UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Writer identification in handwritten text images using deep neural networks.

DEGREE FINAL WORK

Degree in Computer Engineering

Author: Akshay Punjabi Punjabi

Tutor: Enrique Vidal Ruiz

Experimental director: Jose Ramon Prieto Fontcuberta

Course 2019-2020

Acknowledgement

I wish to express my deepest gratitude to Enrique Vidal, who offered the opportunity to work with the PRHLT research center, and to Jose Ramon Prieto Fontcuberta, who provided advice and assistance during the course of this work.

I am also extremely grateful to my parents, who have always supported me through hard times, for allowing me to study hard to this day and to have made a tremendous contribution to my life and education. I thank my brother, who has a Cambridge C2 level of English, to review my work and to share very interesting insights he gave me during the writing process. I would like to thank my friends and my classmates for those moments of happiness and joy. Thanks also to all the professors who were part of my studies at UPV, for enriching me with such valuable knowledge.

To everyone involved in this journey, thank you.

Resum

La identificació de l'escriptor que ha escrit un text és necessària en diverses aplicacions d'anàlisi i reconeixement de documents. En particular, es requereix per a l'organització de manuscrits històrics en arxius i biblioteques.

Els mètodes tradicionals solien basar-se en una segmentació de l' text escrit en caràcters individuals, seguit d'una anàlisi més o menys heurístic de la forma dels caràcters. Aquest enfocament ha proporcionat alguns resultats en casos simples on el text és prou clar com per permetre la detecció i extracció de caràcters individuals. però falla completament quan l'escriptura és informal o desordenada i / o quan es consideren estils d'escriptura antics. Per tant, els treballs modernes segueixen enfocaments holístics en els que s'analitza l'estil d'escriptura sense detecció o extracció prèvia de caràcters o paraules. A més, les tècniques de aprenentatge automàtic estan demostrant ser molt adequades per a abordar el problema d'identificació de l'escriptor d'una manera totalment holística i sense segmentació. En aquesta direcció, les xarxes neuronals convolucionals profundes són els models més prometedors que s'han provat fins ara.

Aquesta proposta consisteix en el desenvolupament de mètodes d'aprenentatge profund per a la classificació d'una imatge de text, o una part d'ella, en classes corresponents a les possibles mans que han escrit el text. El treball també inclourà l'experimentació amb conjunts de dades públiques de referència, així com en altres conjunts de dades del món real, que consisteixen en manuscrits històrics.

Paraules clau: identificació de l'escriptor, aprenentatge profund, xarxes neuronals, CNN, xarxa neuronal convolucional, pàgines escrites a mà, processament d'imatges de documents, identificació de l'escriptor de documents històrics, reconeixement de patrons

Resumen

La identificación del escritor que ha escrito un texto es necesaria en varias aplicaciones de análisis y reconocimiento de documentos. En particular, se requiere para la organización de manuscritos históricos en archivos y bibliotecas.

Los métodos tradicionales solían basarse en una segmentación del texto escrito en caracteres individuales, seguido de un análisis más o menos heurístico de la forma de los caracteres. Este enfoque ha proporcionado algunos resultados en casos simples donde el texto es lo suficientemente claro como para permitir la detección y extracción de caracteres individuales. Pero falla completamente cuando la escritura es informal o desordenada y/o cuando se consideran estilos de escritura antiguos. Por lo tanto, los trabajos modernas siguen enfoques holísticos en los que se analiza el estilo de escritura sin detección o extracción previa de caracteres o palabras. Además, las técnicas de aprendizaje automático están demostrando ser muy adecuadas para abordar el problema de identificación del escritor de una manera totalmente holística y sin segmentación. En esta dirección, las redes neuronales convolucionales profundas son los modelos más prometedores que se han probado hasta ahora.

Esta propuesta consiste en el desarrollo de métodos de aprendizaje profundo para la clasificación de una imagen de texto, o una parte de ella, en clases correspondientes a las posibles manos que han escrito el texto. El trabajo también incluirá la experimentación con conjuntos de datos públicos de referencia, así como en otros conjuntos de datos del mundo real, que consisten en manuscritos históricos.

Palabras clave: identificación del escritor, aprendizaje profundo, redes neuronales, CNN, red neuronal convolucional, páginas escritas a mano, procesamiento de imágenes de documentos, identificación del escritor de documentos históricos, reconocimiento de patrones

Abstract

The identification of the writer who has written a piece of text is needed in several applications of document analysis and recognition. In particular, it is required for the organization of historical manuscripts in archives and libraries.

Traditional methods used to relay on a segmentation of the written text into individual characters, followed by more or less heuristic analysis of the shape of the characters. This approach has provided some results in simple cases where the text is clear enough to allow the detection and extraction of individual characters. However, it fails dramatically when the writing is sloppy and/or when ancient scripts are considered. Therefore, modern works follow holistic approaches where the writing style is analyzed without character or word detection or extraction. Moreover, machine learning techniques are proving highly adequate to tackle the writer's identification problem in a fully holistic, segmentation-free way. In this direction, deep convolutional neural networks are the most promising models tried so far.

This proposal consists of the development of deep learning methods for the classification of a handwritten image, or a part thereof, into several classes corresponding to possible hands which have written the text. This work will also include experimentation on public-domain, benchmark datasets, as well as on other real-world datasets consisting of historical manuscripts.

Key words: writer identification, deep learning, neural networks, CNN, convolutional neural network, handwritten pages, document image processing, historical document writer identification, pattern recognition

Contents

List of Figures

List of Tables

CHAPTER 1
Introduction

For centuries, handwriting has been an important part of civilization, not only for the daily communication between human beings but also for the transmission of wisdom between generations. Accordingly, there is a vast literature of historical works of earlier centuries with great historical significance that need to be researched. However, many of these writings are quite difficult to read due to calligraphy or the language. Because of that, information such as who wrote the text, where or when it was written is difficult to extrapolate. All these details are of great importance for researchers as a means to understand, for example, how an author has evolved, differentiating between authors or inferring where and when was a manuscript written.

Manual extraction about the information of the huge amount of handwritten documents is very time consuming and expensive. Therefore, there is a need for automatic methods that help researchers in this process. In this way, fields like Handwritten Text Recognition (HTR) have emerged to extract text automatically from printed or handwritten images. Consecutively, Natural Language Processing (NLP) focuses on processing this huge amount of text and categorizing it and Natural Language Understanding (NLU) focuses on understanding the language or the meaning of the data.

In this bachelor thesis, however, we focus on writer identification of a document solely using the visual attributes, without taking into account the content of the text. This is also known as text-independent writer identification. Historians could use this system to verify document authorship or determine the authorship of an ancient unlabeled document. Furthermore, modern applications that include forgery detection and forensic science in criminal cases or financial activities such as confirmation of bank procedures would also benefit from writer identification.

1.1 Problem Statement

An effective automatic writer identification method would facilitate the research of old manuscripts and save a lot of time and money. Most writer identification methods follow traditional handcrafted approaches. With the success of deep neural networks in all kinds of computer vision tasks, it would be interesting to use them in writer identification. This is the motivation of our bachelor thesis. We propose to create a writer identification system based on a deep learning neural network. The neural network would receive a handwritten image and output an ordered list of the most probable writers.

1.2 Objectives

Our main objectives for this project are:

- Study the state of the art of writer identification approaches including deep learning approaches as well as traditional approaches.
- Build a deep neural network model that performs writer identification and evaluate it on typically used contemporary datasets, to contrast them with other studies using these datasets.
- Analyze and test the impact of different sized image inputs in the deep neural network built.
- Test the performance of our deep neural network on historical manuscripts.

1.3 Structure of the bachelor thesis

The bachelor thesis is structured in eight chapters. In Chapter 2, we explain the theoretical concepts behind neural networks and how they work. Chapter 3, introduces the task of writer identification, describing the concepts and methodology followed in this field. In Chapter 4, an up to date study is conducted of the state of the art methods in writer identification, which includes traditional methods as well as new deep learning methods. In Chapter 5, we introduce our proposed model. Chapter 6, presents the experiments conducted along with the results obtained. Software and hardware specifications and the datasets used for our experiments are also described in this chapter. We will give our conclusions in Chapter 7 presenting our findings and proposing possible future works. Finally, in Chapter 8 we will explain the relationship of the bachelor thesis to the computer engineering degree.

CHAPTER 2
Theoretical Background

2.1 Pattern Recognition and Computer Vision

Human beings have been searching for patterns in data since their existence. It is our curiosity of questioning the nature around us and the patterns it follows that has gotten us so far. Pattern recognition is the process of automatically discovering patterns in a sequence of data by using mathematical models and computers. With this information, we can automatically perform a variety of tasks such as classification of data into different categories.

In this chapter, we will first start by talking about the history of computer vision and how it lead to the current state of the field in which neural networks have been the protagonist. After that, we will talk about some of the theoretical foundations of neural networks and their derivatives.

Computer Vision

Computer Vision is one subfield of pattern recognition which explores how computers can understand and obtain patterns from images or videos. Several tasks exist in computer vision recognition such as resolving if an image data contains some particular object, feature, or activity. Common recognition tasks are:

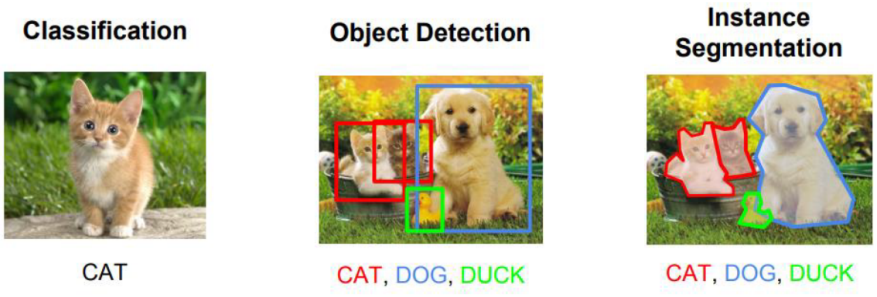


Figure 2.1: Illustrating the difference between classification, object detection, and object segmentation.

- **Image classification.** Classifying objects in an image to different predefined categories. For example, in the left image of Figure 2.1 a cat image is classified to the class cat. For a dog image, it would be classified as the class dog. Other examples include identification of a specific person's face or fingerprint, identification of handwritten digits, or identification of a specific vehicle.
- **Object detection.** Particular objects or parts of an object are detected in images. Normally it involves specifying coordinates of the object or bounding boxes. Some examples are the

detection of vehicles and pedestrians in a road for autonomous vehicles or the estimation of the number of people in a location. In the center image of Figure 2.1 you can see how a bounding box is created for each animal.

- **Image segmentation.** All pixels of the images are classified. For example, in the right image of Figure 2.1 you can see how some pixels are correctly categorized to different types of animals and the rest of the pixels are categorized as background. It differs from object detection as it is very specific and categorizes pixel by pixel which allows us to get more meaningful results. Common applications are in medical imaging, for instance, locating tumors, better visualization of medical images, or detecting regions of abnormalities. Object detection tasks could also be approached this way to achieve better solutions.

History of Computer Vision

The ability to see with our eyes is a remarkable capacity essential for human endeavors. In the 1960s, a series of experiments in cats and monkeys done by David Hubel and Torsten Wiesel led to understanding the information processing in the visual cortex for which they later won the 1981 Nobel Prize in Physiology or Medicine.

Inspired by the findings of Hubel and Wiesel, the field of computer vision began at universities in the 1960s. They were trying to mimic the human visual system to start the creation of artificial intelligence. Initial works were so promising about the future of artificial intelligence that they were convinced that it would be achieved very soon. Their optimism soon faded away as they failed to obtain any meaningful result. Early computing resources were not enough to handle the complexity of the problems. As such, the research for this area stagnated and scientists stopped pursuing it altogether. One important paper created in this time is the Neocognitron [1], which would inspire in the future the creation of neural networks.

After more than 50 years, in 2012, a breakthrough was made in neural networks. A convolutional neural network called AlexNet [2] won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [3] . The ILSVRC is an annual image classification competition that allows researchers to evaluate algorithms on a very large dataset of images. Until then, the error rate for ILSVRC competitors was around 26.2%. Surprisingly, AlexNet won the competition with an error rate of 15.3%. With these results, AlexNet set a precedent that started a new era for neural networks and computer vision tasks.

Nowadays, convolutional neural networks have become the best algorithms for computer vision. Numerous different tasks ranging from classification to segmentation, have greatly benefited from deep learning algorithms. Each year new developments in neural networks achieve lower and lower error rates (Figure 2.2), even surpassing the performance of human beings.

2.2 Machine Learning

Machine Learning algorithms are capable to learn the relationship between the input data and the output data and as a consequence perform complex tasks. It is quite remarkable how powerful these algorithms are. Machine learning is a very wide field and there exist multiple types of machine learning algorithms like k-nearest neighbor (KNN), linear regression, logistic regression, Support Vector Machines (SVM), neural networks, and more.

As such, there are several topics to explore in the field of machine learning. However, in this project we will work fundamentally with neural networks, so we will only focus and explore the theoretical concepts behind neural networks [1].

[1] Most of the theory comes from the book *Deep Learning* by Ian Goodfellow [5]

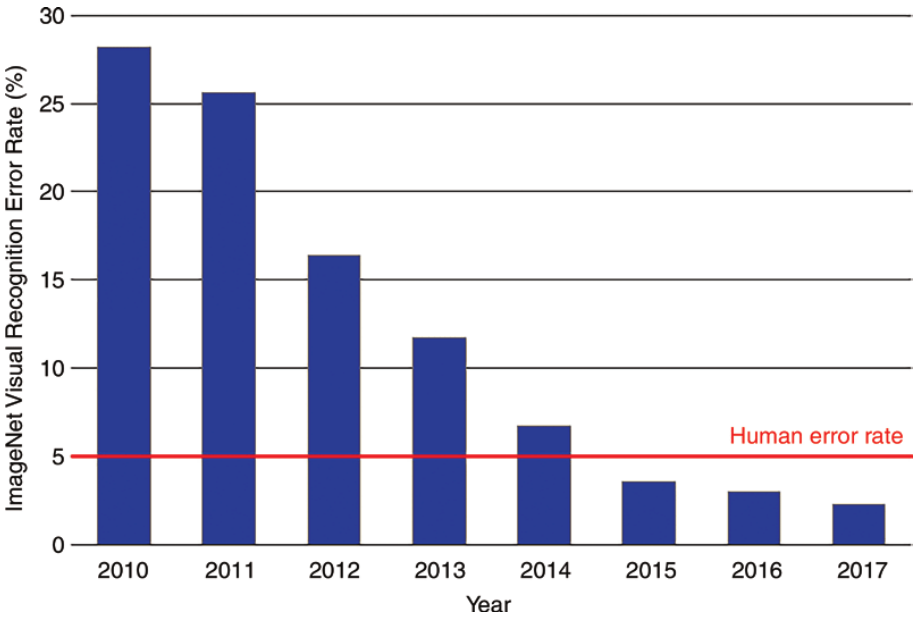


Figure 2.2: ILSVRC competition errors each year. Image from [4].

2.2.1. The goal of Machine Learning

Normally when we try to solve some sort of problem through a computer we manually design algorithms that give the computers the specific instructions to perform a task. In machine learning, instead, we give the computer information in the form of data or patterns that it can learn from, and let the computer try and figure out what those patterns are so that it can perform a task on its own.

The objective of a machine learning algorithm is to map an input space to another output space $f(x) = y$. Finding this exact function is not possible so we instead try to approximate f to a hypothesis function $h(x) = y$. The goal of machine learning is learning and finding the function h.

2.2.2. Types of Machine Learning

As the book *Artificial Intelligence: A Modern Approach* puts it:

> *Learning is a search through the space of possible hypotheses for one that will perform well, even on new examples beyond the training set.*
>
> Page 695, Artificial Intelligence: A Modern Approach, 3rd edition, 2015,

Depending on how a machine learning algorithm trains with the input data and how it produces an output, learning can be categorized in supervised, semi-supervised, or unsupervised.

- **Supervised Learning.** The learning algorithm uses as an input train data with is correspondent target data, tunes and learns the features and verifies the predicted target to the true target. After that, a test data distinct from the training data is used to measure the accuracy of the algorithm. In the case of discrete categories, it is called classification and in the case of continuous variables, it is called regression. An example of a regression problem would be the prediction of the market price of a house in a certain neighborhood.
- **Unsupervised Learning.** In the case of unsupervised learning, the training data does not contain the corresponding target values. The goal of unsupervised learning is then to get

as good as an approximation as possible of similar groups within the data (clustering) or to determine the distribution of data within the input space (density estimation).

- **Semi-supervised Learning.** Semi-supervised learning is halfway between supervised learning and unsupervised learning. The training examples use both labeled samples and unlabeled samples. The idea surges so that all the available data is used. In some cases, it has been shown that this approach is better than unsupervised learning [6].

2.3 Neural Networks

In this section, we explain the theoretical fundamentals of neural networks. Particularly, we will talk about the *perceptron* which is the fundamental unit of a neural network and the *multi-layer perceptron* which is an extension of the perceptron.

2.3.1. Perceptron

As many of our creations, artificial neural networks were inspired by biological processes, hence the word artificial. Broadly speaking, a biological neural network consists of neurons that are connected and receive electrical signals from other neurons as seen in Figure 2.3. Neurons process these input signals and can be activated or not.

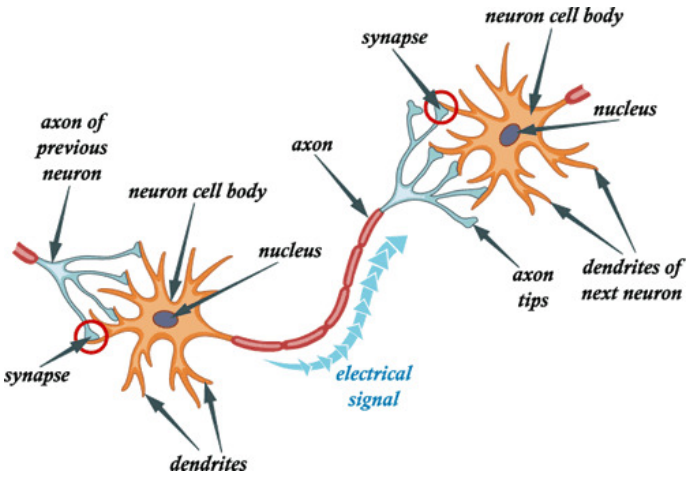


Figure 2.3: Biological neural network. Image from [7].

To create the artificial neural network (ANN) instead of having biological neurons we will use what we are going to call *units*. Units are going to be connected with each other with edges. As seen before, we want to find h that maps input to outputs to perform some sort of classification. Given inputs x_1 and x_2 of a simple example ANN we will try to approximate the function h:

$$h(x_1, x_2) = w_0 + w_1 x_1 + w_2 x_2 \tag{2.1}$$

where w are the weights we will need to adjust to determine what values to multiply our inputs to get some sort of result. Of course, we also need a threshold to perform classification. We will use the simple function g:

$$g(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.2}$$

This is also called as *activation function*, a function that determines when it is that this output becomes active. Function 2.2 works only for binary classification, if we do not want a binary

classification and we want real values a better function would be the *logistic sigmoid function* that has an S-shaped curve:

$$g(x) = \frac{e^x}{e^x + 1} \tag{2.3}$$

The logistic sigmoid function produces a real value between 0 and 1. Figure 2.4 shows these different activation functions graphically:

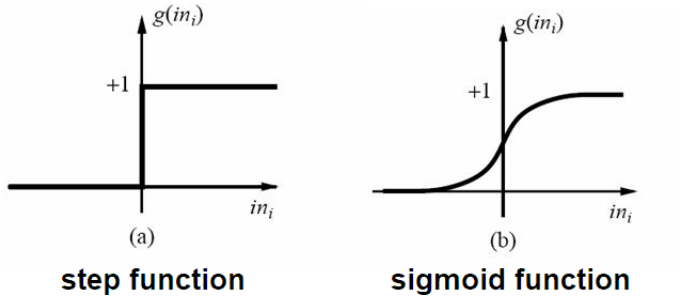


Figure 2.4: Basic activation functions. (a) step function, output would be 0 or 1. (b) logistic sigmoid, output between 0 and 1. Image from [8].

With an activation function h would become:

$$h(x_1, x_2) = g(w_0 + w_1 x_1 + w_2 x_2) \tag{2.4}$$

For now, the example described before only had two inputs but we could have as many inputs as necessary. Generalizing, the general formula of the model is:

$$h(x_1 ... x_n) = g(\sum_{i=1}^{n} x_i w_i + w_0) \tag{2.5}$$

and the visual representation can be seen in Figure 2.5. This is also called *perceptron* and was proposed in 1958 by Roseblatt [9]. This model is the foundation of neural networks algorithms.

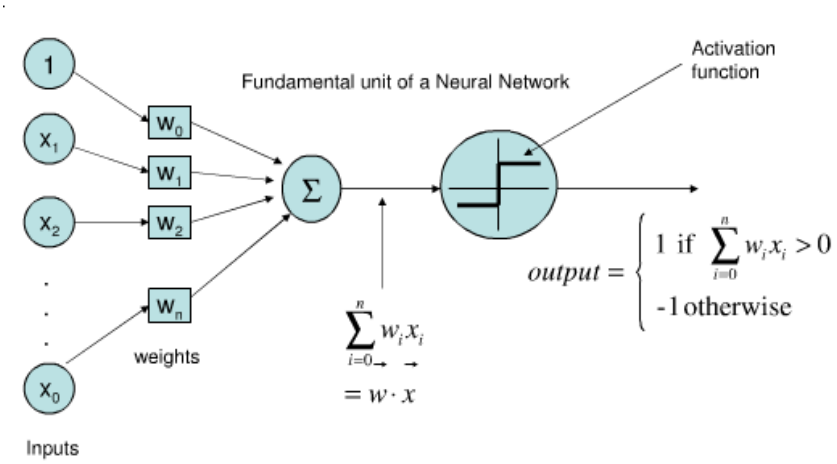


Figure 2.5: The Perceptron, the fundamental unit of a neural network. Image from [10].

2.3.2. Multi-layer Perceptron

The perceptron proposed before can handle the classification of linearly separable data but is not able to handle more complex non-linearly separable data. Because of this, the perceptron was extended to *multi-layer perceptron* (MLP), which is also sometimes called a *feed-forward neural*

network. An MLP network (Figure 2.6) has input layers, output layers, and at least one hidden layer. The universal approximation theorem stated by Cybenko in 1989 [11] says that an MLP with only one single hidden layer and with sigmoid activation functions can approximate any continuous function. This means an MLP can be applied to solve any type of problem. However, it is more efficient to use multiple hidden layers. MLPs with a large number of hidden layers are called *deep neural networks* (DNN).

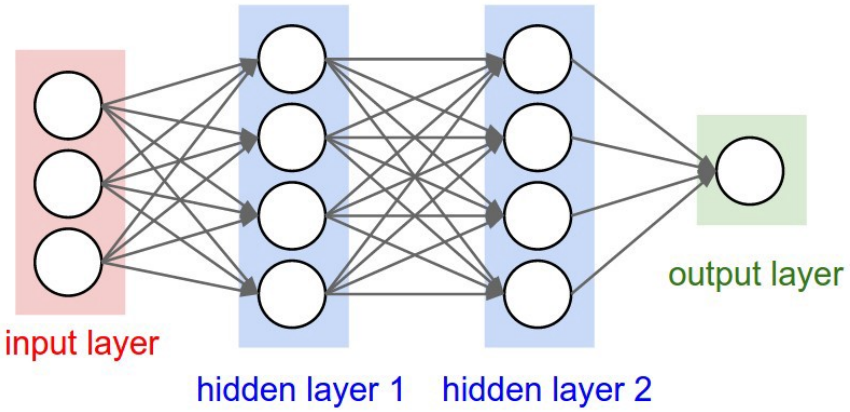


Figure 2.6: A Multi-layer perceptron with three inputs, two hidden layers and one output. Image from [12].

Optimization

To measure how well the model is working and try to get even better models we need a way to evaluate the model. For that, we use a loss function, which expresses how poorly a model performs. Most neural networks use *Cross Entropy Loss*:

$$L(y_t, y_p) = -\sum_{c}^{C} y_t \cdot \log y_p \tag{2.6}$$

where y_t is the true class of the input, y_p is the predicted class by the model and C is the number of classes.

With the information of the loss function the model can adapt at run-time with the objective to minimize this loss function and change its parameters accordingly. The typical algorithm for this procedure is *Gradient Descent* which minimizes the loss function by calculating in which direction of the state space the loss is minimized and following that direction.

Algorithm 2.1 Gradient descent algorithm

PARAMETERS
Learning rate η, weights w, loss function L, number of data points N
Initialize weights w randomly
loop
Calculate gradients based on *all* data points direction that will lead to decreasing loss:
$\Delta w = w - \eta \cdot \frac{1}{N} \sum_i^N L(y_t^i, y_p^i)$
Update weights according to gradient: $w = w - \Delta w$
end loop

This algorithm takes into account all data points which is not feasible in practical situations where we have huge amount of data points. Instead we use *Stochastic Gradient Descent* (SGD) where we use only a *batch* of data points:

Algorithm 2.2 Stochastic gradient descent algorithm

PARAMETERS
Learning rate η, weights w, loss function L, batch size N_b
Initialize weights w randomly
loop
Calculate gradients based on a *batch* of N_b data points direction that will lead to decreasing loss: $\Delta w = w - \eta \cdot \frac{1}{N_b} \sum_i^{N_b} L(y_t^i, y_p^i)$
Update weights according to gradient: $w = w - \Delta w$
end loop

where new parameters η which are the learning rate and batch size N_b are tuned to find a good model. Choosing these parameters is critical for creating a good neural network model. A large learning rate may perhaps miss an optimal minimum. On the other hand, a small learning rate may slow down the training time or get stuck in a local minimum. In practice, different values are tested until finding a suitable one.

Backpropagation

The first problem that arises with MLPs is that we cannot know how to change the weight values of these new hidden layers. So the *backpropagation* [13] algorithm was proposed:

Algorithm 2.3 Backpropagation algorithm

Compute the derivative of the loss function with respect to the output layer
for layer in layers-1 **do**
Compute the derivatives of the loss function with respect to the inputs of layer+1
Compute the derivatives of the loss function with respect to the weights between layer and layer-1
Compute the derivatives of the loss function with respect to layer-1
end for
Update weights

Given the loss of the output layer, the backpropagation algorithm estimates the amount of the error that is caused by each unit in the hidden layers. After that, it propagates the corresponding error of each unit to the preceding hidden layers to update the weights accordingly. This is achieved by using derivatives (remember the derivative of a function measures the ratio of change of the output depending on the change of the input) and recursive applications of the chain rule.

2.3.3. Rectified Linear Unit

The true power of MLPs is due to non-linear activation functions. Up until 2012, the non-linear sigmoid function was used predominantly. Hereafter, the use of the sigmoid function was mostly replaced with the *rectified linear unit* (ReLU):

$$g(x) = \max(0, x) \tag{2.7}$$

The advantages of ReLU over the sigmoid function include fewer vanishing gradient problems compared to sigmoid and that it is much more efficient and faster. In Figure 2.7 a visual illustration is shown.

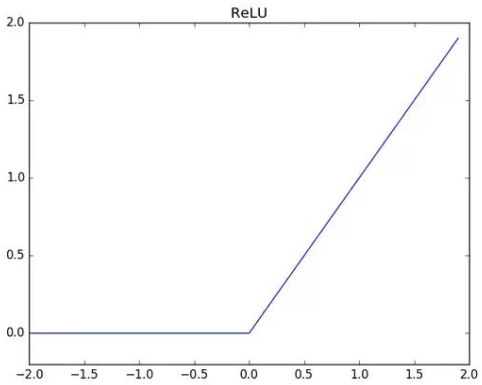


Figure 2.7: ReLU activation function. The output is 0 for negative values, 0 to ∞ for positive values.

2.4 Layers of deep neural networks

The hidden layers of a deep neural network are formed by different types of layers. In this section, we will explain some of the typically used layers.

2.4.1. Normalization Layers

One of the most common problems while creating neural network models is when a model memorizes the training set and does not generalize well. As a result, it performs worse on new data or test set. This is called *overfitting*. It especially occurs in small datasets or working with big models with a lot of parameters that can easily overfit in small datasets. In that event, regularization techniques are necessary.

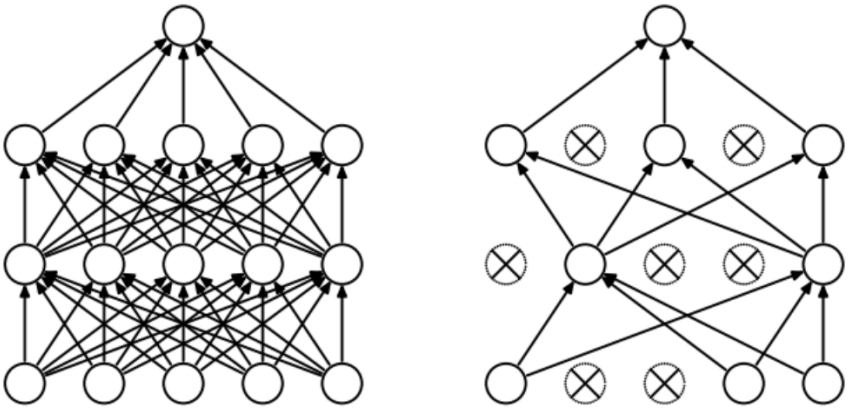
Dropout

Srivastava et al. [14] propose Dropout as a technique for addressing the overfitting problem. Dropout consists of randomly dropping units (hidden and visible) along with its connections with a probability p (normally $p = 0.5$). A visual representation can be seen in Figure 2.8. Because of that, the next layer of the network will have to adapt to work with less information, making the hidden unit more robust without the need to rely heavily on other hidden units and as a result, generalizing better.

This technique has been widely adopted in all types of neural network applications and is commonly used when there are signs that the model is overfitting in the training set and performing worse on the testing set. One of the drawbacks is longer training times, and more epochs would be necessary until a good model is obtained. Another drawback is that it adds another hyperparameter to be tuned. In our models, we use Batch Normalization [15] instead, which solves other problems in neural networks and also removes the need to use dropout. In the next section, we will explain Batch Normalization.

Batch Normalization

Initially, when a neural network is trained, input data is provided with certain distribution to the input layer. For the next layer, the output of the previous layer becomes the input of the current layer, but this input has no longer the same distribution as the initial data. The change of distributions between layers in DNNs while training is called *Internal Covariate Shift*. Ioffe and Szegedy [15]



(a) Standard Neural Net (b) After applying dropout.

Figure 2.8: Applying Dropout to a neural network. (a) A neural network with 2 hidden layers. (b) Applying dropout to the network on (a). Units that have been dropped are marked with X. Image from [14].

put forward this issue of distribution change in each layer which as a result also makes the optimizer readjust for the change each time. A fixed distribution would be more advantageous for training.

For this reason, they propose the *Batch Normalization* algorithm to reduce this internal covariate shift which in consequence increases the training speed substantially. A normalization step that fixes the means and variances of layer inputs is applied. This normalization is contained inside a new trainable layer which is part of the neural network. Normally, it is included after the convolution layer. This new layer, however, adds more parameters to train: γ and β used to shift and scale the activations obtained in the normalization step and the batch mean μ_B and variance σ_B.

Solving the internal covariance shift problem removes optimizer readjustment for the change of distribution each time. This allows us to use much higher learning rates without the risk of divergence. Besides, the regularization of the model with batch normalization removes the need for Dropout.

2.4.2. Convolution Layers

In traditional feed-forward networks, each unit is connected to all the units of the next layer. This is also called a *fully-connected layer* (FC). In the case of having images as input, this would mean for each pixel there is a weight to be trained. For example, an image of size $224 \times 224 \times 3$ (3 color channels) would need a total of 150528 weights for only one hidden layer. With more and more layers, this quickly becomes unfeasible for training a neural network. Moreover, it does not capture the spatial dependencies between pixels in an image.

Image convolution is the process of applying a filter that adds each pixel value of an image to its neighbors, weighted according to a kernel matrix. Image convolution allows us to extract useful high-level features out of images, such as the edges that are shown in Figure 2.9.

In a convolution layer, the kernel moves through an image, as it is a moving window, performing element-wise multiplication with the part of the input it is currently on, adding up the results into a single output pixel. After applying convolution what we obtain as an output is a feature map with size $H_o \times W_o$, which can be determined with the following equations:

$$W_o = \frac{W_i + 2P - K_w}{S} + 1 \tag{2.8}$$

$$H_o = \frac{H_i + 2P - K_h}{S} + 1 \tag{2.9}$$

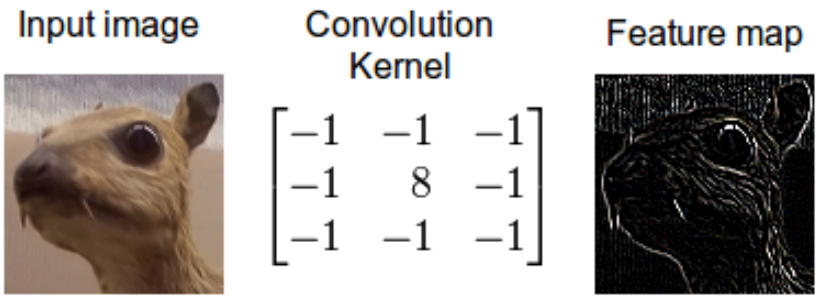


Figure 2.9: Example of a convolution for edge detection. Source [16].

where W_i and H_i are the width and height of the input image, K_w and K_h are the width and height of the kernel matrix, P is the amount of zeros to add in the boundaries as padding and S is the number of pixels by which the window moves after each operation.

Convolutional neural networks (CNN) contains a set of convolutional layers where image convolution is applied to an input image with many input filters learned automatically by the network. Pooling layers are further used to down-samples images (explained in section 2.4.3) and FC layers are used to make the final classification. It offers lower computation needs than if we just use all the pixels to train the network. Additionally, it successfully captures spatial dependencies, it is translation invariant and requires much less processing power.

In Figure 2.10, an example of the convolutional neural network pipeline can be seen. First, a sequence of convolutional layers extracts the feature maps of the image. For each of these convolutional layers, the features are down-sampled with a pooling layer. After that, we flatten the final output feature map to be able to pass it over a fully-connected layer. Finally, an MLP classifies the image based on the features obtained with the convolutional layers.

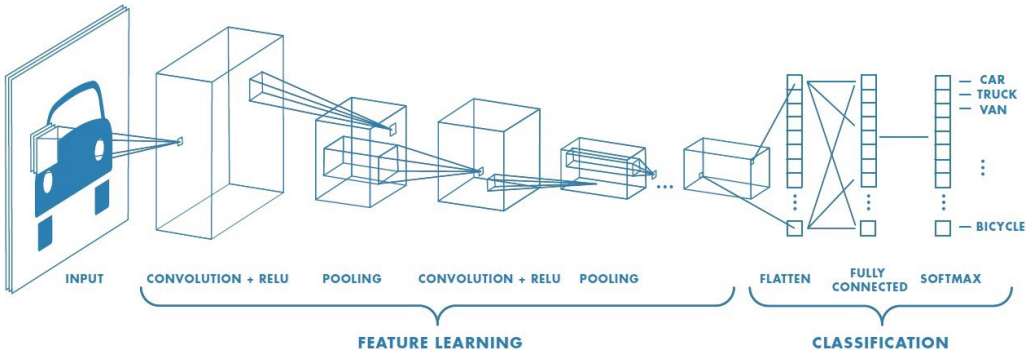


Figure 2.10: Example architecture of a CNN. Source [17].

2.4.3. Pooling Layers

Even when applying convolution to an image these images are usually still very big. When we look at an image we are not interested in obtaining features from each pixel, but in regions of pixels. That's were a *pooling layer* can help us. The pooling layer aggregates the spatial activation of each input feature map from a convolutional neural layer. The pooling is done in each feature map independently and the final result is smaller sized feature maps. The dimension reduction of pooling layers reduce the number of parameters and, as a consequence, decrease the computational load. Most basic and popular pooling methods are Max Pooling and Average Pooling:

- **Max Pooling** returns the maximum value from the portion of the image covered by the kernel window. For example, a max-pooling operation with stride 2 and filter size 2×2 over

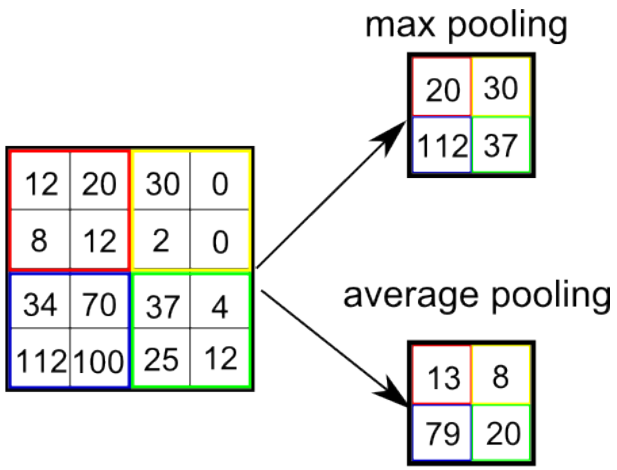


Figure 2.11: Most basic and popular pooling methods. Source [18].

a feature map of size 4×4 would return a 2×2 feature map with the max value of each kernel.

- **Average Pooling** on the other hand returns the average of all the values from the portion of the image covered by the kernel window. Visual illustrations of both methods in Figure 2.11.

Max Pooling has shown better performance than Average Pooling, thus it is more commonly used in literature.

2.5 Influential neural networks

In this section, we will review some of the most influential deep neural networks in the last couple of years. We will start first with AlexNet [2], the network that popularized DNN. After that, we will cover another popular deep neural network, named VGG [19], which improved upon AlexNet and has achieved very high performance. Finally, we will talk about residual networks (ResNet) [20], which has set a new standard in the field and has become the most widely used network in recent years.

2.5.1. AlexNet

As mentioned before, the success of AlexNet [2] in the ILSVRC competition popularized the concept of neural networks. Let us see the specific details that made it so effective.

In the aspect of engineering details, AlexNet was one of the first neural networks to train in multiple GPUs in parallel, making it much faster to train. This proved that training neural networks are feasible despite the big amount of parameters to train. They also applied data augmentation *while* training the network. In this manner, they generated random image patches of size 224×224 that were randomly horizontally reflected, and also randomly RGB color shifted.

In the aspect of neural network architecture, AlexNet was a CNN starting with five convolutional layers, followed by three fully-connected layers and ending with a softmax of 1000 class labels. Max pooling was applied to the second, third, and fifth convolutional layers. ReLU activation function was applied to the output of every layer. Dropout was also applied in the first two fully-connected layers. Figure 2.12 depicts the architecture described for two GPUs.

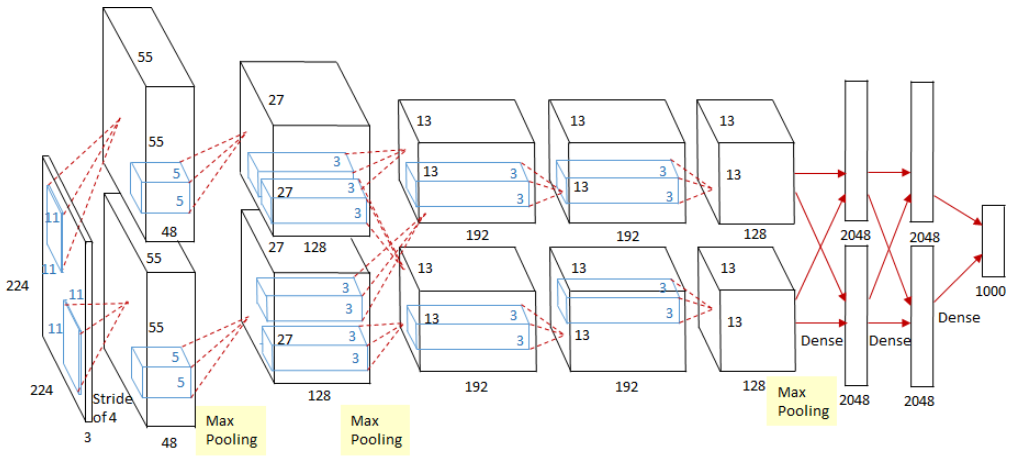


Figure 2.12: Architecture of AlexNet. The network is divided into two different branches because each branch is handled by a different GPU for parallel computation. Source [21].

2.5.2. VGG

VGG [19] is a DNN that won the ILSVRC 2014 challenge. Its architecture was inspired by AlexNet but differed in some key aspects. First of all, it was a much deeper neural network, about 16-19 layers. To be able to do this without having a huge number of parameters they used very small 3×3 convolutional filters. AlexNet had used filters of size 11×11. They provide different configurations of their network, but all the configurations share a common structure. They start with numerous convolutional layers in which the amount varied depending on the depth needed. Next, they are followed by three fully-connected layers, like AlexNet, with softmax of 1000 classes. All layers also use the ReLU activation function. The same data augmentation as AlexNet was applied and also the same dropout. The configurations can be seen in Figure 2.13.

ConvNet Configuration					
A	A-LRN	B	C	D	E
11 weight layers	11 weight layers	13 weight layers	16 weight layers	16 weight layers	19 weight layers
input (224 × 224 RGB image)					
conv3-64	conv3-64 **LRN**	conv3-64 **conv3-64**	conv3-64 conv3-64	conv3-64 conv3-64	conv3-64 conv3-64
maxpool					
conv3-128	conv3-128	conv3-128 **conv3-128**	conv3-128 conv3-128	conv3-128 conv3-128	conv3-128 conv3-128
maxpool					
conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256	conv3-256 conv3-256 **conv1-256**	conv3-256 conv3-256 **conv3-256**	conv3-256 conv3-256 conv3-256 **conv3-256**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512	conv3-512 conv3-512 **conv1-512**	conv3-512 conv3-512 **conv3-512**	conv3-512 conv3-512 conv3-512 **conv3-512**
maxpool					
FC-4096					
FC-4096					
FC-1000					
soft-max					

Figure 2.13: Architecture of different configurations of VGG. Each configuration A to E increasingly varies on the depth of weight layers. In bold the difference between the previous configuration. Source [19].

VGG became significantly important in the field of neural networks because it demonstrated that deeper networks achieve higher performance. This finding has become a crucial part of the design of neural networks and has been used in all future approaches.

2.5.3. Residual Networks

Typical networks continuously use stacked layers to fit a desired underlying mapping $h(x)$. In these networks, it can be seen that a degradation problem arises the deeper the networks get in. The accuracy gets saturated and then degrades rapidly. This is called the *vanishing gradient problem*.

He et al. [20] suggest a new method to solve the vanishing gradient problem. They propose to fit the stacked non-linear layer to a residual mapping $f(x) = h(x) - x$ with the assumption that the residual function $h(x) - x$ is an approximate mapping of $h(x)$. In this way $h(x) = f(x) + x$. They adopt this residual learning creating the following elements for the neural network architecture. First, a building block defined as:

$$y = f(x, w_i) + x \tag{2.10}$$

where x and y are the input and the prediction vectors. $f(x, w_i)$ is the new function to be learned. This function can be represented by a CNN. $f + x$ is computed with element-wise addition on two feature maps and by a shortcut connection.

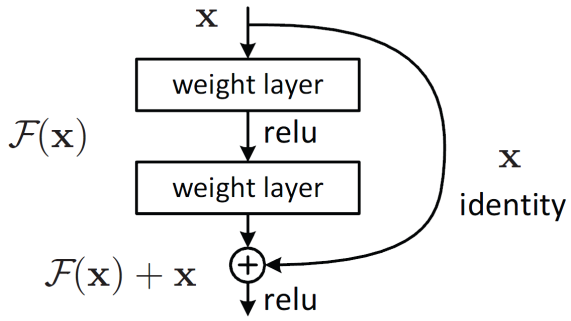


Figure 2.14: Residual learning: a building block. Taken from [20].

A residual network (ResNet) can be created with different configurations. Some examples can be seen in detail in Figure 2.15. The difference between configurations depends on the number of layers and the size of those layers. Each layer is made up of *blocks*, which are made up of convolutional layers, batch normalization layers, and shortcut connections. A shortcut connection is simply a direct connection between the input and output of a block. In Figure 2.14 there is an example block with an identity residual connection.

With this idea, they compared plain VGGs network with their proposed ResNet. The results show that their model manages to overcome the degradation problem and demonstrate accuracy gains when the depth increases. It got state of the art results in the ImageNet dataset with the added benefit of having fewer parameters (18% of VGG19[2] for ResNet34)

With this model, they got 1st place on the ILSVRC 2015 classification task and it has become one of the most popular models used for computer vision tasks. Due to these facts, we use these models for our experiments.

[2]To distinguish between configurations the model name is followed by the number of layers, for example, for configuration D in Figure 2.13 we will call it VGG16.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure 2.15: Architectures for ImageNet. Building blocks are shown in brackets with the numbers of blocks stacked. Down-sampling is performed by conv3_1, conv4_1, and conv5_1 with a stride of 2. Taken from [20].

CHAPTER 3
Writer Identification

Handwriting is a behavioral biometric (other examples are voice, signature, gait) that varies between individuals. When writing letters of a word, each person is used to writing in unique, characteristic shapes, called allographs. As an example, we can see a clear variation of handwriting from three different writers in Figure 3.1. It is this clear variation that permits writer identification even when the same text is written.

Figure 3.1: An obvious difference of character shapes and writing style can be seen between three different writers. Image from [22].

Historian and forensic examiners are some of the people who use writer identification heavily in their fields where they manually try to identify the authorship of some handwritten document. Manually operating with large numbers of handwriting documents involves laborious work and a lot of time. Current databases of handwritten documents are usually quite large, and the hope is that automated methods would assist researchers to work through these large collections. Hence, writer identification aims to design computer algorithms that would identify the most probable author given a sample from an unidentified author. The objective is not getting 100% right who is the author, but rather, that the system at least finds the author, for example, in its top 10 list (getting 100% accuracy that the author is in this list would be a desirable system).

The typical writer identification system will get a query unlabeled sample that will be contrasted with a database of samples of known authorship to produce the most probable list of writers. A conceptual example of the system can be seen in Figure 3.2.

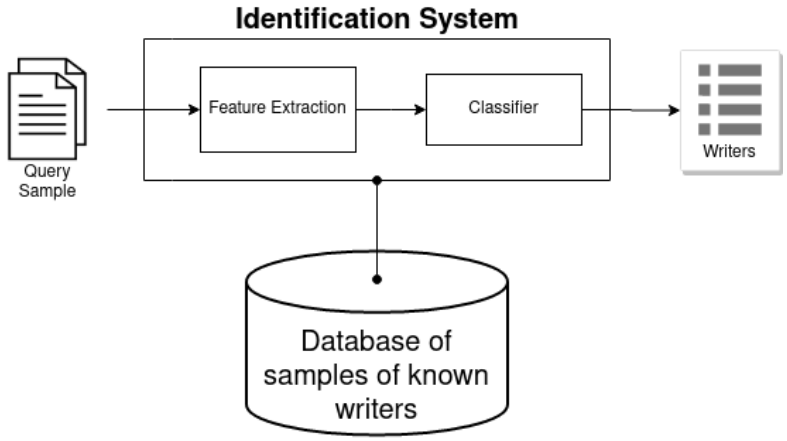


Figure 3.2: The pipeline of a writer identification system.

3.1 Typology

Writer identification can be categorized depending on the data source and its text dependency.

Data source. On the one hand, it can be *Offline* which means images of handwritten physical pages. This is the type of data in our case. CNN would be a good deep learning-based choice to work with these images as the features are associated with words, characters, lines, or paragraphs. On the other hand, the data source can be *Online* which consists of handwritten data associated with temporal information. Usually, this is obtained by devices like tablets and a stylus which records the pressure, speed, and angle on the surface of the tablet. In comparison to offline data, online data has more available information with the speed and order of the writing (offline has only the final result), so usually, models in this domain have better accuracy. Recurrent Neural Networks (RNN) are better suited for this task because of the time dimension of the data. Zhang et al. [23] has good results on an end to end model which uses an RNN model.

Text dependency. *Text-dependent* methods use specific text itself as information to distinguish between authors. Because text-dependent methods need that the same text is written. One example of tasks using these methods is signature verification where two signatures (same text) must be contrasted to determine if it's written by the same author. In contrast, *text-independent* methods use the image characteristics of the words or letters and work for any text. Our work follows this last line.

3.2 Methods

All writer identification methods consider two essential aspects. The first aspect is feature extraction, in which discerning features are extracted from the document sample to later be contrasted with other samples. The second aspect is a classification in which different algorithms are used to compute the comparison of features and determine the authorship. The most critical aspect is feature extraction for the creation of a good writer identification system.

Writer identification methods can be divided into two categories:

- *Feature-based methods* compare the handwriting samples according to grapheme, structural or textural features. Texture features inform of texture changes in the handwriting that normally occur because of how the pen is grabbed by the writer or because of the writing inclination used. Grapheme features reveal the specific character shapes of the writer as seen in Figure 3.1. Structural features are more related to a higher level of handwriting

characteristics, like contours and how letters are connected. Figure 4.2, Figure 4.3 and Figure 4.1 show examples of texture features, grapheme features and structural features respectively.

- *Deep learning-based methods.* The rapid rise of DNNs has attracted researchers to use them in this domain. Most of the researchers use the DNN as a feature extractor and use other techniques to classify based on the features extracted by the network. A few of them try to build an end to end classification system with the DNN. In this work, we follow the latter path of building an end to end classification system.

3.3 Evaluation

There are different evaluation metrics used for the writer identification task. The evaluation metrics used are:

- **Top-k accuracy.** The most common metric used for any neural network model is accuracy which measures the percentage of correctly predicted outputs among the total number of predictions and it would be the same as top-1 accuracy with k=1. Top-5 accuracy means that given the five highest probability predictions of the model any of these five predictions must match the true target. Generalizing, Top-k accuracy is measuring the percentage of when the correct prediction is in one of the k largest predicted values of the model.

  In our experiments we measure accuracy (k=1), Top-5 (k=5) and Top-10 accuracy (k=10).

- **Mean Average Precision (mAP).** This metric is a common evaluation method for information retrieval systems. It is calculated by taking the Average Precision (AP) over all ranks k obtained by a given query q:

$$\mathrm{AP}(q) = \frac{\sum_{k=1}^{n} P(k) r(k)}{R} \tag{3.1}$$

  where k is the rank in the sequence of retrieved documents, n is the number of retrieved documents, $P(k)$ is the precision at rank k in the ordered list of retrieved documents, R is the number of relevant documents, $r(k)$ is an indicator function equaling 1 if the item at rank k is a relevant document, zero otherwise.

  The mAP is computed as the mean of AP for each query of the set of queries Q:

$$\mathrm{mAP} = \frac{\sum_{q \in Q} \mathrm{AP}(q)}{|Q|} \tag{3.2}$$

CHAPTER 4
State of the art

In the last chapter, we reviewed some of the concepts and algorithms required to approach the offline text-independent writer identification problem. Now in this chapter let us look at some of the work done in this field in the last few years. This analysis of the state of the art is mainly based on [24] which does a review of methods between 2011 and 2016 across three major languages: English, Chinese, and Arabic.

The chapter will follow the categorization proposed in this analysis and we will include another category called deep learning methods, as it was not considered in this work. All results are listed on Table 4.1.

4.1 Feature-based methods

Feature-based methods concentrate on extracting distinguishing features for writer identification. Feature-based methods are further separated into three groups: texture-based methods, structure-based methods, and grapheme-based methods. Texture based approaches focus on textural features between the text and the image while the structure-based approach focuses on structural features of the word and character positions. Grapheme-based methods generate from the handwritten texts a codebook of graphemes with bag-of-words or bag-of-features techniques which are then used for identification with the use of histograms.

4.1.1. Structure-based methods

There is one interesting study [25] in which they used the width of ink traces for writer identification. They created the Quill feature which is a probability distribution of the relation between the ink direction and the ink width. Pen properties and writer's movement style are some examples of Quill features. They also propose a variant of Quill named Quill Hinge which has good results and proves that ink width patterns are useful.

Most common features in this category are based on the scale invariant feature transform (SIFT) algorithm. SIFT features (Figure 4.1) encompass the entire structural information and is invariant to the scale and orientation of the characters. The advantage of SIFT features is that its relative positions do not change from image to image thus is useful for handling word and characters structures in a text.

In [28] they used SIFT due to these properties. The authors first segment the words from the handwritten images and extract SIFT descriptors (SDs) and the respective scale and orientations (SOs) from the segmented words. From the SDs and SOs, they obtain signatures and histograms respectively, which then are used to distinguish the style between writers.

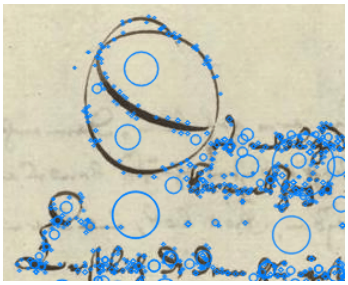


Figure 4.1: SIFT features on a handwritten sample image of the ICDAR 2017 dataset [26]. Image from [27].

The same authors also propose [29], where they modify descriptors (SDs and triangular descriptor (TD)) to incorporate orientation information and named them as modified SD or MSD. They use the proposed MSDs as they think orientation is important for writer identification and SDs do not take into account the orientation. Similar to before they use MSDs to create the MSD Histogram (MSDH) which is computed by using the bag-of-words technique. To further improve performance TD Histogram (TDH) based on triangular contour points of handwriting is computed. Finally, identification is determined with the distance between MSDHs and TDHs. Experimental results show that this method is more effective than the mentioned before with the unmodified SIFT descriptor.

All of these methods have the advantage of being language insensitive and can work well in different languages.

4.1.2. Texture-based methods

Textures of handwriting texts are constructed based on the characteristic properties of the handwritten image. After that, a feature extraction algorithm is chosen and classification is done computing the distance between features.

One of the first studies was of Said et al. [30] where the texture feature was extracted by Gabor filters and grey level co-occurrence matrix (GLCM). He et al. [31] used Hidden Markov Tree (HMT) model in the wavelet domain for feature extraction. Helli and Moghaddam [32] used Gabor and XGabor filter for feature extraction.

Two more popular textural descriptors are Local Binary Patterns (LBP) and Local Phase Quantization (LPQ) which have been used in a wide variety of applications with great success. Bertolini et al. [33] attempted approaching writer identification with these descriptors. A texture representation image is generated by compressing handwriting utilizing overlapping individual connected components. As an example we can see a texture generated in Figure 4.2 where the text is compacted removing all spaces but the original inclination of the characters is maintained. After that, LBP and LPQ are used for feature extraction. Finally, the classification is done by a Support Vector Machine (SVM).

In [34] the authors propose the identification of different writers by making use of novel direction, curvature, and tortuosity based geometrical features. Furthermore, the paper proposed the improvement of state of the art edge-based directional features by using a filled moving window instead of edge moving window and chain code-based features by using a fourth-order chain-code list to improve discriminative power.

Texture-based approaches need a considerable amount of data to be able to extract reliable features, which is not always available.

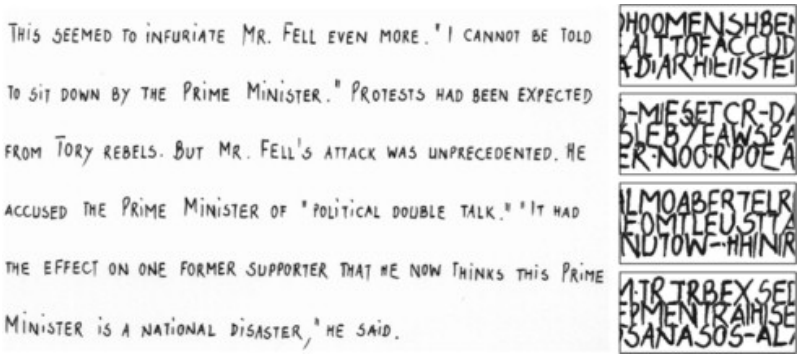


Figure 4.2: Texture generated from a handwritten sample image on the IAM dataset with the method of Bertolini et al. [33].

4.1.3. Grapheme-based methods

Grapheme based methods focus on the features of the characters of the handwriting itself. Features like contours of characters, shapes, style, width, height, etc. The traditional pipeline of grapheme based approach follows four stages: preprocessing of the handwriting document; generation of a reference base codebook (Figure 4.3); feature extraction; and the classification stage. The main phase is the generation of a codebook which consists of a collection of grapheme features. There are many ways to generate this collection of features as well as there many ways to extract the features from this collection.

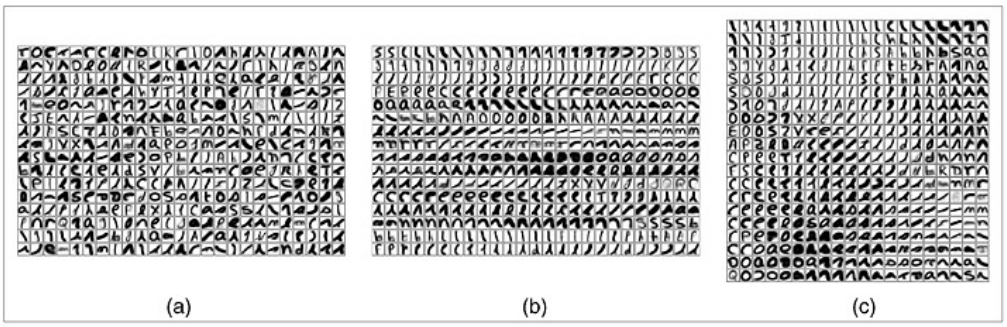


Figure 4.3: Examples of codebooks with 400 graphemes. Taken from [22].

Jain and Doermann [35] proposes the generation of features for the codebook with K-Adjacent Segment (KAS). The KAS method has better performance as a single feature descriptor than other methods where many features are combined to create good acceptable results.

Kirli et al. [36] proposes a novel dynamic windows based feature extraction model. These dynamic windows can adapt to any type of handwriting. They use the dynamic windows to extract features from three special writing zones. As for classification for the writer identification they use k-nearest neighbor (K-NN), Gaussian Mixture Models (GMM), and Normal Density Discriminant Function (NDDF) Bayes classifiers.

Although graphemes features are normally used for the codebook generation, Fiel and Sablatnig [37] extracted SIFT descriptors. After that, the histogram of occurrences of a new document is then compared to the ones in the codebook. Finally, classification is done with the nearest neighbor method. Using local SIFT descriptors has the added advantage of not needing binarization preprocessing, avoiding information loss.

Ghiasi and Safabakhsh [38] introduce two new efficient methods for the generation of codebooks from contours. The first method uses the actual pixel coordinates of contour fragments while the other method computes the linear piece-wise approximation using segment angles and lengths. For

feature extraction, they use the occurrence histogram of the shapes in the codebook as people's writing of characters is quite different and varies from person to person. They especially focus on small fragments of the handwriting that frequently appear in various characters. The use of small fragments leads to a faster generation of the codebook. Although generating the codebook is fast, computational requisites are larger.

As we have seen, the usual approach includes using a single codebook of graphemes features. The use of several different codebooks as an ensemble technique might be helpful to improve the performance of writer identification. Khalifa et al. [39] applies this approach of an ensemble of codebooks complemented with kernel discriminant analysis using spectral regression (SR-KDA). SR-KDA is chosen as a way to reduce dimensionality and avoid over-fitting problems that arise with combining multiple codebooks. The method improved classification accuracy using many distinct codebooks from randomly generated grapheme features. This ensemble of codebooks has shown an 11% increase compared to the use of a single codebook.

Garz et al [40] adopt a simpler and faster approach in contrast with the studies mentioned before where complex methods and heavy preprocessing are done. Their approach consists of using a set of novel descriptors extracted from geometrical interest points at various scales like from strokes, junction, endings, and loops. The proposed descriptors reduce the compute time compared to other methods and are more simpler and efficient to use. Another advantage of their approach over other methods is that it does not require any type of image processing such as binarization or segmentation which may introduce errors for the identification process. The only drawback of this approach is the amount of data required to produce a good model with appropriate results.

More recently [41], used a combination of SIFT and RootSIFT descriptors in a set of Gaussian mixture models (GMMs). SIFT and RootSIFT descriptors are extracted from handwritten word images. Then, similarity and dissimilarity GMMs of every writer are generated using these descriptors. After that, a new histogram-based method is used to generate an intermediate prediction score. Finally, a score fusion function is used to get the final prediction score. With their method, they have obtained the current best results in the Firemaker [42] dataset.

4.1.4. Combination of structure and grapheme based methods

Bulacu and Schomaker [22] methods work both at texture level and the character-shape (allograph or grapheme) level. At the texture level, they use contour-based joint directional probability distribution functions (PDFs) that encode orientation and curvature information of the writing style. At the character-shape level, they use a random pattern generator of ink-trace fragments. The base codebook is generated by segmenting the allograph into several fragments. Finally, the writer is identified with the PDF of the patterns of the writing. The authors have also proposed several new features, such as edge direction distribution, edge-hinge distribution, and directional co-occurrence.

Siddiqi and Vincent [43] also extracted features at the allograph level and texture level. At the allograph level, they simply create a codebook of the most similar shapes used. At the texture level, they extract features from the chain code of the handwriting contours. Finally, KNN with $k = 1$ is used for classification.

The combination of different types of features makes these types of approaches very effective for writer identification and are interesting to investigate.

4.2 Deep Learning based methods

Deep learning methods have been gaining significant momentum and is having wide success in many different fields, such as computer vision, speech recognition, and natural language processing,

to name just a few. Interestingly, there have been few works in the field of writer identification and most of the work is based on feature extraction for text retrieval and not end to end classification.

One of the first attempts at writer identification with deep learning methods was proposed by Fiel and Sablatnig [44] in 2015. The authors proposed to train a CNN over the input images which consisted of segmented words and line images and used the second last fully connected layer as a feature vector after training is finished. The obtained feature vector is then compared to precalculated feature vectors of the dataset by nearest neighbor classification. Note that they did not use full pages as data but segmented words and lines images which were also binarized, normalized, and deskewed. The data was also artificially augmented by random rotations of each image by -25 to 25 degrees. The CNN model used was the popular AlexNet.

In the same year, [45] used a very similar approach differing in a few respects. They also used the second to last layer of a CNN as a feature vector. However, in their case, the feature vector that formed the second last layer was encoded to form a global feature vector through Gaussian mixture models (GMM) super vector encoding. Apart from that, the CNN architecture used has minor differences and the input data was in this case small image patches of size 32×32 with no preprocessing. They obtained better results than the model proposed by Fiel and Sablatnig.

Another interesting work is of [46]. First, the authors state that a large amount of data is necessary to create a good model. For that reason, they propose a data augmentation technique to generate 500 handwriting samples for each writer by generating extra lines in the images by permuting words obtained by word segmentation on the original image. Secondly, they find that global appearance is a more suitable feature than local features, so they extract global features of the entire image instead. This differs from previous work were local features of smaller patches of the entire image are extracted. As the other research mentioned before, they also extract features with a CNN similar to AlexNet but use a joint bayesian technique to extract the most similar of the precalculated features in comparison with the obtained feature. The results reported in this paper are the best results for the ICDAR13 dataset of the CVL dataset.

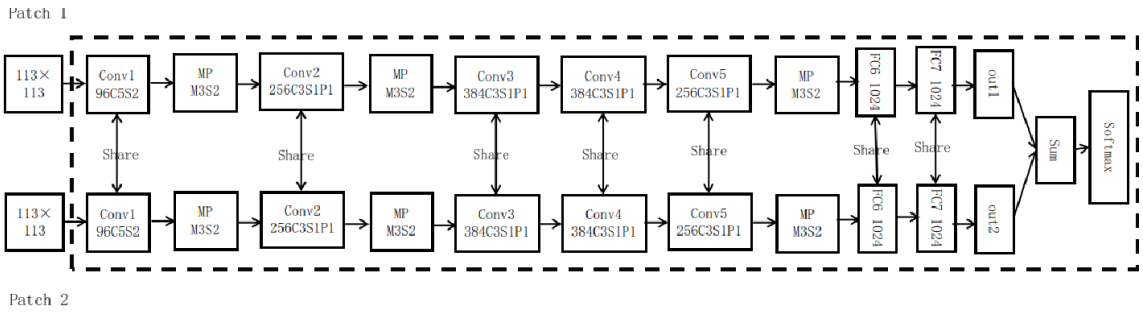


Figure 4.4: Architecture of DeepWriter proposed by Xing et al. [47].

One successful approach that involves classification and not feature extraction, was of DeepWriter [47] a deep multi-stream convolutional neural network consisting of two branches sharing the convolutional layers. The architecture of the proposed model can be seen if Figure 4.4. The two branches are based on the AlexNet architecture and take a pair of adjacent 113×113 image patches as input for each branch. They obtained good results on the IAM dataset.

Nguyen et al. [48] propose another end to end CNN classifier which extracts local features and combines the extracted features to form global features. First, they randomly sampled 64×64 image patches from each writer to form n-tuple images. Afterward, every image from n-tuple images is sent to two different branches of the CNN, one that extracts features at the sub-region level and the other at the character level. The sub-region level branch targets individual writing strokes features. The character level branch captures character shape features. Finally, the extracted local features are aggregated into global features and sent to a softmax classifier to make a prediction.

A very recent study [49] suggests a novel approach to writer identification by using word images. They call their DNN implementation *FragNet*. FragNet has two branches: a feature pyramid branch and a fragment branch. The feature pyramid branch is a traditional CNN used to extract feature maps in different scales from an input word image. For the fragment branch, the inputs consist of fragments that are segmented from the input word image and feature maps on the feature pyramid. This is achieved inside the network by cropping out a square region from a feature map of a convolutional layer. This fragment branch is trained so that the network can learn useful information on the fragments. The combined pieces of evidence of all fragments are taken into account to make the final prediction of the writer authorship. Figure 4.5 illustrates the architecture of FragNet.

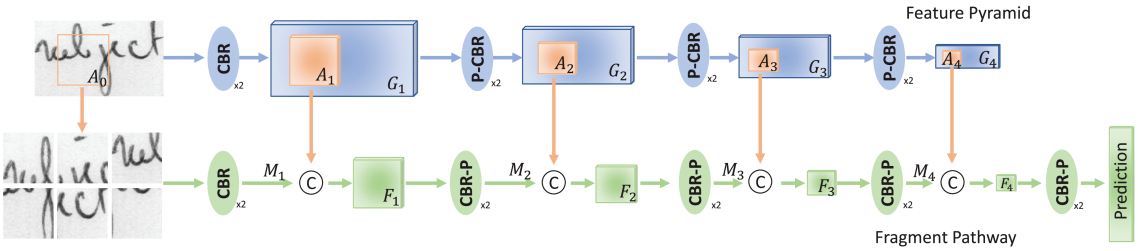


Figure 4.5: A FragNet network has two branches: feature pyramid (blue color) which accepts the whole word image as the input and Fragment branch (green color) which accepts the fragment as the input. (P)-CBR means the sequences of P: max-pooling, C: convolutional, B: batch normalization and R: ReLU layers. C with circle is the concatenate operation. x2 means that two blocks stacked together.G_i and F_i are the i-th feature maps in the feature pyramid and fragment branch, respectively. [49]

One of the major advantages of FragNet is that it can be interpreted. FragNet makes decisions based on fragments, therefore fragments that contributed the most can be visualized.

Structure-based approaches								
Year	Feature	Classifier	Ref.	DB	Writers	Top1 (%)	Top5 (%)	Top10 (%)
2014	MSDH + TDH	KNN	[29]	IAM	657	97.1	98.8	99.2
				ICDAR13	250	95.2	98.4	99
2014	SDS + SOH	Euclidean	[28]	IAM	657	**98.5**	99.1	99.5
				Firemaker	250	92.4	96.2	98.8
2012	Quill–Hinge	NN	[25]	IAM	657	97	–	98
				Firemaker	251	86	–	97
Texture-based approaches								
Year	Feature	Classifier	Ref.	DB	Writers	Top1	Top5	Top10
2016	Chain code	KDA	[34]	IAM	650	82.7	–	92.2
2013	Texture LPQ	SVM	[33]	IAM	650	96.7	–	–
Grapheme-based approaches								
Year	Feature	Classifier	Ref.	DB	Writers	Top1	Top5	Top10
2019	SIFT + RootSIFT	GMM	[41]	IAM	650	97.85	–	–
				Firemaker	250	**97.98**	–	–
				CVL	310	99.03	–	–
2016	p(Is,I), p(IBOS)		[40]	IAM	657	86.9	91.6	94.7
2015	Graphemes	SR-KDA	[39]	IAM	657	92	93	97
2013	Connected	KNN, x2	[38]	IAM	650	94.8	–	–
				Firemaker	250	95.2	–	99.2
2012	SIFT	x2	[37]	IAM	650	93.1	–	–
2011	KAS	SVM	[35]	IAM	650	92.1	94.5	95.8
2011	Global and local	KNN, GMM, Bayes	[36]	IAM	93	98.76	–	–
Combination of structure and grapheme based methods								
Year	Feature	Classifier	Ref	DB	Writers	Top1	Top5	Top10
2010	Codebook and contour	KNN	[43]	IAM	650	91	–	97
2007	Contour PDFs and ink trace	PDFs	[22]	IAM	650	89	–	97
				Firemaker	250	83	–	95
				Large	900	87	–	96
Deep Learning-based approaches								
Year	Feature	Classifier	Ref.	DB	Writers	Top1	Top5	Top10
2020	CNN with word fragments (FragNet-64)		[49]	IAM	657	96.3	–	–
				Firemaker	250	97.6	–	–
				CVL	310	99.1	–	–
2019	CNN with tuples of images of size 64x64		[48]	IAM	650	93.14	–	–
				Firemaker	250	93.56	–	–
				Large	900	**94.75**	–	–
2016	Multi-stream CNN (DeepWriter)		[47]	IAM	657	97.3	–	–
2016	CNN	Joint Bayesian	[46]	CVL	310	**99.7**	99.8	100
				ICDAR13	350	**99**	99.2	99.6
2015	CNN	GMM	[45]	CVL	311	99.4	–	–
				ICDAR13	250	98.9	–	–
2015	CNN	KNN	[44]	CVL	311	98.9	99.3	99.5
				ICDAR13	250	88.5	96	98.3

Table 4.1: Summary of state of the art in writer identification methods on IAM [50], Firemaker [42], IAM and Firemaker combined (Large), ICDAR 2013 [51] and CVL [52] datasets with respect to the type of approach. In bold best results in the respective dataset.

CHAPTER 5
Approach

As we have seen in the previous chapter there have been few deep learning works in the field of writer identification. Interestingly, all of the approaches used as an input training sample for the neural network, a small image patch. To the best of our knowledge, none of the works use a full page as training input. As a result, we propose a model that works with full-page images as well as comparisons with versions of the model working with image patches. In this chapter, we will describe the implementation details of the proposed model. The next chapter will show the results of our experiments.

5.1 Model

The neural network architecture used in our work is the well-known ResNet (Explained in 2.5.3), in specific the ResNet18 version. Larger versions of ResNet were not used as they reported virtually the same results while ResNet18, which is a much smaller model. Some preliminary tests were done on other popular models like VGG[19], AlexNet [2], and DenseNet[53]. Table 5.1 shows the size and the number of parameters of each model.

Model	Size	Parameters
AlexNet	233 MB	61 Millions
VGG11	507 MB	132 Millions
VGG16	528 MB	138 Millions
DenseNet121	33 MB	7 Millions
ResNet18	45 MB	11 Millions
ResNet34	83 MB	21 Millions
ResNet50	98 MB	25 Millions
MobileNet v2	14 MB	3 Millions

Table 5.1: Size and model parameters for each model tried.

Most of them reported worse results and required larger training time because of their large number of parameters, so they were not included in our study. Other preliminary tests on MobileNet(v2)[54] reported good results, which is an interesting finding because it has a much lower number of parameters and the model size is also only a few megabytes, making it perfect for mobile devices. However, in this study we will only work with ResNet18, MobileNet could be studied in future work in case of the need for a small model.

ImageNet pre-trained models are often used in neural networks for faster training and better performance. We also benefited from using pre-trained models. For this reason, we use pre-trained models on ImageNet in all our experiments. As a result, our base model is a ResNet18 (Figure 5.1)

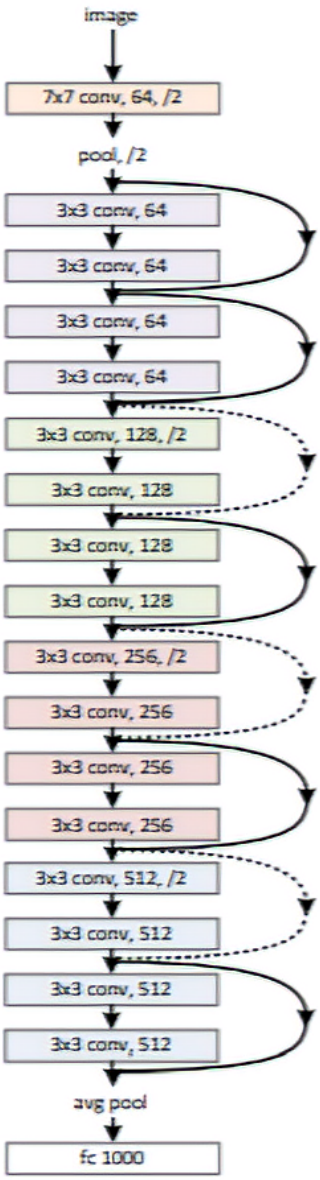


Figure 5.1: Our base ResNet18 model. The last layer output is changed to the number of classes of the trained dataset. Modified from [20].

pre-trained on ImageNet with the output changed to the number of classes in the respective text datasets discussed in 6.1. With this pre-trained model, we train it again in each dataset mentioned. This is also called *fine-tuning*. Fine-tuning is done because pre-trained models have already learned features for their tasks and these features could also be useful for other tasks. With this, there is no need to train the model from scratch, which saves time and resources.

Additionally, we fine-tune two variants of our models based on the input image. The first variant uses a full page as input. The second variant uses patch images of varying sizes, ranging from 100×100 to 1500×1500.

For our pages based model, a full page which has been processed with text padding (explained in Section 6.1.4) is received as training input. As test input, it also receives a full page.

Concerning our models that use patch images as input, as training input we extract randomly from a page image, n number of square patches with size $a \times a$, where n and a are chosen intuitively by hand. Random extraction of patches implies that any patch could be extracted from all the possibilities, meaning that there could be heavily overlapped patches.

As testing/validation input, we extract the same sized patch $a \times a$ but with stride $a \times a$ so that no overlapping patches are extracted and all the image information is received as input. We could have followed the same procedure as in the training phase, extracting random patches, but we believe obtaining all possible non-overlapping patches is a more consistent and robust manner to tackle the testing phase since it removes the random factor.

Furthermore, our patch models perform a voting scheme for improved accuracy. This proposed scheme can be seen in Figure 5.2. From all the patches extracted from a given page, predictions are computed to determine the probabilities of the possible writers. The maximum predicted writer for each patch is considered as a vote. As a result, the writer who receives the most votes is considered as the predicted writer.

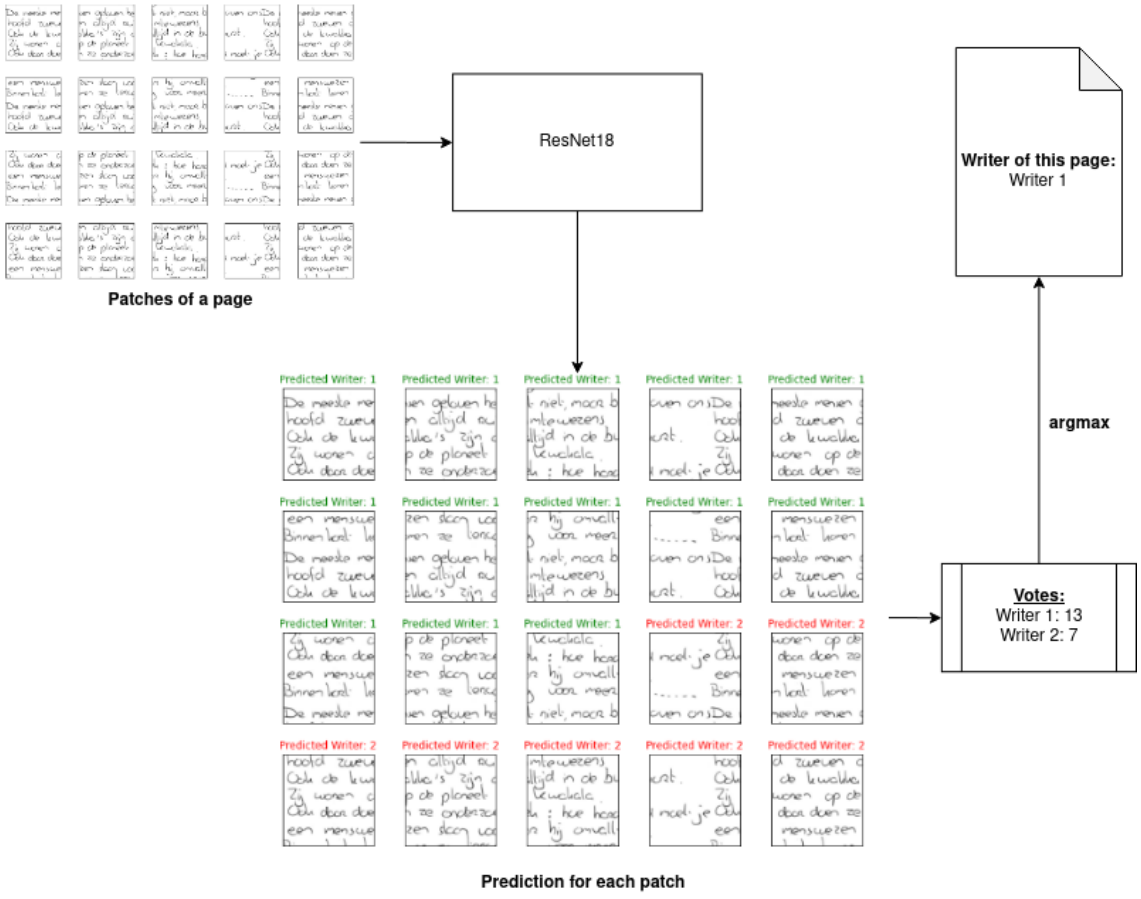


Figure 5.2: Visual illustration of the voting scheme used for our patch models. In green the true writer.

CHAPTER 6
Experiments and results

In this chapter, we present a set of experiments that investigate the performance of our proposed model. Our main contributions are the experiments that will measure the impact of page images and different image patch sizes.

First, we introduce the benchmark datasets used for our experiments in Section 6.1. After that, we give details about our hardware and software specifications in Section 6.2. Section 6.3 will specify the chosen hyperparameters for our model and the procedure followed. Next, our experiments will start in Section 6.4. In Section 6.4.1, we will analyze how our model performs with patch images following common sizes used in literature as input data. In Section 6.4.2 we further analyze our models on different sizes of patch images that were not tried previously. Additionally, we will evaluate our model on the proposed full pages as input model in Section 6.4.3. Section 6.4.4 experiments with historical handwritten data. Finally, in Section 6.5 we present the results of our best models and compare them with the state of the art.

Accuracy, top-5 accuracy, and top-10 accuracy (all explained in 3.3) were reported to measure the performance of the experiments. Confidence intervals are not shown for clarity but the worst-case 95% interval is $\pm$ 3%.

6.1 Datasets

In this section, we will describe the benchmark datasets used in this project and how we processed these data to prepare them for our experiments.

6.1.1. Firemaker

Firemaker [42] is a dataset in which 250 writers provide four handwritten pages each. Each page has a different writing condition and all are written in Dutch. The first page consists of a copy of a specific text so that each of the 250 writers is copying the same text. On the second page, writers use the same text mentioned before but written in upper case. The third page again contains the text copied before but writers were given the condition of writing in a style different than their natural style. The fourth and final page is a description of a cartoon comic image therefore each writer provides a different text. As they did in [48], we use the first page as a training and validation set and the fourth page as a test set. The second page written in uppercase is unusual to find in real case datasets and the third page could be used for forged binary classification tasks but is not useful for classification tasks and for that reason we did not use these pages for our experiments.

More information can be found in the document inside the dataset file [55] and some sample images are shown in Figure 6.1.

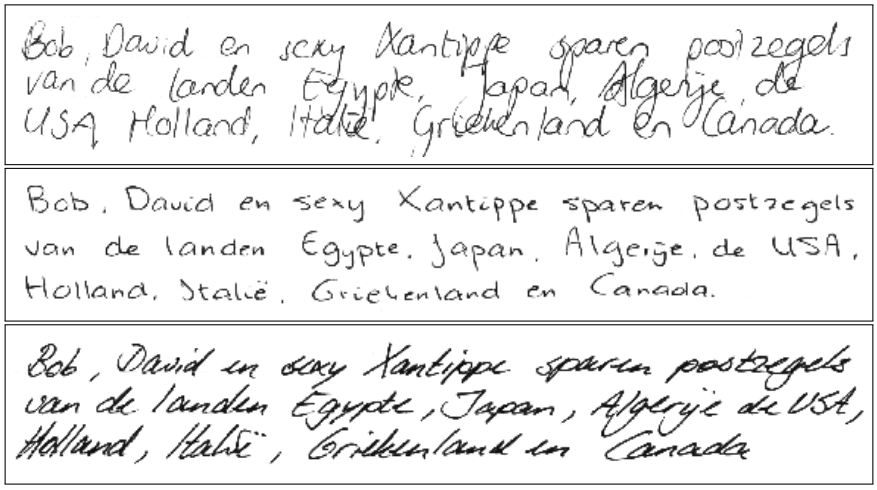


Figure 6.1: Three sample images from different writers that wrote the same text. Extracted from the Firemaker dataset.

6.1.2. IAM Handwriting Database

The IAM Handwriting Database was first published at the ICDAR 1999 and is described in [50]. It contains 1539 pages of handwritten text from 657 writers who provide a different amount of pages. They also provide labeled text lines and labeled words using automatic segmentation and are manually verified, for solving text recognition tasks, which is not our case. Sample images from three different writers can be seen in Figure 6.2.

Since the number of pages of each writer ranges from 1 to 58 pages in the original database, we modified the dataset to have exactly two pages for each writer. For writers who contributed more than two pages, we only keep the first two, and for writers with only one page, we cut the respective pages in half. This is the procedure typically used in literature and it will be useful for comparisons with the state of the art.

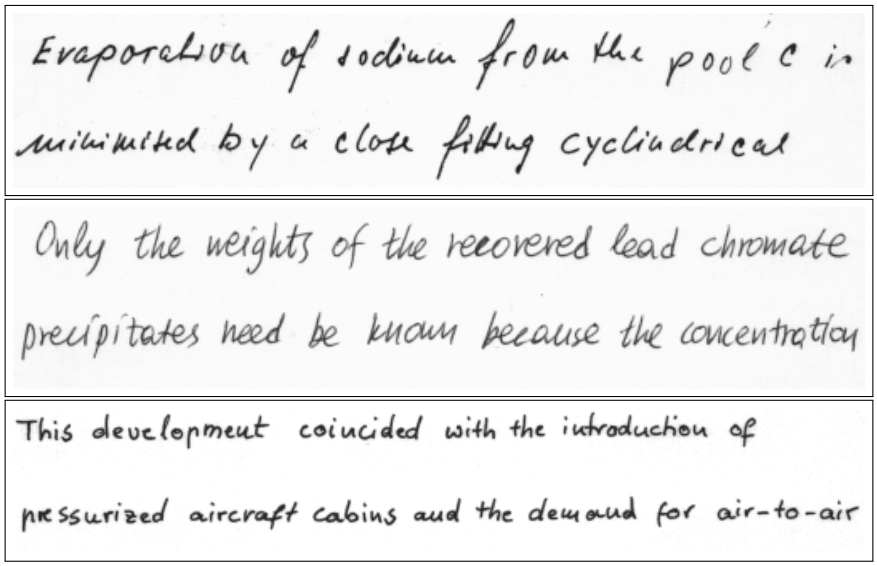


Figure 6.2: Three sample images from different writers extracted from the IAM dataset.

6.1.3. ICDAR 2017

ICDAR 2017 dataset was proposed on the 2017 Competition on Historical Document Writer Identification in the International Conference on Document Analysis and Recognition (ICDAR) [26]. The objective of the competition was to retrieve the most similar documents given a query document which consists of a historical handwritten page from the 13th to 20th century. For this purpose, they provide a training dataset of 394 writers which provide 3 pages each and a testing dataset of 720 writers which provide 5 pages each. These images consist of color images but they also provide the same sets as binarized images. Figure 6.3 presents some images of this dataset.

Each of these sets has disjoint writers so that participants extract features from the training set independent of the labels of the writers. This can be categorized as unsupervised feature learning in the case of deep learning architectures. This is not the objective in our case as ours is classification, we classify each page to a known writer. We use the test dataset of 3600 writers which provide 5 pages each, as an assessment of our models in historical handwritten data as well as for transfer learning experiments.

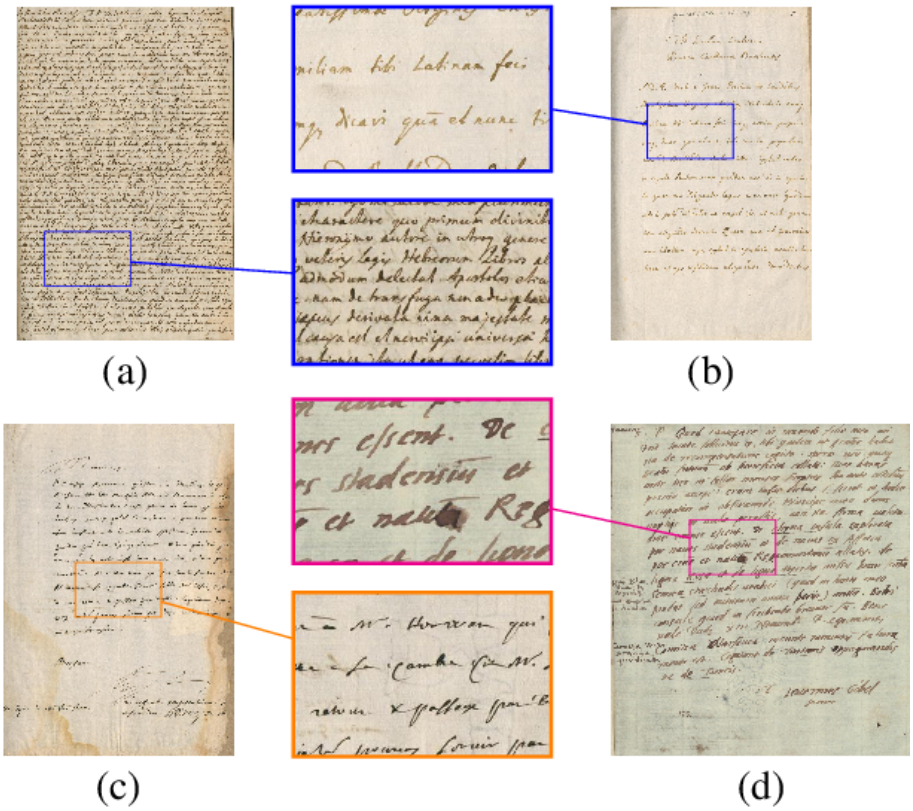


Figure 6.3: Four sample pages from the ICDAR 2017 dataset. (a) and (b) come from the same writer. (c) and (d) come from different writers. Image from [56].

It is interesting to note, that most of the participants in this competition used handcrafted features and performed better than the only system relying on deep learning, which had the worst results. It seems deep learning methods perform poorly in this dataset. This can also be seen in [27].

6.1.4. Data Preprocessing

As all these datasets were developed for tasks different from ours, we had to prepare our train, validation, and test sets. In this section, we give details of the steps followed for each dataset.

A division of training data and testing data was performed for each dataset explained in 6.1. For Firemaker we use the first page of the provided dataset as training and the fourth page as testing. In the case of IAM, for writers who provided two or more images, we use the first image as training as the second for testing. For writers who only provided one page, two halves are created, one for training and one for testing. Finally, in ICDAR 2017 from the 5 pages provided in the test set we use 3 as training, 1 as validation, and 1 for testing. All this information is also presented in Table 6.1.

Dataset	Writers	Training	Validation	Testing	Total pages
Firemaker	250	1 page		1 page	500
IAM	657	1 page/ half page		1 page/ half page	1314
ICDAR17	720	3 pages	1 page	1 page	3600

Table 6.1: Summary of the datasets constructed.

For all the pages in every dataset, a cropped image is obtained with handwritten images as compact as possible to avoid unnecessary blank backgrounds. As a result, we obtain a set of different sized cropped images. Fixed-size images are needed for neural networks. For this purpose, we define a novel technique called *text padding*. First, text padding pads a cropped image with a white background to the maximum height and width of all cropped images. Then, text padding fills the blank space with the original text, performing a copy from left to right and top to bottom, as shown in Figure 6.4. This also can be seen as data augmentation. So, the aim of this technique is twofold: getting fixed image sizes and data augmentation. Apart from text padding, no other data augmentation techniques were used, even during training.

Normalization is done in every dataset by finding the mean and standard deviation of the dataset and subtracting this mean of each image and dividing it to the standard deviation.

6.2 Hardware and software specifications

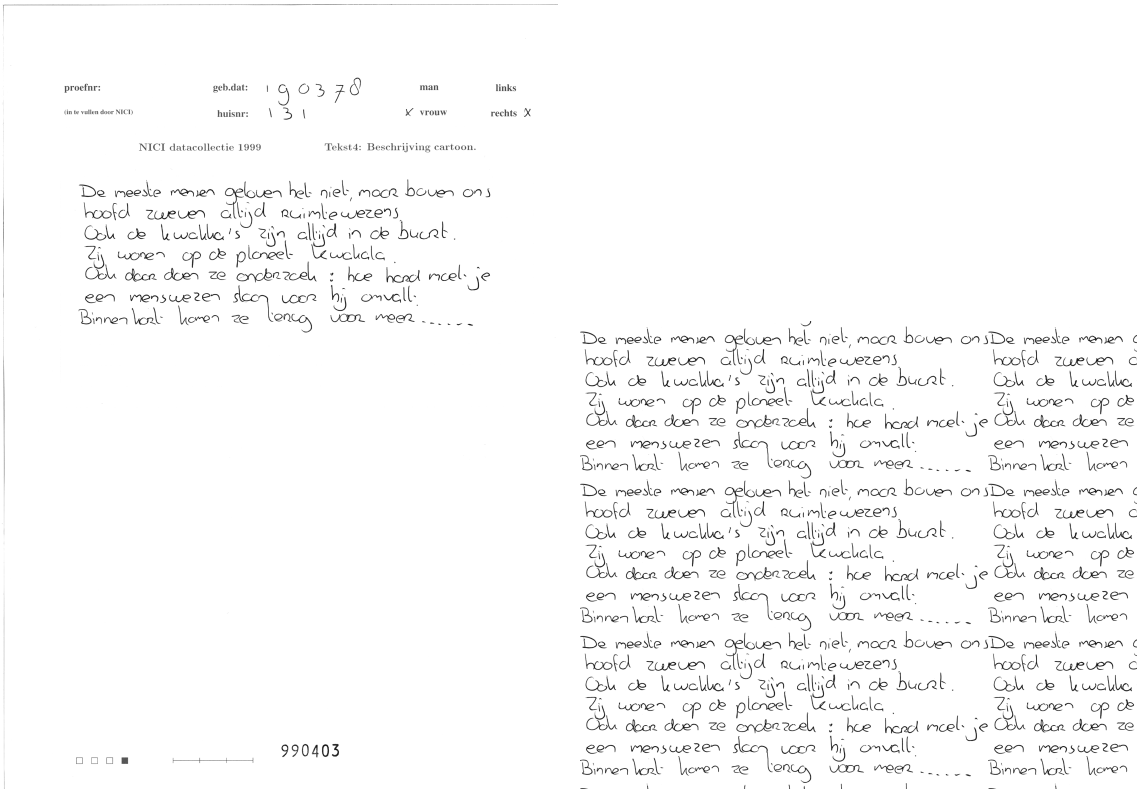
Hardware

For training, we use two Nvidia GeForce RTX2080 GPUs provided by the PRHLT Research Center. The CUDA version used is the latest at the time of the writing (version 10.2).

Most of the time we use distributed training in these GPUs. We also use 16-bit precision operations for faster training instead of the normally used 32-bit precision operations which are used by default.

Software

All the implementation is programmed in Python. Specifically, for creating our neural network models the open-source machine learning framework Pytorch version 1.5[57]. Pytorch is a fast and flexible framework that leverages machine learning algorithms and is designed to provide easy to learn tools for creating deep learning models. Its torchvision model zoo [58] provides popular state of the art models architectures pre-trained on Imagenet [3], ready to be loaded and used for finetuning or feature extraction. We use these models to create deep learning models and train them in the context of our experiments.



(a) **(b)**

Figure 6.4: Example of the proposed text padding. (a) original page (b) cropping and applying text padding to (a).

Readers are invited to learn more about the implementation and to replicate the experiments, using our publicly available code in:

https://github.com/akpun/writer-identification

6.3 Hyperparameters optimization

Hyperparameters are parameters of the model that can be changed to influence the model performance. Some hyperparameters involve changing parameters of an algorithm, for example, the learning rate of the optimizer function. We call them *algorithm hyperparameters*. Others involve changing a part of the model architecture, for example, which optimizer algorithm to use. We call them *model hyperparameters*. Neural networks are infamous for requiring extensive hyperparameter tuning. In this section, we specify the procedure we followed for each hyperparameter.

Algorithm hyperparameters

The algorithm hyperparameters that need to be tuned in our case include the patch size of the image, the number of patches to extract from a single page sample, the learning rate, and batch size.

For the case of patch size and the number of patches, as mentioned before, n is the number of patches extracted with size $a \times a$ where n and a were chosen intuitively by hand. Therefore, we evaluated the performance of each different size a and number n with a fixed batch size of 32 and a learning rate of 0.01. The set of sizes a tested include 256, 300, 400, 500, 600, 800 and 1000. The set of values n includes 10, 32, 64, 100, and 200.

After obtaining the best combination of n and a, we first tune the batch size and learning rate in the ICDAR 2017 dataset because is the only dataset that has a disjoint validation set. IAM and Firemaker datasets use a validation set extracted from the training set. With the best parameters obtained in ICDAR 2017, we can estimate the approximate range of the hyperparameters with good accuracies for other datasets. This is done with the assumption that the hyperparameters follow approximately the same distribution for all handwritten text datasets and our experiments appear to work well with this assumption. For the case of batch size, the range is between 4 and 32 and it is also limited by the data size. For the learning rate, it ranges between 0.0001 and 0.005.

On IAM and Firemaker we follow the same procedure of finding n and a, but for the batch size and learning rate, a random search is done in the mentioned ranges to find the best accuracy.

For the full page case, we only need to tune the batch size and learning rate. That being so, a random search is done in the same manner as in the patch case, trying to find the best batch size and learning rate combination.

Model hyperparameters

Some parts of the model architecture can be interchanged with other methods for different results. We chose to do all our experiments with the same methods. In the case of the optimizer algorithm, the optimizers that were considered were Adam [59], AdamW [60], and stochastic gradient descent (SGD). SGD was the optimizer chosen as it reported better results than the other alternatives. Cross Entropy Loss is used as our loss function. Several different loss functions exist, but we have not considered them as cross-entropy is a robust loss function that is known to work well.

6.4 Experiments

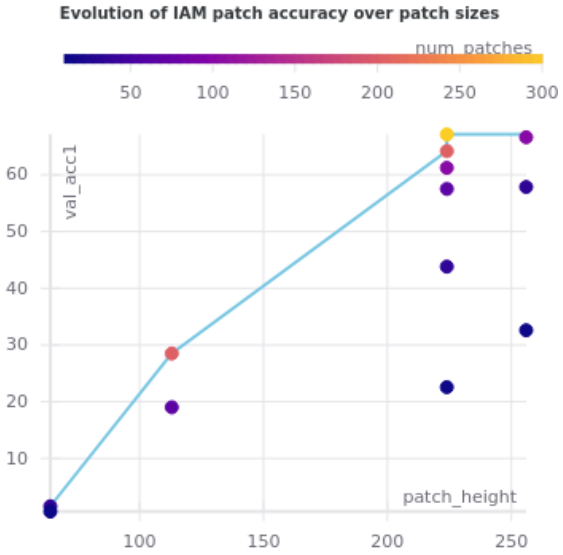
In this section, we will do a series of experiments to end up with a good performing model. We start with an initial experiment working with commonly used patch image sizes in literature. Next, we will extend this experiment trying even bigger patch sizes that never have been tried in literature. After that, we will experiment with our proposed full-page input. Furthermore, we will investigate the performance of our models in historical manuscripts, which is known to be a challenging task. Finally, we present the results of our best models and compare them with the state of the art.

6.4.1. Experiment 1: Commonly used patch sizes in literature

First, we try our model with small-sized image patches. To do so, we employ the patch sizes found in literature: 32×32 [45], 64×64 [48], 113×113 [47, 49], 224×224 [46] and 256×256 used by the University of Fribourg on the competition of ICDAR17. Normally, for patches of small sizes, some preprocessing is done to choose patches that have a minimum type of information (there may be the case where blank patches are extracted). This involves using different techniques for measuring the amount of information in a patch, for example, the number of edges could be counted to determine if there are any letters or words. However, we do not employ any preprocessing. We expect that the model ignores uninformative patches. We train the models for 20 epochs but use a fixed batch size of 32 and learning rate 0.01. Table 6.2 and Figure 6.5 presents the best results for each patch size.

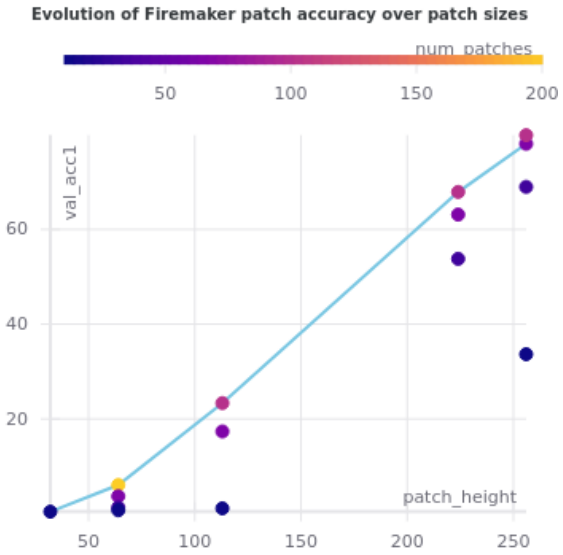
Dataset	Num patches	Patch size	Top1 (%)	Patch Accuracy (%)
IAM	32	64	1.2	1.6
	200	113	87.5	28.5
	300	224	95.73	67.1
	100	256	95.12	68
Firemaker	10	32	0.4	0.4
	200	64	13.2	6
	100	113	72.8	23.34
	100	224	92.4	67.9
	100	256	96	79.45

Table 6.2: Best results with the patch size used in literature. Top1 indicates the accuracy of the pages using the voting scheme. Patch accuracy indicates the accuracy of the individual prediction of each patch.



(a) IAM accuracy

(b) IAM patch accuracy



(c) Firemaker accuracy

(d) Firemaker patch accuracy

Figure 6.5: Best results with the patch size used in literature. (a) and (c) indicates the accuracy of the individual prediction of each patch. (b) and (d) indicates the accuracy of the pages using the voting scheme. The blue line indicates the best results for each patch size.

The graphs show that accuracy improves with bigger patch sizes. With a patch size of 256, it reaches 95% and 96% accuracy for IAM and Firemaker, respectively. It seems that our model is not capable of discerning any valuable features with small patch sizes, which can be observed with dramatically low performance. Still, we can see in plots 6.5b and 6.5d that our voting scheme seems to improve the accuracy noticeably for these cases. If we look at the number of patches, we can see that a larger number of patches increases the performance of the model considerably, even for cases where the patch size are small. Overall, we have obtained good results with patch sizes 224 and 256. However, with the steep upward trajectory with bigger patch sizes, it will be interesting to see until what size the accuracy peaks.

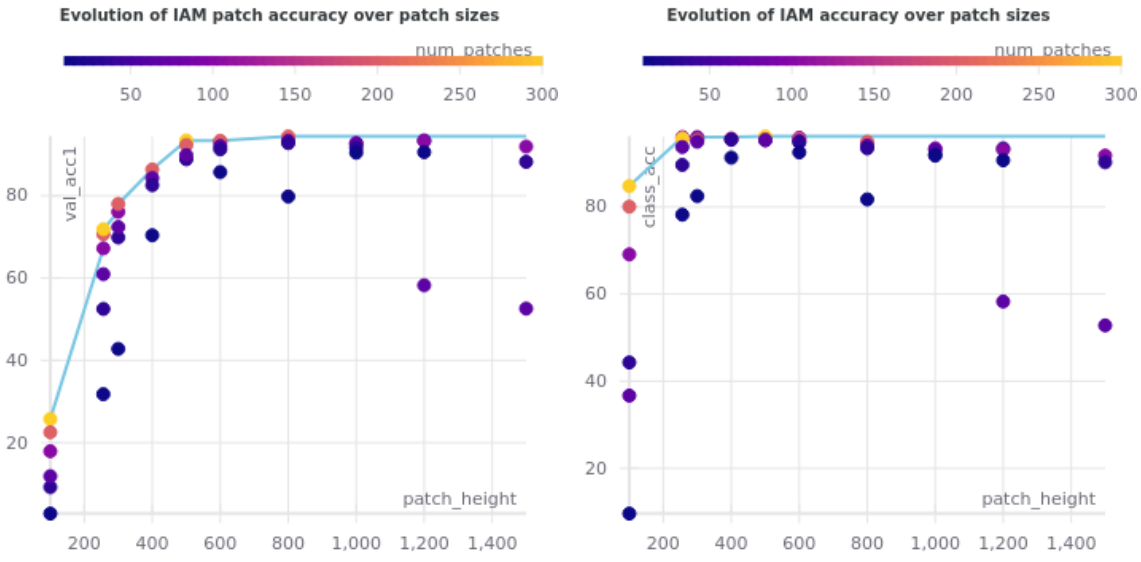
6.4.2. Experiment 2: Using bigger patch size

Looking at the results of our previous experiments, patch accuracy seems to be improved with bigger patch sizes. This raises the question of whether even larger patch sizes will perform better. We evaluate this hypothesis in the following experiments.

The chosen image sizes to test were of 300[1], 400, 500, 600, 800, 1000, 1200 and 1500. Table 6.3 and Figure 6.6 and 6.7 show our best results.

Dataset	Num patches	Patch size	Top1 (%)	Patch Accuracy (%)
IAM	100	600	96.3	93.1
Firemaker	100	1500	99.2	99.2

Table 6.3: Best results with big patch sizes never used in literature. Top1 indicates the accuracy of the pages using the voting scheme. Patch accuracy indicates the accuracy of the individual prediction of each patch.



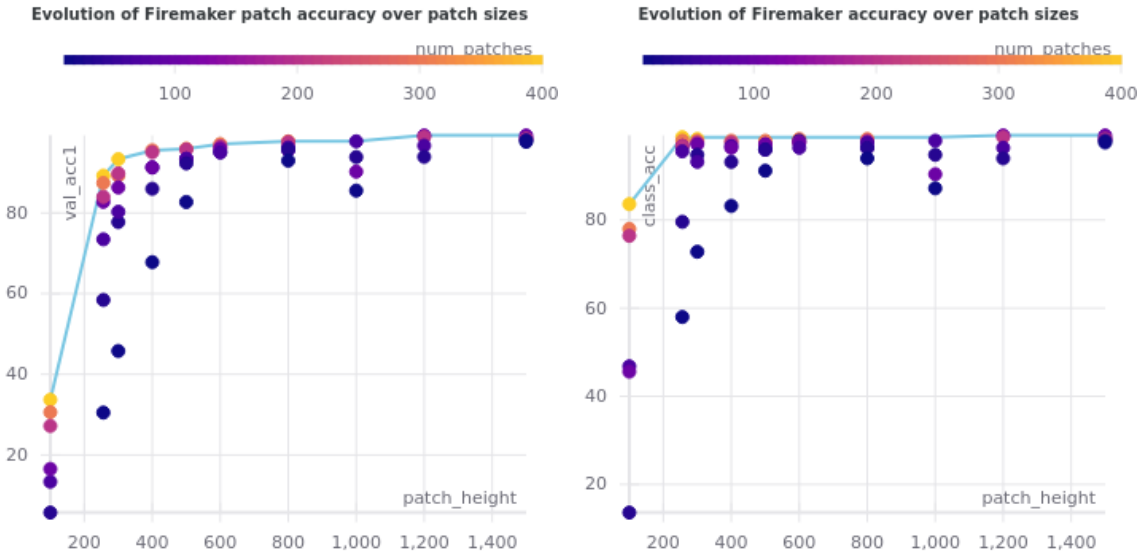
(a) IAM accuracy

(b) IAM patch accuracy

Figure 6.6: Best results with the bigger patch sizes never used in literature with IAM dataset. (a) indicates the accuracy of the individual prediction of each patch. (b) indicates the accuracy of the pages using the voting scheme. The blue line indicates the best results for each patch size.

For the IAM dataset, in the case of patch accuracy, it peaks with size 600 with accuracy 93% and after that, it plateaus or slightly decreases with bigger sizes. In the case of the real voting based accuracy, with a size of 300, it already achieves 93% accuracy. The best result is obtained with size 600 and after that, the results barely vary with only 1% difference.

[1]This refers to a square patch of 300×300. To simplify we state a single value.



(a) Firemaker accuracy

(b) Firemaker patch accuracy

Figure 6.7: Best results with the bigger patch sizes never used in literature with Firemaker dataset. (a) indicates the accuracy of the individual prediction of each patch. (b) indicates the accuracy of the pages using the voting scheme. The blue line indicates the best results for each patch size.

For the Firemaker dataset, both the patch accuracy and the voting accuracy peak at size 600 with an accuracy of 96%, however, it keeps improving with increasing sizes. With the biggest size, we were able to obtain an incredible result of 99.2%.

Overall, with bigger patches, we have obtained excellent results. It seems that having big patches provides a lot of information to the neural network. Combining it with the voting scheme, the model has less margin of error and obtains better results.

6.4.3. Experiment 3: Using full-page images as input

As mentioned before, all the deep learning approaches up to date use small image patches from the original page. Observing that no studies use full pages as input images, we decided to use full pages to see how well can the neural network perform with a full page.

We train our network for 100 epochs with different combinations of batch sizes and learning rates. The best combinations for each dataset are shown in Table 6.4. The accuracy graphs for each dataset can be seen in Figure 6.8.

Dataset	Batch size	Learning rate	Top1 (%)	Top5 (%)	Top10 (%)
IAM	9	0.002	91.34	97.4	98
Firemaker	6	0.003	98.32	99.6	100

Table 6.4: Best hyperparameters found for pages with IAM and Firemaker.

We obtain a top 1 test accuracy of 91.32% in IAM and 98.32% in Firemaker. The results seem to be quite satisfactory. Firemaker, in particular, achieves the state of the art results.

The results indicate that our proposed page model is less effective than all other models. The voting scheme used for the patch models helps significantly it to perform better. The reason is that when testing, the model has a bigger margin of committing errors in patch images, while in the case of pages there is a single image.

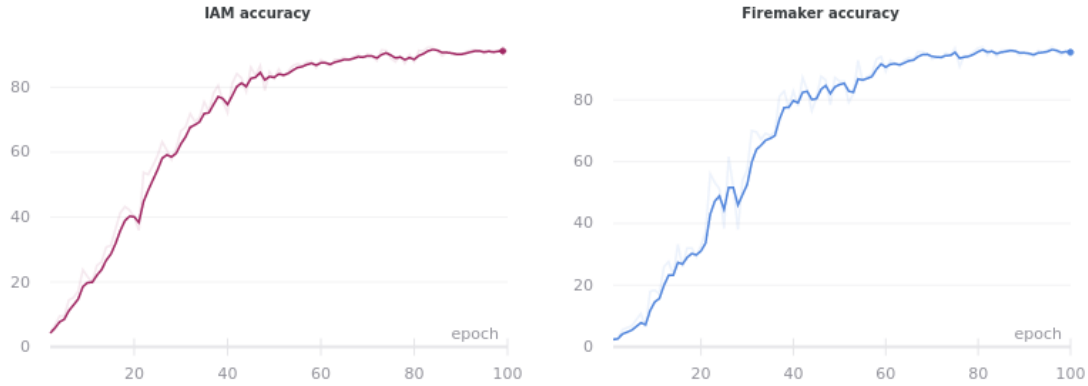


Figure 6.8: Validation accuracy on IAM and Firemaker with the pages model.

6.4.4. Experiment 4: Historical Documents

Previous experiments were done on IAM and Firemaker, which are contemporary datasets. Historical manuscripts have proven to be more challenging for writer identification. These types of documents are quite noisy, suffer deterioration, and are normally written in cursive which is more difficult to understand. In these circumstances, it is more than interesting to know how well can our model cope with these types of datasets. We use the ICDAR 2017 dataset for this purpose. On the one hand, Table 6.5 and Figure 6.9 shows our results with the pages model.

Dataset	Batch size	Learning rate	Top1 (%)	Top5 (%)	Top10 (%)
ICDAR17	9	0.002	75.42	86.8	89.9

Table 6.5: Best hyperparameters found for pages on ICDAR17

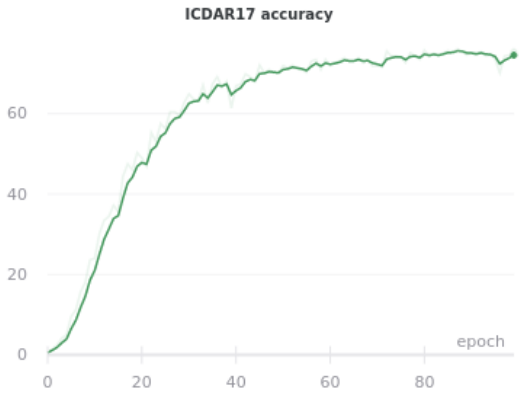


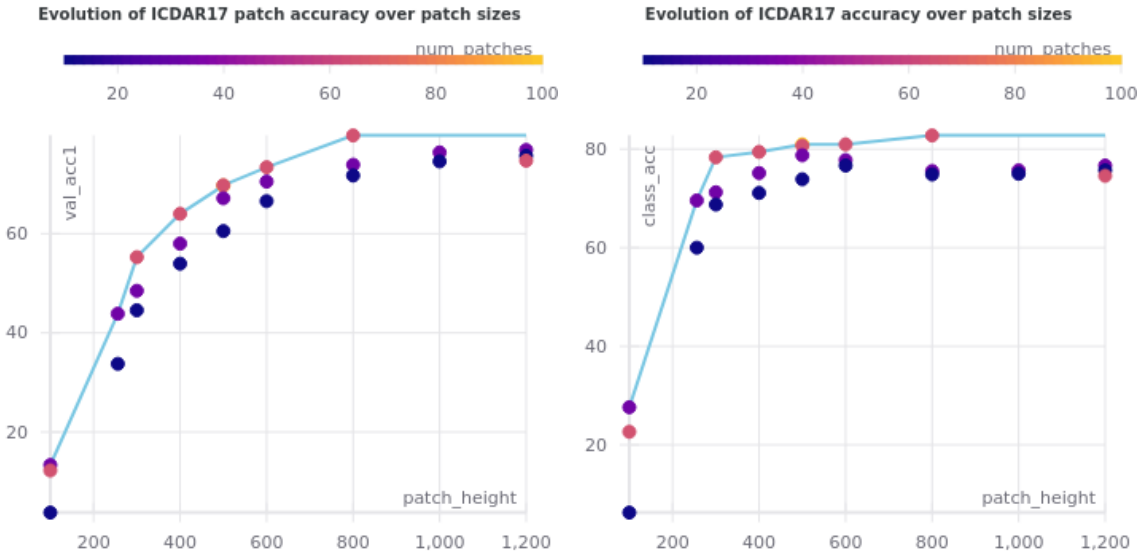
Figure 6.9: Validation accuracy on ICDAR17 with the pages model.

On the other hand, Table 6.6 and Figure 6.10 shows our results with the best performing model with patch images.

Dataset	Num patches	Patch size	Top1 (%)	Patch accuracy (%)
ICDAR17	64	800	83.75	80.1

Table 6.6: Best results found for patches in ICDAR17.

As seen before in the contemporary datasets, the big patches based model performs better. Our best results are obtained with a patch size of 800. Increasing more the patch size does not improve the performance, on the contrary, it slightly decreases. Besides this observation, based on our



(a) ICDAR 2017 accuracy

(b) ICDAR 2017 patch accuracy

Figure 6.10: Best results with the bigger patch sizes never used in literature with ICDAR 2017. (a) indicates the accuracy of the individual prediction of each patch. (b) indicates the accuracy of the pages using the voting scheme. The blue line indicates the best results for each patch size.

results, we can confirm the difficulty that faces writer identification on these types of datasets. In general, the accuracy does not seem to go beyond the 80% mark. A comparison could be done with the results obtained by the team of researchers of the University of Fribourg on the competition of ICDAR 2017 [26] with 47.8% accuracy or with [27] with 88.9% accuracy. But it is not comparable, as our test set is different because their test set was provided for writer retrieval purposes.

6.5 Results

In this section, we compare our results with the state of the art methods on the IAM and Firemaker datasets in Tables 6.7 and 6.8. We include all methods up to date, as far as we have studied, including handcrafted approaches and deep learning approaches. These results may not be comparable because they are assessed for different tasks or carry out an evaluation in a different manner. Still, we can say our methods have obtained a new state of the art results in Firemaker and competent results in IAM. New deep learning approaches are producing incredible results for writer identification, but handcrafted methods still are quite powerful.

Year	Feature	Classifier	Ref.	Writers	Top1 (%)	Top5 (%)	Top10 (%)
2020	CNN with word fragments (FragNet-64)		[49]	657	96.3	–	–
2019	SIFT + RootSIFT	GMM	[41]	650	97.85	–	–
2018	CNN with tuples of images of size 64x64		[48]	650	93.14	–	–
2016	Multi-stream CNN (DeepWriter)		[47]	657	97.3	–	–
2016	p(Is,I), p(IBOS)		[40]	657	86.9	91.6	94.7
2016	Chain code	KDA	[34]	650	82.7	–	92.2
2015	Graphemes	SR-KDA	[39]	657	92	93	97
2014	MSDH + TDH	KNN	[29]	657	97.1	98.8	99.2
2014	SDS + SOH	Euclidean	[28]	657	**98.5**	99.1	99.5
2013	Texture LPQ	SVM	[33]	650	96.7	–	–
2013	Connected	KNN, x2	[38]	650	94.8	–	–
2012	Quill–Hinge	NN	[25]	657	97	–	98
2012	SIFT	x2	[37]	650	93.1	–	–
2011	KAS	SVM	[35]	650	92.1	94.5	95.8
2010	Codebook and contour	KNN	[43]	650	91	–	97
2007	Contour PDFs and ink trace	PDFs	[22]	650	89	–	97
2020	CNN with patches of size 600		Ours	657	96.3	–	–

Table 6.7: Summary of state of the art in writer identification methods in IAM dataset. In bold best results.

Year	Feature	Classifier	Ref.	Writers	Top1 (%)	Top5 (%)	Top10 (%)
2020	CNN with word fragments (FragNet-64)		[49]	250	97.6	–	–
2019	SIFT + RootSIFT	GMM	[41]	250	97.98	–	–
2019	CNN with tuples of images of size 64x64		[48]	250	93.56	–	–
2014	SDS + SOH	Euclidean	[28]	250	92.4	96.2	98.8
2013	Connected	KNN, x2	[38]	250	95.2	–	99.2
2012	Quill–Hinge	NN	[25]	251	86	–	97
2007	Contour PDFs and ink trace	PDFs	[22]	250	83	–	95
2020	CNN with patches of size 1500		Ours	250	**99.2**	–	–

Table 6.8: Summary of state of the art in writer identification methods in Firemaker dataset. In bold best results.

CHAPTER 7
Conclusions and future works

This bachelor thesis proposed developing a writer identification system with deep neural networks. In this chapter, we summarize the main conclusions we can derive from our work as well as proposing future lines of work. We will also explain how we achieved the initial objectives proposed in the project.

Our first objective was to study the state of the art of writer identification approaches. In Chapter 4 we did an exhaustive summary of writer identification methods, which included both traditional methods and deep learning methods. We focused especially on papers that included their results in benchmark datasets, and we list all these results in Table 4.1. We realized during this review that the proposed deep learning approaches have worked only in patch images as input data, instead of using the full pages as input data. Thus, one of our approaches included experimenting with full pages.

After studying and analyzing the state of the art, our second objective was to create an end to end writer identification deep neural network model. In Chapter 5, we developed a model that would be the basis of several distinct approaches. Our different approaches vary depending on the type of data, which can be patch images of different sizes and page images. In this manner, the main contributions we have provided include a novel data augmentation method, a patch-based model that uses a voting scheme, and a page-based model. We call the novel data augmentation technique *text padding*, which instead of performing padding with blank values to an image, it fills the padded area with handwritten text. Our patch-based models compute the most probable writer for each patch and determine the writer for the full page based on the most voted writer over all the patches. Finally, our page based model uses a single image of the full page to determine the writer.

We evaluate all our models and contrast them with known benchmark datasets in Chapter 6. We obtained the state of the art results in the Firemaker dataset and competitive results on the IAM dataset with big patch images. This seems to indicate that, at least in our model, global features over the page can help us to better identify a writer than local features over words or lines. We hypothesize that features like the spacing between lines or the spacing between words may play a role in our improved results. Further studies need to be done to determine the real reasons behind this phenomenon.

Other objectives proposed in our project were also included in Chapter 6. These objectives were: analyzing the impact of different sized image inputs and testing our model in historical manuscripts.

On the one hand, for the first objective, we created some graphs where the performance was measured for each patch size. We observed an increase in performance with bigger patch sizes reaching nearly 100% accuracy.

On the other hand, for the second objective, we evaluated our patch models as well as the page models in the ICDAR 2017 dataset which consists of historical handwritten pages. Our models

consistently achieve over 70% accuracy, our best model reaching over 80% accuracy. These results indicate that our models are competent even for historical data but still proves that these type of data are still more challenging than contemporary datasets.

For future work, several ideas can be explored to better understand the reasons behind the performance and improve model performance.

In the case of patch-based models, if even more performance is needed we can improve our voting system. The proposed voting system takes into account only the absolute number of most voted writers. This can be modified to determine the sum of the probabilities for each writer and make a decision based on these probabilities. Secondly, we could extract even bigger patches because, in the end, our proposed text padding method allows us to create any size of patches. However, there is probably a point of diminishing returns and it requires more computational resources, so it should be explored cautiously.

For the case of historical datasets, the noise in these datasets can be damaging to our models. Some good methods are needed to deal with the noise. There have been works of some neural networks that perform well for cleaning the noise on documents, which could be used to solve this problem. Data transformations, like random brightness reduction or addition, vertical flips, and random Gaussian blur in the training dataset could help the model to be prepared for noisy inputs.

One of the limitations of the use of big patches is the need for large computational resources. We could try to decrease this memory usage in some manner. Preliminary tests on small models like MobileNet v2 obtain good results, which could be used as an alternative for lower memory requirements.

Finally, our models only have been tested on English and Dutch languages. It would be interesting to test our model in other Latin languages like Spanish or Italian, or even in foreign languages like Chinese, Arabic, or Hindi.

CHAPTER 8
Degree relationship

This project could be created thanks to the knowledge obtained in the courses and some other transversal competencies learned in my time at the university.

In particular, Machine Learning has helped enormously in understanding the inner workings of neural networks, allowing me to write Chapter 2 without much difficulty. This corresponds to the "Comprehension and integration" transversal competency.

Moreover, the experience obtained in programming related courses, especially courses where the chosen language was Python, helped me write code fluently without the need to delve into learning the programming language. Another tool I used was LaTeX for writing and organizing the project document. Additionally, I had to use Linux to be able to use the provided GPUs. I did not have any problems because I already worked a couple of types in the Linux environment in many courses. Knowing all these tools corresponds to the "Specific instrumentation" transversal competency.

For the realization of this bachelor thesis, I had to explore a new field, writer identification. Writer identification is a very niche field in which I started with practically no knowledge. After this project, my knowledge in the field has grown considerably and this goes together with the "Continous Learning" transversal competency.

The creation of a good neural network model is not an easy task. Lots of models were created and tried, most of them failed. Nevertheless, in the end, I was able to obtain a model with an incredible state of the art results. "Creativity, innovation and entrepreneurship", "Application and practical thinking" and "Analysis and problem solving" were transversal competencies essential for overcoming the problems that arose during the creation of the neural network model.

Software management tools and concepts were also used, for example, version control with Git and Conda for package management. Finally, my work would have not been completed without my handmade Kanban boards for task management, and my calendar for time management. Personally, I would never work on a project without a plan. I learned this through transversal competencies like "Design and project" and "Planning and time management".

Bibliography

[1] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[2] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[4] Curtis Langlotz, Bibb Allen, Bradley Erickson, Jayashree Kalpathy-Cramer, Keith Bigelow, Tessa Cook, Adam Flanders, Matthew Lungren, David Mendelson, Jeffrey Rudie, Ge Wang, and Krishna Kandarpa. A roadmap for foundational research on artificial intelligence in medical imaging: From the 2018 nih/rsna/acr/the academy workshop. *Radiology*, 291:190613, 04 2019.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[6] Aarti Singh, Robert Nowak, and Jerry Zhu. Unlabeled data: Now it helps, now it doesn't. In *Advances in neural information processing systems*, pages 1513–1520, 2009.

[7] Connectionist ai. 2018. http://www.mysearch.org.uk/website1/html/106.Connectionist.html.

[8] The differences between artificial and biological neural networks. 2018. https://towardsdatascience.com/the-differences-between-artificial-and-biological-neural-networks.

[9] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[10] Kritika Verma and Pradeep Singh. An insight to soft computing based defect prediction techniques in software. *International Journal of Modern Education and Computer Science*, 7:52–58, 09 2015.

[11] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989.

[12] Neural network with two hidden layers. 2018. https://brilliant.org/wiki/feedforward-neural-networks/.

[13] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[14] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 06 2014.

[15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift.

[16] Kernel. 2018. https://en.wikipedia.org/wiki/Kernel_(image_processing).

[17] A comprehensive guide to convolutional neural networks. 2018. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks.

[18] Stack overflow, visualizing pooling. 2017. https://stackoverflow.com/questions/44287965/trying-to-confirm-average-pooling.

[19] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition.

[21] Review: Alexnet, caffenet. 2018. .

[22] Marius Bulacu and Lambert Schomaker. Text-independent writer identification and verification using textural and allographic features. *IEEE transactions on pattern analysis and machine intelligence*, 29(4):701–717, 2007.

[23] X. Zhang, G. Xie, C. Liu, and Y. Bengio. End-to-end online writer identification with recurrent neural network. *IEEE Transactions on Human-Machine Systems*, 47(2):285–292, 2017.

[24] Gloria Jennis Tan, Ghazali Sulong, and Mohd Shafry Mohd Rahim. Writer identification: A comparative study across three world major languages. *Forensic science international*, 279:41–52, 2017.

[25] AA Brink, J Smit, ML Bulacu, and LRB Schomaker. Writer identification using directional ink-trace width measurements. *Pattern Recognition*, 45(1):162–171, 2012.

[26] Stefan Fiel, Florian Kleber, Markus Diem, Vincent Christlein, Georgios Louloudis, Stamatopoulos Nikos, and Basilis Gatos. Icdar2017 competition on historical document writer identification (historical-wi). volume 01, pages 1377–1382. IEEE, 2017.

[27] Vincent Christlein, Martin Gropp, Stefan Fiel, and Andreas Maier. Unsupervised feature learning for writer identification and writer retrieval.

[28] Xiangqian Wu, Youbao Tang, and Wei Bu. Offline text-independent writer identification based on scale invariant feature transform. *IEEE Transactions on Information Forensics and Security*, 9(3):526–536, 2014.

[29] Youbao Tang, Wei Bu, and Xiangqian Wu. Text-independent writer identification using improved structural features. In *Chinese Conference on Biometric Recognition*, pages 404–411. Springer, 2014.

[30] H.E.S. Said, T.N. Tan, and K.D. Baker. Personal identification based on handwriting. *Pattern Recognition*, 33(1):149 – 160, 2000.

[31] Zhenyu He, Xinge You, and Yuan Yan Tang. Writer identification of chinese handwriting documents using hidden markov tree model. *Pattern Recognition*, 41(4):1295 – 1307, 2008.

[32] Behzad Helli and Mohsen Ebrahimi Moghaddam. A text-independent persian writer identification based on feature relation graph (frg). *Pattern Recognition*, 43(6):2199 – 2209, 2010.

[33] Diego Bertolini, Luiz S Oliveira, E Justino, and Robert Sabourin. Texture-based descriptors for writer identification and verification. *Expert Systems with Applications*, 40(6):2069–2080, 2013.

[34] Somaya Al-Maadeed, Abdelaali Hassaine, Ahmed Bouridane, and Muhammad Atif Tahir. Novel geometric features for off-line writer identification. *Pattern Analysis and Applications*, 19(3):699–708, 2016.

[35] R. Jain and D. Doermann. Offline writer identification using k-adjacent segments. In *2011 International Conference on Document Analysis and Recognition*, pages 769–773, 2011.

[36] Önder Kırlı and M Bilginer Gülmezoğlu. Automatic writer identification from text line images. *International Journal on Document Analysis and Recognition (IJDAR)*, 15(2):85–99, 2012.

[37] S. Fiel and R. Sablatnig. Writer retrieval and writer identification using local features. In *2012 10th IAPR International Workshop on Document Analysis Systems*, pages 145–149, 2012.

[38] Golnaz Ghiasi and Reza Safabakhsh. Offline text-independent writer identification using codebook and efficient code extraction methods. *Image and Vision Computing*, 31(5):379–391, 2013.

[39] Emad Khalifa, Somaya Al-Maadeed, Muhammad Atif Tahir, Ahmed Bouridane, and Asif Jamshed. Off-line writer identification using an ensemble of grapheme codebook features. *Pattern Recognition Letters*, 59:18–25, 2015.

[40] Angelika Garz, Marcel Würsch, Andreas Fischer, and Rolf Ingold. Simple and fast geometrical descriptors for writer identification. *Electronic Imaging*, 2016(17):1–12, 2016.

[41] F. A. Khan, F. Khelifi, M. A. Tahir, and A. Bouridane. Dissimilarity gaussian mixture models for efficient offline handwritten text-independent identification using sift and rootsift descriptors. *IEEE Transactions on Information Forensics and Security*, 14(2):289–303, 2019.

[42] Vuurpijl L. Schomaker, L.R.B. Forensic writer identification: A bench-mark data set and a comparison of two systems, technical report. *Nijmegen University, Tilburg, The Netherlands*, 2000.

[43] Imran Siddiqi and Nicole Vincent. Text independent writer recognition using redundant writing patterns with contour-based orientation and curvature features. *Pattern Recognition*, 43(11):3853–3865, 2010.

[44] Stefan Fiel and Robert Sablatnig. Writer identification and retrieval using a convolutional neural network. In *International Conference on Computer Analysis of Images and Patterns*, pages 26–37. Springer, 2015.

[45] Vincent Christlein, David Bernecker, Andreas Maier, and Elli Angelopoulou. Offline writer identification using convolutional neural network activation features. In *German Conference on Pattern Recognition*, pages 540–552. Springer, 2015.

[46] Youbao Tang and Xiangqian Wu. Text-independent writer identification via cnn features and joint bayesian. In *2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, pages 566–571. IEEE, 2016.

[47] Linjie Xing and Yu Qiao. Deepwriter: A multi-stream deep cnn for text-independent writer identification.

[48] Hung Tuan Nguyen, Cuong Tuan Nguyen, Takeya Ino, Bipin Indurkhya, and Masaki Nakagawa. Text-independent writer identification using convolutional neural network. 121:104–112, 2019.

[49] Sheng He and Lambert Schomaker. Fragnet: Writer identification using deep fragment networks.

[50] Urs-Viktor Marti and Horst Bunke. The iam-database: an english sentence database for offline handwriting recognition. *International Journal on Document Analysis and Recognition*, 5:39–46, 2002.

[51] F. Kleber, S. Fiel, M. Diem, and R. Sablatnig. Cvl-database: An off-line database for writer retrieval, writer identification and word spotting. In *2013 12th International Conference on Document Analysis and Recognition*, pages 560–564, 2013.

[52] G. Louloudis, B. Gatos, N. Stamatopoulos, and A. Papandreou. Icdar 2013 competition on writer identification. In *2013 12th International Conference on Document Analysis and Recognition*, pages 1397–1401, 2013.

[53] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[54] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018.

[55] Louis Schomaker, Lambert; Vuurpijl. Firemaker image collection for benchmarking forensic writer identification using image-based pattern recognition. 2000. https://zenodo.org/record/1194612#.XqHAb8szbmE.

[56] Vincent Christlein, Lukas Spranger, Mathias Seuret, Anguelos Nicolaou, Pavel Král, and Andreas Maier. Deep generalized max pooling. *2019 International Conference on Document Analysis and Recognition (ICDAR)*, pages 1090–1096, 2019.

[57] Adam Paszke and Gross et al. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[58] Torchvision models. 2020. https://github.com/pytorch/vision/tree/master/torchvision/models.

[59] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015.

[60] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.