UNIVERSITY OF TARTU
Institute of Computer Science
Computer Science Curriculum

Joonas Lõmps

Affecting Factors on Optical Handwritten Character Recognition Accuracy

Masters's Thesis (30 ECTS)

Supervisor: Amnir Hadachi, PhD

Tartu 2018

Accuracy Affecting Factors for Optical Handwritten Character Recognition

Abstract:
Optical character recognition (OCR) refers to a technique that converts images of typed, handwritten or printed text into machine-encoded text enabling automatic processing paper records such as passports, invoices, medical forms, receipts, etc. Pattern recognition, artificial intelligence and computer vision are all research fields that enable OCR. Using OCR on handwritten text could greatly benefit many of the emerging information systems by ensuring smooth transition from paper format to digital world. Nowadays, OCR has evolved into a multi-step process: segmentation, pre-processing, feature extraction, classification, post-processing and application-specific optimization. This thesis proposes techniques to improve the overall accuracy of the OCR systems by showing the affects of pre-processing, feature extraction and morphological processing. It also compares accuracies of different well-known and commonly used classifiers in the field. Using the proposed techniques an accuracy of over 98% was achieved. Also a dataset of handwritten Japanese Hiragana characters with a considerable variability was collected as a part of this thesis.

Keywords:
Optical character recognition, Computer vision, Machine learning, Feature extraction, Morphological processing

CERCS: P170, P176

Optilist käekirjatuvastust mõjutavad tegurid

Lühikokkuvõte:

Optiline kirjatuvastus viitab tehnikale, mis konverteerib trükitud, kirjutatud või prinditud teksi masinkodeeritud tekstiks, võimaldades sellega paberdokumentide nagu passide, arvete, meditsiiniliste vormide või tsekkide automaatset töötlemist. Mustrituvastus, tehisintellekt ja arvuti nägemine on kõik teadusharud, mis võimaldavad optilist kirjatuvastust. Optilise kirjatuvastuse kasutus võimaldaks paljudel kasvavatel informatsiooni süsteemidel mugavat üleminekut paberformaadilt digitaalsele. Tänapäeval on optilisest kirjatuvastusest väljaskasvanud mitme sammuline protsess: segmenteerimine, andmete eeltöötlus, iseloomulike tunnuste tuletamine, klassifitseerimine, andmete järeltöötlus ja rakenduse spetsiifiline optimiseerimine. See lõputöö pakub välja tehnikaid, millega üleüldiselt tõsta optiliste kirjatuvastus süsteemide täpsust, näidates eel-töötluse, iseloomulike tunnuste tuletamise ja morfoloogilise töötluse mõju. Lisaks võrreldakse erinevate enimkasutatud klassifitseerijate tulemusi. Kasutades selles töös mainitud meetodeid saavutati täpsus üle 98% ja koguti märkimisväärselt suur andmebaas käsitsi kirjutatud jaapani keele hiragana tähestiku tähti.

Võtmesõnad:

Optiline kirjatuvastus, Arvuti nägemine, Masinõpe, Iseloomulike tunnuste tuletamine, Morfoloogiline töötlus

CERCS: P170, P176

Acknowledgment

The author would like to thank the Chair of Distributed Systems of the Institute of Computer Science at the University of Tartu, Ms Eri Miyano and her students from the Department of Languages of Asian Region at the University of Tartu for helping in the data collection campaign of the Hiragana dataset used in this research work and most importantly the guidance of my supervisor Dr. Amnir Hadachi from the Intelligence Transport Systems lab at the University of Tartu.

Contents

1 Introduction

The proof that we have engaged the digitalization era can be seen in our handling of day to day personal and professional activities. The technological advancement has made technology a part of our daily traditions among all fields and domains like industry [BA17], medicine [HSPD10], education [AKHI15], transportation [AAP$^+$08], etc. As a result, researches and also industrial are using their resources to develop technologies capable of preforming document analysis and automated information extraction presented on paper in handwriting and initially addressed to human interpretation.

The widespread use of personal computers, smartphones and tablets has introduced and amplified the need and the use for digital information. Nevertheless, the transition is far from being completed, as we are still using paper format and have old legacy archives practices that need to be processed. Smoothing of this transition and transformation still offers challenges that must be addressed. Hence, a need for an effective way to digitalize large amount of handwritten documents or files.

Optical character recognition can be seen as the art of recognizing written characters with the help of machines. It can be separated into two different modes: online and offline mode. Online mode usually refers to a real-time handwriting recognition using a digitizer with timed sequence of pen coordinates. The two-dimensional coordinates of the special pen on an electronic surface are recorded and analysed. However, offline mode uses digitalized scans of already written or printed texts as an input. Online mode yields good results due to the fact that more information can be captured in the online mode such as the direction, speed and order of strokes, which is not usually the case for offline mode [PS00].

Generally handwriting recognition systems include the following steps: pre-processing, feature extraction and recognition. In some cases segmentation, post-processing and application-specific optimization are added. Segmentation can serve the purpose of segmenting text to characters or characters smaller parts. Post-processing can use the help of language specific semantics to increase the accuracy and application-specific optimization narrows the input to something specific like a form or something similar to that. This thesis focuses on investigating the effects of pre-processing, feature selection and classification on the OCR systems accuracy based on handwritten Japanese Hiragana characters.

The thesis is split into five chapters. The first chapter focuses on prior work and state of art in optical character recognition. It gives an overview of which techniques are generally used, as well as shows in which direction the current research is heading.

The second chapter talks about the methodology adopted in the thesis to conduct the investigation of the effects of pre-processing, feature selection and classification. Also explains each part in depth.

The third chapter introduces our datasets and introduces used parameters used in the classifiers and the overall framework of classification.

The fourth chapter titled "Experimental Results" presents the findings of our investigation and explains the possible reasons for the results.

The fifth chapter called "Conclusion" wraps up our investigation and notes all the work on in this thesis. In this chapter we also point out the techniques that have positive or negative effects on the optical character recognition system. It also notes the possible improvements and ideas that were not yet explored in this thesis but might be worth looking into.

2 Related Work

The field of handwriting recognition research has gained a lot of momentum because of its importance in handling large amounts of data such as addresses written on envelops, forms, surveys, checks and archives of anything else written by hand in need of digitalization. Generally handwriting recognition process follows three main steps: pre-processing, feature extraction, and classification. Sometimes additional steps like segmentation, semantic post-processing or application specific optimizations are added [BRB$^+$09, Alg13].

2.1 Pre-processing

The pre-processing is an important phase in handwriting recognition since it is crucial for good recognition accuracy. The authors of [CMLS15] proposed pre-processing approaches based on filtering and Hough transform for removing the noise which had a significant impact on the recognition rate. Equally, the authors of [HZK07] propose a pre-processing approach for capturing handwriting in online mode, by starting with the removal of duplicate points(points that have the same coordinates), followed by the elimination of hooks using point interpolation and sharp point detection, and topping it off with smoothing and normalization. Their work shows an average of 10% increase for digits, 13.5% for uppercase characters and 16% for lower case characters. Also, there are studies giving clear insight about the impact of pre-processing on the handwritten recognition rate as illustrated in [JH17], where the authors demonstrate the impact of pre-processing phase on the increase of the recognition accuracy. It is clear from the literature that pre-processing plays a huge roll in achieving good results in handwriting recognition.

2.2 Feature extraction

Additionally, recognition accuracy remains dependent also on feature extraction, which can either be structural or statistical. Statistical representation of handwritten characters focus on extracting the distribution of foreground pixels of their image intensities to characterize the characters. The most commonly used approaches are as follows:

- *Histogram projection*: is a way to represent characters' by projecting them into different direction and create a 1-Dimensional or 2-Dimensional vector. It focuses on counting the number of pixels in each row and column in an image to construct a vector [KYS88].

- *Zoning*: is a feature that divides the image containing the character or characters into several zones; then the intensities of pixels in each zone are added and converted into a vector [RR09, SLR$^+$14].
- *Crossings and distances*: is about the numbers of crossings of a contour by a line segment in a specific direction. The character frame is partitioned into a set of regions, following different directions and from each partition a feature is extracted [RS14].

Structural features are founded around the characters topological and geometrical characteristics. They extract global and local properties of the characters using topological and geometrical representations. These features help to discover the main structure components that build up the characters and are also useful for representing characters with high variability and distortions. The most used approaches in the category are as follows:

- *Chain coding*: is the process of mapping the strokes of a character into 2-Dimensional spaces [MB01].
- *Topological structure*: is about extracting and counting features as lines, curves, crossing locations, loops etc. It uses the patterns and strokes in defining the topological aspect of the creation of the character [SR13].
- *Geometrical properties*: is about measuring and estimating geometrical physiognomy of the characters like the ratio between the width and height of the bounding box of the frame, relative distance between the start point of the character and the last one, comparison between the length of two successive strokes, or changes in the curvature [FSTV97, IA13].
- *Graphs*: are related to the aspect of extracting some properties of the characters such as stroke, curve or cross points and using them to build a relational graph. The written character is transformed into a representation using graphs [LFC09].

2.3 Classification

The actual recognition if performed in the classification phase. One of the most commonly used techniques is support vector machine (SVM). SMV is a classifier that used hyperplanes in a high dimensional space to perform the classification. For example in [Saa14] the authors present a pooling SVM approach for handwritten digit recognition achieving good recognition rate at low computational cost.

A combination of neural network and k-nearest neighbour classifiers designed by the authors of [MPV13] managed to achieve an accuracy of 99.3% on highly distorted real-life images from gas- and electricity-meters. They based their training data on the angle of the digits instead of the pixels, so the technique is insensitive to the digit rotation.

Additionally, hidden Markov chain based approach as illustrated in [Sen94] using the extracted skeleton graph, built using edge detection, achieved an average performance of 98% accuracy on a dataset that did not have a high variability of the writing style.

Usage of neural networks (NN) for handwriting recognition is quite popular. There are many applications where NN are used for character recognition. However, in order to reach optimal recognition rates a lot of work needs to be done regarding the feature extraction and training process [GSM11].

2.4 Summary

From the literature it is clear that a lot of work and progress has been done on offline handwriting recognition. However, for online or real-time recognition the challenges are still there to make these algorithms work with the same accuracy and performance.

There are works exploring usages of different techniques to speed up the training process. For example, in [LLW10] the authors propose a learning algorithm based on analysis-by-synthesis paradigm where they used the compact representation and convolute mixture of primitives to represent the handwriting movement primitives. This speeds up the process since the identification of the activation times of the primitives is done while the primitive factorization is occurring.

From these perspectives, this thesis focuses on investigating the affects of the commonly used techniques on optical handwriting character recognition accuracy.

3 Adopted Methodology

This section gives a more precise overview of the methodology used in this thesis. The methodology consists of four steps: pre-processing, morphological processing, feature extraction and classification (fig. 1). Each step contains several sub-steps or variations. Pre-processing contains normalization, grayscaling, smoothing and binarization. This is followed with morphological erosion or dilation and for comparison skipping of this step. In the next phase, five different features are extracted, which are then used to train the four different classifiers.

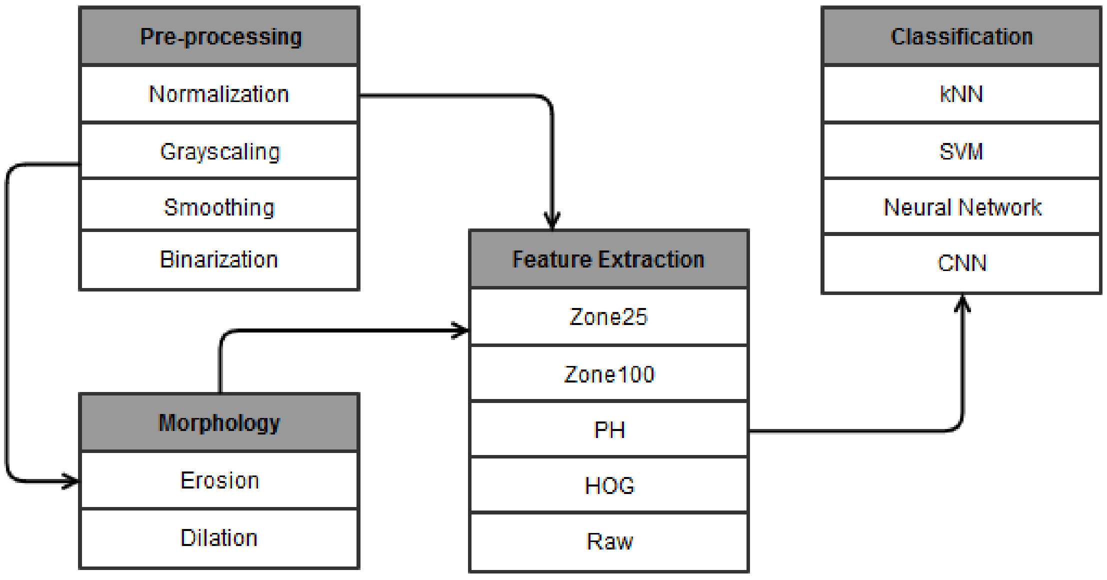


Figure 1. The Adopted Methodology

3.1 Pre-processing

Image pre-processing is an essential part of any computer vision system. It aims to enhance the image features, improve its data by removing any unwilling distortions and making sure that it is ready to be fed into the image recognition system. Commonly used techniques in computer vision include denoising, color enhancement, artifact removal, image stabilization high dynamic range, etc. Due to the fact that we are using still images of handwritten text, our pre-processing consists of four steps. To make characters comparable to one another they need to have same dimensions. For this we chose 50x50 pixels, a total of 2500 pixels per character image. Normalization takes input image of a character, removes the whitespace from all four sides of the character and then resizes it to 50x50 pixels. The normalized image is then converted to grayscale, leaving it with only pixel intensity values and is represented exclusively by different shades of gray. It is then smoothed with a Gaussian blur to reduce the noise. The smoothed image then goes into the last step of our pre-processing, which is binarization. We use the Otsu's method [Ots79], leaving all the foreground pixels white and background pixels black. Step by step progress of pre-processing is visualized on figure 2.

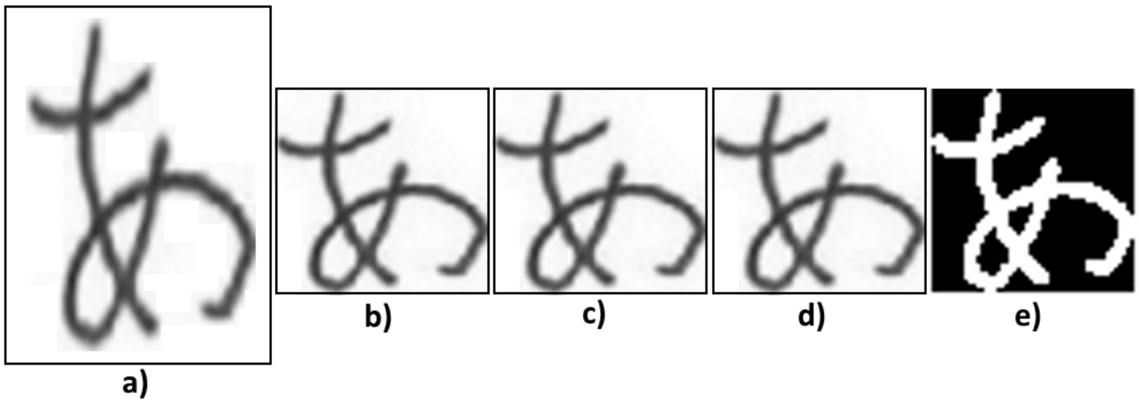


Figure 2. Pre-processing steps: a) input image, b) normalized image, c) grayscaled image, d) smoothed image, e) inverse binarized image

3.2 Morphological processing

Morphology is the study of shakes. Mathematical morphology uses sets to describe shapes. It is used for the analysis and processing of geometrical structures. Most commonly applied to digital images, but also works well on graphs, spatial structures, solids and surface meshes. Mathematical morphology was initially developed to be used on binary images, but was later extended for grayscale images [Ser83, Ste86].

Correct use of mathematical morphology can lead to noise removal, image enhancement, image segmentation or even edge detection, all while preserving the shape of the images and eliminating irrelevancies [HSZ87]. It uses operations defined in set theory to transform input images as sets.

Morphological operations describe the interaction of an input image and a given structuring element. The latter one is usually small compared to the input image. Input images serve as the images to be modified but the structuring element in conjunction with the operation determine the details of the modifications.

This thesis focuses on the two operations that make up all other operations in mathematical morphology: erosion and dilation.

3.2.1 Erosion

Erosion is a morphological transformation that combines two sets using the vector subtraction of set elements (Fig 3). If A and B are sets in the Euclidean space E and A is a binary image, then the erosion of the binary image A by the structuring element B is defined as:

$$A \ominus B = \{x \in E \mid x + b \in A, \forall b \in B\} \tag{1}$$

Example of an erosion operation:

$$\begin{gathered} A = \{(0,0),(0,1),(1,0),(1,1),\\ (1,2),(1,3),(2,1),(2,2),\\ (2,3),(2,4),(3,2),(3,3),(4,5)\}\\ B = \{(0,-1),(-1,0),(0,0),(0,1),(1,0)\}\\ A \ominus B = \{(1,1),(2,2),(3,2)\} \end{gathered}$$

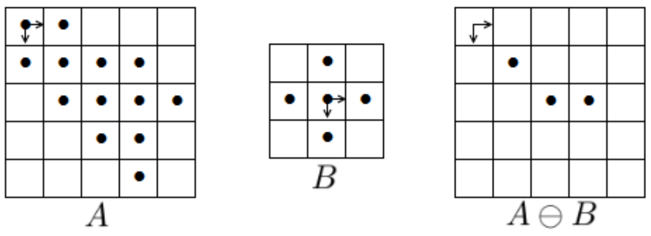


Figure 3. Illustration of the erosion process

This is the definition used for erosion operation by [HSZ87]. A simple way to think about is that the structuring element B is slid across the binary image A and where B is contained in A, its origin point $(0,0)$ is present in $A \ominus B$. Figure 4 shows the input and output images on a Japanese "あ" character.

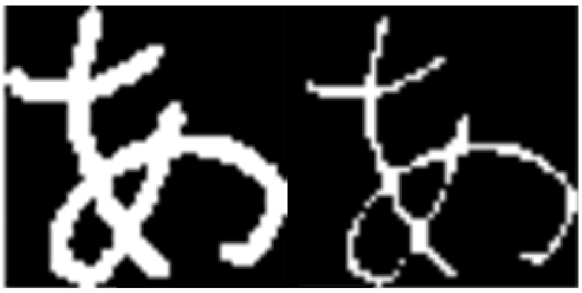


Figure 4. On the left input image for erosion, on the right output of erosion

3.2.2 Dilation

Dilation is a pseudo-inverse of the erosion. Instead of combining two sets using vector subtraction of set elements, it uses their addition (Fig. 5). If A and B are sets in Euclidean space and A is a binary image, then the dilation of the binary image A by the structuring element B is defined as:

$$A \oplus B = \{c \in E \mid c = a + b, \exists a \in A, \exists b \in B\} \tag{2}$$

Example of dilate operation:

$$A = \{(1,1),(1,2),(2,2),(2,3),(3,3)\}$$
$$B = \{(1,-1),(0,0),(0,1)\}$$
$$A \oplus B = \{(2,0),(1,1),(2,1),(3,1),$$
$$(1,2),(2,2),(3,2),(4,2),$$
$$(1,3),(2,3),(3,4),(2,4),(3,4)\}$$

This is the definition used for erosion operation by [HSZ87]. Another way to view it is that the structuring element B can be slid across the binary image A and wherever the origin point of B matches a point on A, the structuring element B can be stamped onto image A, creating $A \oplus B$. An example result of dilation process on an actual character is shown on figure 6.

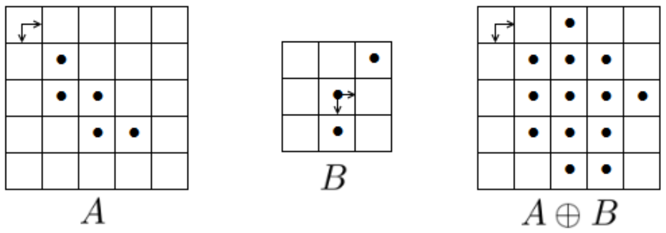


Figure 5. Illustration of dilation process

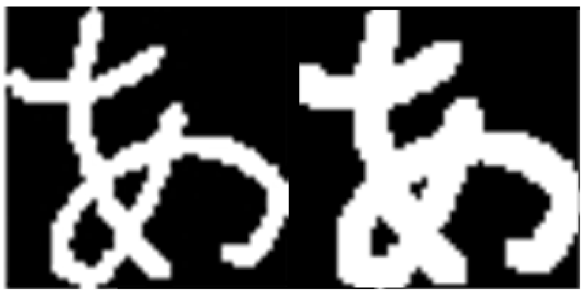


Figure 6. On the left input image for dilation, on the right output of dilation

3.3 Feature Extraction

Image representation plays one of the main roles in a recognition system. To achieve the desired results it is usually suggested not to use a simple binary representation of an image. A more compact characteristic rendition of an image is required to avoid unnecessary complexity and to improve the overall recognition accuracy for many of the recognition systems [AYV01].

Distinguishing between different classes of characters while remaining invariant to characteristic differences within the class can be done by extracting a set of features for every class represented in the data [OLS99]. In the following section, we describe the features this thesis covers.

3.3.1 Raw data

This thesis considers raw data as a separate feature to compare the other features with, as we want to show that the extracted features play an important roll in a recognition system. Here, raw data indicates the output of pre-processing, which in our case is a vector with 2500 values.

3.3.2 Projection Histogram

A projection histogram (PH) can be used to represent a two-dimensional image signal into a one-dimensional signal. This thesis combines vertical and horizontal projection histograms to detect both vertical and horizontal chracteristic features of the characters. How it works with a binary image is that it counts the foreground pixels in each column and row in an image and turns those values into a feature vector with a size of $n + m$, where n and m represent the dimensions of the image. An example of it can be see on figure 7.

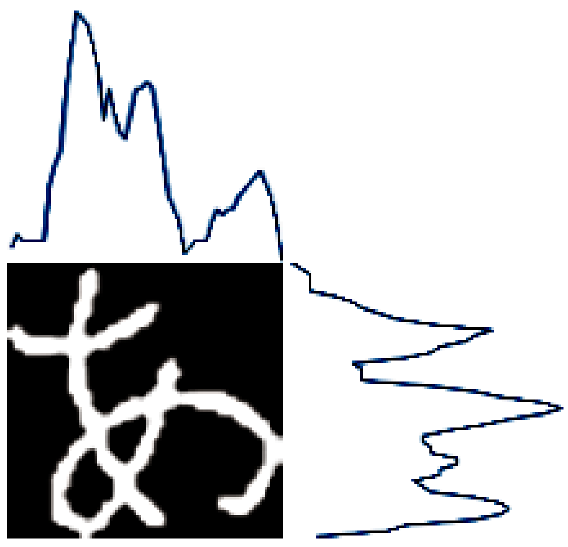


Figure 7. Input image with vertical and horizontal projection histograms

3.3.3 Zoning

Zoning describes a feature that divides the input image into a set of predefined zones, with predefined dimensions and shapes, and adds up the intensities of the foreground pixels in any given zone [SLR$^+$14]. The values are then transformed into a vector with a size of l, where l denotes the number of zones.

If $f(i, j)$ is a digital image and $I(i, j)$ is the pixel intensities of that image, then the summation pixel intensities $I(i, j)$ of an image $f(i, j)$ in the k^{th} zone are calculated by the following equation:

$$V_k = \sum_{i=1}^{n} \sum_{j=1}^{m} I_k(i,j) \tag{3}$$

where $1 \leq k \leq l$, where l is the number of zones, n and m are the dimensions of the zone.

The final vector is a list of intensities of every zone.

$$V = [V_k], 1 \leq k \leq l \tag{4}$$

In our approach, instead of adding up the intensities we just count the foreground pixels in every zone as we have binary input images. We tested with two variations of zoning features, zone25 and zone100 where the number indicates the number of equally sized squares the input image is split into and the results can be seen on figure 8.

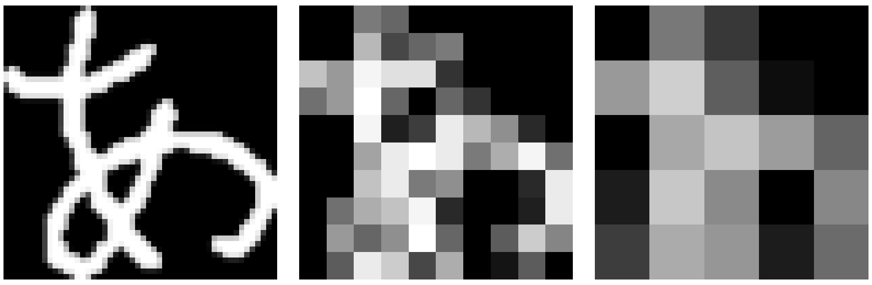


Figure 8. On the left the input image is shown. In the middle, the result of a zone100 feature is shown. On the right there is the output from zone25 feature.

3.3.4 Histogram of Oriented Gradients

The basic idea of histogram of oriented gradients (HOG) is that even without exactly knowing the corresponding gradient or edge position, the distribution of local intensity gradients or edge directions can often quite well characterize the objects appearance and shape [DT05].

This is achieved by dividing the image window into small regions called *cells*. For each of those, a local one-dimensional histogram of gradient directions or edge orientations is accumulated over the pixels in that cell. To increase the accuracy of the descriptor it is useful to normalize the cell values with the values of the local histograms from a slightly larger region containing multiple cells, called *blocks*. The resulting normalized descriptor blocks form the final representation of HOG [DT05] and the visualized version of it can be seen on figure 9.

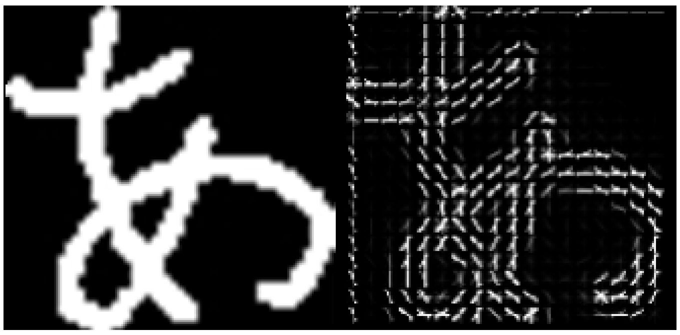


Figure 9. Input image on the left and extracted and visualized HOG feature on the right

3.4 Classification

Classification is the problem of identifying to which category a new observation belongs, based on a set of observations where the category has been previously identified. An algorithm that implements classification is called a classifier, but can also be used for a mathematical function, that assigns a category to the input data.

Classification in machine learning can be divided into two: supervised learning and unsupervised learning. Supervised learning refers to a case where a set of correctly identified observations exists. The latter of the two is known as clustering and groups data into categories based on some similarity or distance. All of the classifiers covered in this thesis use supervised learning techniques.

Supervised learning itself usually consists of training and testing phases. Let there exist a dataset where all the observations have correct classes attached to them and we want to measure the accuracy of our classifier or the whole recognition system together. Usually the dataset will be divided into training and testing sets, most commonly with $80 : 20$ distribution. The training set is used to train the classifier in question and the testing set is used to validate the trained classifier. This is a commonly used technique in the industry and research to measure the effectiveness of a classifier. On another note we want to show that not only the classifier affects the effectiveness of a recognition system.

The following sections introduce classifeirs used in this thesis in more detail.

3.4.1 k-Nearest Neighbours

Since the beginning of the 1970's k-Nearest Neighbours (kNN) has been a widely used technique in statistical estimation and pattern recognition to classify new observations based on a majority vote of its neighbours found in the training set [CH67]. The mentioned neighbours are found based on a similarity measure, most commonly based on Euclidean or absolute distance between the test sample and training samples.

Formally kNN can be defined as follows [HS04]: Let

$$T = \{(c_i, x_i), \quad i = 1, 2, ..., n_T\} \tag{5}$$

be the training set, where $c_i \in 1, 2, ..., y$ signifies the class membership and vector $x_i = (x_{i1}, x_{i2}, ..., x_{ip})$ denotes the feature vector values. For any new observation (c, x) its nearest neighbour $(c_{(1)}, x_{(1)})$ in the training set is determined by

$$d(x, x_{(1)}) = min_i(d(x, x_i)) \tag{6}$$

and $c_{(1)}$, the class of the nearest neighbour, is set as the predicted class c of the new observation (c, x). Where $x_{(j)}$ and $c_{(j)}$ describe the jth nearest neighbour of x and its class.

As previously mentioned the commonly used distance functions is the Euclidean distance

$$d(x_i, x_j) = \sqrt{(x_{i1} - x_{j1})^2 + (x_{i2} - x_{j2})^2 + ... + (x_{ip} - x_{jp})^2} = \sqrt{\sum_{s=1}^{p} (x_{is} - x_{js})^2} \tag{7}$$

or the absolute distance

$$d(x_i, x_j) = |x_{i1} - x_{j1}| + |x_{i2} - x_{j2}| + ... + |x_{ip} - x_{jp}| = \sum_{s=1}^{p} |x_{is} - x_{js}| \tag{8}$$

The parameter k is selected by the used and indicates how many of the closest neighbours the classifier bases its prediction of the new observations class on.

Let k_r be the number of observation in the previously found nearest neighbours, that belong to class r:

$$\sum_{r=1}^{y} k_r = k. \tag{9}$$

The new observation is predicted to be a member of class l with

$$k_l = max_r(k_r) \tag{10}$$

In case of a tie for max occurrences, the overall summation of the distances per class is used as the tiebreaker.

Basic majority voting classification has a drawback that is tied to the class distribution in the dataset. Classes that appear more frequently in the dataset tend to also dominate the prediction of the new observation and due to their large number they tend to commonly appear among the k nearest neighbours.

In addition, selection of the parameter k depends on the data. Larger k helps reduce the effects of the noise in the classification but on the other hand blurs the boundaries between different classes (Fig 10).

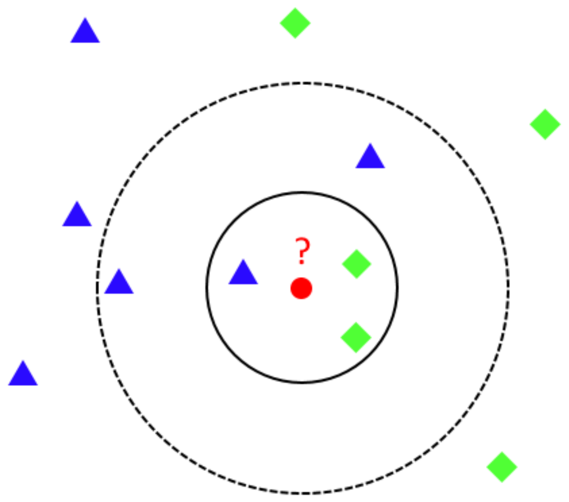


Figure 10. Example of kNN classification. Red circle indicates the test sample to be classified to either the first class of blue triangle or the second class of green diamonds. Middle solid circle indicates k = 3 which classifies the test sample to the second class as there are two samples from the second class and one from the first. If k = 5, or the dashed line, the test sample is classified to the first class, as there are three samples from the first class and two from the second class.

The votes received by the nearest neighbour's algorithm are sometimes weighted based on their distance from the test sample. This reduces the influence of further away samples and can achieve better accuracy.

3.4.2 Support Vector Machine

Support Vector Machine is a supervised machine learning algorithm that uses the optimal separating hyperplane to separate the two classes and maximize the distance to the closest point from either class [Sai96]. This provides a unique solution to the separating hyperplane problem and maximizing the margin between the two classes leads to improved performance of the classification.

Finding the optimal separating hyperplane can be seen as a optimization problem defined by the authors of [FHT01] as follows.

Let the training data consist of N pairs $(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)$, with $x_i \in \mathbb{R}$, $y_i \in -1, 1$ and a hyperplane defined by

$$x : f(x) = x^T\beta + \beta_0 = 0, \tag{11}$$

where β is a unit vector: $||\beta|| = 1$.

$$\begin{gathered}\max_{\beta,\beta_0,||\beta||=1} \quad M \\ \text{subject to } y_i(x_i^T\beta + \beta_0) \geq M,\ i = 1, ..., N\end{gathered} \tag{12}$$

These conditions ensure that the decision boundary defined by β and β_0 is at least a signed distance M from all the points and that we are looking for the largest possible M and associated parameters. The $||\beta|| = 1$ constraint can be get rid of by replacing the conditions with

$$\frac{1}{||\beta||}y_i(x_i^T\beta + \beta_0) \geq M, \qquad \text{or equivalently} \qquad y_i(x_i^T\beta + \beta_0) \geq M||\beta||. \tag{13}$$

Any β and β_0 satisfying these inequalities mean that any positively scaled multiple also satisfies them, we can set $||\beta|| = 1/M$ making (12) equivalent to

$$\begin{gathered}\min_{\beta,\beta_0} \frac{1}{2}||\beta||^2 \\ \text{subject to } y_i(x_i^T\beta + \beta_0) \geq 1,\ i = 1, ..., N.\end{gathered} \tag{14}$$

The constraints define the margin around the linear decision boundary of thickness $1/||\beta||$ and we choose β and β_0 to maximize its thickness turning it into a convex optimization problem or better yet, Lagrange function to be minimized with respect to β and β_0 as

$$L_P = \frac{1}{2}||\beta||^2 - \sum_{i=1}^{N} \alpha_i [y_i(x_i^T \beta + \beta_0) - 1]. \tag{15}$$

Setting the derivatives to zero

$$\beta = \sum_{i=1}^{N} \alpha_i y_i x_i \tag{16}$$

$$0 = \sum_{i=1}^{N} \alpha_i y_i \tag{17}$$

and substituting these in (15) a so-called Wolfe dual is obtained

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{k=1}^{N} \alpha_i \alpha_k y_i y_k x_i^T x_k \qquad \text{subject to} \quad \alpha_i \geq 0. \tag{18}$$

The solution is obtained by maximizing the L_D while satisfying the Karush-Kuhn-Tucker conditions, which include (16), (17), (18) and

$$\alpha_i [y_i(x_i^T \beta + \beta_0) - 1] = \forall i. \tag{19}$$

From which we can see that

- if $\alpha_i > 0$, then x_i is on the boundary of the margin;
- if $\alpha_i = 0$, then x_i is not on the boundary of the margin.

Equation (16) shows that the solution vector β is defined in linear combination of the support points x_i - ones defined to be on the boundary of the margin. Figure 11 a) shows a linear optimal separating hyperplane for an example case where it exists, meaning the two classes are perfectly linearly separable.

In reality, data or classes that can be perfectly separated with a linear optimal separating hyperplane are rare, which is why a generalized version has been derived for a nonseparable case, where the classes can not be separable by a linear boundary.

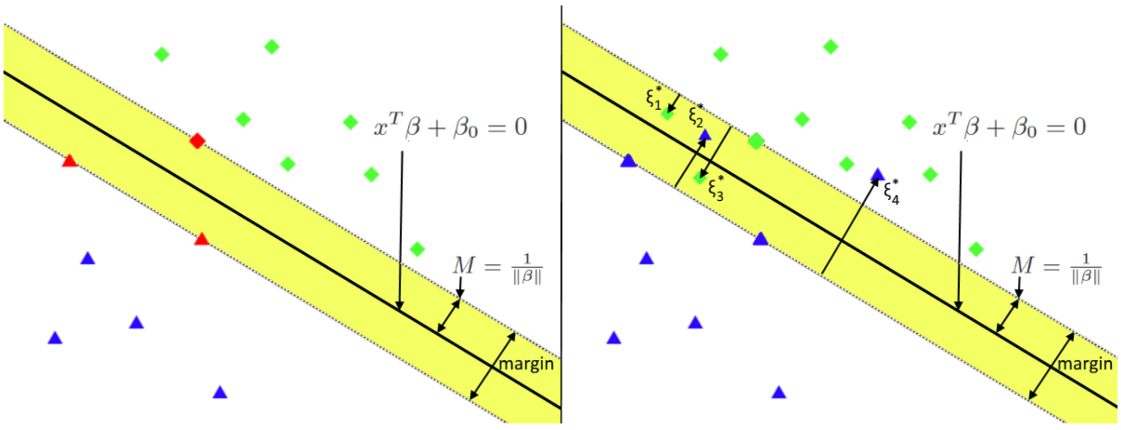


Figure 11. Support vector classifiers. Panel on the left shows a linearly separable case where the decision boundary is the solid line and the yellow area is the maximal margin ($2M = 2/||\beta||$). On the right we have a nonseparable case, where the ξ_j^* are points on the wrong side of their margin by $\xi_j^* = M\xi_j$, while the points on the correct side have $\xi_j^* = 0$.

Continuing the previously used notation, equation (12) can be more conveniently rephrased using the distance $2M$, or the margin as

$$\begin{gathered}\min_{\beta,\beta_0} ||\beta|| \\ \text{subject to } y_i(x_i^T\beta+\beta_0) \geq 1, i = 1, ..., N,\end{gathered} \tag{20}$$

where the $||\beta|| = 1$ constraint has been dropped, as $M = 1/||\beta||$.

A way to deal with the overlap in feature space is to still maximize M while allowing for some points to be on the wrong side of the margin by defining slack variables $\xi = (\xi_1, \xi_2, ..., \xi_N)$. The standard way to modify the constraint in (12) is a bit unnatural, measuring the overlap in relative distance:

$$y_i(x_i^T\beta + \beta_0) \quad \geq \quad M(1 - \xi_i), \tag{21}$$

as this keeps results in a convex optimization problem. As in (14) the norm constraint on β can be dropped and by defining $M = 1/||\beta||$ we can rewrite (12) as

$$\min ||\beta|| \quad \text{subject to} \quad \begin{cases} y_i(x_i^T\beta + \beta_0) \geq 1 - \xi\forall i, \\ \xi \geq 0, \ \sum \xi_i \leq \text{constant}. \end{cases} \tag{22}$$

To find the Support Vector Classifier it is convenient to re-express (22) in its equivalent form

$$\begin{aligned} &\min_{\beta,\beta_0} \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{N} \xi_i \\ &\text{subject to} \quad \xi_i \geq 0, y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i \forall i, \end{aligned} \tag{23}$$

where C, or the "cost" parameter, replaces the constant in (22).

The primal Lagrange function of it is

$$L_P = \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i[y_i(x_i^T\beta + \beta_0) - (1 - \xi_i)] - \sum_{i=1}^{N}\mu_i\xi_i, \tag{24}$$

which is minimized with regard to β, β_0 and ξ_i. Setting the respective derivatives to zero, we get

$$\beta = \sum_{i=1}^{N}\alpha_i y_i x_i, \tag{25}$$

$$0 = \sum_{i=1}^{N}\alpha_i y_i, \tag{26}$$

$$\alpha_i = C - \mu_i, \forall i, \tag{27}$$

in addition to positivity constraints α_i, μ_i, $\xi_i \geq 0 \forall i$. We can obtain the Lagrangian Wolfe dual objective function by substituting (25),(26) and (27) into (24)

$$L_D = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{i'=1}^{N}\alpha_i\alpha_{i'}y_i y_{i'}x_i^T x_{i'}, \tag{28}$$

giving us a lower bound on the function (23) for any feasible point. We maximise L_D subject to $0 \geq \alpha_i \geq C$ and $\sum_{i=1}^{N}\alpha_i y_i = 0$. The Karush-Kuhn-Tucker conditions also include the constraints

$$\alpha_i[y_i(x_i^T\beta + \beta_0) - (1 - \xi_i)] = 0, \tag{29}$$

$$\mu_i\xi_i = 0, \tag{30}$$

$$y_i(x_i^T\beta + \beta_0) - (1 - \xi_i) \geq 0, \tag{31}$$

for $i = 1, ..., N$. Equations (25) - (31) characterize the solution to the primal and dual problem.

Solution for β comes from (25) in the form of

$$\hat{\beta} = \sum_{i=1}^{N} \hat{\alpha}_i y_i x_i, \tag{32}$$

with nonzero coefficients $\hat{\alpha}_i$ only for observations i for which constraints in (31) are exactly met, they are the so called support vectors, since $\hat{\beta}$ is represented in terms of them alone.

So far we have described how to find linear boundaries in the input feature space. The procedure can be made more flexible by enlarging the feature space using basis expansions [FHT01]. Better training-class separation is generally achieved with linear boundaries with the enlarged space, which translates to nonlinear boundaries in the original space. The procedure remains same once the basis functions $h_m(x), m = 1, ..., M$ are selected. The support vector classifier is fitted using input features $h(x_i) = (h_1(x_i), h_2(x_i), ..., h_M(x_i)), i = 1, ..., N$, and produce the function $\hat{f}(x) = h(x)^T\hat{\beta} + \hat{\beta}_0$.

The SVM classifier is an extension of this idea, allowing the enlarged space to scale extremely, infinitely in some cases. It seems that the computation could become prohibitive or overfitting could become an issue, but SVM has techniques to deal with those issues.

The optimization problem (24) and its solution can be represented in a way that only involves the input features via inner products using the trandormed feature vectors $h(x_i)$, as for particular choices of h, the inner products are computationally very cheap.

The Lagrange dual function (28) has the form

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} \langle h(x_i), h(x_{i'}) \rangle. \tag{33}$$

We can see from (25) that $f(x)$ can be written

$$\begin{aligned} f(x) &= h(x)^T \beta + \beta_o \\ &= \sum_{i=1}^{N} \alpha_i y_i \langle h(x), h(x_i) \rangle + \beta_0. \end{aligned} \tag{34}$$

Just like before, a_i, β_0 are determined by solving $y_i f(x_i) = 1$ in (34) for any x_i where $0 < a_i < C$.

Actually, the transformation $h(x)$ does not need specified, but we need to know the kernel function

$$K(x, x') = \langle h(x), h(x') \rangle \tag{35}$$

that computes the inner products in the transformed space. K has to be a symmetric positive definite function, one of the most popular choices for K in the literature is dth-Fegree polynomial: $K(x, x') = (1 + \langle x, x' \rangle)^d$.

This is called the kernel trick and from (34) we see that the solution can be written as

$$\hat{f}(x) = \sum_{i=1}^{N} \hat{\alpha}_i y_i K(x, x_i) + \hat{\beta}_0. \tag{36}$$

With this we know how we can differentiate between two classes using SVMs. Upgrading a SVM to the multi-class case is not too straightforward due to outputs not being on a calibrated scale making comparing them to each other hard. This is why the two different approaches used are **one-versus-all** and **one-versus-one**.

In one-versus-all we train K binary classifiers, $f_K(x)$, where the data from class k is treated as positive case, and all other classes as negative.

The other approach is one-versus-one, where we train $K(K-1)/2$ classifiers to discriminate all pairs $f_{K,K'}$ and the point is classified into the class that has the most votes. [FHT01]

3.4.3 Neural Network

Nowadays, the term Neural Network can be heard almost everywhere, a lot of companies are generating a great deal of *hype* surrounding it and making it seem like an all capable mythical unicorns. In reality, normal Neural Networks are just nonlinear statistical models [FHT01].

A neural network is a classification model, usually represented with a network diagram as in figure 12. Applying this network for K-class classification, there are K units are the right, with the kth unit modeling the probability of class k. The measurements $Y_k, k = 1, ..., K$, each coded as a $0 - 1$ variable for the kth class, total of K classes.

Features Z_m are derived from linear combinations of the inputs X_p, and the target Y_k

is modeled as a function of linear combinations of the Z_m,

$$\begin{aligned} Z_m &= \sigma(\alpha_{0m} + \alpha_m^T X), m = 1, ..., M, \\ T_k &= \beta_{0k} + \beta_k^T Z, k = 1, ..., K, \\ f_k(X) &= g_k(T), k = 1, ..., K, \end{aligned} \tag{37}$$

where $Z = (Z_1, Z_2, ..., Z_M)$, and $T = (T_1, T_2, ..., T_K)$.

The $\sigma(v)$ denotes the activation function, that performs a certain fixed mathematical operation on it. Commonly chosen to be the one of the followings:

- ***Sigmoid*** - its non-linearity has the mathematical form of $\sigma(v) = 1/(1 + e^{-v})$ and is plotted on figure 13. It squashes the real-valued inputs into a range between 0 and 1, where large negative number become 0 and large positive numbers become 1. Sigmoid has seen persistent use in the history due to its nice interpretation as a firing rate of a neuron, but recently in practice it has fallen out of favor and is rarely used. Reasoning behind that are its two major drawbacks, it saturates and kills gradients and the outputs are not zero-centered.

- ***Tanh*** - is simply a scaled sigmoid neuron where the following holds $\tanh(x) = 2\sigma(2x) - 1$. This makes tanh zero-centered, but its activations still saturate. In practice tanh non-linearity is preferred to the sigmoid non-linearity. Tanh squashes the real-valued inputs to the range $[-1, 1]$ and can be seen plotted on figure 14.

- ***ReLU*** - The Rectified Linear Unit has grown in popularity in the last years. Its activation is simply thresholded at zero $f(x) = \max(0, x)$ as shown on figure 14. It is faster compared to *tanh / sigmoid* neurons, but the units can be fragile and die with large gradients causing a neuron to never activate again, but this can be remedied with proper setting of the learning rate.

Sometimes neural networks have an additional $bias$ unit feeding into every unit in the hidden layer and output layers that add an additional constant input feature, which captures the intercepts α_{0m} and β_{0k} in model (37).

The final transformation of the outputs T is done by the output function $g_k(T)$, which in early works in K-class classification was the identity function $g_k(T) = T_k$, but was abandoned in favor of the $softmax$ function

$$g_k(T) = \frac{e^{T_k}}{\sum_{l=1}^{K} e^{T_l}}. \tag{38}$$

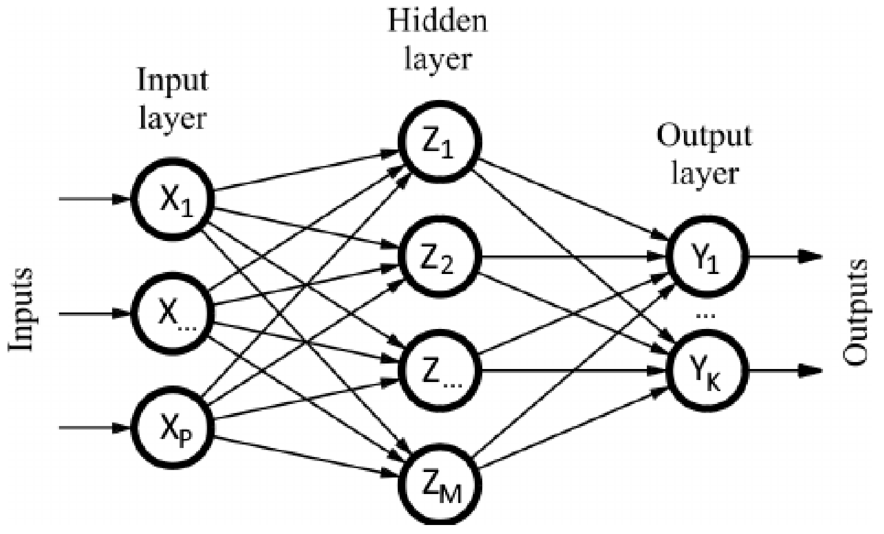


Figure 12. Sample of a single hidden layer, feed-forward neural network.

This produces positive estimates in the range $(0, 1)$ that sum to one.

As shown on figure 12 the network consists of *input, hidden* and *output* layers. Units on the *hidden* layer Z_m, are called *hidden units* because the values Z_m are not directly observed and can be thought as basis expansion of the original inputs X, making the neural network a standard linear model, or linear multilogit model, using transformations as inputs. In most of the neural nets there are more than one *hidden* layer.

Introduction of the nonlinear transformation σ enlarges the class of linear models.

The unknown parameters that make the model fit the training data well, often called weights, are sought after in the neural networks. We denote the complete set of weights by θ, which consists of

$$\begin{aligned} \{\alpha_{0m}, \alpha_m;\ m = 1, 2, ..., M\}\ M(p+1)\text{weights}, \\ \{\beta_{0k}, \beta_k;\ k = 1, 2, ..., K\}\ K(M+1)\text{weights}. \end{aligned} \tag{39}$$

For classification either squared error or cross-entropy (deviance) is used:

$$R(\theta) = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{ik} \log f_k(x_i), \tag{40}$$

and the classifier is $G(x) = \text{argmax}_k f_k(x)$. The softmax activation function and the cross-entropy error function make the neural network model a linear logistic regression model in the hidden units where the parameters are estimated by maximum likelihood.

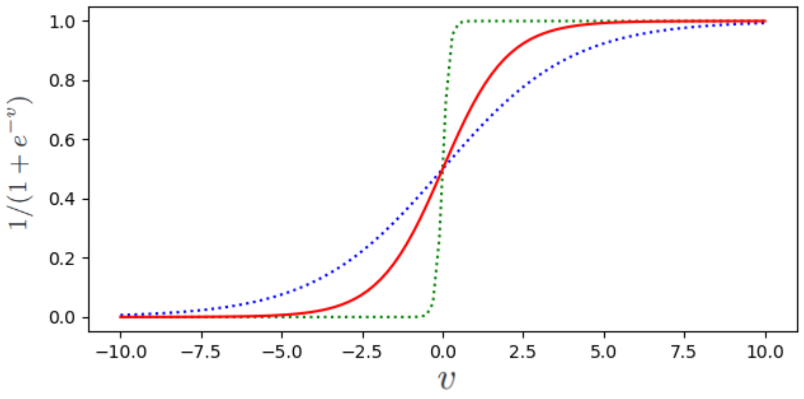


Figure 13. Plot of the sigmoid function $\sigma(v) = 1/(1+e^{-v})$ (red) used in the hidden layers of a neural network. Also included are $\sigma(sv)$ for $s = \frac{1}{2}$ (blue) and $s = 10$ (green), where s is s scale parameter controlling the activation rate.

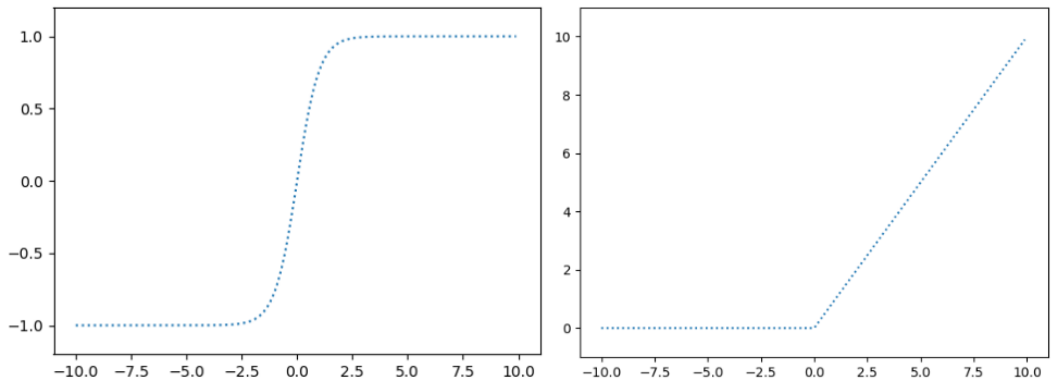


Figure 14. **Left**: The tanh activation function $\tanh(x) = 2\sigma(2x) - 1$. **Right**: Rectified Linear Unit (ReLU) activation function $f(x) = \max(0, x)$

To avoid likely overfitting of the solution global minimizer of $R(\theta)$ should be replaced by minimizing $R(\theta)$ by gradient decent, which in this setting is called *back-propagation*. The compositional form of the model makes the gradient easily derivable by using the chain rule for differentiation.

Let $z_{mi} = \sigma(\alpha_{0m} + \alpha_m^T x_i)$, from (37) and let $z_i = (z_{1i}, z_{2i}, ..., z_{Mi})$, then we have

$$\begin{aligned} R(\theta) &= \sum_{i=1}^{N} R_i \\ &= \sum_{i=1}^{N} \sum_{k=1}^{K} (y_{ik} - f_k(x_i))^2, \end{aligned} \tag{41}$$

with derivatives

$$\begin{aligned} \frac{\partial R_i}{\partial \beta_{km}} &= -2(y_{ik} - f_k(x_i)) g_k'(\beta_k^T z_i) z_{mi} \\ \frac{\partial R_i}{\partial \alpha_{ml}} &= -\sum_{k=1}^{K} 2(y_{ik} - f_k(x_i)) g_k'(\beta_k^T z_i) \beta_{km} \sigma'(\alpha_m^T x_i) x_{il}. \end{aligned} \tag{42}$$

According to these derivatives, a gradient decent update at $(r+1)$th iteration has the form

$$\begin{aligned} \beta_{km}^{(r+1)} &= \beta_{km}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \beta_{km}^{(r)}}, \\ \alpha_{ml}^{(r+1)} &= \alpha_{ml}^{(r)} - \gamma_r \sum_{i=1}^{N} \frac{\partial R_i}{\partial \alpha_{ml}^{(r)}} \end{aligned} \tag{43}$$

where γ_r is the learning rate. Now (42) can be written as

$$\begin{aligned} \frac{\partial R_i}{\partial \beta_{km}} &= \delta_{ki} z_{mi}, \\ \frac{\partial R_i}{\partial \alpha_{ml}} &= s_{mi} x_{il}. \end{aligned} \tag{44}$$

The quantities δ_{ki} and s_{mi} are errors from the current model at the output and hidden layer units that satisfy

$$s_{mi} = \sigma'(\alpha_m^T x_i) \sum_{k=1}^{K} \beta_{km} \delta_{ki}, \tag{45}$$

known as the back-propagation equations. Taking this into account, the updates in (43) can be implemented with a two-pass algorithm. Forward pass fixes the current weights and computes the predicted values $\hat{f}_k(x_i)$ from formula (37). The errors δ_{ki} are computed in the backward pass and then back-propagated via (45) to get errors s_{mi}. Finally, both sets of errors are used to compute the gradients for the updates in (43), via (44). [FHT01]

3.4.4 Convolutional Neural Network

Convolutional Neural Networks, or Convolutional Networks, are neural networks for processing data that has a known grid-line topology. Most commonly used with image data, which can be thought as a two-dimensional grid of pixels or with time-series with sample at regular time intervals as one-dimensional data. The name indicates that the network uses a mathematical operation called convolution, which is a specialized kind of linear operation in at least one of their layer instead of general matrix multiplication.

A convolution is an operation on two functions of a real-valued argument which is easier to follow with example of two functions one might use.

Let's say that we are tracking an object with a sensor. The sensor provides a single output $x(t)$, the position of the object at time t. Both x and t are real valued, meaning, we can get a different reading from the sensor at any time.

Suppose that our sensor is noisy and to obtain a less noisy estimate of the object's position, we would like to average several measurements. We want it to be a weighted average that gives more weight to recent measurements making them more relevant. This can be done by using a weighting function $w(a)$, where a is the age of a measurement. Applying this at every moment, the new function s providing a smoother estimate of the position of the object is reflected as following:

$$s(t) = \int x(a)w(t-a)da. \tag{46}$$

This operation i called convolution and is typically denoted with an asterisk:

$$s(t) = (x * w)(t). \tag{47}$$

In our example, for the output to be a weighted average, w needs to be a valid probability density function and has to equal to 0 for all negative arguments, or it will look into the future. In general, convolution is defined for any functions for which the above integral is defined and may be used for other purposes besides taking weighted averages. [GBC16]

In terminology, x or the first argument to the convolution is commonly referred as the input, and the second argument w as the kernel. Its output can be referred as the feature map.

When working with real data on a computer, the data is more discretized, so it might be more realistic to assume that our sensor provides a measurement once per second making the values of t integers. Assuming that x and w are defined only on integer t, a discrete convolution can be defined:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a). \tag{48}$$

In machine learning applications, the input data is typically a multidimensional array, thus making the kernel a multidimensional array of parameters, adapted by the learning algorithm. These arrays are referred to as tensors. Since every element of the input and kernel must be explicitly stored, it is assumed that these functions are zero everywhere but in the finite set of points for which the values are stored.

Like previously mentioned convolutions are commonly used over more than one axis at a time, it is also important to use a same dimensional kernel. In case of a two-dimensional image I as the input, we also use a two-dimensional kernel K making the convolution equal to:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n), \tag{49}$$

or by using the commutative properties of convolution its equivalent

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n). \tag{50}$$

The commutative property arises from flipping the kernel relative to the input. The kernel is flipped only to obtain the commutative property.

An example of convolution applied to a two-dimensional tensor is shown on figure 15.

Discrete convolution can be viewed as multiplication by matrix, but the matrix has several entries constrained to be equal to other entries. For univariate discrete convolution, each row of a matrix is constrained to be equal to a row above shifter by one element, also known as s Toeplitz matrix. In two dimensions, a doubly block circulant matrix corresponds to convolution. In addition, convolution usually corresponds to a very sparse matrix, the reason being that the kernel is usually much smaller than the input image. [GBC16]

Convolution uses three important ideas that help improve machine learning:

- **Sparse interaction** - In traditional neural network every output unit interacts with every input unit due to the use of matrix multiplication between different

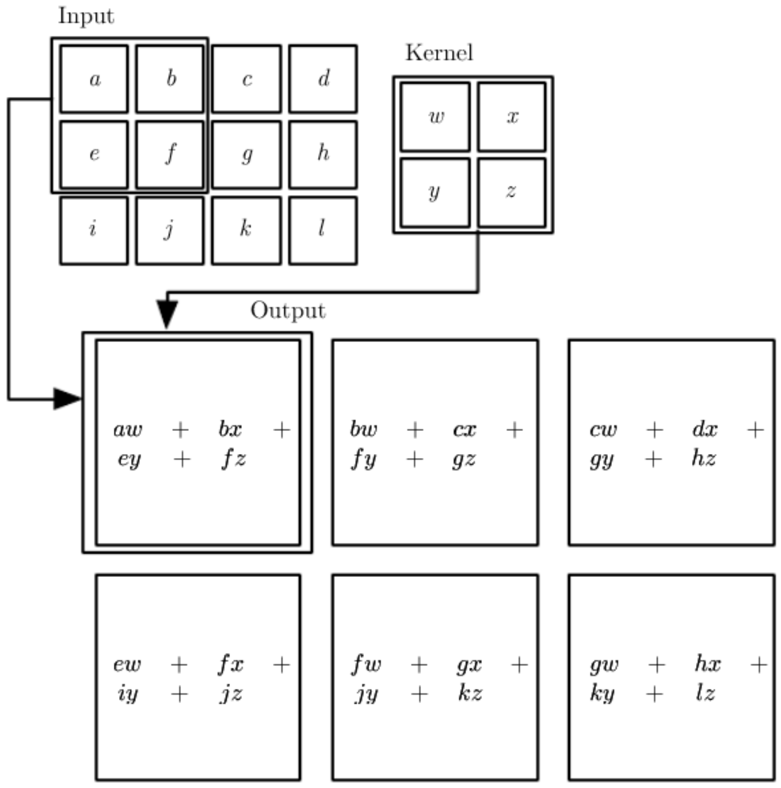


Figure 15. An example of two-dimensional convolution without kernel flipping. Output is restricted to positions where the kernel lies entirely within the image. Boxes with arrows indicate how the upper-left element of the output tensor is formed by applying the kernel to the upper-left region of the input tensor. [GBC16]

layers. Convolutional networks typically have sparse interactions, also referred to as sparse connectivity or sparse weights, which is achieved by making the kernel smaller than the input. Instead of using all the whole input we detect small, meaningful features with kernel with a fraction of the input. This leads to storing fewer parameter, which improves the models statistical efficiency and reduces the memory requirements. Also, computing the output requires fewer operations. These improvements in efficiency are usually quite significant. With m inputs and n outputs, the matrix multiplication requires $m \times n$ parameters, and needs $O(x \times n)$ runtime. By limiting the number of connections each output to k, then the sparsely connected approach requires only $k \times n$ parameters and $O(k \times n)$ runtime. In practice, it is possible to achieve good result and good performance while keeping k multiple orders of magnitude smaller than m. A graphical demonstration of sparse connectivity can be seen on figures 16 and 17. Additionally, in deep convolutional network, units in the deeper layers indirectly interact with a larger portion of the input, as shown in figure 18.

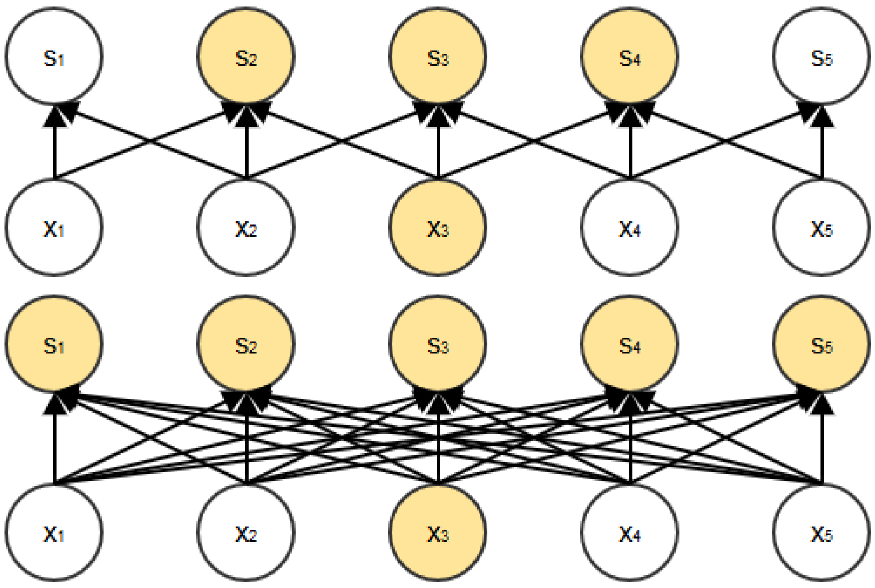


Figure 16. Sparse connectivity highlighting one input unit and the units it affects. On the top we see a s formed by convolution with a kernel of width 3, showing that only three outputs are affected by x_3. On the bottom we see a case where s is formed by matrix multiplication, showing a non-sparse connectivity, where all outputs are affected by x_3.

- **Parameter sharing** - Using the same parameter for multiple functions in a model. Traditional networks use each element in the weight matrix once for computing the output layer. However, in a convolutional neural net, the whole kernel is used at every position of the input, making it possible to learn just one set or parameters instead of a set of parameters for every location. This reduces the storage requirements of the model to k parameters, and k is usually several orders of magnitude smaller than m making the convolution significantly more efficient than dense matrix multiplication.

- **Equivariant representation** - Usage of convolution causes the layer to have a

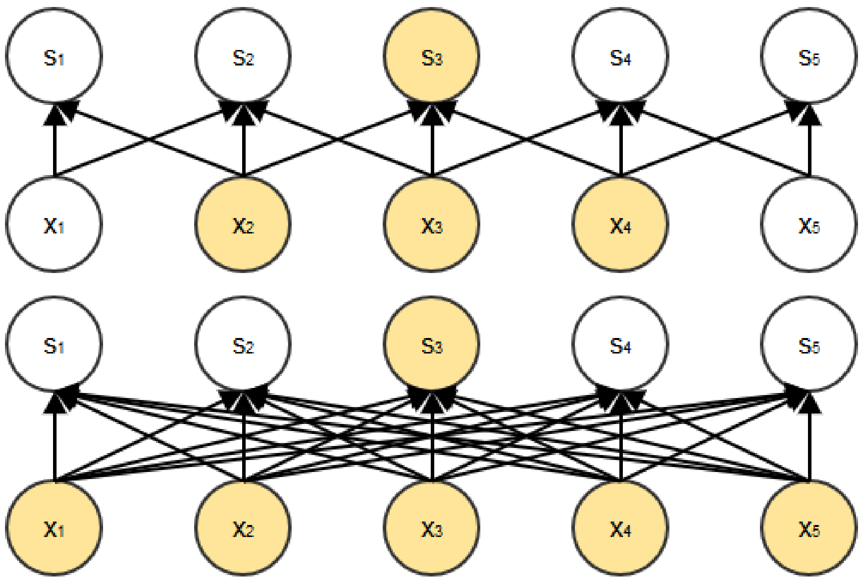


Figure 17. Sparse connectivity highlighting one output unit and the units it is affected by, also known as the receptive field. On the top we see a s formed by convolution with a kernel of width 3, showing that only three inputs affect s_3. On the bottom we see a case where s is formed by matrix multiplication, showing a non-sparse connectivity, resulting in all inputs affecting the output unit s_3.

property called equivariance to translation. This means that the functions changes to input change the output in the same way. In mathematical terms, a function $f(x)$ is equivariant to function g if $f(g(x)) = g(f(x))$. A commonly used example is shifting data: let I be a function returning image brightness at certain coordinates and g be a function mapping one image function to another, such that $I' = g(I)$ is the image function with $I'(x, y) = I(x - 1, y)$, meaning it shifts ever pixel of I one unit to the right. Applying transformation to I, then applying convolution results with the same output as if we applied convolution to I' and then applied the transformation g to the output. Convolutions are not naturally equivariant to all transformations, such as changes in rotation or scale of an image.

A commonly used layer after convolutional layer in a convolutional neural networks is a layer that that uses a pooling function to further modify the output. A pooling function replaces the output of the previous layer at a certain location with a summary of statistic of the nearby layers. There are a few commonly used pooling functions are

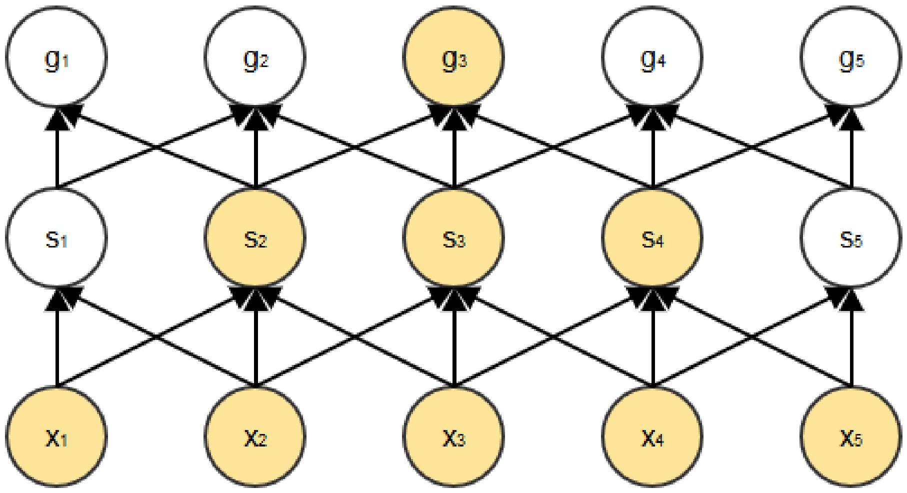


Figure 18. Units on deeper layers of a convolutional network have a larger receptive field of the units in the shallow layers. Meaning, units in the deeper layers can be indirectly connected to all of most of the input image.

- **max pooling** - outputs the maximum output within a predefined $m \times n$ neighbourhood,
- **average pooling** - outputs the average of a neighbourhood,
- **weighted average** - based on the distance from the central pixel.

The added value of the pooling layers reveals itself in making the representation approximately invariant to small translations of the input, meaning that if the input is translated by a small amount, the values for most of the pooled outputs will not change, as demonstrated on figure 19.

The use of pooling layers can greatly improve the statistical efficiency of the network, as well as, improve it computationally as the pooling region can be space k units apart making the following layer have roughly k times fewer inputs to process.

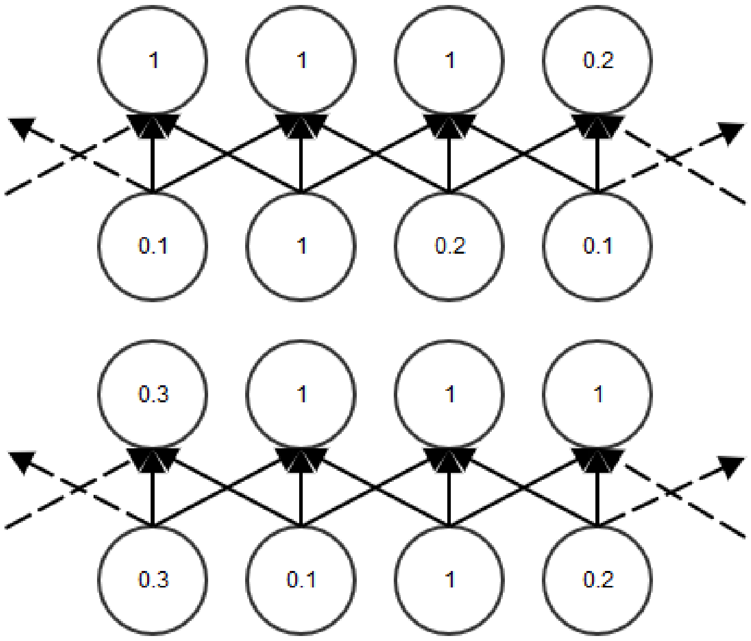


Figure 19. Max pooling example. On the top we see an initial input and output of a max pooling layer. On the bottom we see the same network, but the input has been shifted to the right by one. In the input all the values have changed, but only half of the values in the output are affected, displaying that max pooling units are sensitive to only the maximum value in the neighborhood, not its exact location.

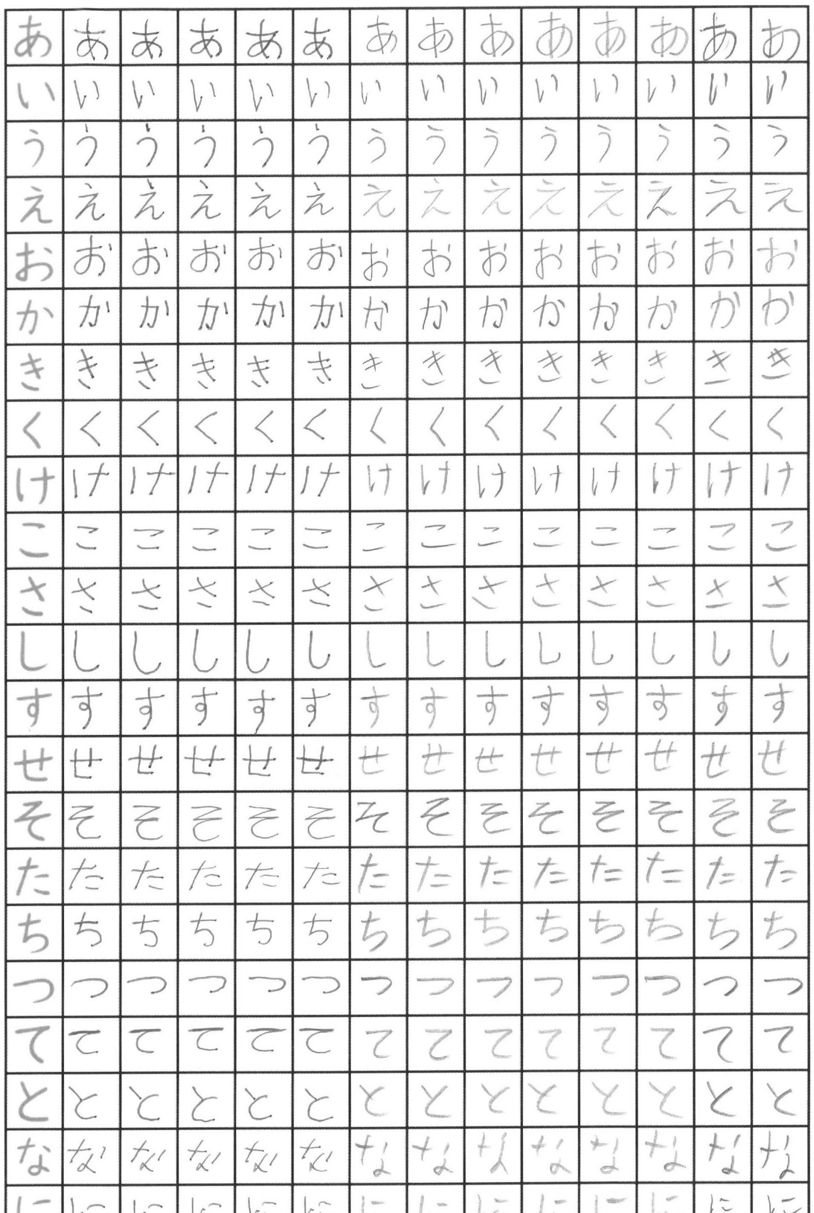
4 Data and Classification Framework

This section introduces the two different datasets used in this thesis that were used to conduct the experiments on. Also, it goes over the parameters used in the classifiers and the neural network architectures used.

4.1 Datasets

The first dataset of handwritten Hiragana characters used in this thesis was collected by the author with the help of Tartu University's Department of Languages of Asian Region and its students.

Technique used to collect the dataset were template forms (fig. 20) that were handed out to students that were studying Japanese at that time. The students were asked to fill the forms.



あ	あ	あ	あ	あ	あ	あ	あ	あ	あ	あ	あ	あ	あ
い	い	い	い	い	い	い	い	い	い	い	い	い	い
う	う	う	う	う	う	う	う	う	う	う	う	う	う
え	え	え	え	え	え	え	え	え	え	え	え	え	え
お	お	お	お	お	お	お	お	お	お	お	お	お	お
か	か	か	か	か	か	か	か	か	か	か	か	か	か
き	き	き	き	き	き	き	き	き	き	き	き	き	き
く	く	く	く	く	く	く	く	く	く	く	く	く	く
け	け	け	け	け	け	け	け	け	け	け	け	け	け
こ	こ	こ	こ	こ	こ	こ	こ	こ	こ	こ	こ	こ	こ
さ	さ	さ	さ	さ	さ	さ	さ	さ	さ	さ	さ	さ	さ
し	し	し	し	し	し	し	し	し	し	し	し	し	し
す	す	す	す	す	す	す	す	す	す	す	す	す	す
せ	せ	せ	せ	せ	せ	せ	せ	せ	せ	せ	せ	せ	せ
そ	そ	そ	そ	そ	そ	そ	そ	そ	そ	そ	そ	そ	そ
た	た	た	た	た	た	た	た	た	た	た	た	た	た
ち	ち	ち	ち	ち	ち	ち	ち	ち	ち	ち	ち	ち	ち
つ	つ	つ	つ	つ	つ	つ	つ	つ	つ	つ	つ	つ	つ
て	て	て	て	て	て	て	て	て	て	て	て	て	て
と	と	と	と	と	と	と	と	と	と	と	と	と	と
な	な	な	な	な	な	な	な	な	な	な	な	な	な
に	に	に	に	に	に	に	に	に	に	に	に	に	に

N / Jaapani keel II

Figure 20. Example of a filled template.

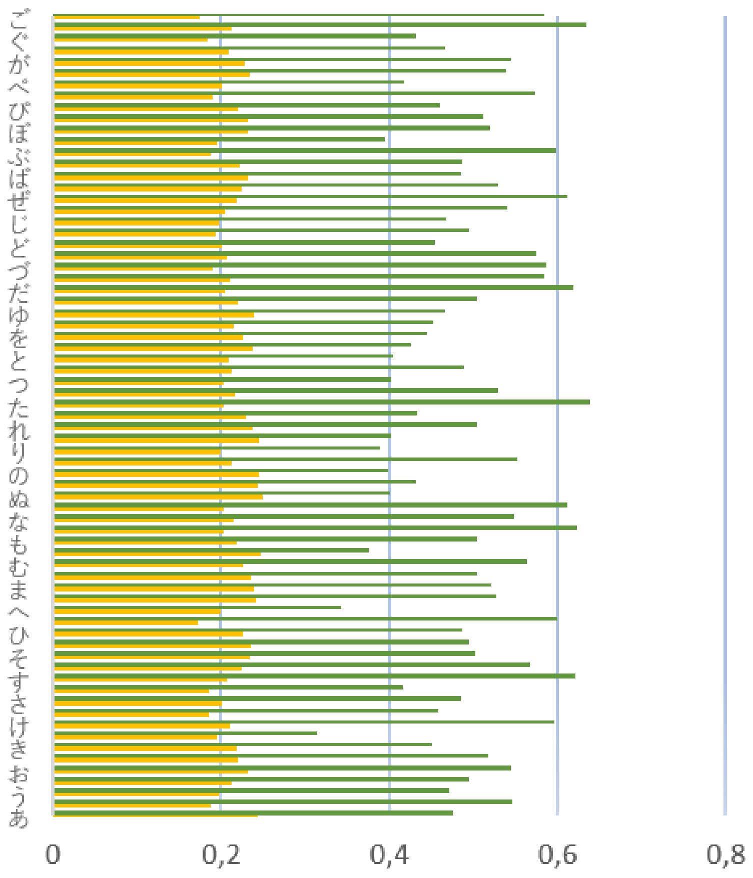


Figure 21. Dataset variability per character. Green represents characters variability before pre-processing and yellow after pre-processing.

The dataset consists of 12211 unique representations of Hiragana characters, approximately 172 unique samples for each of the 71 different Hiragana characters in 28 different handwriting from students with varying experience with the language. This gives our dataset a significant average variability about 49% before pre-processing and 22% after. Per character variabilities are shown on figure 21.

Additionally, a dataset from Electrotechnical Laboratory (ETL) Character Database [ETL16] is tested with, which also includes handwritten Hiragana characters in ETL-8 dataset. This dataset contains handwriting samples from 160 different writers totaling in 12000 unique samples. The provided images are pre-segmented, centered gray-scale characters that are 64 x 64 in size.

Both of the datasets include sample with diacritics, known as *dakuten* and *handakuten*, which are shown on figure 22.

The latter dataset is used to compare the best resulting combination of techniques from the methodology with the best result from the article [Tsa16].

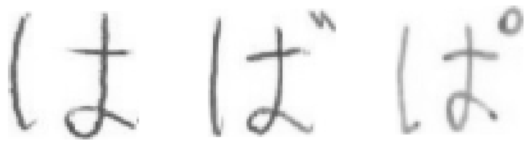


Figure 22. Samples of diacritics in Hiragana. On the left we have a regular "ha", in the middle we have *dakuten* version of it, "ba" and on the right we have *handakuten* version "pa".

4.2 Classification framework

For all combinations of different methods, the metric used to compare them to each other was the classification accuracy, which is the percentage of correctly classified example from the testing dataset. The used dataset was split 80:20, for training and testing respectively.

Experiments with kNN classifier showed that the most optimal $k = 3$ and $k = 5$. Also, distance wise weighted versions of it were also explored.

With SVM two different kernels were used: 3rd degree polynomial and linear. Both of them with penalty parameter C at 2.67.

Classic neural networks were used in several different hidden layer configurations. All layers used ReLU as the activation function and Adaptive Moment Estimation (Adam), solver [KB14] for weight optimization. The different layer setups are listed in table 1. Training was carried out for 60 epochs.

Convolutional neural networks were also used in a few different network configurations. Similarly to classic neural network testing also the ReLU activation function was used and weight optimization Adam with the default parameters as provided in the paper [KB14] $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All trainings were carried out over 40 epochs.

Convolutional layers use a stride of 3. Max-pooling layers are calculated over a 2×2 pixel window with a stride of 2.

Table 1. Explored classic neural network configurations. Input layer size depends on the feature vector, HL refers to hidden layer and the numbers in the table indicate the number of units in that layer. All output layers have 71 units, which is the number of possible classes.

	Input	**HL 1**	**HL 2**	**HL 3**	**HL 4**	**Output**
mlp_1	*	50	50	-	-	71
mlp_2	*	100	80	-	-	71
mlp_3	*	200	100	-	-	71
mlp_4	*	50	30	20	-	71
mlp_5	*	100	80	80	-	71
mlp_6	*	50	30	20	20	71

For every network the last output layer is a fully-connected layer with softmax classifier and 71 units, which is the number of possible classes. Used network configurations are shown table 2.

4.3 Implementation details

K-Nearest neighbour, support vector machine and classic neural network models used in this thesis were implemented in scikit-learn [PVG$^+$11], a python machine learning library. The convolutional neural networks were implemented in Theano [BBB$^+$10] using the Keras interface [C$^+$15].

5 Experimental Results

As previously mentioned, the primary metric for all tasks was the classification accuracy. Summarized results of all can be seen on in Table 3. Generally, more parameters in classifier lead to better classification accuracies on the testing set. First we will go over the results classifier wise, then feature wise and lastly try to summarize the patterns.

Table 2. Convolutional neural network configurations. In convolutional layers the number indicates the height and the width of the convolution window.

cnn_1	cnn_2	cnn_3	cnn_4	cnn_5	cnn_6	cnn_7
input						
conv-5	conv-10	conv-5	conv-10	conv-10	conv-10	conv-10
conv-5		conv-10	conv-10	conv-20	conv-10	conv-20
maxpool						
conv-10	conv-20	conv-20	conv-20	conv-40	conv-20	conv-40
conv-10		conv-40	conv-20	conv-80	conv-20	
					conv-40	
maxpool						
	conv-40	conv-80	conv-40	conv-160	conv-160	conv-160
			conv-40			conv-160
	maxpool					
FC-71-softmax						

5.1 Classifier wise

Looking at the results of k-nearest neighbour variations it can be seen that with both $k = 3$ and $k = 5$ the results do not differ but a significant margin. On the other hand, the weighted variants with the same k tend to achieve uniformly higher accuracies with any of the tested cases.

Support vector machines seem to prefer zoning and raw features. The two different kernels show us how a linear kernel can not always find the best separating hyperplane, as it is out preformed by its polynomial counterpart in almost all of the tested cases.

The classical neural networks follow the pattern that increasing the number of units in layers and making the network deeper by adding additional hidden layers the classification accuracy increase. It can be noticed that the results are better if none of the hidden layers have lesser units than the input or the output layers. This is very clearly demonstrated with the disappointing accuracies with raw feature.

Accuracies achieved by the convolutional neural networks did not disappoint. This is where the highest classification accuracy was achieved. Model cnn_3 in combination with raw pre-processed data reached 98.1247% accuracy with the testing set. Identically to classical neural networks, deeper and wider configurations of convolutional neural networks achieve higher accuracy. But all of the tested models seem to perform rather well with zone100 and raw features.

5.2 Feature wise

Feature wise we can say that the PH and HOG are the features that underperformed compared to other features while reaching 85.65% with pre-processed data with cnn_5 model and 87.08% with dilated data on mlp_4 model respectively.

To the authors surprise the smallest feature, zone25, with only 25 values achieved 90.90% accuracy with polynomial support vector machines. This result comes at 100 times reduction of the initial data.

Previously mentioned features larger version, zone100, is the most uniformly and consistently performing feature. It does see a small drop in the average accuracy with neural networks with smaller layers, but overall it is clearly a feature worth investing time in due to its simplicity, and its characteristic of keeping the initial form of the character intact with 25 times less features than initially. Also, table 3 shows that exactly this feature has the most highest accuracies per classifier.

Next we will go over the results for the raw data. As already mentioned, the highest accuracy recorded in this thesis was achieved with raw pre-processed data and cnn_7 model. Most of the convolutional models performed well with raw data, but the same can be said for both of the support vector machine models and even the k-nearest neighbour. Classical neural networks had difficulties with raw features and the author believes that it can be accounted to the too small sizes of the hidden layers.

5.3 Copmarison and Errors

Comparing the best performing combination of the pre-processing, feature extraction and classifier, pre-processed without morphological processing raw data on cnn_7 model, on the hiragana character dataset used in [Tsa16] achieved a result of 97.6% versus their 96.5%.

In all of the tested cases the most problematic characters were the diacritics *dakuten* and *handakuten* as shown in the previous section. The reason being, that about 90% of the character is same for all of those cases, which leaves only 10% of the data to differ between the three different but very similar characters.

5.4 Summary

To summarize the results, it can be said that pre-processing of the input data could be one of the most important parts of a optical character recognition system. The results clearly show that in almost all of the cases, pre-processing the input data significantly increases the accuracy of the whole system. Also, it is important to bare in mind that the parameters of the classifier need to be tuned according to the input vector size, as the models mlp_1-mlp_6 demonstrated. Additionally, morphological processing can increase the classifiers performance.

Table 3. Classification accuracies for all tested combinations of neural networks. First row indicates the used feature. Second row indicates from which data the feature was extracted, normal meaning features were extracted from just a gray-scaled image, eroded and dilated indicate that preprocessing (prep.) was followed by either one. The highest accuracies per classifier are highlighted in bold. ! shows that can not be ran with this configuration.

	zone25				zone100				Projection Histogram				Histogram of Oriented Gradients				raw			
Classifier	normal	eroded	prep.	dilated	normal	eroded	prep.	dilated	normal	eroded	prep.	dilated	normal	eroded	prep.	dilated	normal	eroded	prep.	dilated
kNN3	69.6112	88.4036	88.9206	88.9950	72.2953	88.0521	89.3548	**90.6948**	54.2556	73.6435	76.8776	79.4913	62.6220	63.9495	69.0323	77.4483	75.9388	75.9388	83.8131	88.8255
kNN5	70.5790	87.8536	88.5773	88.8830	72.8246	87.1257	88.7758	**90.2440**	55.3350	73.6931	77.1629	78.8999	62.9611	64.5037	69.5285	76.7122	75.4963	76.3524	83.0397	88.2713
kNN3w	71.0835	88.9330	89.2142	89.5658	73.4781	88.5070	90.2564	**91.4516**	56.9892	76.2242	79.0943	80.8106	64.3921	66.7122	71.4185	78.5277	78.2548	78.5567	85.1861	89.8346
kNN5w	71.7328	88.3623	89.6650	89.0695	74.3093	88.2010	89.7146	**90.8478**	57.9901	75.3226	78.7717	80.6452	64.5409	66.5881	71.0422	78.7304	76.8900	78.1514	84.2887	88.9454
SVM_lin	72.7047	86.1456	87.0141	87.9239	73.4078	89.4955	89.9090	89.3714	50.2068	73.3251	71.8776	73.9868	50.7858	15.3019	14.8056	21.5467	88.1307	88.3788	90.7497	**90.8985**
SVM_poly3	74.7312	88.9785	90.5997	90.9031	77.5682	92.1836	92.8619	**93.1032**	57.1340	80.7940	82.0885	82.2291	22.3739	2.8122	3.3085	6.9892	77.6607	85.0666	90.6978	92.6913
mlp_1	67.4814	83.4285	83.3499	83.4078	70.0207	85.3763	**85.9181**	85.2399	52.4648	74.8180	76.2779	74.0405	78.8007	82.4814	84.1315	83.7179	33.0976	11.5012	4.7312	2.8784
mlp_2	70.1902	85.0124	86.1456	84.8015	72.0017	87.2374	**87.7130**	87.2498	53.9868	76.4847	77.7378	76.1456	79.5782	83.7924	85.0910	85.9719	63.3292	49.4334	40.8387	44.3788
mlp_3	70.6907	87.0761	87.1092	86.0959	73.9744	88.5070	**88.8710**	88.7593	54.6278	78.4119	79.3714	78.6890	80.6948	84.6071	86.3358	87.0802	75.6948	69.9173	71.7783	69.3052
mlp_4	69.4955	85.3019	85.6452	84.8966	71.9818	87.1092	**87.5021**	87.2002	54.2060	77.0182	77.6220	76.5385	77.1257	81.3110	82.0265	82.8784	72.0389	65.2357	63.4119	59.6857
mlp_5	69.6526	85.4218	85.5170	85.0207	72.5517	87.2043	87.4855	**87.6261**	54.3797	76.9065	77.8536	76.9479	78.2010	81.2903	82.6592	83.1266	74.3755	68.4864	71.4557	67.0389
mlp_6	62.8453	80.3309	80.6989	80.1530	65.6203	81.7122	81.7535	**81.8693**	47.6261	71.4723	70.9222	70.8635	70.0207	69.0157	69.2639	68.3623	60.6493	53.9909	49.3962	56.7873
cnn_1	!	!	!	!	59.8858	79.0052	80.8397	**81.6958**	47.2890	64.5739	64.6962	65.3893	13.3306	6.4003	7.7456	5.3403	64.7370	74.2356	71.7488	64.4516
cnn_2	!	!	!	!	70.1997	87.5254	89.4822	87.8108	54.3823	75.7439	77.9046	74.8878	32.6131	38.8096	35.7113	34.9368	89.1606	86.9955	85.8948	86.7101
cnn_3	!	!	!	!	73.8279	91.2759	92.0097	91.8467	53.0778	79.8206	81.2474	81.3697	60.2935	69.2213	1.4268	69.7920	**95.2710**	87.3216	94.12963	93.6404
cnn_4	!	!	!	!	73.3387	89.4007	91.0721	91.4798	54.2193	78.5161	78.1084	78.3530	68.9767	1.4268	1.4268	70.7297	94.2927	**95.8010**	93.3550	88.7077
cnn_5	!	!	!	!	77.9861	93.8035	94.5780	93.8035	57.1137	83.6526	85.6502	81.6143	74.5209	75.4586	77.9864	78.5161	**96.7386**	90.7052	91.3256	89.6045
cnn_6	!	!	!	!	76.1924	93.1512	93.6812	91.8874	54.4639	80.0244	83.3265	81.0843	72.6049	75.0509	1.4268	77.3562	**95.5972**	92.2136	90.6237	93.8035
cnn_7	!	!	!	!	76.4370	93.5589	94.0888	93.1512	58.2959	81.9856	83.3265	82.9596	75.1732	73.3387	73.9094	77.4153	98.0801	97.47248	**98.1247**	96.2965

6 Conclusion

In this chapter we will be assessing the results and outputs of the thesis. Let's first recite the most important parts of the thesis.

It first introduces what OCR is, how it is used and how it could be used in the future. It gives a brief overview of different types of optical character recognition and the main components of it.

Next it goes over a wide range of literature on the topic and quickly introduces the techniques used in the same field of science, regarding all the different parts of the OCR system.

This is followed by the introduction of the adopted methodology used in the thesis. All parts of the methodology are explained in depth, the pre-processing, the morphological processing, feature extraction and lastly the classification including the classifiers themselves.

Moreover, a completely new dataset of Japanese hiragana characters with relatively high variability due to the number of different handwritings and varying level of experience in the language is collected. This dataset will be made publicly available and freely accessible to every researcher that has interest in it. Also, the implementation details are shared including the used classifier models and their parameters.

Lastly, and most importantly, the thesis goes over the results of the experiments in detail. First, viewing the results classifier wise and explaining the most probable reasoning for the differences in the accuracy. Secondly, going over the results feature and pre-processing technique wise, applying similar reasoning to explain the differences. It shows the importance of proper pre-processing in addition to feature selection and classifier parameterizing.

Compared to the reviewed literature this thesis achieved a surprisingly high accuracy maxing out at 98.12%. Furthermore, using the explored techniques a 1.1% higher accuracy than the previous work by [Tsa16] was reached on a completely different dataset, validating the used techniques.

There are still several parts of OCR that could be added and researched in order to increase the accuracy even further or developing an application that could work from images of handwritten text. Thus, in the future additional research into segmentation and semantical post-processing could be done.

References

[AAP$^+$08] Christos-Nikolaos E Anagnostopoulos, Ioannis E Anagnostopoulos, Ioannis D Psoroulas, Vassili Loumos, and Eleftherios Kayafas. License plate recognition from still images and video sequences: A survey. *IEEE Transactions on intelligent transportation systems*, 9(3):377–391, 2008.

[AKHI15] Wali Mohammad Abdullah, Kazi Lutful Kabir, Nazmul Hasan, and Sharmin Islam. Development of a smart learning analytics system using bangla word recognition and an improved document driven dss. In *Computer and Information Engineering (ICCIE), 2015 1st International Conference on*, pages 75–78. IEEE, 2015.

[Alg13] Yasser M Alginahi. A survey on arabic character segmentation. *International Journal on Document Analysis and Recognition (IJDAR)*, 16(2):105–126, 2013.

[AYV01] Nafiz Arica and Fatos T Yarman-Vural. An overview of character recognition focused on off-line handwriting. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 31(2):216–233, 2001.

[BA17] Ekrem Baser and Yusuf ALTUN. Optical and handwritten digit recognition by using neural network on nao humanoid robot. *International Journal of Scientific Research in Information Systems and Engineering (IJSRISE)*, 2(3), 2017.

[BBB$^+$10] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: A cpu and gpu math compiler in python. In *Proc. 9th Python in Science Conf*, volume 1, 2010.

[BRB$^+$09] Federico Boschetti, Matteo Romanello, Alison Babeu, David Bamman, and Gregory Crane. Improving ocr accuracy for classical critical editions. In *International Conference on Theory and Practice of Digital Libraries*, pages 156–167. Springer, 2009.

[C$^+$15] François Chollet et al. Keras: Deep learning library for theano and tensorflow. *URL: https://keras. io/k*, 7:8, 2015.

[CH67] Thomas Cover and Peter Hart. Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27, 1967.

[CMLS15] Edgard Chammas, Chafic Mokbel, and Laurence Likforman-Sulem. Arabic handwritten document preprocessing and recognition. In *Document Analysis and Recognition (ICDAR), 2015 13th International Conference on*, pages 451–455. IEEE, 2015.

[DT05] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[ETL16] ETLCDB. Electrotechnical laboratory character database, 2016.

[FHT01] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. *The elements of statistical learning*, volume 1. Springer series in statistics New York, 2001.

[FSTV97] Pasquale Foggia, Carlo Sansone, Francesco Tortorella, and Mario Vento. Character recognition by geometrical moments on structural decompositions. In *Document Analysis and Recognition, 1997., Proceedings of the Fourth International Conference on*, volume 1, pages 6–10. IEEE, 1997.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[GSM11] Anshul Gupta, Manisha Srivastava, and Chitralekha Mahanta. Offline handwritten character recognition using neural network. In *Computer Applications and Industrial Electronics (ICCAIE), 2011 IEEE International Conference on*, pages 102–107. IEEE, 2011.

[HS04] Klaus Hechenbichler and Klaus Schliep. Weighted k-nearest-neighbor techniques and ordinal classification. 2004.

[HSPD10] Andreas Holzinger, Martin Schlögl, Bernhard Peischl, and Matjaz Debevc. Preferences of handwriting recognition on mobile information systems in medicine: Improving handwriting algorithm on the basis of real-life usability research. In *e-Business (ICE-B), Proceedings of the 2010 International Conference on*, pages 1–8. IEEE, 2010.

[HSZ87] Robert M Haralick, Stanley R Sternberg, and Xinhua Zhuang. Image analysis using mathematical morphology. *IEEE transactions on pattern analysis and machine intelligence*, (4):532–550, 1987.

[HZK07] Bing Quan Huang, YB Zhang, and Mohand Tahar Kechadi. Preprocessing techniques for online handwriting recognition. In *Intelligent Systems Design and Applications, 2007. ISDA 2007. Seventh International Conference on*, pages 793–800. IEEE, 2007.

[IA13] Saad M Ismail and Siti Norul Huda Sheikh Abdullah. Geometrical-matrix feature extraction for on-line handwritten characters recognition. *Journal of Theoretical and Applied Information Technology*, 49(1):86–93, 2013.

[JH17] Lõmps Joonas and Amnir Hadachi. The impact of morphological processing and feature selection on handwriting recognition accuracy. In *The 4th international Conference on Multimedia, Scientific Information and Visualization for Information Systems and Metrics*, 2017.

[KB14] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[KYS88] Wang Kai, YY Yang, and Ching Y Suen. Multi-layer projections for the classification of similar chinese characters. In *Pattern Recognition, 1988., 9th International Conference on*, pages 842–844. IEEE, 1988.

[LFC09] JC Lee, TJ Fong, and YF Chang. Feature extraction for handwritten chinese character recognition using xy graphs decomposition and haar wavelet. In *Signal and Image Processing Applications (ICSIPA), 2009 IEEE International Conference on*, pages 10–14. IEEE, 2009.

[LLW10] Min Liu, Tong Liu, and Guoli Wang. A compact representation of handwriting movements with mixtures of primitives. In *Intelligent Control and Automation (WCICA), 2010 8th World Congress on*, pages 1629–1634. IEEE, 2010.

[MB01] U-V Marti and Horst Bunke. Using a statistical language model to improve the performance of an hmm-based cursive handwriting recognition system. In *Hidden Markov models: applications in computer vision*, pages 65–90. World Scientific, 2001.

[MPV13] Oliviu Matei, Petrica C Pop, and H Vălean. Optical character recognition in real environments using neural networks and k-nearest neighbor. *Applied intelligence*, 39(4):739–748, 2013.

[OLS99] Il-Seok Oh, Jin-Seon Lee, and Ching Y. Suen. Analysis of class separation and combination of class-dependent features for handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(10):1089–1094, 1999.

[Ots79] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE transactions on systems, man, and cybernetics*, 9(1):62–66, 1979.

[PS00] Réjean Plamondon and Sargur N Srihari. Online and off-line handwriting recognition: a comprehensive survey. *IEEE Transactions on pattern analysis and machine intelligence*, 22(1):63–84, 2000.

[PVG$^+$11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[RR09] SV Rajashekararadhya and P Vanaja Ranjan. Zone-based hybrid feature extraction algorithm for handwritten numeral recognition of two popular indian scripts. In *Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, pages 526–530. IEEE, 2009.

[RS14] Amjad Rehman and Tanzila Saba. Neural networks for document image preprocessing: state of the art. *Artificial Intelligence Review*, 42(2):253–273, 2014.

[Saa14] Jose M Saavedra. Handwritten digit recognition based on pooling svm-classifiers using orientation and concavity based features. In *Iberoamerican Congress on Pattern Recognition*, pages 658–665. Springer, 2014.

[Sai96] Stephan R Sain. The nature of statistical learning theory, 1996.

[Sen94] Andrew William Senior. Off-line cursive handwriting recognition using recurrent neural networks. 1994.

[Ser83] Jean Serra. *Image analysis and mathematical morphology*. Academic Press, Inc., 1983.

[SLR$^+$14] Panyam Narahari Sastry, TR Vijaya Lakshmi, NV Koteswara Rao, TV Rajinikanth, and Abdul Wahab. Telugu handwritten character recognition using zoning features. In *IT Convergence and Security (ICITCS), 2014 International Conference on*, pages 1–4. IEEE, 2014.

[SR13] Tanzila Saba and Amjad Rehman. Effects of artificially intelligent tools on pattern recognition. *International Journal of Machine Learning and Cybernetics*, 4(2):155–162, 2013.

[Ste86] Stanley R Sternberg. Grayscale morphology. *Computer vision, graphics, and image processing*, 35(3):333–355, 1986.

[Tsa16] Charlie Tsai. Recognizing handwritten japanese characters using deep convolutional neural networks, 2016.

Appendix

I. Publications

Lõmps Joonas and Amnir Hadachi. The impact of morphological processing and feature selection on handwriting accuracy. In *The 4th international Conference on Multimedia, Scientific Information and Visualization for Information Systems and Metrics*, 2017

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public