goal. This could be to learn how to play a game, or to drive a vehicle in a simulation. The algorithm gets only information on how good or bad he interacts with the environment. For example, in learning to play a game, this information could be the winning or losing the game.

The classification of the supervised learning can be further distinguished into

- **Multi-Class**: Multi-class are characterized by more than two classes for one label. For example, the label weather could be sunny, rainy or cloudy, which are three classes for the label weather.
- **Multi-Label**: Multi-label describes the fact, that one label could have multiple classes for one example. For example, a document has the classes "politics", "sports" and "science". But sometimes, a document can be classified to two of the classes like politics and sports.

The learning problem of this thesis is associated to the supervised learning and more precisely to a multi-class problem. Our dataset consists of serveral images, and to every images a label is associated. The dataset will be described in Chapter 3.

2.1.2 Underfitting and Overfitting

A machine learning algorithm must perform well on unseen data. Their ability is called generalization. The generalization error is measured on a test set. A model may not perform well on unseen data, for two reasons. The model may not have enough capacity and underfits the underlying function or it has too much capacity and overfits the underlying function. If a model is underfitting, then the training error will result high as

well as the test error. If the model is overfitting, then the training error will be low and the test error will be high. The goal is to find a model, which has a low generalization error. The typical curves for training and generalization error are depicted in 2.3.

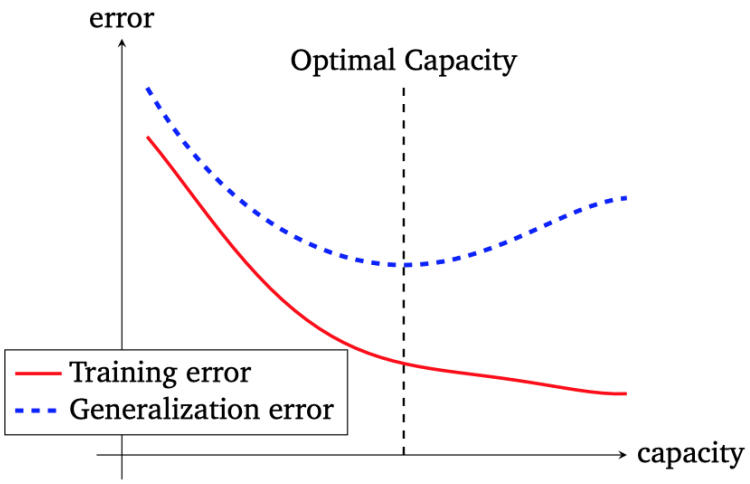


Figure 2.3: Shows typical curves for training and generalization error in dependency of the capacity of the model. Optimal capacity is reached at the minimal generalization error. Left of the optimal capacity is the model underfitting. On the right side of the optimal capacity is the model overfitting. The generalization error has typically a U-shaped curve.

In Figure 2.4, three diagrams are depicted, which show the same noisy sampling of a function. The samples are used learn a model, which describes the underlying function. The left diagram learned a model with a low capacity. So the training error is high and a test set would also produce a high test error. In the center is depicted a model with a higher capacity. This shows a good approximation of the underlying function. Nevertheless the model has a training error, because we sampled the function with noise. But this model will have on a test set the best generalization error compared to the other two models. The third diagram shows a model with a high capacity. The training error will be low, but the learned model is not a good approximation of the function, which would result on a test set with a high error.

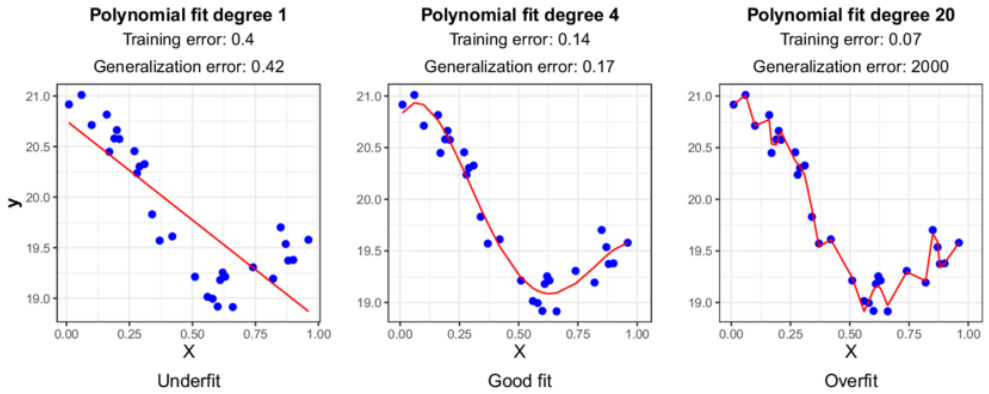


Figure 2.4: The three diagrams show three different models, which try to fit the sampled points. The models are described by a polynomial of degree 1,4,15. The left model underfits the underlying function. The model in the center is a good approximation of the underlying function. The right model overfits the underlying function: it has the smallest error when it is used to fit the sampled points, but on unseen samples, it provide a high error.

2.1.3 Hyperparameter and Model Selection

Hyperparamters are parameters that belongs to the machine learning algorithm. With these parameters, the behavior of the algorithm can be changed; to be precise, some of the parameters are used regulate the capacity of the model. These parameters will not be learned during the training, but will be set by the user at the start of the learning process. In case of neural networks, this could be the learning rate, or the architecture of the model (number of layers or width of layers). In other words, hyperparameters are parameters, which are not learned during the learning process.

Normally, we test several models with different hyperparameters and choose the model with the lowest error. If we do this test only on the training set, then the lowest error is reached with the model that has the highest capacity. This is not useful, as we saw in the section about overfitting. Actually want the model with the lowest generalization error. So the training set will be further split in a validation set. Thus we have three data sets: a training, a validation and a test set. The validation set is only used to measure the generalization error, and is not used for the training. The model with the smallest generalization error will then

be used as the best model. But the predicted performance on the validation set has a bias, because we chose the model, that maximizes the validation set. Therefore, the performance should not be measured on the validation set. That is why we have a third dataset, the test set. On this test set could we now predict the performance of the model, and this is a good approximation, how good the performance will be on unseen data.

2.2 Artificial Neural Networks

An *Artifical Neural Networks* (ANN) – or short only Neural Networks (NN) – is a construction inspired by the human brain, as the word neural indicates. There are similarities between a nerve cell and a node belonging to an ANN. A nerve cell have dendrites handling the cell input and if the stimuli is large enough the signal is passed on to new cells through axons. ANNs are composed of nodes with weighted input, an activation function and output. In figure 2.5 we can see a comparison between a neuron and its counterpart in Artificial Neural Networks.

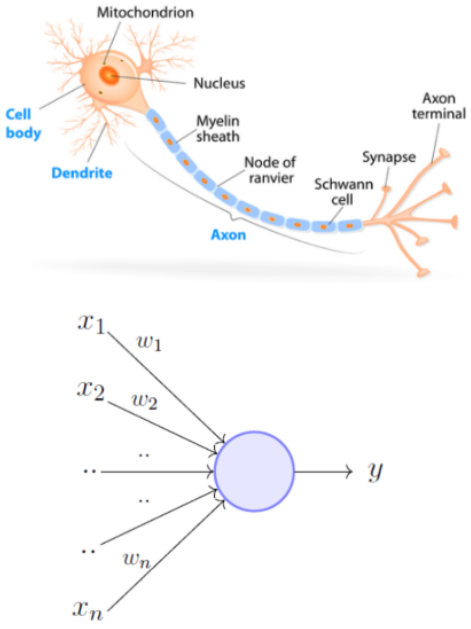


Figure 2.5: Comparison between neuron and its counterpart in Artificial Neural Networks.

Due to ANNs capacity and black box behaviour it is easy to think of it as complicated. Large complex structures may be built with amazing results, but the smallest elements are rather simple. In this section we shall start with the simplest possible network and successively build larger and more complex structures, ending with CNNs, which have been used in this project.

2.2.1 Perceptron

The perceptron is the simplest Artificial Neural Networks algorithms, introduced in its simplest version by Rosenblatt in 1958, as supervised machine learning algorithm for binary classification. It can be considered an ANN composed by one artificial neuron. The perceptron (see figure 2.6) has multiple inputs $(x_1, x_2, ..., x_n)$, and only one output y . Furthermore, the perceptron has a bias input, which is called x_0 and has the typical value of -1. For every input a linear combination with the weights $(w_0, w_1, ..., w_n)$ is derived, that is

$$z = \sum_{i=0}^{n} x_i \cdot w_i$$

After the linear combination is computed, the value z is inserted into the activation function $\sigma(z)$. This function activates the output, if a threshold is fulfilled. Several activation functions, with different properties can be used, but for the simplicity, we will use the Heavyside-function defined as:

$$\sigma(z) = \begin{cases} 0 & \text{if } z < 0 \\ 1 & \text{if } z \geq 0 \end{cases}$$

So, the final form is:

$$y = \sigma(\sum_{i=0}^{n} x_i \cdot w_i)$$

By using this activation function, the perceptron has the ability to split the hyperspace with linear boundaries. This means that the perceptron can only learn problems that are linearly separable. For many tasks, this is not suffcient. By introducing multilayer perceptrons, however it is possible to solve nonlinear problems, as described in the following.

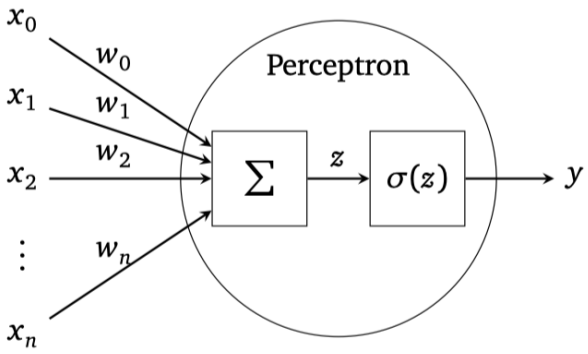


Figure 2.6: One perceptron with n inputs and one output. The circle describes the perceptron, in which is applied the linear combination and the activation function.

2.2.2 Multilayer Perceptron

The multilayer perceptron also known as feedforward neural networks, are a multilayered networks of perceptrons. This means, that several perceptrons are linked together. In Figure 2.7, an example feedforward neural network is depicted. A feedforward neural network consists of multiple layers, which can be classified as follows:

- *Input layer*: it is the first layer of the network. It consists of a set of n input nodes, without processing capacity, associated with the n inputs of the network: $x_i \in \mathbb{R}, \quad i = 1, \ldots, n$;
- *Hidden Layers*: they are $L - 1$, whit $L \geq 2$ different inner layers, each consisting of several perceptrons, which are called units;
- *Output Layer* it is the last layer of the network. It consists of $K \geq 1$ neurons whose outputs constitute the outputs of the network $y_i \in \mathbb{R}, \quad i = 1, \ldots, K$

The depth of a network, $L + 1$, defines the number of layers and the width of a layer defines the number of units. For example, the network in Figure 2.7 has a depth of three. The width of the hidden layers is four and the width of the input and output layers is two. The bias units are indicated with $+1$, and do not contribute count to the width of the layer.

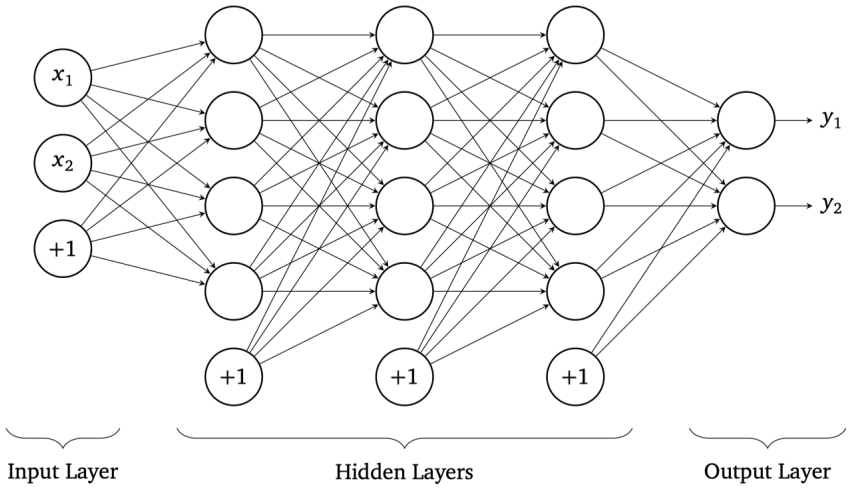


Figure 2.7: One perceptron with n inputs and two output. The circle describes the perceptron, which includes the linear combination and the activation function.

In this example, the units between every layer are fully connected (FC). This means that every unit of layer $l + 1$ has the output of all units from layer l as input. The number of links, and consequently the number of weights, will fastly increases, when the width of the layers increases. If layer l has a units and layer $l - 1$ has b units, then the number of weights $|W_l|$ between these layers are $|W_l| = a \cdot (b + 1)$. The $+1$ comes from the bias input. So it is not uncommon that modern NNs have millions of parameter, which must be learned.

Through this concatenation of several layers, the feedforward neural networks gets the capability to solve non-linear problems, if the activation function is not linear. If the activation function was linear, then the network would have the capability to solve only linear problems, because the composition of linear functions produces again a linear function. So, the activation function should be a non linear function. In Section 2.2.4 we show some useful activation functions.

Furthermore, a feedforward neural network has a very powerful property. With only one hidden layer with sufficient units and a suitable activation function, like the sigmoid function, the network can approximate, with arbitrary precision, every continuous functions on a closed and bounded subset of $\mathbb{R}^n$. This means that a network with a single layer has the ability to represent any function. However it is not guaranteed that the training algorithm will find this function. In fact it is possible, that the training algorithm misses to find the correct value for the parameters that the training algorithm chooses the wrong function due to overfitting. Furthermore, the single layer may be infeasible large and it exists no heuristic how large the layer should be.

In general, deeper networks are better and have a lower generalization error compared to shallow networks only one hidden layer. For example in the recognition of objects in images, deeper networks learn in the lower layers how to extract edges and corners, and in higher layers compose this information onto more complex structures.

For the designing of NNs, it is practical to use a layer as a unique building block. So the layers of this section, will be called fully connected (FC), because all units of the layer are connected with the input of the successive layer. Later, we will present other types of layers, like convolution, pooling or dropout layer. Commonly, a layer has only one specific task. Furthermore, the layers could be seen as a function $f^{(i)}(\mathbf{x}) = \mathbf{y}$, which manipulates the input $\mathbf{x}$ to produce the output $\mathbf{y}$. So a feedforward neural network can be described as a composition of functions with

$$f(\mathbf{x}) = f^{(n)}(\dots(f^{(2)}(f^{(1)}(\mathbf{x})))$$

where $f(\mathbf{x})$ describes the complete network.

2.2.3 The training

So far, we have not described how the learning work for ANNs. In this section, we consider only the supervised training. This means that we

have a training set of input vectors $\{\mathbf{x}_n\}$, where $n = 1, ..., N$, with a corresponding set of output vectors $\{\mathbf{y}_n\}$. The goal of the training is finding the function $f(\mathbf{x}; \mathbf{w}) = \mathbf{y}$ of the ANN so that the function approximates the training data. $\mathbf{w}$ describes the parameters that are learnable during the training. In our case, these are the weights on the links between the units. A loss function is used. A simple loss function $J(\mathbf{w})$ is the mean square error (MSE) that computes the squared difference between $f(\mathbf{x}; \mathbf{w})$ and $\mathbf{y}$. Furthermore, the loss function computes the average loss over the complete training set

$$J(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} ||f(\mathbf{x}_n; \mathbf{w}) - \mathbf{y}_n||^2 \tag{2.1}$$

The loss function $J(\mathbf{w})$ is used to minimize the error between the training data and the ANN with respect to $\mathbf{w}$. For the minimization of the loss function, we will use a gradient descent method that needs the gradient of the loss function with respect to $\mathbf{w}$. To calculate the gradient, the backpropagation algorithm, introduced later in this section is used. The gradient descent method is an iterative optimization algorithm that updates the weights with the help of the gradient. In the simplest version, it has the following form:

$$\mathbf{w} \leftarrow \mathbf{w} - \epsilon \cdot \nabla J(\mathbf{w})$$

where ϵ describes the learning rate. There are some modifications of the gradient descent, which will be introduced later. The gradient descent methods do not find necessarily the global minimum. The gradient descent can find a local minimum or, in the worst case, it can stuck in a saddle point.

In the following, we show the algorithm for the backpropagation. After this we introduce a loss function for the classification problem and then we describe the modification of the gradient descent for a faster convergence.

Backpropagation

The backpropagation is an algorithm that propagates the error from the loss function back through the network to compute the gradient $\frac{\partial J(\mathbf{w})}{\partial w_{j,k}^{(l)}}$. Backpropagation is not the learning algorithm of the ANN, but rather an efficient method to compute the gradient with respect to the weights. The learning algorithm is the gradient descent, which needs the gradient of the loss function to update the weights. The gradient has the following recursively equation:

$$\frac{\partial J(\mathbf{w})}{\partial w_{j,k}^{(l)}} = \delta_j^{(l)} \cdot y_k^{(l-1)}$$

with

$$\delta_j^{(l)} = \begin{cases} \frac{\partial J(\mathbf{w})}{\partial y_j^{(l)}} \cdot \sigma_l'(z_j^{(l)}) & \text{if } l \text{ is the output layer} \\ (\sum_{i=1}^{q} \delta_j^{(l+1)} \cdot w_{i,j}^{(l+1)}) \cdot \sigma_l'(z_j^{(l)}) & \text{if } l \text{ is a hidden layer} \end{cases}$$

where $y_j^{(l)}$ describes the output of unit j in layer l and q is the size of units in layer $l+1$. The function σ_l is the activation function of layer l and $z^{(l)} = \sum_{i=1}^{K} w_{j,i}^{(l)} \cdot y_i^{(l-1)}$ is the activation value from unit j in layer l. In Figure 2.8 is shown one unit of the layer l.

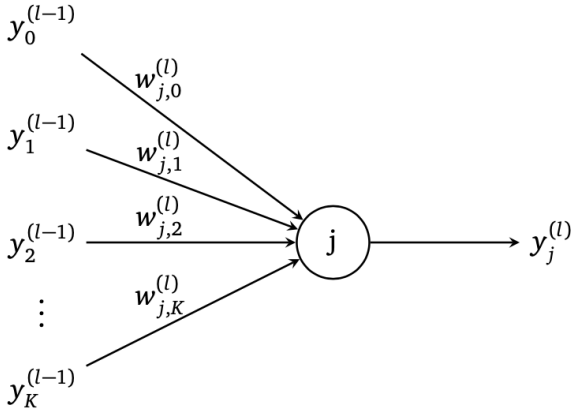


Figure 2.8: Shows one unit of the layer l.

Before the backpropagation process can be started, the forward propagation must be computed to generate the activation values $z^{(l)}$ and the output of $y^{(l)}$. After all gradients are computed, the gradient descent step can be applied, with

$$w_{j,k}^{(l)} \leftarrow w_{j,k}^{(l)} - \epsilon \cdot \frac{\partial J(\mathbf{w})}{\partial w_{j,k}^{(l)}}$$

After this, a new iteration can be started. This is repeated, until an stop criterion is reached. This could be a maximal number of iterations or when the loss reaches a predefined value.

Loss Function

In equation 2.1, we introduced the MSE loss function. This is only an example of many loss functions that can be used. A loss function maps values from one or more variables onto a single value of $\mathbb{R}$, and this value represents the loss. The loss describes the discrepancy between the function $f(\mathbf{x}; \mathbf{w})$ and the target value $\mathbf{y}$. The learning of the neural network means that a minimum of the loss function is achieved, which are described by the loss function. Therefore, the loss function J($\mathbf{w}$) is minimized with respect to $\mathbf{w}$, i.e. the parameters of the neural network.

There exists different loss functions and the choice of the loss function depends on the learning problem. In regression problems, the MSE loss function is a good choice. For the problem of classification it is not practicable. Therefore, loss functions for multi-class problems have been defined. One is cross-entropy loss, also known as log loss. It calculates a loss between two distributions, with

$$L(\mathbf{p}, \mathbf{q}) = -\sum_{i=0}^{K} p_i \cdot \ln q_i \tag{2.2}$$

where $\mathbf{p}$ is the target distribution, $\mathbf{q}$ is the distribution computed by the neural network (prediction) and K is the number of classes.

In fact, for classification, the prediction is a probability vector, meaning it represents predicted probabilities of all classes, summing up to 1. In order to obtain this prediction, the softmax activation function is used at the output layer. We introduce this function later. The target is a one-hot vector, meaning it has 1 on a single position (groundtruth) and 0's everywhere else. The learning goal is now to minimize the loss between this two distributions.

Finally, we would like add a note to Equation 2.2. This is not the final loss function $J(\mathbf{w})$, because this describes only the loss of one example in the training set. Like in Equation 2.1 for the MSE, the average of the cross entropy loss over all training examples is computed.

Gradient Descent

The gradient descent algorithm is an iterative optimization method to find a local minimum of a function, in our case $J(\mathbf{w})$. For a better overview, consider again equation 2.2.3, that is

$$\mathbf{w} \leftarrow \mathbf{w} - \epsilon \cdot \nabla J(\mathbf{w})$$

The choice of the learning rate ϵ is important, because if we use a too small step size, the algorithm may a high number of iterations and it could stuck in a local minimum. If ϵ is too large, it may happen that the domain area with the best solution is bypassed. Hence it is one of the main hyperparameters of a neural network and should be chosen carefully. The starting point is also important, because different points result in different paths to a better (local) minimum.

For the gradient descent some interesting extensions exists.

Stochastic Gradient Descent: The first and most important extension is the descent of the Stochastic Gradient (SGD) with mini-batch. It iterates the gradient descent with a small subset of examples from the training set, instead of the full training set. Normal gradient descent is very impractical as far as computation time and update

rate are concerned. For an update, work out the complete training set, which can contain millions of examples. This means that it has to calculate millions of forward propagations before it can calculate a backpropagation step to update the weights. With SGD with mini-batches, it makes a weight update with a small sample size, but with this huge amount of updates it will find the right direction, which minimizes the loss.

$$\mathbf{w} = \mathbf{w} - \frac{\epsilon}{m} \cdot \sum_{i=0}^{m} \nabla J_m(\mathbf{w})$$

where

$$J(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} ||f(\mathbf{x}_n; \mathbf{w}) - \mathbf{y}_n||^2 = \frac{1}{2N} \sum_{n=1}^{N} J_n(\mathbf{w})$$

At this point, it is appropriated to introduce two common terms in the training of neural networks:

- **Iteration**: one iteration describes one update with the gradient descent. In other words, it is one learning step.
- **Epoch**: one epoch describes one pass through the complete training set. Normally, one epoch consists of several iterations, because the size of the training set is much bigger than the batch size.

Momentum Update: A further extension of the gradient descent is the momentum update. The learning with normal gradient descent is sometimes slow, but it can be accelerate by using the momentum method. The path of the normal gradient descent goes mostly in zigzag lines to the minimum. With the help of the momentum

method, a decaying moving average of the past gradients is estimated. The formula for the gradient descent with momentum update is the following

$$\boldsymbol{\phi} \leftarrow \mu\boldsymbol{\phi} - \nabla J(\mathbf{w})$$

$$\mathbf{w} \leftarrow \mathbf{w} + \boldsymbol{\phi}$$

It introduces a further hyperparameter $\mu \in [0;1)$, which describes the decaying rate of the momentum. A higher value slows down the decaying rate. Commonly values for μ are 0.5, 0.9 and 0.99. The momentum helps to stay in the direction of the previously gradient updates.

ADAM: Adaptive Moment Estimation tries to overcome the problem of updating by modifying the learning rate in a different way for each parameter and according to the stage of learning. It is therefore a member of the family of adaptive methods. In particular, the algorithm calculates the mean of the gradient and of the square of the gradient with exponential decay; the parameters β_1 and β_2 control the decay of these moving averages. The authors recommend the values for these parameters, which are in fact the default values also in every framework that supports Adam.

2.2.4 Activation Functions

The activation function $\sigma(z)$ is a fundamental function of a neural network, which yields the network the power to solve non-linear problems. If the units have no activation function or only a linear activation function, then the neural network could only solve linear problems, no matter how deep the neural network is. So, the activation function should have a non-linear characteristic. A further property for the activation function is that it must be continously differentiable. Otherwise, the backpropagation algorithm will not work, because it cannot compute the gradient.

At the beginnings of the research on neural networks, were saturated activation functions used. This means that the activation was limited to $(0,1)$ or $(-1,1)$. As activation functions the *sigmoid* or *tanh* functions were used. Both have the problem that with small or big values of z the value of the gradient goes to zero, and the convergence of the learning decreases. In other words, the training of the neural network needs much more time. Another problem is the vanishing or exploding of the gradient, if the network is deep. The vanishing gradient occurs in earlier layers, because, through the backpropagation, the gradient will often be multiplied with small values. Many multiplications with small values between 0 and 1 tends toward zero. The exploding gradient, can happen, if the weights are big, and the activation is near 0. At this point the derivative of sigmoid and tanh has reached their maximum.

Recently, new activation functions were developed, like the ReLU. In this section, we show two activation functions that are used in this thesis: the ReLU and the Softmax.

ReLU- Rectifiere Linear Unit: Actually, the Rectifire Linear Unit (ReLU) is the most used activation function for neural networks. The formula for this function is simple

$$\sigma(z) = \max(0, z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$

This means, that the function is 0, if z is negative otherwise the value is z . The ReLU activation is depicted in 2.9. Interesting by the function is linear for positive values and not saturated. A further property, which makes ReLU so popular is the fast computation of the derivative. The derivative is either 0 (at negative values) or 1 (at positive values). Thanks to this fact, the vanishing of the gradient is no more a problem, because a multiplication by 1 doesn't change the error.

Glorot et al. [10] showed, that ReLU has without pre-training better results as tanh or softplus. Furthermore, Krizhevsky et al. [15] have applied ReLU in combination with Convolutional Networks and

have a 6 time faster convergence as with the same network with the activation function tanh.

Batch normalization with ReLU is heavily used in modern network architectures. We introduce batch normalization in the next section. ReLU with batch normalization increases the learning speed of the network.

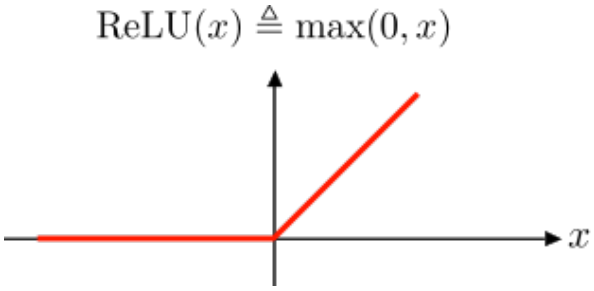


Figure 2.9: Diagram for the activation functions of ReLU and ELU

Softmax: The main application of the softmax activation function, is representing a probability distribution over K classes; therefore it is only used in the output layer. The softmax is applied to the complete output layer and is defined by

$$\sigma(\mathbf{z})_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \qquad \text{for } j = 1, \ldots, K$$

It scales every output of $\sigma(\mathbf{z})_j$ to a range between 0 and 1, such that $\sum_{j=1}^{K} \sigma(\mathbf{z})_j = 1$. This represents a probability distribution. Thanks the exponential function, the largest value in z will be highlighted and the other values will be suppressed. For example, let $z = [8, 5, 3]$, then is the activation $\sigma(z) = [0.9523, 0.0474, 0.0003]$. The value 8 gets the major portion of the available space with respect to the other values.

2.2.5 Regularization

Regularization are methods to control the overfitting or rather to improve the generalization error. There are different methods to apply regularization. Some of the methods are common approaches in machine

learning, whereas others are only usable for neural networks. We describe three regularization methods, which are used in this thesis.

L2-Regularization: Let $\mathbf{w} = (w_0, \ldots, w_N)$ and $\boldsymbol{\omega} = (w_1, \ldots, w_N)$, i.e $\boldsymbol{\omega}$ is the weights vector without the bias weight.
L2-Regularization is a regularization methods, which is added to the loss function $J(\mathbf{w})$. It penalizes the weights $\boldsymbol{\omega}$ from $\mathbf{w}$, but not the bias weights. The regularization term $\Omega(\mathbf{w}) = \lambda \frac{1}{2} ||\boldsymbol{\omega}||^2$ is added to the loss function. This results in a new loss function

$$\tilde{J}(\mathbf{w}) = J(\mathbf{w}) + \Omega(\mathbf{w})$$

which is used to minimize the loss of the neural network. Furthermore, it is added a further hyperparameter λ , which represents the strength of the regularization.

So, by adding the quadratic weight to the loss function, weights with a high value are penalized, because they increase the loss. In other words, the L2-regularization brings the weights closer to zero.

Batch Normalization: The training of a network changes the weights on every layer. This change has the effect that the input distribution changes during updates of previously layers. The authors of batch normalization called this effect internal covariate shift. To counteract the change of the distribution during the learning, they introduced the batch normalization. It is achieved through a normalization step that fixes the means and variances of each layer's inputs. Ideally, the normalization would be conducted over the entire training set, but to use this step jointly with stochastic optimization methods, it is impractical to use the global information. Thus, normalization is restrained to each mini-batch in the training process. Let B to denote a mini-batch of size m of the entire training set. The empirical mean μ_B and variance σ_B^2 of B could thus be denoted as

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2$$

For a layer of the network with d-dimensional input, $x = (x^{(1)}, ..., x^{(d)})$, each dimension of its input is then normalized (i.e. re-centered and re-scaled) separately,

$$\tilde{x_i}^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{{\sigma_B^{(k)}}^2 + \epsilon}}$$

where $k \in [1, d]$ and $i \in [1, m]$. $\mu_B^{(k)}$ and ${\sigma_B^{(k)}}^2$ are the per-dimension mean and variance, respectively. ϵ is added in the denominator for numerical stability and is an arbitrarily small constant. The resulting normalized activation $\tilde{x_i}^{(k)}$ have zero mean and unit variance, if ϵ is not taken into account. To restore the representation power of the network, a transformation step then follows as

$$y_i^{(k)} = \delta^{(k)}\tilde{x_i}^{(k)} + \beta^{(k)}$$

where the parameters $\delta^{(k)}$ and $\beta^{(k)}$ are subsequently learned in the optimization process.

Formally, the operation that implements batch normalization is a transform $BN_{\delta^{(k)},\beta^{(k)}} : x_{1...m}^{(k)} \rightarrow y_{1...m}^{(k)}$ called the Batch Normalizing transform. The output of the BN transform $y^{(k)} = BN_{\delta^{(k)},\beta^{(k)}}(x^{(k)})$ is then passed to other network layers, while the normalized output $\tilde{x_i}^{(k)}$ remains internal to the current layer.

At test time, μ and σ are replaced by the average, which is collected during the training. So, it is possible to predict a single example, without to have a minibatch.
Through the application of batch normalization, the learning rate

can be increased, and this results in a faster training. Furthermore, the accuracy is incread compared to the same network without batch normalization.
The batch normalization helps the activation function ReLU to learn faster and have a better performance. Batch normalization with ReLU is heavily used in modern network architectures [13, 3, 26].

Dropout: Dropout was developed by Srivastava et al. [25] and is a regularization method for neural networks. The key idea is, that units will be randomly dropped for every iteration of the training. This means, that the dropped units and their connection are zero. This should prevent the co-adapting of the units. During the testing the dropout is not available, and all units are used for the predicting.

Srivastava et al. [25] tested dropout on different datasets and they had on all an improvement of performance. The drawback of dropout is the increasing training time, because not all weights a trained in one iteration. If one unit is dropped, then are all depending gradients are zero and therefore the corresponding weight will be not updated.

Dropout could be seen as a training of a subset of the model. Every iteration builds a different version of the model, and every weight is updated with another set of weights. This prevents the co-adapting of the units, because a unit cannot more rely that another unit is available.

A similar dropout is the Spatial Dropout, which is introduced by Tompson et al. [27]. It is used in convolution neural networks to drop complete feature maps and not only single units. This brings us to the next section, where we introduce convolution neural networks.

2.3 Convolutional Neural Networks

Convolutional neural networks, which we will refer to with the abbreviation CNN, are an evolution of the standard deep artificial networks

characterized by a particular architecture that is extremely advantageous for visual (and non-visual) tasks, which has made them very effective and popular over the years. They were inspired by the biological research of Hubel and Wiesel [14] who, studying the brains of cats, discovered that their visual cortex contained a complex structure of cells. The latter were sensitive to small local parts of the visual field, called receptive fields. They thus acted as perfect filters for understanding the local correlation of objects in an image. As these systems are the most efficient in nature for understanding images, the researchers attempted to simulate them. Modern CNN were proposed by Yann LeCun et al in 1998 [17]. The information for the CNN are taken from the the Stanford Lecture CS231n [1].

2.3.1 Architecture

CNNs are deep neural networks made up of different layers that act as feature extractors and a fully connected network at the end, which acts as a classifier, as shown in Figure 2.10.

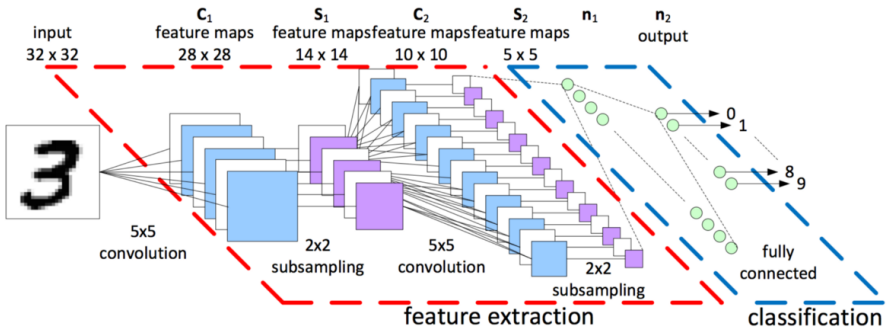


Figure 2.10: Architecture of a CNN that classifies numbers: the division between the layers that act as feature extractor and the final classifier is highlighted

The layers where image features are extracted are called convolution layers and are generally followed by a non-linear function and a pooling step. Another example of image processing layers, may be the contrast normalization layer, see Figure 2.11.

Convolution and pooling have the purpose of extracting the characteristics, while the non-linear unit serves to strengthen the strongest

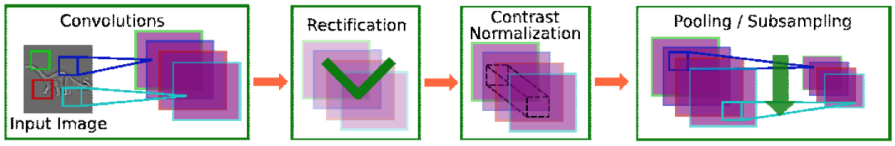


Figure 2.11: The different layers typical of a CNN

characteristics and weaken the less important ones, that is, those that have stimulated the neurons less. From Figure 2.10, we can also note that, for each input image, different groups of images correspond in the various layers, which are called feature maps. The feature maps are the result of the convolution operation carried out through a bank of filters, also called kernels, which are matrices with useful values to search for certain characteristics in the images. Finally, once the convolutional layers are finished, the feature maps are "unrolled" into vectors and given to a "classical" neural network which performs the final classification.

Convolutional Layer

A digital grayscale image can be considered as a two dimensional matrix real or discrete values. Each value of the matrix is called a pixel and its indices are also called coordinates: each pixel $\boldsymbol{X}(m,n)$ represents the intensity in the position indicated by the indices. A “filter” or “kernel” is defined as a matrix $\boldsymbol{W}$, usually (3x3), with weights to be applied to the image through a transformation in order to produce a filtered output. The transformation is carried out through the convolution operation between the input image and the filter. The convolution, discrete in the case of digital images, can be defined as:

$$\boldsymbol{F}(m,n) = \boldsymbol{X}(m,n) \circledast \boldsymbol{W}(m,n) = \sum_{j=-\infty}^{\infty} \sum_{i=-\infty}^{\infty} \boldsymbol{X}(i,j) \cdot \boldsymbol{K}(m-i, n-j)$$

where $\boldsymbol{F}$ is the feature maps. Each pixel of $\boldsymbol{F}$ is thus the result of a weighted sum by $\boldsymbol{K}$ of the sub-region which has its center in the pixel indicated by the coordinates m, n. An example of a convolution is shown in Figure 2.12.

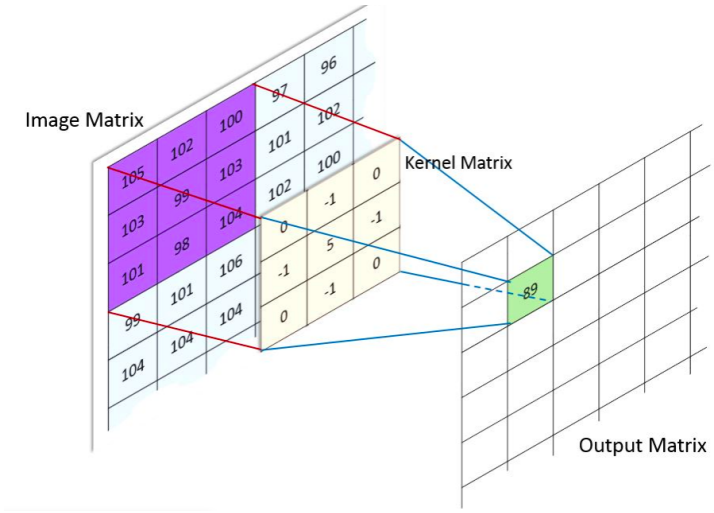


Figure 2.12: Example of convolution

Actually, a convolutional layer does not only take in two-dimensional matrix as in the case of grayscale images, but three-dimensional matrix. In fact, a color image has normally three channels (red, green, blue), and every channel is described by a two dimensional matrix. The output of a convolution layer is again a three dimensional matrix, with feature maps of two dimensions and this x times for the number of filters for this layer. Every filter produces a feature map. In Figure 2.13 an example is provided. The two figures show the connection between the input and two output units of the feature map. The input is showed with three channels. This could be the color channels R,G,B of an image, or the feature maps of a previous layer. The input is described by a three dimensional matrix. As output one feature map is depicted. For more feature maps the behavior is similar. The activation function and the bias are omitted for this example, but would be also applied on the units. The bias are shared, which means, that one bias weight are used for all units in one feature map. The weights $\boldsymbol{W}_{i,o}$ are a 3×3 matrix, which describe one kernel, between channel i of input image $\boldsymbol{X}_i$ and the feature map $\boldsymbol{F}_o$. The weights $\boldsymbol{W}_o$, describe one filter and are shared for one feature map. This means, that the same weights are used for the computation of all units in one feature map. Then for convolutional layer with input three-dimensional matrix the output or $o - th$ feature map is calculated with:

$$\boldsymbol{Z_o} = \sigma(\sum_{i=0} \boldsymbol{X}_i \circledast \boldsymbol{W}_{i,o} + b_o \mathbf{1}) \tag{2.3}$$

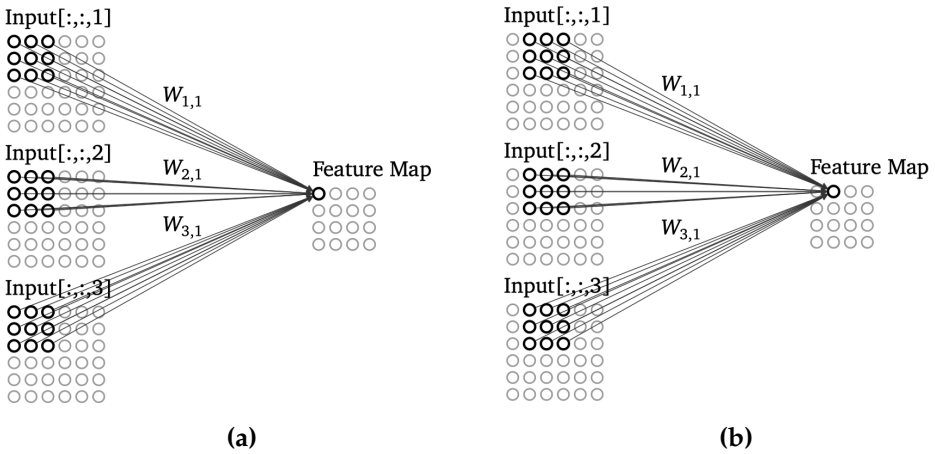


Figure 2.13: The two figures show the connection between the input and two output units of the convolution layer; the convolution layer has only one feature map at the output. The activation function and the bias are omitted for a better overview. The local connectivity is depicted in the two figures, with a kernel size of 3×3. (a) and (b) shares the same weights for the computation of the feature map. For another feature map, it would be used new weights. $\boldsymbol{W}_{i,o}$, defines a 3×3 weight matrix, which is used between the input channel i and the output channel o. The output channel is called feature map.

The convolution in a convolution layer are multiple convolutions, as are added up, which we can see in Equation 2.3. Without the addition of the convolutions of the input channels, we would not get a combination of features, which are essential for a CNN.

As already mentioned, these convolutions are performed between the input image/s and an arbitrary number of filters. These filters have values such as to obtain recognition of certain characteristics at the output. The values of the filters are initially chosen randomly, and are then improved at each iteration using the backpropagation algorithm, seen in 2.2.3. By doing so, the network trains its filters to extract the most important features of the examples of the training set.

There are several hyperparameters to set in the convolution layers:

1. The measure of the filter F: also called receptive field. Each filter looks for a specific feature in a local area of the image. Typically they are 3x3, 5x5 or 7x7.
2. The number K of filters: for each layer, this value defines the depth of the output of the convolution operation. In fact, by placing the feature maps one on top of the other, you get a cube in which each "slice" is the result of the operation between the input image and the corresponding filter. The depth of this cube depends precisely on the number of filters.
3. The *stride* S: defines how many pixels the convolution filter moves at each step. If the stride is set to 2, the filter will skip 2 pixels at a time, thus producing a smaller output.
4. The *padding* P: define the amount of pixels added to an image when it is being processed by the kernel of a CNN or pooling layer. It is used to preserve the output size. In general, when the stride $S = 1$, a value of $P = (F - 1)/2$ guarantees that the output will have the same size as the input.

We can calculate the number of parameters p in a convolutional layer knowing the number of input channels I and the value of the hyperparameters, with

$$p = (F^2 \cdot I + 1) \cdot K$$

When processing images with CNNs, three-dimensional inputs are generally received, characterized by the height H_1, the width W_1 and the number of color channels D_1. Knowing the parameters specified above, you can calculate the size of the output of a convolution layer:

$$H_2 = (H_1 - F + 2 \cdot P)/S + 1$$

$$W_2 = (H_1 - F + 2 \cdot P)/S + 1$$

$$D_2 = K$$

In this regard, observe an example of "volume of neurons" of the first convolution layer in Figure 2.14. Each neuron is spatially connected

only to 1 local region of the input but for the whole depth (i.e. the 3 color channels). Note that there are 5 neurons along the depth and they all look at the same input region.

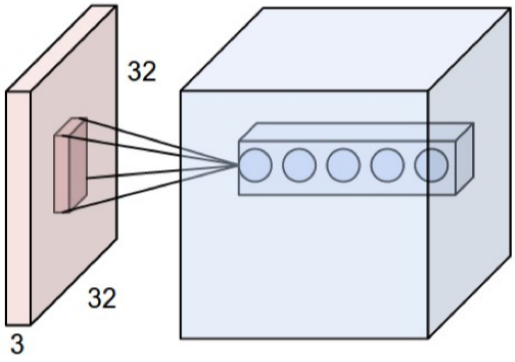


Figure 2.14: Each neuron is connected to only 1 local region of the input but at all depth (i.e. color channels). The depth of the output is given by the number K of filters, in this case 5

ReLU Layer

After the convolution layer the features maps are passed in input to the ReLU 2.9 layer. The ReLU layer is very important for a CNN. This can be mainly due to 2 reasons:

1. the polarity of the features is very often irrelevant to recognize objects
2. the ReLU prevents two characteristics, both important but with opposite polarity, from canceling each other when pooling.

Pooling Layer

Another property that you would like to have to improve results on artificial vision is the recognition of features regardless of the position in the image, because the goal is to strengthen the effectiveness against translations and distortions. This can be achieved by decreasing the spatial resolution of the image, which favors a higher computational speed and is at the same time a countermeasure against overfitting, since the number of parameters decreases.

The pooling layer gets N images of a resolution in input and outputs the same number of images, but with a resolution reduced to some extent, usually by 75%. In fact, the most common form of pooling layer uses $2x2$ filters, which divide the image into non-overlapping 4-pixel areas and choose a single pixel for each area.

The criteria by which to choose the winning pixel are different:

- *Average pooling*: the average value over the pixels of the pool is calculated
- *Median pooling*: the median of the pixel values in the pool is calculated
- *LP-pooling*: the p-norm of the pixel matrix is calculated
- *Max pooling*: the pixel with the highest value is chosen

Of these, the one that has proved most effective is max pooling, Figure 2.15. Across the network, you will gradually have a higher number

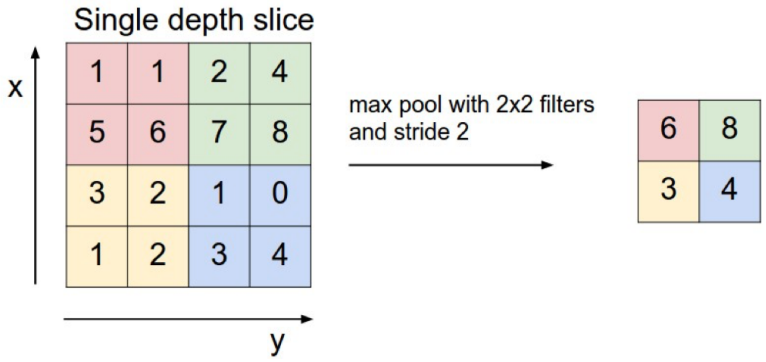


Figure 2.15: Max pooling: at the output the image will have 1/4 of the starting pixels.

of feature maps and a decrease in input resolution. These factors combined together give a strong degree of invariance to the geometric transformations of the input.

Fully connected Layer

In the fully connected layer, all inputs from the convolution layers are fed into a normal, fully connected neural network that will act as a classifier.

Figure 2.16 shows the complete architecture of a CNN. Note how the resolution of the image is reduced at each pooling layer (also called subsampling) and how each pixel of the feature maps derives from the receptive field on the set of all the feature maps of the previous level.

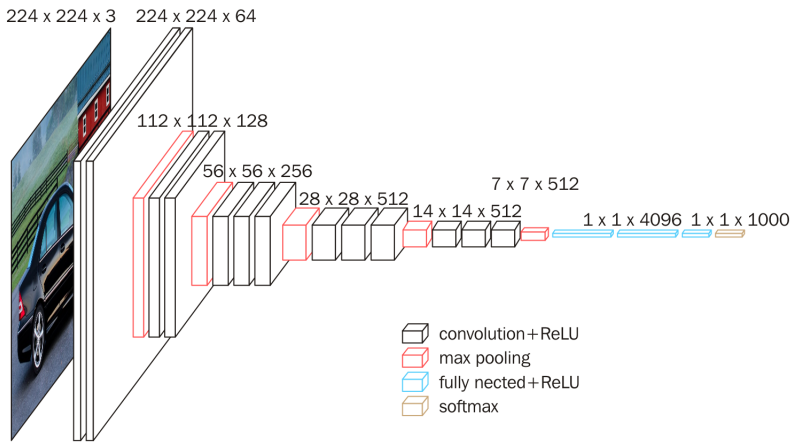


Figure 2.16: CNN Architecture

Figure 2.17, instead, shows a CNN in the classification deed of a car. The network filters are displayed during the various levels of processing and then terminate in a completely connected layer that gives a probability in output. This probability is then translated into a score, from which the winning class is chosen.

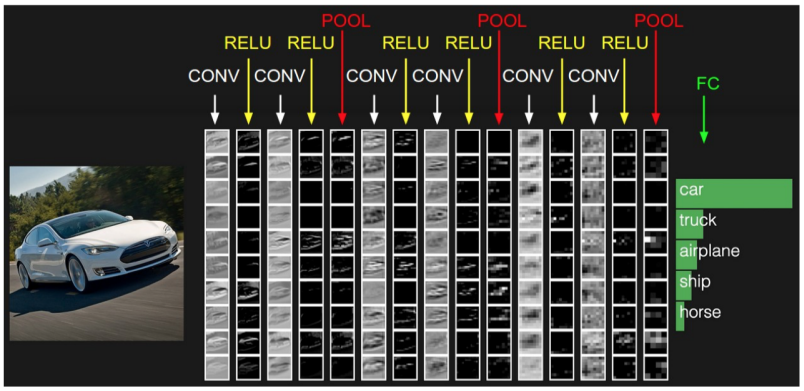


Figure 2.17: Typical CNN in a classification task; the winning class is the one with the highest probability, indicated at the end.

2.3.2 Depth-wise Separable Convolution

A standard convolution layer of a neural network, as seen previously, involve (without considering the bias) $F^2 \cdot I \cdot K$ parameters. For an input channel of 3 and output of 256 with 5x5 filter this will have 19200 parameters. Having so much parameters increases the chance of overfitting. To avoid such scenarios, have been proposed different convolutions.

Depth-wise Separable Convolution originated from the idea that depth and spatial dimension of a filter can be separated - thus the name separable. This type of convolution separates the process in 2 parts: a depthwise convolution and a pointwise convolution.

1. **Depthwise Convolution**: In this first part we operate a convolution on the input image without changing the depth. We do so by using I kernels of shape $QxQx1$, where a typical choice for Q is 3 or 5. For example (see Figure 2.18), if we have an input image of 12x12x3, we can use 3 kernels of shape 5x5x1 in order to get a 8x8x3 feature map (convolutions with no padding and a stride of 1).

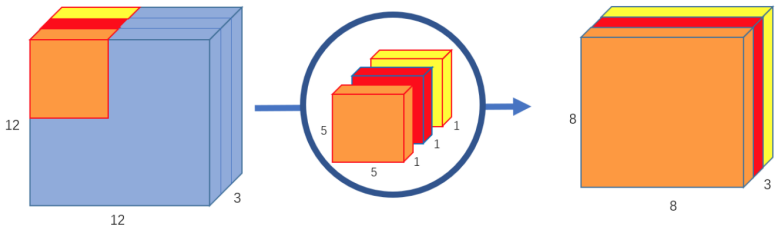


Figure 2.18: Depthwise Convolution

2. **Pointwise Convolution**: The pointwise convolution is so named because it uses a 1x1 kernel, or a kernel that iterates through every single point. This kernel, however, has a depth of as many channels as the input image has. We can create S $1x1xI$ kernels in order to increase the number of channels of each image. For example (see Figure 2.19), if we have a input image of 8x8x3, we can use 256 kernels of shape 1x1x3 in order to get a 8x8x256 feature map (convolutions with a stride of 1).

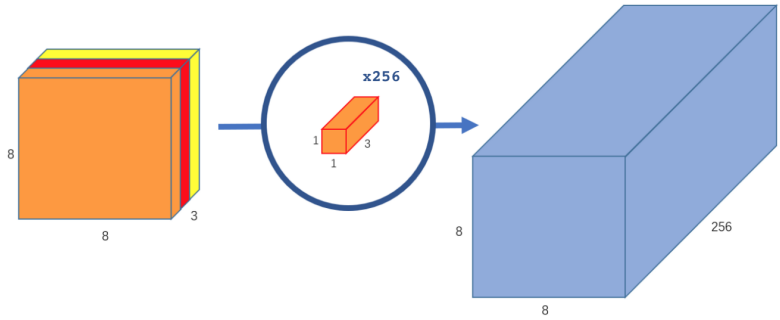


Figure 2.19: Pointwise Convolution

In the above example, the number of parameters using Depth-wise Separable Convolution is: 3x5x5x1 + 1x1x3x256 = 843, much less than the normal convolution, that has 19,200 parameters.

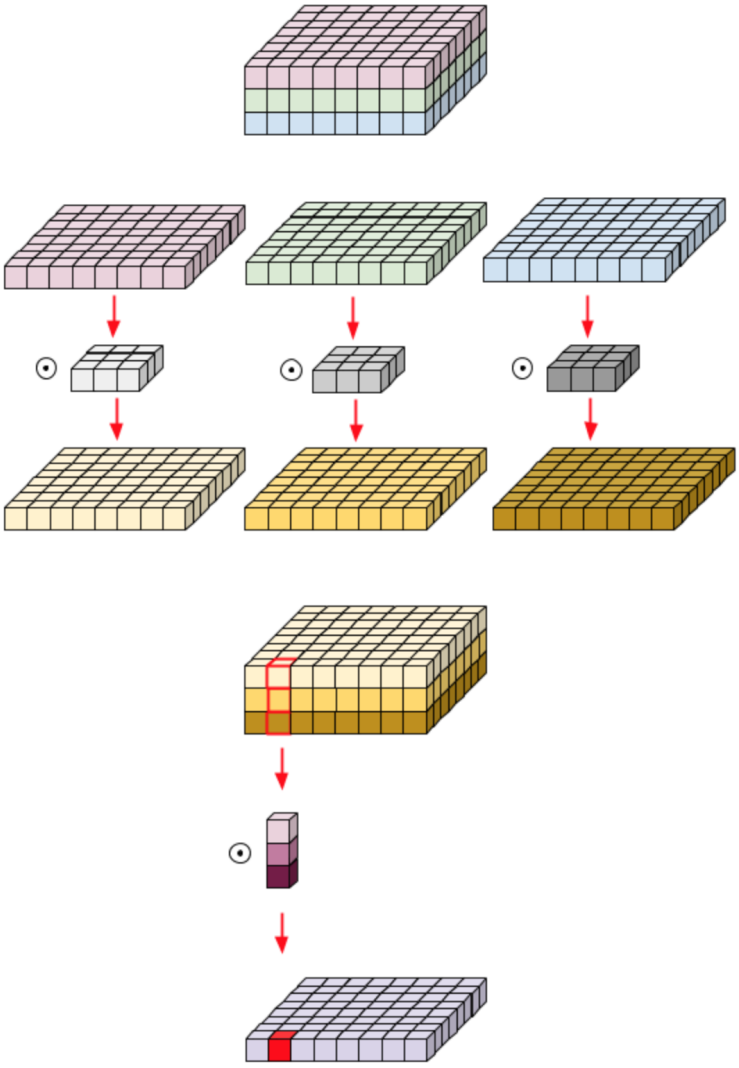


Figure 2.20: Depth-wise Separable Convolution

3

The Datasets

In this thesis two different dataset have been used, one containing annotated images of Egyptian hieroglyphs found in the Pyramid of Unas and one containing images selected and annotated by Prof. Massimiliano Franci. Before the images are fed into the Neural Network, the images contained in these dataset should be merged into a single dataset, preprocessed and augmented. The preprocessing steps consist of change the colorspace, denoising and scaling. This chapter starts with an overview of the initial dataset and after this, the preprocessing and data augmentation techniques will be presented.

3.1 Images

The first dataset (D_1) contains 4310 grayscale photos of dimentions 50x75 taken from the walls inside the pyramid of Unas. The photos are taken from the book *The Pyramid of Unas* by Alexandre Piankoff [22]. Each photo rappresent a hieroglyphs and it is labeled with the class hieroglyph it belongs to. In this dataset there are 172 different classes.

The second dataset (D_2) contains 1310 rgb images of variable dimensions with 48 different classes. Also in this case, each image rappresent a hieroglyph and it is labeled with the class it belongs to.

The hieroglyphs in the images were labelled according to the Gardiner Sign list [9], where each hieroglyph is labelled with an alphabetic character followed by a number. The alphabetic character represents the

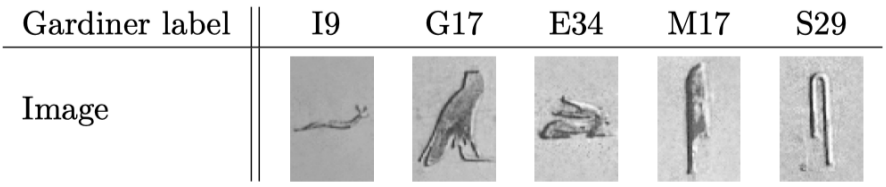


Figure 3.1: Examples of some images labeled according the Garniner Sign list

class of the hieroglyph, for example, class 'A' contains hieroglyphs about a man and his occupations while class 'G' contains hieroglyphs of birds. The entire Gardiner Sign list consist of more than 1000 hieroglyphs in 25 classes.

Starting from these two datasets we then obtained a single dataset D. The joint dataset is not just simple the union of the D_1 and D_2 datasets. We decided to take only the images that belong to classes contained in both datasets.
So, let

$$D_1 = \{(x_i^{(1)}, y_i^{(1)})\} \quad \text{with} \quad i = 1, \ldots, |D_1|$$

$$D_2 = \{(x_j^{(2)}, y_j^{(2)})\} \quad \text{with} \quad j = 1, \ldots, |D_2|$$

and let Y_1 , Y_2 the set of all the different labels in D_1 e D_2.
Then,

$$D = \{(x_k, y_k)\} \quad \forall y_k \in Y_1 \cap Y_2$$

$|Y| = |Y_1 \cap Y_2|$ is the number of classes contained in our dataset D. In our case there are 40 different classes, so $|Y| = 40$. In figure 3.2 we can see a histogram that shows the number of images for each class. It is possible to see that the dataset is quite unbalanced. This is one of the reasons why we used data augmentation.

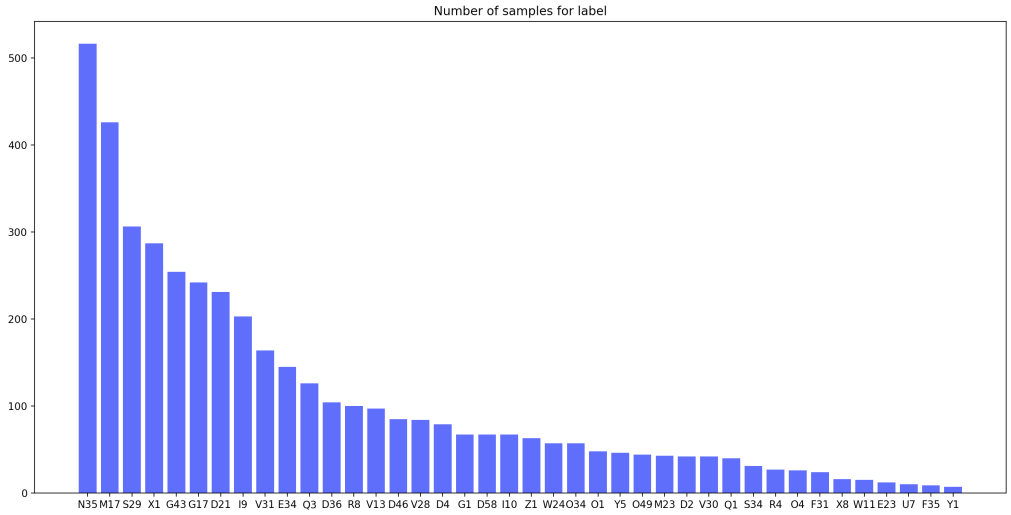


Figure 3.2: Number of images for each label

3.2 Preprocessing

In this section, we will see the preprocessing techniques used to obtain a good starting dataset for the training of our neural network. The goal is to create a dataset containing images of the same size in grayscale. The size chosen for the images is 100x100. This choice is due to the fact that some models of neural networks do not adapt to input dimensions smaller than a certain value.

To achieve this goal the image must pass through the preprocessing pipeline, which consists of three different steps, as depicted in Figure3.3

However not all images will perform all steps. In fact, the images of D_1 are already in grayscale and do not require to perform the denoise step.

We apply denoising to images that have a marginal noise that differs from their background. As we can see from figure 3.4, some images of D_2 show this border noise .

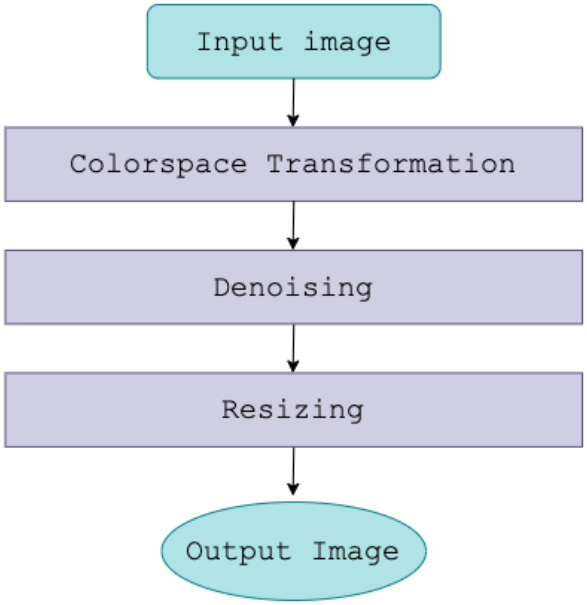


Figure 3.3: Flowchart of the Preprocessing Pipeline.

Figure 3.4: Some figures with border noise

To remove this noise, (see Figure 3.5), a background image was generated following a distribution of a Gaussian field of mean μ and standard deviation σ equal to the pixels intensity mean and standard deviation of the original image without considering the figure and the border. The size of the background is chosen as the smallest square shape that can hold the original image. In this way the image will keep its aspect radio even after resizing without induce a loss in height/width information from the original hieroglyph. After getting the output image, it will be resized to 100x100.

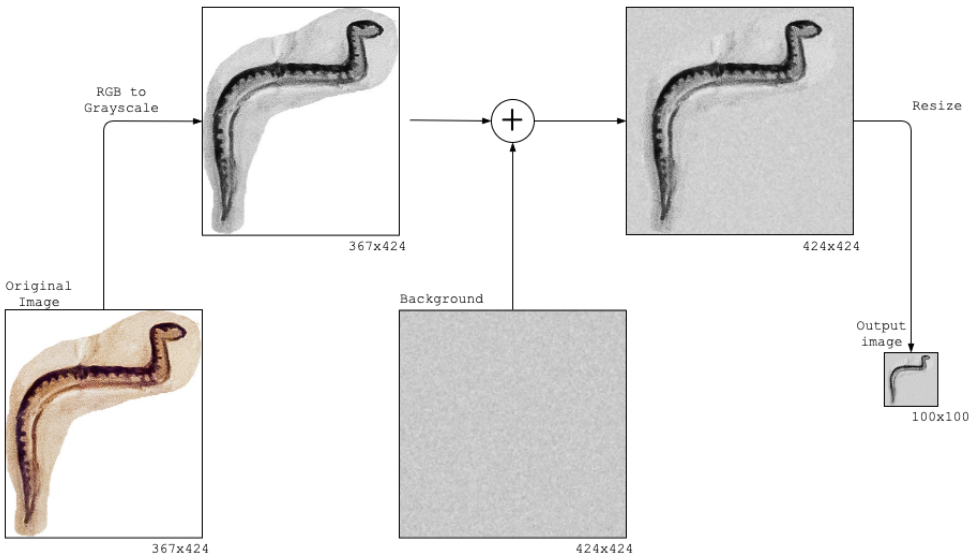


Figure 3.5: Example of image filled background and resize

3.3 Train Validation and Test Sets

After having carried out the preprocessing of the images and having obtained a single dataset, it was necessary to divide the latter into three parts. Validation and test sets were separated from the rest of the dataset, as they must not be augmented. The only augmentation technique that will be used on the test set is the horizontal flip, in order to verify that the network generalizes well. The size chosen for the test and validation set is 15% for each one of the entire dataset. The remaining part is the training set. It, unlike the latter, was subjected to all the data augmentation techniques that we will see later.

3.4 Data Augmentation

Augmentation is widely used in Deep Learning to improve the results [7]. It increases the size of the training set and, if it is unbalanced, it can improve the balance. This results in reducing the overfitting and helps to increase the performance of the CNN. Transformations, like rotation and zooming, helps the CNN to be invariant toward rotation and different scales.

Augmentation transforms the image, so that a new image is produced. This could be reached with a simple affine transformation like translation, zooming or rotation.

In order to rebalance the training set,it was increased according to the quantity of images present for each label. The number of images that will be increased for each class is:

$$n_{aug}^{(C_i)} = \begin{cases} \frac{\sum_{i=0}^{N} |C_i|}{N} - |C_i| = \mu - |C_i| & \text{if } |C_i| < \mu \\ 0 & \text{otherwise} \end{cases}$$

where $n_{aug}^{(C_i)}$ is the number of images that will be increse for the class C_i, $|C_i|$ is number of images of class C_i in D, and $|D|$ is the size of dataset D.

After that the training set was downsampled. The maximum number of images that a class can contain is set to two times the average of the image for label into the training set, μ.

In Figure 3.6 we can see the augmentation and the downsampling on the train dataset.

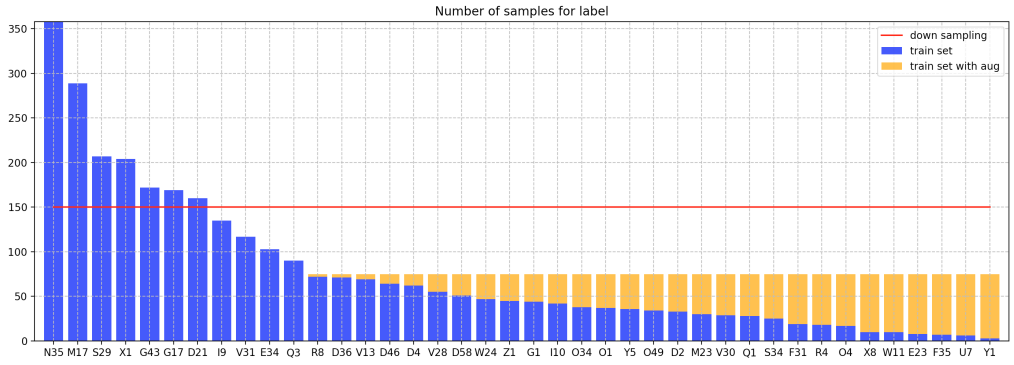


Figure 3.6: Data augmentation and downsampling on the train dataset

We implemented different augmentation transformation, which can be independently used.

The following list describes the used augmentations:

Translate: The image is randomly translated to the x-axes and y-axes in an appropriate range.

Zoom: The image will be zoomed in an appropriate range. It helps the Neural Network to be invariant for different scales of the hieroglyphs.

After applying this, all images in the dataset have been horizonally flipped. Make the network invariant to the horizonal flipped operation is very important, as the ancient Egyptians allowed the hieroglyphs to be written in 3 directions, either from left to right, right to left and from top to bottom. For this reason it is possible to find the same hieroglyph facing both left and right.

4

Image classification

In this chapter we will present the model architectures for the classification of Egyptian hieroglyphs, which are used in the next chapter for evaluation. At the beginning of the search for a good CNN for the task, we tested three existing models, by using both the transfer learning method and normal training, used for one of the big classification problems which was the ImageNet challenge. The three CNN performance was very good. We compared the various architucture in terms oh both performance and computational cos and, we developed our model architecture in order to improve this two factors.

4.1 Transfer Learning

The main reason for the success of transfer learning [28] is that models trained on huge dataset learn general features that facilitate the training process with small datasets, often obtaining better performances with respect to models trained from random weights. Transfer learning consist of using a pre-trained model, or a subpart of this, which is transferred to another task. Two common approaches for transfer learning exist:

1. A first way for transfer learning uses the pre-trained model as feature extractor [21, 23]. This means that the pre-trained model is

used up to a specific layer for the generation of features. These features are then used as input of another machine learning algorithm to train the other task. It is also possible to build on the top of the transferred layers some further layers, which are then learned, but the transferred layers are kept frozen. So, the task of the pre-trained model is only to extract the features, and the features are then used to train the task.

2. The second approach is the fine-tuning [29]. The pre-trained model or only a part of the model is transferred and on the top will be added further layers. But now, the weights of the transferred layers will be also trained. It is also possible, to keep some lower layers of the transferred model frozen.

What is the benefit of the transfer learning? The pre-trained models are mostly learned on big datasets with millions of images. This has the benefit that the model have learned useful features in the lower layers. These features could be useful for similar tasks. A nice side effect is, that the training is faster, if the pre-trained model was applied on a similar tasks, because the weights are initialized in a suitable way.

In this thesis, we used the first approch. So, we have take the pre-trained model layers without the last layers (often fully connected layers) and we have freezem, so as to avoid destroying any of the information they contain during future training rounds. On top the pre-trained model we have trained fully connected layers. It's composed by GlobalAveragePooling layer, Dense layer with 128 units, Dropout with 0.15 and Dense layer with 40 (number of class). As output layer, we use a softmax activation. After that, the new layers are trained on our training set.

4.2 Model used

The models used as tests are ResNet-50, Inception-v3, and Xception. All of them have achieved excellent results on the ImageNet challenge.

4.2.1 ResNet50

ResNet, which was proposed in 2015 by researchers at Microsoft Research [13] introduced a new architecture called Residual Network, in order to solve the problem of the vanishing/exploding gradient. In this network, we use a technique called skip connections . The skip connection skips training from a few layers and connects directly to the output. The approach behind this network is instead of layers learn the underlying mapping, we allow network fit the residual mapping. So, instead of learning a direct mapping of $\mathbf{x} \rightarrow y$ with a function $H(\mathbf{x})$ (a few stacked non-linear layers), let us define the residual function using $F(\mathbf{x}) = H(\mathbf{x}) - \mathbf{x}$, which can be reframed into $H(\mathbf{x}) = F(\mathbf{x}) + \mathbf{x}$, where $F(\mathbf{x})$ and $\mathbf{x}$ represents the stacked non-linear layers and the identity function (input=output) respectively. The author's hypothesis is that it is easy to optimize the residual mapping function $F(\mathbf{x})$ than to optimize the original, unreferenced mapping $H(\mathbf{x})$.

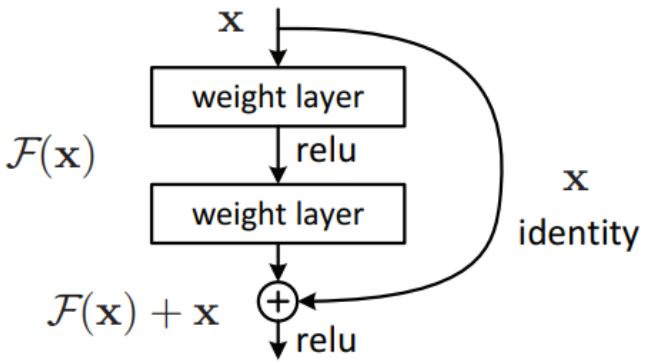


Figure 4.1: Residual block

This network was designed to process images with the size of 224 × 224 pixels. We used the Resnet50 with the same image size in order to obtain maximum performance.

The architecture of ResNet50, depicted in Figure 4.2, contains 50 layers. Stages 1-4 contain blocks of length 3, 4, 6 and 3 respectively, where each block consists of three convolutional layers. The ResNet-50 has over 23 million trainable parameters.

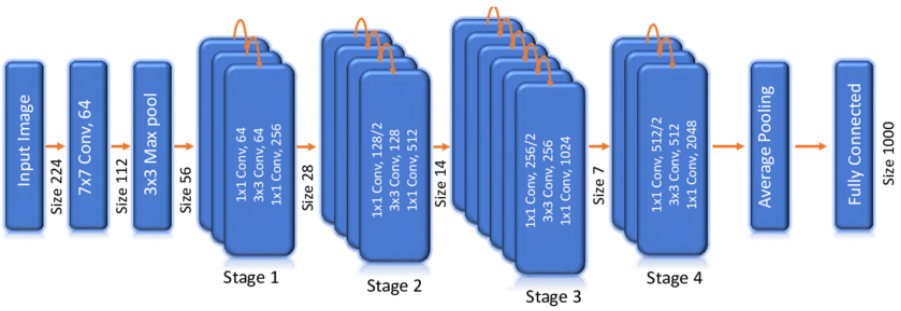


Figure 4.2: ResNet-50 architecture

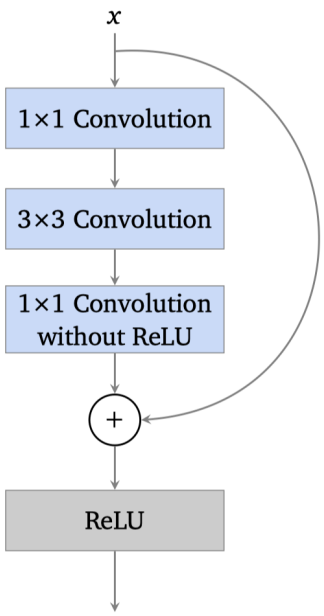


Figure 4.3: Example of a residual block in ResNet-50

For a detailed description of the network, we recommend the paper of Kaiming He et al. [13].

4.2.2 Inception-V3

The Inception-v3 model is the third version of the GoogleNet model. The network has a relative depth of 42 layers. All convolution layer uses the activation function ReLU, and apply the batch normalization. The Inception-v3 starts with default convolution and pooling layers, to reduce the dimensional size of a feature map. Then it starts with the

main characteristics of the network, the inception modules. The inception modules are parallel running convolution layers with different kernel sizes. One type of inception block is depicted in Figure 6.1, but there are two further types of inception blocks, which follows the same principles, but with other kernel sizes. The idea behind the inception module is, that it can extract similar features on different scales. The 1×1 kernel sizes has the purpose of the reduction of the number of feature maps, before is applied the expensive 3×3 convolution layer. For a detailed description of the network, we recommend the paper of Szegedy et al. [26].

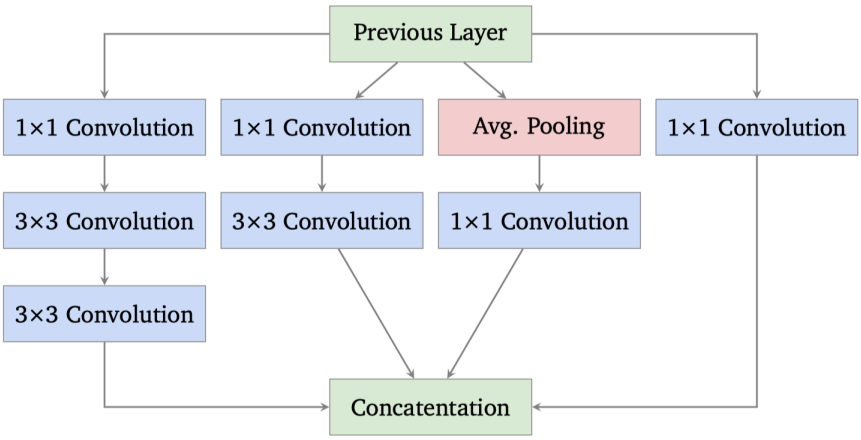


Figure 4.4: Example of an inception module in Inception-v3. The two successively 3×3 convolution layers have the same receptive field as a 5×5, but with the benefit, that they are faster to compute. The average pooling is executed with a stride of (1,1), so the pooling has no down sampling character. Furthermore, all layers applying padding, to produce the same spatial dimension of the feature maps, which are concatenated at the end of the inception module.

The network was designed for the ImageNet challenge and process images with the size of 299×299 pixels. We used the Inception-v3 with the same image size in order to obtain maximum performance.

4.2.3 Xception

Xception, stands for Extreme version of Inception. In this new version a canonical Inception Block, Figure 4.5(a), is simplified with a extreme

version of Inception Block, Figure 4.5(b).

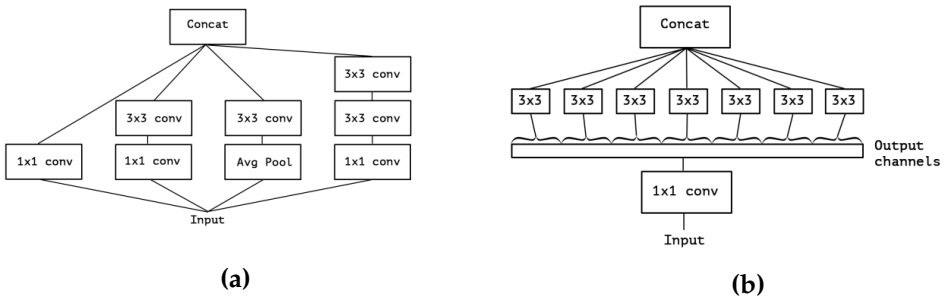


Figure 4.5: (a) A canonical Inception module (Inception V3). (b) An "extreme" version of our Inception module, with one spatial convolution per output channel of the 1x1 convolution.

This extreme form of an Inception module is almost identical to a depthwise separable convolution seen in section 2.20. The only difference is the order of the operations: depthwise separable convolutions as usually implemented (e.g. in TensorFlow) perform first channel-wise spatial convolution and then perform 1x1 convolution, whereas Inception performs the 1x1 convolution first. However, the authors argue that this difference is unimportant, in particular because these operations are meant to be used in a stacked setting. So the order of the operation can be changed.

Therefore this architecture relies on two main points :

- Depthwise Separable Convolution
- Skip connection between Convolution blocks as in ResNet

Both of these points have already been covered in section 2.20 and 4.2.1 respectively. In figure 4.6 we can see the Xception architecture. For a detailed description of the network, we recommend the paper of François Chollet [3].

As InceptionV3, the net was designed to process images with the size of 299 × 299 pixels. So, we used this network with the same image size in order to obtain maximum performance.

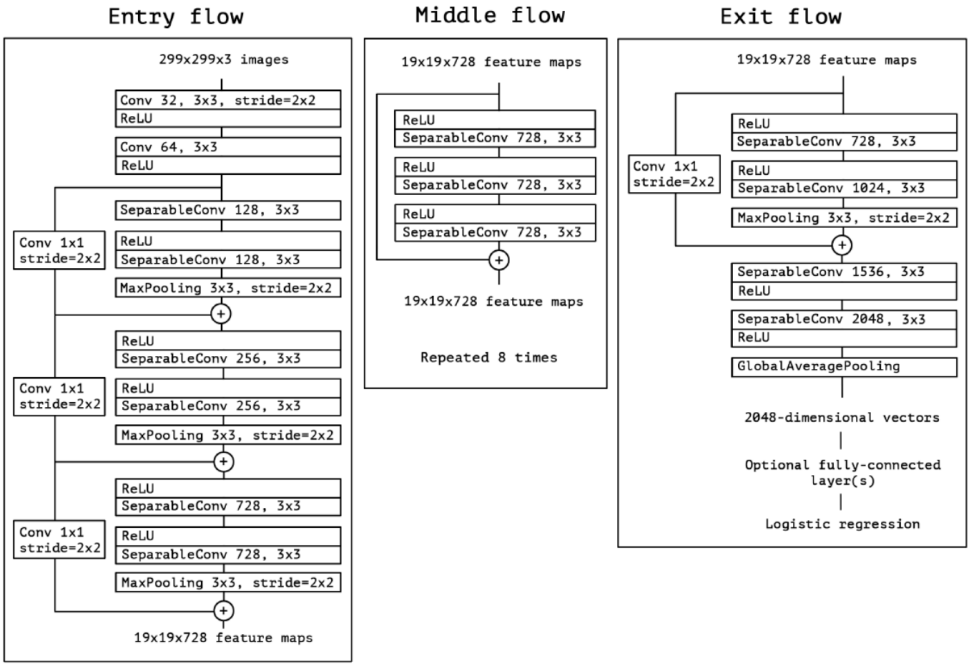


Figure 4.6: Xception architecture

4.3 Our Model

Our principal goal was to develop a model, with similar performance, as those models previously described. Furthermore, our model should be faster than these models.

Designing a CNN includes tuning several hyper-parameters. Hyperparameters can be divided into two types:

1. Hyperparameters that determine the network structure such as:
    - *Kernel Size* –the size of the filters.
    - *Stride* –the rate at which the kernel pass over the input image.
    - *Padding* –the amount of pixels added to an image when it is being processed by the kernel of a CNN or pooling layer.
    - *Hidden layer* –layers between input and output layers.
    - *Activation functions* –allow the model to learn nonlinear prediction boundaries.

2. Hyperparameters that determine the trained network such as:
    - *Learning rate* –it regulates the update of the weights at the end of each batch.
    - *Learning rate decay* -function to decrese the learning rate value every x loop
    - *Gradient Descend method* - optimization algorithm for finding a local minimum to train the net (see 2.2.3)
    - *Regularizations* - methods to control the overfitting and improve the generalization error (see 2.2.5).
    - *Number of epochs* –iterations over the entire training dataset.
    - *Batch size* –number of patterns shown to the network before the weights are updated.

In the next paragraphs we will illustrate the choices of these hyperparameters.

4.3.1 Architecture

To reduce the computation time in a effective way, we decided to use a default input size of 100×100 pixels. At this resolution, humans can easily recognize the details of the hieroglyphic, indicating that this resolution is not too strict for recognition.

At the basis of our model, there are the Separable Convolutional Layer 2.3.2, a feature present in Xception model, which reached the best performance on the task.

The network consists of 6 blocks. The first is the input block and have two standard convolution layers with 64 number of filter and a kernel size of 3x3 . In this block there are two max pooling layer with a kernel size of 3x3 and a stride of 2x2. This choice allows to reduce the spatial dimension of the feature maps and therefore to reduce the computation time. The last block is the output block and have a separable convolution layer with 512 filter followed by dropout regularizator, fully connected layer and softmax. In the middle there are four blocks

Params in architecture		
Architecture	**Total**	**Trainable**
Our	498,856	494,504
Resnet50	23,663,400	23,610,280
InceptionV3	21,884,168	21,849,736
Xception	20,942,864	20,888,336

Table 4.1: Number of params for CNN architecture

and each of these has two separable convolutional layers. The first two have 128 filters while the last two 256.
All convolution layer are followed by batch normalization. ReLu functions are used after batch normalization or max pooling.

Compared to previous architectures, this have much less trainable parameters (see Table 4.1). It only has 498,856, of which 494,504 trainable.

The architecture of the model is showed in Figure 4.7

4.3.2 The training

Our architecture was implemented and trained using Keras [4]. Keras is an open-source software library that provides a Python interface for artificial neural networks. Keras acts as an interface for the TensorFlow [2] library. TensorFlow is an end-to-end open source platform for machine learning. It can run on numerous types of CPUs and even on GPUs, thanks to the support of languages such as CUDA or OpenCl. Keras abstracts the concepts of TensorFlow, so that it is easy to model a layered network. Furthermore, it provides the possibility, to train the defined model.

In order to train our model it was necessary to choose some hyperparameters (see 4.3) and the right loss function. The loss function used to optimize the model during training is the categorical cross entropy. As optimization method ADAM(Adaptive moment estimation) is used, a variant of the stochastic gradient descent (SGD), with a batch size of 32.

To improve the generalization we have used four regularization methods. The first is the L2-Loss to regularize the weights in the fully connected layer, the second is an adaption of the learning rate, the third is the dropout layer and the four is the batch normalization after each convolution. The adaption of the learning rate decreases the learning rate by a factor of 2, every 15 epochs. The initial learning rate was set to is 0.001. This value and the learning rate decay parameter were chosen empirically, after having performed several test with different values.

4.3.3 Testing

The validation and test sets, which we described in 3.3, were used to validate and test the trained model.

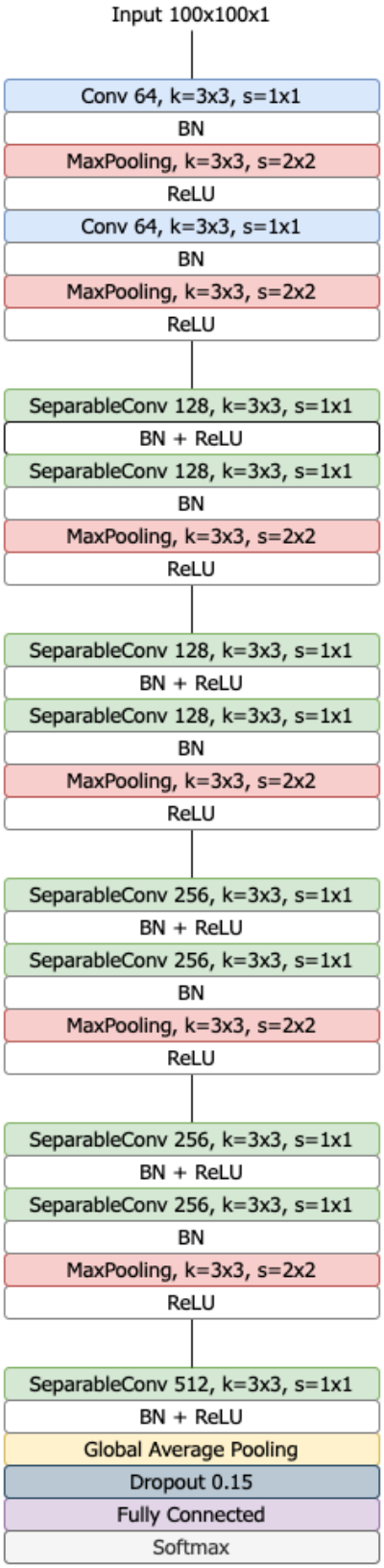


Figure 4.7: Our model architecture

5

Results

This chapter evaluates the performance of our Net to predict the hieroglyphic classes, in comparison also to three convolutional neural networks which was presented in the previously chapter. Computation times related to training and predictions will be evaluated and accuracy and loss trends will be shown during training. Additionally, a modified version of our architecture, which includes residual connections, will be tested. Finally, the results of the cross validation and the benefits of augmentation will be shown. The evaluation is applied to the test set, which represent the performance on unseen data. Before we start with the performance evaluation, we introduce the evaluation metrics.

5.1 Evaluation Metrics

For the evaluation of the performance of the task several metrics will be used in order to quantify the quality of the predictions. For the evaluation the accuracy and F1-Score as main metric are used. The accuracy is used in most of the related works, so it is useful for comparison, and, in addition, the value is easy to interpret. The F1-score is used to check for discrepancy in a single class. A drop of the macro F1-score means that one class is badly classified. To define these metrics it is necessary to introduce the confusion matrix, as they can all be derived from it.

Confusion Matrix

A common way for evaluating the performance of a classification is too look at the confusion matrix. Let n is the number of classes, a *confusion matrix* X is an nxn matrix with the left axis showing the true class (as known in the test set) and the top axis showing the classes assigned to the items. Each element x_{ij} of the matrix is the number of items with true class i that were classified as being in class j.

		Predicted Number			
		Class 1	Class 2	…	Class ***n***
Actual Number	Class 1	x_{11}	x_{12}	…	x_{1n}
	Class 2	x_{21}	x_{22}	…	x_{2n}
	⋮	⋮	⋮	⋮	⋮
	Class ***n***	x_{n1}	x_{n2}	…	x_{nn}

Figure 5.1: Confusion Matrix

The *recall* for the class i R_i, is defined as the ratio of the number of items in class i correctly predicted to the number of samples in this class

$$R_i = \frac{x_{ii}}{\sum_j x_{ij}}$$

The *precision* for the class i, P_i, is defined as the ratio of the number of the items in class i correctly predicted to the number of predictions for this class

$$P_i = \frac{x_{ii}}{\sum_i x_{ij}}$$

To compute the macro recall $Recall$, and macro precision $Precision$, it will be calculated as the average for R_i and P_i over all classes

$$Recall = \frac{\sum_i R_i}{n}$$

$$Precision = \frac{\sum_i P_i}{n}$$

These values are necessary for the calculation of the F1-score.

Macro F1-Score

The macro F1-score $F1score$ is the harmonic mean of the macro recall and the macro precision.

$$F1score = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall}$$

It combines the two measurements into one score; the score reaches its best value at 1 and the worst at 0. The combination of macro recall and macro precision makes the $F1score$ sensitive to outliers in single classes. This means that if a single or multiple classes are badly classified, this would cause a drop of the $F1score$, even if the error rate does not change. Usually, if the error rate increases, the $F1score$ decreases and viceverse.

Accuracy

The accuracy is given by the sum of the elements placed on the diagonal of the confusion matrix divided by the number of samples N, that is

$$accuracy = \frac{1}{N} \cdot \sum_i x_{ii}$$

The $accuracy$ metric cannot represent misclassified classes, especially if the test dataset has unbalanced classes. For example, suppose we have a class c_1 with 90 samples and another c_2 with 10. If the accuracy shows that 90% of the data is rightly classified, at best this could mean that both c_1 and c_2 have 90% of true prediction. But in the worst case, this could mean that all samples are labeled as c_1 and c_2 has 0% accuracy.

In our case, the test-set classes are not uniformly distributed, as we can see from figure 5.2.

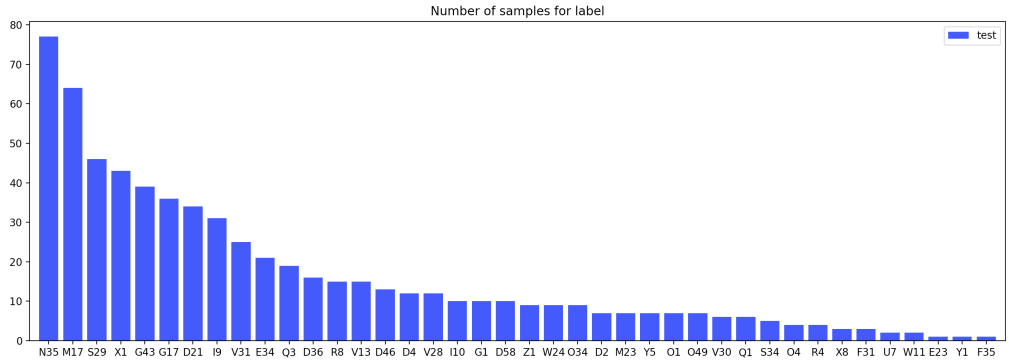


Figure 5.2: Test-set distibution

5.2 Architectures Evaluation

In this section, we report the results in order of accuracy, precision, recall and F1-score, previously mentioned, for our architecture. In addition, the behaviors of the architectures in training and testing will be shown and analyzed. The transfer learning approach did not give the desired results, which are shown in Table 5.1. Table 5.2 report the evaluation of the models trained with traditional approach.

Transfer learning evaluation				
	Metrics			
Architecture	Accuracy	Precision	Recall	F1-Score
Resnet50	0.906	0.882	0.825	0.84
Xception	0.834	0.715	0.72	0.703
InceptionV3	0.864	0.73	0.737	0.717

Table 5.1: Evaluation using the transfer learning approach.

We can see that our model achieves better performance in reference to all evaluation metrics whit respect to other architectures. In Figures 5.3, 5.4, 5.5, 5.6, we can see the trend in the training and testing progress over the epochs of the various models.

CNN evaluation				
	Metrics			
Architecture	Accuracy	Precision	Recall	F1-Score
Our	**0.976**	**0.975**	**0.965**	**0.968**
Resnet50	0.945	0.919	0.905	0.903
Xception	0.956	0.919	0.93	0.919
InceptionV3	0.948	0.917	0.904	0.9

Table 5.2: Evaluation of the architectures tested.

In Figures 5.7 are shown the accuracy trends on the test set with the various architectures.

5.3 Computation Time

In the previous chapter, we mentioned that our net should be faster to be trained than the other three architectures. To compare the training time of our network with the InceptionV3, ResNet50 and Xception, the training steps (gradient updates) per millisecond on different input resolutions was measured. The batch size is set to 32. Times are measured on an Nvidia Tesla T4 and are shown in Figure 5.8. Among all the tested CNN, our net is the fastest with all image resolution. The training time increases exponentially. This has a lot of influence on the CNN, because the feature maps are all increased by a factor of four.

In addition to the computation time related to the training, the time related to the prediction of the images was also calculated with batch-size 1 and 32 on the same GPU. We collected 100 measuring points and averaged them. The results are depicted in Table 5.3. Among all the tested CNN, Our net is the fastest to image prediction at any resolution.

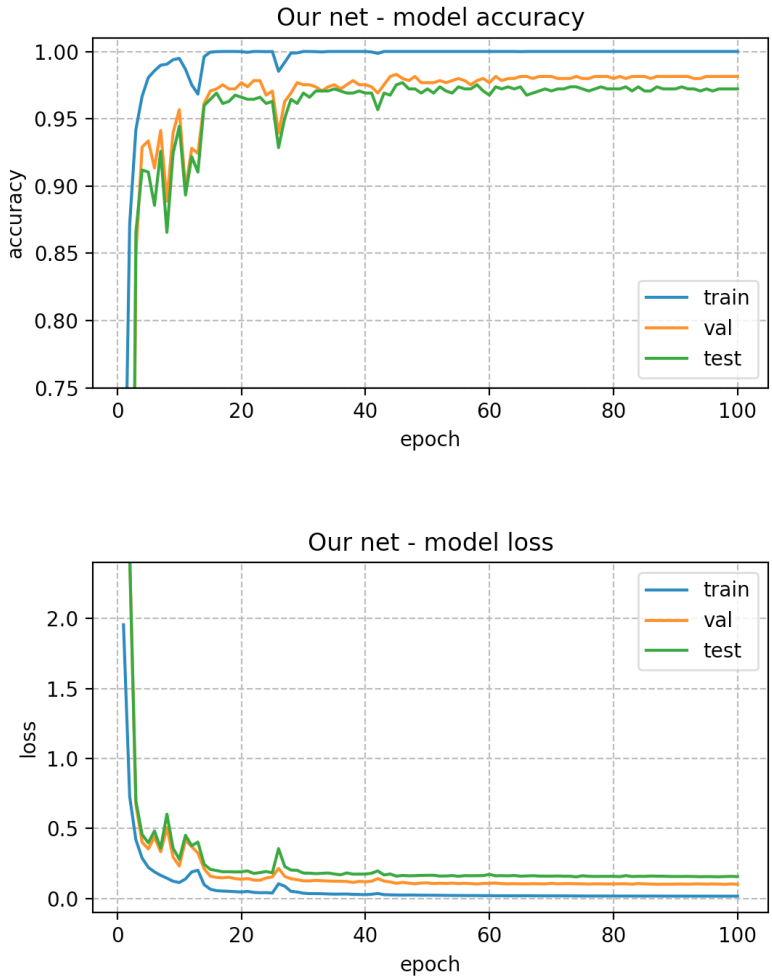


Figure 5.3: Accuracy and loss trend in training, validation and testing using Our architecture

5.4 Residual Connection

Unlike Resnet50, InceptionV3 and Xception our model does not use residual connections. In order to evaluate whether the residual connections could have any benefit, we compared a modified version of our model that includes them on the same dataset. The results are shown in figure 5.9.

Residual connections are clearly not essential in terms of either speed or final grading performance. Figure 5.10 shows the model with the

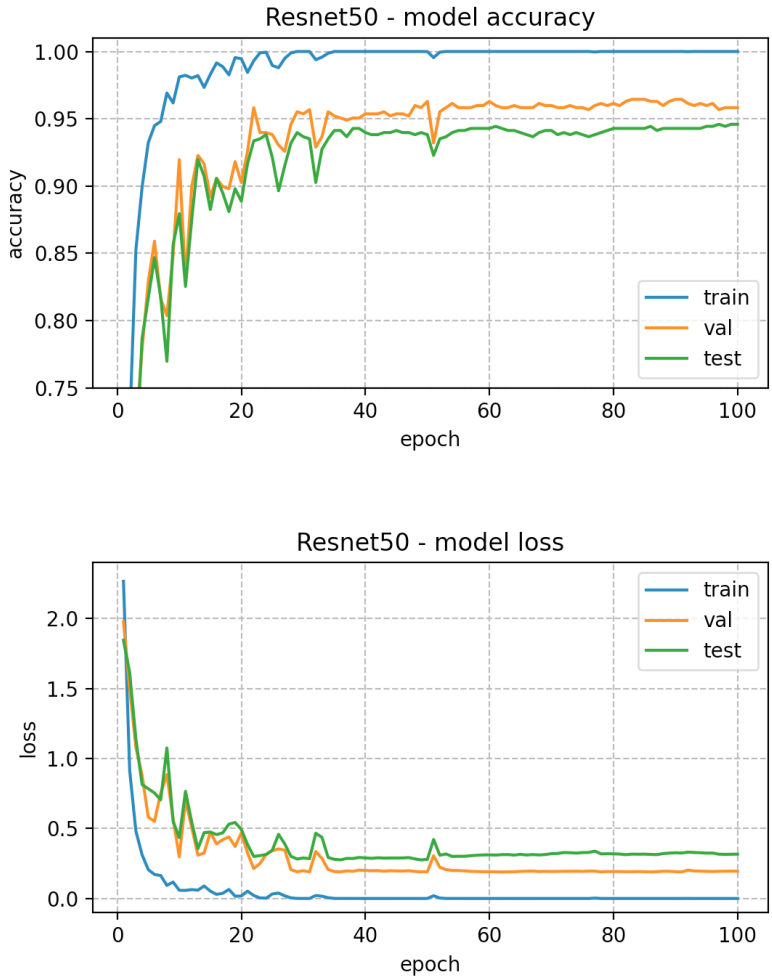


Figure 5.4: Accuracy and loss trend in training, validation and testing using Resnet50 architecture

residual connections. This modified version increases both the parameters and the computation times as convolutional layers are added in the residual blocks.

5.5 K-Fold Cross-Validation

A further method for evaluating and estimating the performance of the network is the Cross-Validation. Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample.

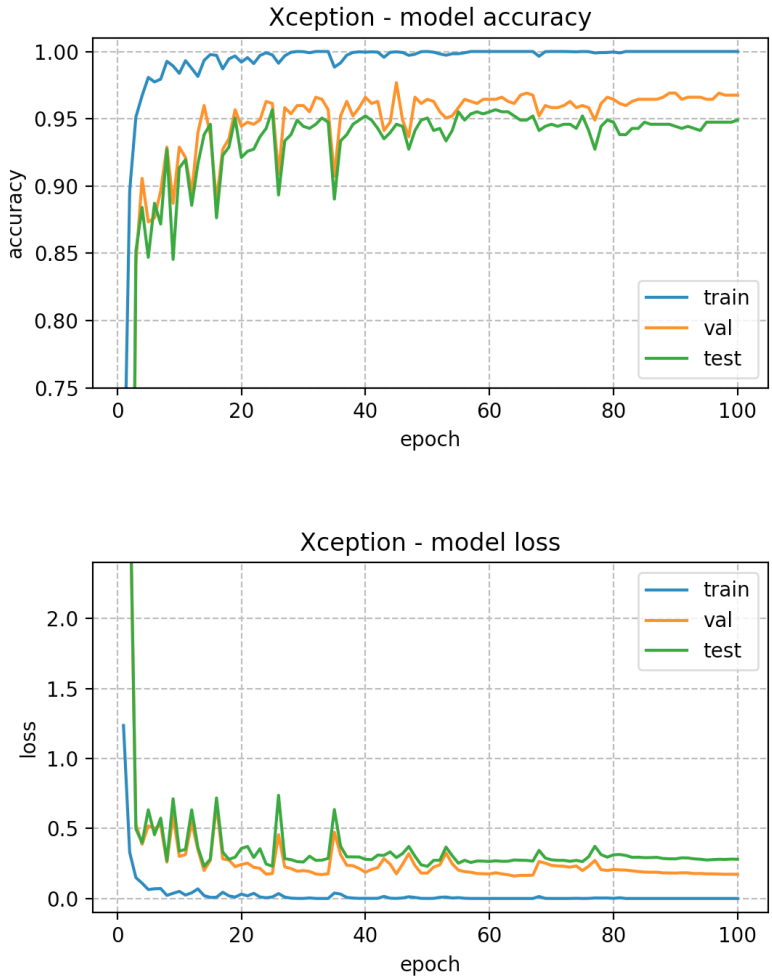


Figure 5.5: Accuracy and loss trend in training, validation and testing using Xception architecture

The procedure has a single parameter called *k* that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called *k*-fold cross-validation. When a specific value for *k* is chosen, such as k=10 becoming 10-fold cross-validation.

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

It is a popular method because it generally results in a less biased or

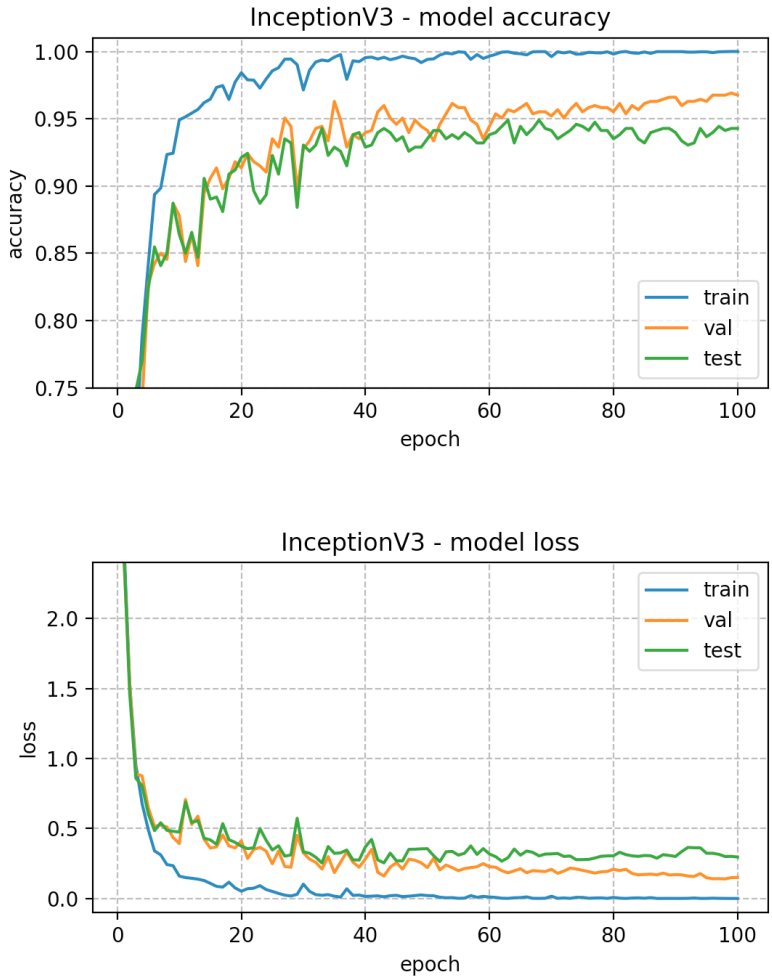
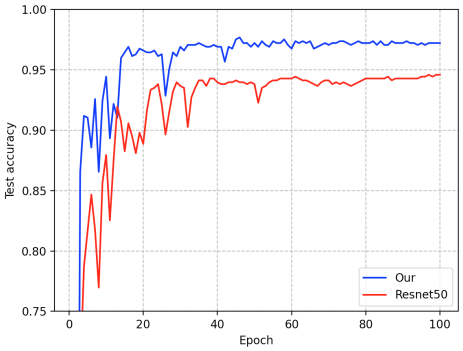


Figure 5.6: Accuracy and loss trend in training, validation and testing using InceptionV3 architecture

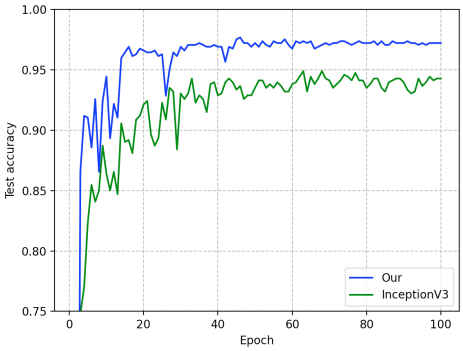
less optimistic estimate of the model skill than other methods, such as a simple train/test split. So, cross-validation is used to reduce the variance in the validation error estimates without having to set aside a lot of data for validation.

The general procedure is as follows:

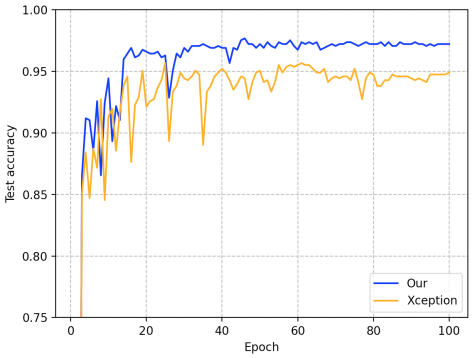
1. Shuffle the dataset randomly.
2. Split the dataset into k groups
3. For each unique group:
    - Take the group as a hold out or test data set



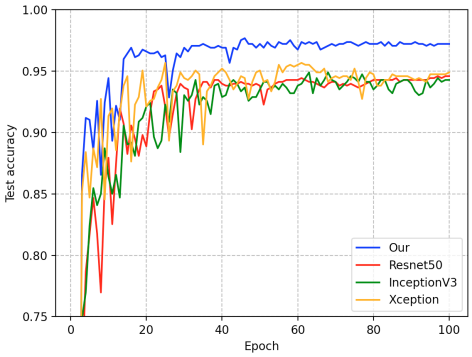
(a) Comparison between Our net and Resnet50 regarding the accuracy progress on the test set



(b) Comparison between Our net and InceptionV3 regarding the accuracy progress on the test set



(c) Comparison between Our net and Xception regarding the accuracy progress on the test set



(d) Accuracy progress on the test-set with the various architectures

Figure 5.7: Comparison between various architecture on test-set accuracy

- Take the remaining groups as a training data set
- Fit a model on the training set and evaluate it on the test set

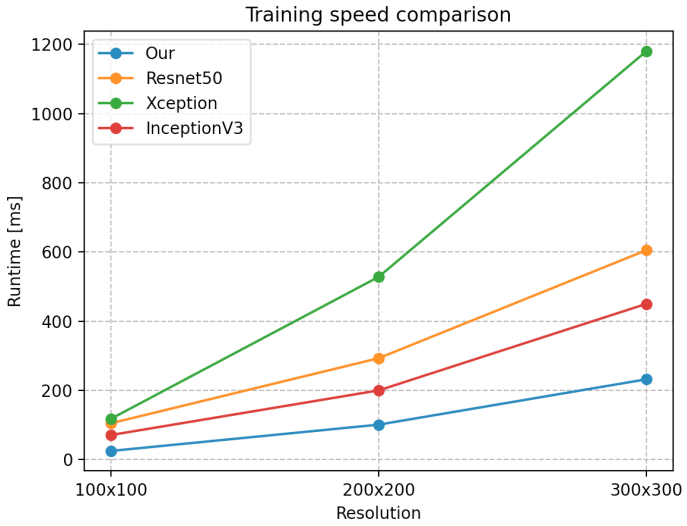


Figure 5.8: Training computation time

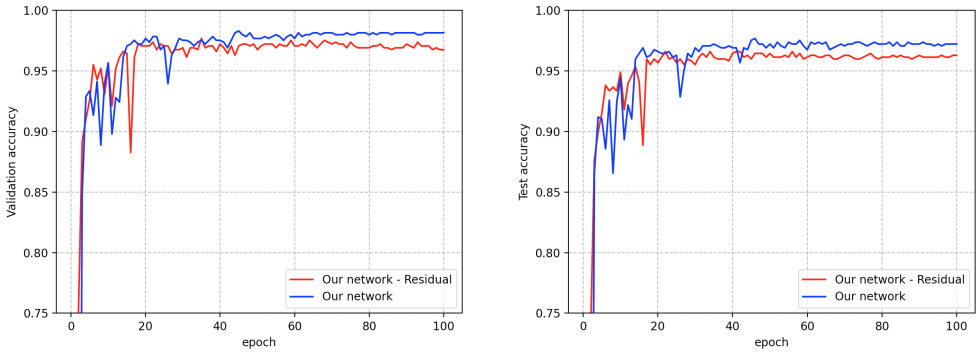
		Prediction runtime [ms]		
		Resolution		
	Architecture	100x100	200x200	300x300
Batch-size 1	Our	**3**	**3**	**4**
	Resnet50	8	10	13
	Xception	7	10	11
	InceptionV3	12	13	14
Batch-size 32	Our	**10**	**22**	**50**
	Resnet50	29	86	181
	Xception	27	110	275
	InceptionV3	20	58	116

Table 5.3: Prediction runtime for batch-size 1 and 32 on Nvidia Tesla P100-PCIE-16 GPU.

- Retain the evaluation score and discard the model

4. Summarize the skill of the model using the sample of model evaluation scores

Importantly, each observation in the data sample is assigned to an



(a) **(b)**

Figure 5.9: Accuracy progress on validation and test sets using Our model with and without residual connections

individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model $k - 1$ times.

If we have unbalanced data, evaluating the models on this split may not give them enough examples of some less populated class, and this could lead to a loss of performance. The solution is to not split the data randomly when we split the dataset, but we can split it so that the ratio between the target classes is the same in each fold as it is in the full dataset. This is called stratification or stratified sampling.

In this thesis, k-fold cross validation stratified was used, with $k = 5$ (see Figure 5.11). For each iteration the train-set was augmented through previous mentioned techniques, as zoom in, translation and horizontal flip. The test-set, instead it has not been altered.

The results of the cross validation are shown in Figure 5.12.

5.6 Image augmentation

In figure 5.13 it is possible to see the effects of the data augmentation on the train-set. Figure 5.13(a) shows the accuracy trend on the test-set. The graph in orange is inherent to the network trained with the train-set to which data augmentation has been applied, while the one in blue is inherent to the network trained with the normal train-set. As we can see

in the first case the accuracy values are better and therefore the effect of the augmentation is remarkable. The effects are even more interesting when we test the model with the totally horizontally flipped test-test (Figure 5.13(b)). In this case, in fact, with the use of augmentation on the train-set, there is a clear improvement in accuracy. It improves from a maximum of 0.79 to a maximum of 0.97.

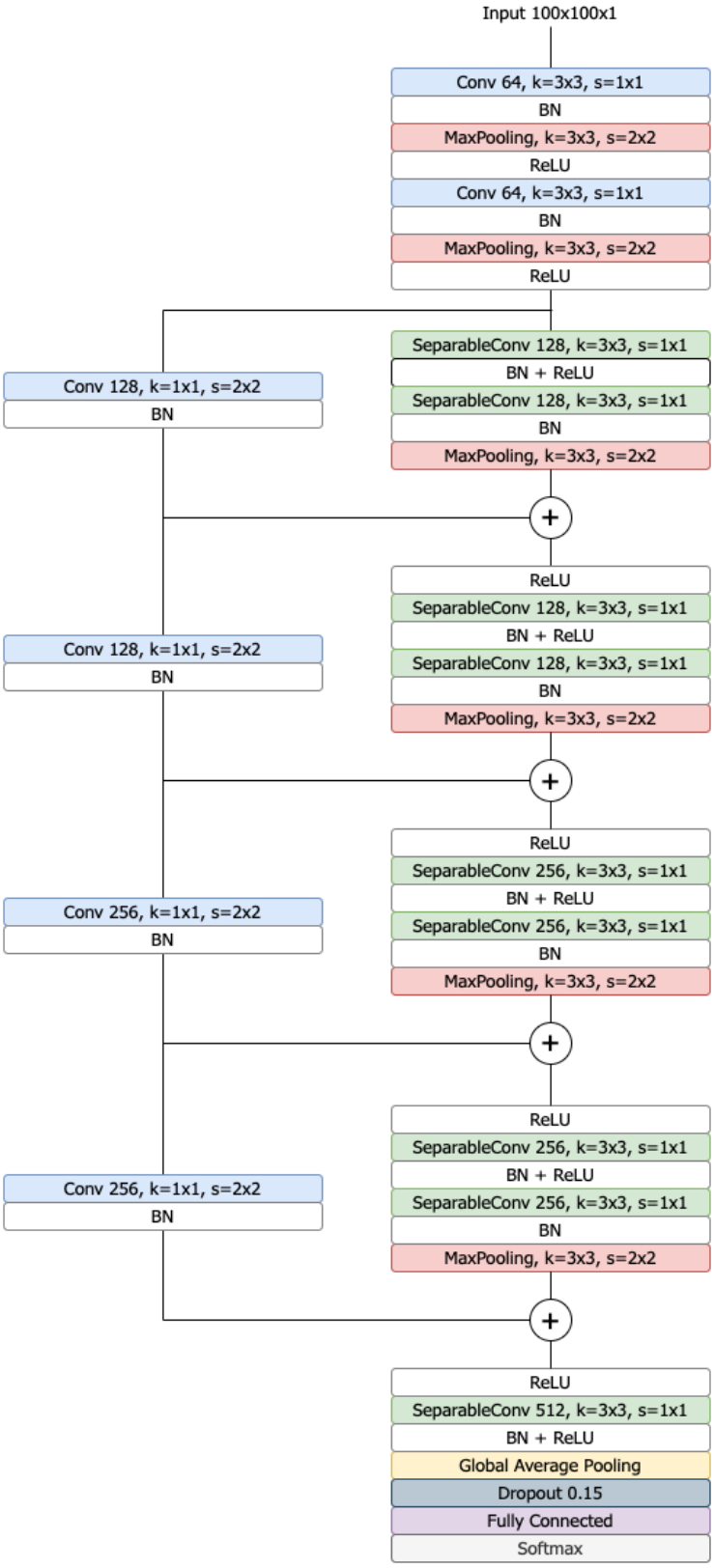


Figure 5.10: Our model with residual connections

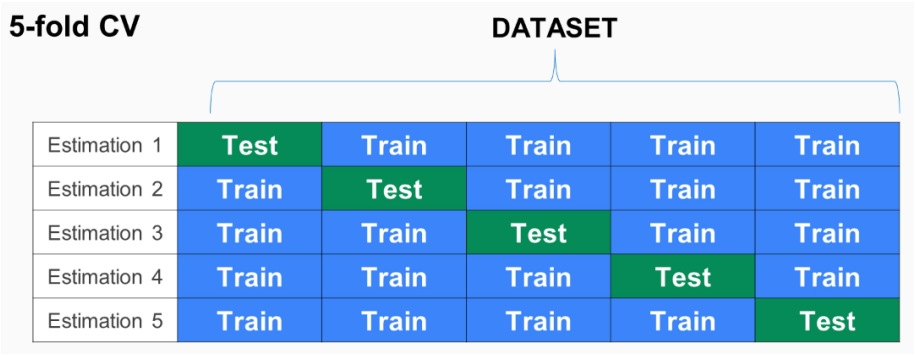


Figure 5.11: 5-Fold Cross Validation

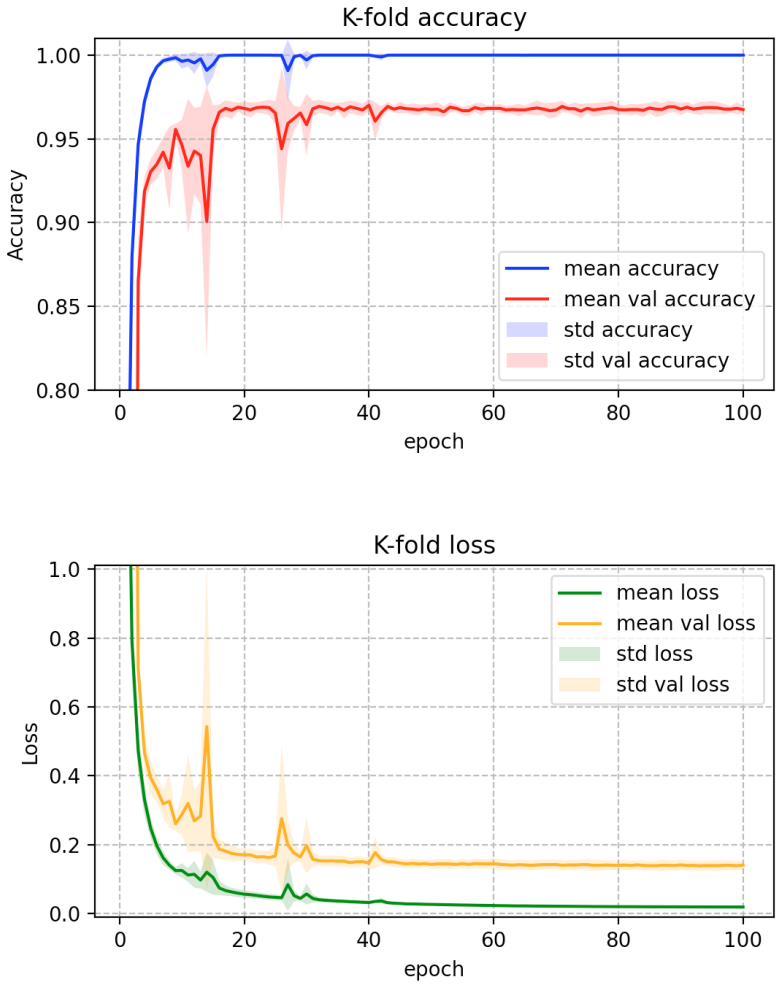
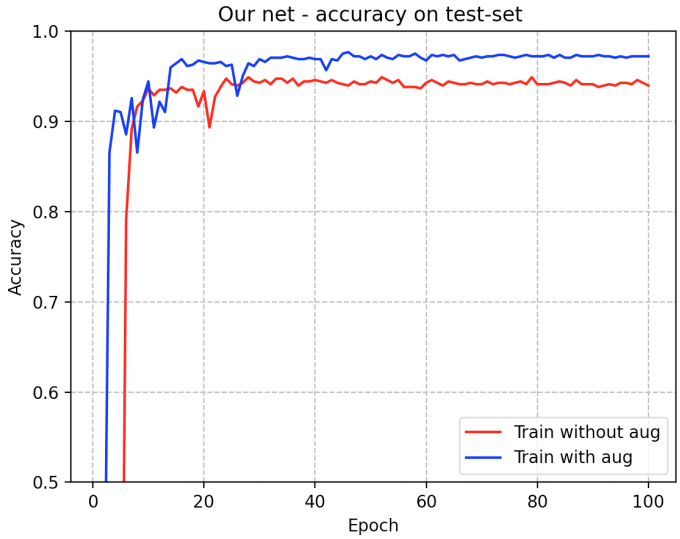
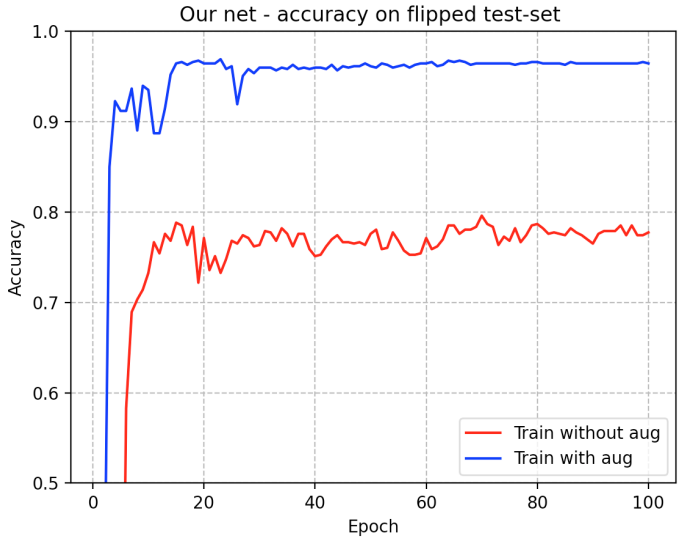


Figure 5.12: Cross Validation



(a) The orange graph shows the accuracy on the test-set having trained the network with the augmented train-set. The graph in blue shows the accuracy on the test-set having trained the network with the normal test-set.



(b) The orange graph shows the accuracy on the flipped test-set having trained the network with the augmented train-set. The graph in blue shows the accuracy on the flipped test-set having trained the network with the normal train-set.

Figure 5.13: Effects of augmentation on the test test.

6

Conclusion

In this thesis, some deep learning methods and their applications have been covered. In particular, interest was given to Convolutional Neural Network (CNN) . CNN is composed of an ensemble of convolution and subsampling layers and with a final prediction layer in order to discern the class label of the input of the network (generally an image).The major objective of deep methods is to extract high level features in order to apply them for classification problems. Starting from this fact, we have developed a new convolution network model with the aim of classifying ancient Egyptian hieroglyphs.

In chapter 2 an overview on machine learning was shown by examining the main characteristics. In addition, the fundamental concepts of deep learning were described. In particular, we initially analyzed the simplest neural network architecture composed of a single neuron, known as perceptron. Starting from this we have presented the most complex architectures, such as multilayer percepron and convolutional networks. Finally, a variant of the traditional convolutions, the depth-wise separable convolutions, were analyzed, showing their main advantages. This variant has been very useful for the purposes of this thesis and has allowed our network to obtain excellent results both in terms of efficiency and speed.

In chapter 3 we have introduced the datasets used and we showed the preprocessing pipeline for preparing an image in input to CNN. in addition, the data augmentation technique used for training the networks was described. In the evaluation, we have seen that the image augmentation of the train-set has much influence on the performance.

In chapter 4 we have introduced three state-of-the-art CNN and then we proposed a new convolutional neural network architecture. We also reported the methods used for model training and the choices of hyperparameters.

In chapter 5, the performance of our network was evaluated and compared with three state-of-the-art CNNs. Furthermore, the results of the cross validation and the positive effects of data augmentation were shown. Finally, a variant of our network, that presents residual connection, was tested.

The results showes an excellent ability of our network in recognizing the hieroglyph class. The model performed better, both from the point of view of effectiveness and computation time, compared to the other three networks. The model reached an accuracy higher than 97%, well two points above the Xception, which reported the best results compared to Resnet50 and InceptionV3 for our task.

Bibliography

[1] J. J. Andrej Karpathy and F.-F. Li. "Lecture Notes to CS231n: Convolutional Neural Networks for Visual Recognition". In: (2015). URL: https://cs231n.github.io.

[2] Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Y. Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: http://tensorflow.org/.

[3] F. Chollet. *Xception: Deep Learning with Depthwise Separable Convolutions*. 2017. arXiv: 1610.02357 [cs.CV].

[4] F. Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[5] D. Cireşan, U. Meier, and J. Schmidhuber. *Multi-column Deep Neural Networks for Traffic Sign Classification*. 2012. arXiv: 1202.2745 [cs.CV].

[6] R. Collobert and J. Weston. "A Unified Architecture for Natural Language Processing: Deep Neural Networks with Multitask Learning". In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. Helsinki, Finland: Association for Computing Machinery, 2008, pp. 160–167. ISBN: 9781605582054. DOI: 10.1145/1390156.1390177. URL: https://doi.org/10.1145/1390156.1390177.

[7] J. M. Dan Cire san Ueli Meier and J. Schmidhuber. "Multi-column deep neural network for traffic sign classification". In: (2012).

[8] D. A. Forsyth and J. Ponce. *Computer Vision - A Modern Approach, Second Edition.* Pitman, 2012, pp. 1–791. ISBN: 978-0-273-76414-4.

[9] A. Gardiner. *Egyptian Grammar*. Griffith Institute, 1957. ISBN: 0900416351.

[10] X. Glorot, A. Bordes, and Y. Bengio. "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by G. Gordon, D. Dunson, and M. Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Nov. 2011, pp. 315–323. URL: http://proceedings.mlr.press/v15/glorot11a.html.

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

[12] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet. *Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks*. 2014. arXiv: 1312.6082 [cs.CV].

[13] K. He, X. Zhang, S. Ren, and J. Sun. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[14] D. HUBEL and T. WIESEL. "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex". In: *The Journal of physiology* 160 (Jan. 1962), pp. 106–154. ISSN: 0022-3751. DOI: 10.1113/jphysiol.1962.sp006837. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/pmid/14449617/?tool=EBI.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

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

[17] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324. DOI: 10.1109/5.726791.

[18] Y. LeCun, Y. Bengio, and G. Hinton. "Deep Learning". In: *Nature* 521 (May 2015), pp. 436–44. DOI: 10.1038/nature14539.

[19] Y. Lecun, B. Boser, J. Denker, D. Henderson, R. Howard, W. Hubbard, and L. Jackel. "Backpropagation applied to handwritten zip code recognition". English (US). In: *Neural Computation* 1.4 (1989), pp. 541–551. ISSN: 0899-7667.

[20] M. A. Nielsen. *Neural Networks and Deep Learning*. Determination Press. ISBN: 0900416351.

[21] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. "Learning and Transferring Mid-level Image Representations Using Convolutional Neural Networks". In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*. 2014, pp. 1717–1724. DOI: 10.1109/CVPR.2014.222.

[22] A. Piankoff. *The pyramid of Unas*. eng. Bollingen series ; 40:5. Princeton, N.J: Princeton University Press, 1969 - 1968.

[23] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. *CNN Features off-the-shelf: an Astounding Baseline for Recognition*. 2014. arXiv: 1403.6382 [cs.CV].

[24] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. *ImageNet Large Scale Visual Recognition Challenge*. 2015. arXiv: 1409.0575 [cs.CV].

[25] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html.

[26] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. *Rethinking the Inception Architecture for Computer Vision*. 2015. arXiv: 1512.00567 [cs.CV].

[27] J. Tompson, R. Goroshin, A. Jain, Y. LeCun, and C. Bregler. *Efficient Object Localization Using Convolutional Networks*. 2015. arXiv: 1411.4280 [cs.CV].

[28] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. *How transferable are features in deep neural networks?* 2014. arXiv: 1411.1792 [cs.LG].

[29] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. *How transferable are features in deep neural networks?* 2014. arXiv: 1411.1792 [cs.LG].