Norwegian University of Life Sciences

Master's Thesis 2022 60 ECTS
Faculty of Science and Technology

Exploring the possibilities of obtaining CNN-quality classification models without using convolutional neural networks

Amir Arfan
MSc Data Science

Preface

The research completed in this thesis represents the finalization of my master's degree at the Norwegian University of Life Sciences (NMBU). I began working on the thesis at the start of 2022, and the journey to this date has been enjoyable and a precious learning experience.

First and foremost, I would like to thank my supervisor Prof. Ulf Geir Indahl. The main ideas for this thesis were shaped when I took the MATH310 course by Prof. Indahl in 2021, who gave me the opportunity to develop these ideas further in this thesis. I am also profoundly thankful to Prof. Indahl for providing knowledge about significant historical figures within machine learning, which has ultimately been a huge motivation source. I also offer my deepest gratitude to my co-supervisor, Prof. Kristian Hovde Liland. The time my supervisors devoted to providing feedback and criticism is immensely appreciated. They have been a source of inspiration, guidance, and knowledge. Their encouraging words and openness to creative approaches have made working on this thesis a pleasant journey.

Furthermore, I would like to thank all my close friends and family for their support. Especially Emilie Giltvedt Langeland and my brother Ali Arfan for being here for me throughout my master's degree.

Finally, I would like to end this preface with a quote that has helped me overcome the challenges I have faced when working on this thesis:

> *"Do not imagine that, if something is hard for you to achieve, it is therefore impossible for any man; but rather consider anything that is humanly possible and appropriate to lie within your own reach too."*
>
> \- Marcus Aurelius, *Meditations*

Amir Arfan
Ås, November 16, 2022

Abstract

In this thesis, we pursue the success of Convolutional Neural Networks for image classification tasks. We explore the possibilities of achieving state-of-the-art performance without explicitly using CNNs on 2D grayscale images.

We propose a Binary Patch Convolution (BPC) framework based on binarized patches from each group of images in a supervised task, eliminating the kernel learning process of CNNs. The binarized patches act as activations of different shapes and are applied using convolution. One of the key aspects of the framework is that it maintains a direct relation between the convolution kernels and the original images. Therefore, we can present a method to measure information content in a feature map for observing relations between different groups. We discuss and test different strategies for selecting groups of images to extract patches from while evaluating their effect on classification accuracy. The practical implementation of the BPC framework allows for many convolution kernels to be evaluated, positively impacting the framework's performance. Ultimately, the proposed framework can extract pertinent features for classification and can be combined with any classifier. The framework is tested on the MNIST and Fashion-MNIST datasets and achieves competitive accuracy, even outperforming related work. We also discuss challenges and future work applicable to the framework.

Furthermore, we have attempted to capture trends in the error of images by proposing an iterative variant of singular value bases classification. The proposed method fails to capture a generalizable error trend; thus, we have recognized that it is a challenging task for images. The process has given valuable insight into how to approach image classification problems.

On top of that, we have examined the effects of negative transfer inherent in an original problem. Our experiments show that models trained on all groups in the data (global) are outperformed by models trained on different combinations of subgroups (local). Our proposed approaches for minimizing negative transfer within a task effectively increase classification accuracy. However, they are infeasible to deploy in practical scenarios due to the computation time introduced. The results are meant to motivate research toward within-task minimization of negative transfer, primarily since the research is focused on doing so in transfer learning.

Contents

List of Figures

List of Tables

Acronyms

AI	Artifical Intelligence.
ANN	Artifical Neural Network.
BNN	Bayesian Neural Network.
BOLS	Bi-objective Least Squares.
BPC	Binary Patch Convolution.
C-KS	Clustered Kennard Stone.
CNN	Convolutional Neural Network.
CRS	Clustered Random Supervised.
CTM	Convolutional Tsetlin Machine.
DANN	Domain-Adversarial Neural Network.
DL	Deep Learning.
FFT	Fast Fourier Transform.
GAN	Generative Adversarial Network.
GPU	Graphical Processing Unit.
ICA	Independent Component Analysis.
LOOCV/LOO-CV	Leave-One-Out Cross-Validation.
ML	Machine Learning.
MLP	Multi-Layer Perceptron.
OLS	Ordinary Least Squares.
ReLU	Rectified Linear Unit.
RS	Random Supervised.
RSVD	Randomized Singular Value Decomposition.
SGD	Stochastic Gradient Descent.
SVD	Singular Value Decomposition.
SVM	Support Vector Machine.
VAE	Variational Autoencoder.

Chapter 1

Introduction

The ability to classify objects is a crucial ability that we humans possess. Our ability improves as we age through experiences and learning. Upon seeing countless examples, we create relatable associations through our understanding that help us label objects into categories. Our understanding is shaped by creating analogies, and new things are represented as something we know [1, p. 57]. This classification ability is used in everyday tasks and solves complex problems. An individual's path in life leads to different experiences, and knowledge accumulation makes us experts in various fields. We guide ourselves towards the correct solution to problems through trial and error. It is only natural that to achieve artificial intelligence (AI) comparable to humans, the ability to classify must be in place, and the fundamentals of our learning process should inspire ways of improving AI.

Current solutions to hand this ability to computers rely on a field of artificial intelligence known as machine learning (ML), which is the capability of learning using data to perform supervised and unsupervised tasks. A key field within ML is image classification. Image classification is used for miscellaneous real-life tasks such as tumor detection [2], digit classification [3], facial recognition [4], autonomous driving [5], and more, which motivates research in this field. Deep Learning (DL) through Convolutional Neural Networks (CNNs) is the current state-of-the-art for image classification and has been responsible for bringing attention to neural networks in general. To improve the ability of machines for image classification and our understanding of where the machines are failing, we should use the information gained from achieving state-of-the-art performance with CNNs, and explore the possibilites of achieving comparable performance without their explicit use.

1.1 Motivation

CNNs trained to classify images produce convolution filters that resemble partial structures of the data. Essentially, they are producing class-specific templates [6]. These templates are applied to images through convolution to extract features, making them easier to distinguish. A direction to explore would be if one can replace these templates with existing images alleviating the training process while retaining the same level of performance as a CNN. Additionally, the role of the operators in CNNs, and the various convolution layers are difficult to interpret [7]. Using existing images to create filter banks and use them as convolution kernels may give us a better understanding of what role convolution plays.

Furthermore, one should seek to improve existing ML approaches by utilizing techniques inspired by humans' learning process. Using the concept of negative transfer and reinforcement, we may try to get an indication of where our classification model fails to aid us in the task. We should also aim to find a method to surpass plateaus using the information attained in the process using multiple classification stages.

1.2 Objectives

This thesis aims to provide a method for obtaining Convolutional Neural Network type of performance on 2D grayscale images without the use of said networks. We seek to find a method which requires no learning of convolution kernels, and maintains the direct connection between convolution kernels and the original images. We wish to take inspiration from the technical aspects of convolutional neural networks. Additionally we aim to use unsupervised learning to aid us in achieving this task. We also wish to explore whether classical image analysis strategies can improve our understanding of the task. While keeping all the objectives in mind, we also require that the proposed approaches are competitive in terms of classification accuracy against state-of-the-art techniques.

This thesis also aims to explore alternative techniques with the goal of both improving and utilizing them. We aim to find an approach for improving classification using *Singular Vector Decomposition* (SVD) bases [8]. Additionally, we wish to examine what improvements can be made by considering the concept of negative transfer within an original problem.

1.3 Structure

The thesis begins by presenting theory relevant to understanding the proposed methods and results displayed. We then introduce and explain the proposed methods, followed by the results produced by applying these methods to practical problems for image classification. Finally, we discuss the results, challenges, and future work.

Appendix A: Datasets, includes information about the datasets used in this thesis. Appendix B: Practical Implementations, contains information about the practical implementations of the methods used. Additionally, Appendix C: BPC Python Package, contains information about one proposed method, Binary Patch Convolution (BPC) as a Python package.

Chapter 2

Theory

The theory chapter will present the central components to understand this thesis's work. This chapter will give an overview of the essential elements of machine learning. Techniques for training various models covering optimization and measuring loss will be presented and discussed. We will also take a look at some essential concepts from Linear Algebra. We will delve into Deep Learning by covering the fundamental subjects. Finally, we will cover Convolutional Neural Networks, as they are a primary inspiration for this thesis.

This chapter is essential for laying the groundwork for this thesis's proposed methods.

2.1 Machine Learning

Machine Learning (ML) is the field of giving machines the ability to perform a particular task. Using definition 2.1.1, we can begin to grasp the concept of ML.

Definition 2.1.1 (Machine Learning)**.** A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E [9, p. 2].

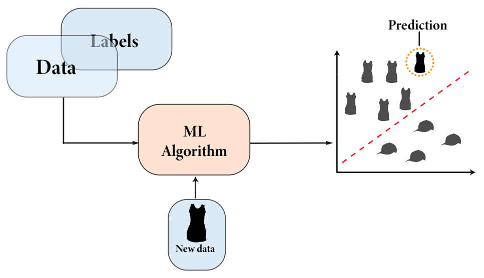
Tasks T is something we want the machine to complete for us and are predefined by us. The goal is to replace labor intelligently by using machine learning algorithms to train predictive models. The performance P measure will give insight into how well the ML algorithm performs and guide the algorithm in the training process. Some relevant metrics for assessing performance are discussed later in Section 2.2.1. Experience E may come in various forms, and experiences are needed to create associations for training and predicting where unseen experiences are essential for evaluating the performance of the ML algorithm.

In this thesis, the focus is on supervised image classification. The ultimate goal is that given an unseen image, the ML algorithm is supposed to create a model that can predict the corresponding label correctly. The experience is thus labeled grayscale images. The performance will then measure how accurately the algorithm can predict the label of an unseen image.

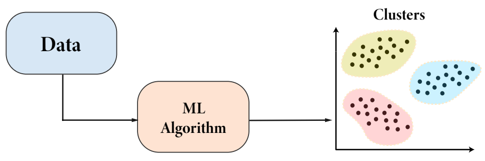
This thesis' machine learning formulation can be summarized as such:

- Task: Pass an unseen image to the model trained by a ML algorithm and get a classification label
- Performance: How accurately does the model predict the actual label
- Experience: Pairs of image $X \in \mathbb{R}^{m \times n \times c}$, and label $y \in \mathbb{R}^G$

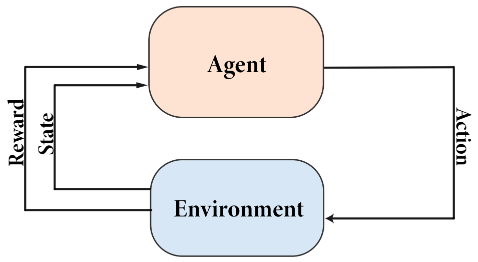
To learn from the experience ML algorithm has three main types of methods: *supervised*, *unsupervised*, and *reinforcement learning*.



(a) A supervised binary classification task. The ML algorithm learns the association between the input data and the labels. Given a new unseen data sample, it predicts its' qualitative target.



(b) An unsupervised clustering task. The data contains no labels, and the ML algorithm has to find patterns to divide the data into separate clusters.



(c) Illustration of a reinforcement learning setup. Showing the relationship between an agent and its environment.

Figure 2.1: Illustration of the three main learning methods for ML algorithms

2.1.1 Data

The experience we consider in this thesis comes in the shape of images. The images can be seen as feature matrices, where each pixel value corresponds to a feature. The images have a height m, width n, and color channel c. An image is denoted by $x \in \mathbb{R}^{m \times n \times c}$. In this thesis, we only consider images with a single channel (grayscale), $c = 1$. For computational purposes, a single image can also be flattened into a one-dimensional vector where the length of the vector, denoted l, is the product of the width and height.

A set of N grayscale images can be stored as a three-dimensional matrix composed of multiple 2D arrays defined as $X \in \mathbb{R}^{N \times m \times n}$ shown in Figure 2.2a. Or it can be stored as a two-dimensional matrix composed of multiple 1D vectors defined as $X \in \mathbb{R}^{N \times l}$ shown in Figure 2.2b.

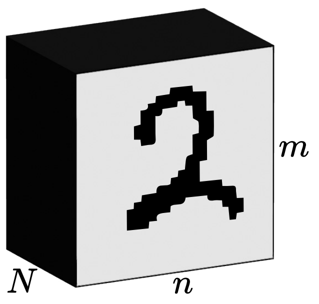
2.1.2 Metrics

In machine learning, prediction metrics provide a high-level understanding of evaluating the performance of a model. Distance metrics, on the other hand, can be used to compare samples. We will now present some relevant metrics for this thesis.

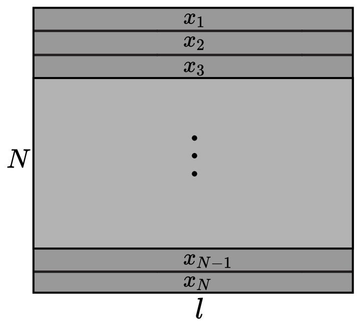
Accuracy

This thesis focuses on classification tasks, with the ultimate goal of predicting a sample correctly. The accuracy of a machine learning model is defined in definition 2.1.2.

Definition 2.1.2 (Accuracy)**.** The accuracy is given by the number of correctly classified samples divided by the total amount of samples. The range of the accuracy is between 0 and 1, and can be presented in decimals or percentages.



(a) 3D collection of N images. Stored as arrays of m rows and n columns.



(b) 2D collection of N images. Stored as vectors with length l.

Figure 2.2: Illustration of how a collection of images can be stored as data

$$\text{Accuracy} = \frac{\#\text{of correctly classified samples}}{\#\text{of total samples}}$$

The *error rate* of a model is simply 1 − accuracy.

Distance Metrics

A commonly used distance metric in machine learning is the Minkowski Distance, as presented in definition 2.1.3.

Definition 2.1.3 (Minkowski Distance)**.** The Minkowski Distance [10, p. 101] between two points X, Y, where $X = (x_1, x_2, \dots x_N)$ and $Y = (y_1, y_2, \dots y_N) \in \mathbf{R^N}$ is given as such:

$$D(X,Y) = \sqrt[p]{\sum_{i}^{N} |x_i - y_i|^p} \tag{2.1}$$

When $p = 2$, the *Minkowski distance* is known as the L_2 or *Euclidean distance*. When $p = 1$, the distance is known as L_1 or *Manhattan distance*. These distance metrics are displayed in Figure 2.3.

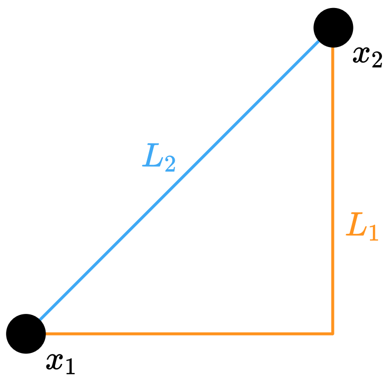


Figure 2.3: Illustration of the L_1 and L_2 distances between two points.

2.1.3 Supervised Learning

The experience comes in pairs of data inputs and labels in supervised learning. The ML algorithm will then create a model based on the associations learned from the experience. Categorical classification is an example of a supervised learning task where the categories can be binary or multinomial. The ML algorithm's mission in binary classification is to label input data between two possibilities correctly. In multinomial classification, the possibilities are more than two. Another example of a supervised learning task is regression, where the goal is to predict a continuous variable. An example of a regression task is to predict the prices of houses.

One of the challenges with supervised learning is the need for annotated data. The process of annotating data can be time-consuming and require expertise. Such as in medical fields where one must be qualified to annotate the data [11]. On the contrary, unsupervised learning requires no labels. However, unsupervised learning cannot return a target value for the input but can be used for acquiring insight into the data.

2.1.4 Unsupervised Learning

Unsupervised learning is often used to find patterns from the given data, which we can use to improve our understanding. In unsupervised learning, the ML algorithms require no labels. The objective of unsupervised learning is to extract information without the need for supervision. Clustering is an example of an unsupervised learning task that aims to separate the data into separate blobs, which we can then use to infer meaningful associations. For instance, if data is collected by watching strikers in football, the features given are time on the field and goals scored. If one wished to separate the players depending on the ratio between playing time and amount of goals scored, manually doing so would be time-consuming. However, players with similar ratios would be placed into the same clusters with clustering, automating the task. Another example of unsupervised learning is representation learning, where the goal is to find the most suitable representation of the data. Suitability will vary between different datasets and different purposes. An example of representation learning is finding a low-dimensional representation, which allows one to plot the data to infer meaningful information, also known as dimensionality reduction. Representation learning is covered in more detail in Section 2.3.1.

One of the challenges with unsupervised learning is finding the suitable algorithm for the task in question. In supervised learning, one has a straightforward performance measure, whether or not the label predicted is correct. On the other hand, for example, in clustering, there is no ground truth for what amount of clusters divides the data in the best manner. As we will see in this thesis, unsupervised learning can be a great tool both by itself and when combined with supervised learning.

2.1.5 Reinforcement Learning

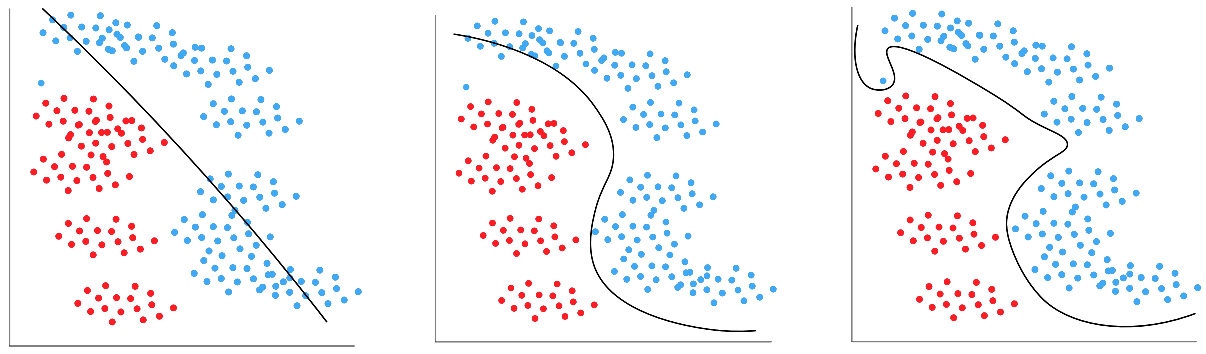
In reinforcement learning, the experience comes in the shape of interactions. The goal is to train an agent to complete a task. The agent interacts with its environment, as seen in Figure 2.1c. Each action it performs is returned with a response from the environment. The performance is measured in terms of a reward from the environment. The reward indicates how well the agent performed the task in question. The response also includes an associated state which holds information about the environment. A typical goal for an agent can be to win a board game, where the state of the environment contains information about possible outcomes.

Reinforcement learning is out of the scope of this project. However, the key insights can be beneficial to remember when designing future ML methods. In essence, with reinforcement learning, one wants the agent to find the optimal series of actions to achieve the defined goal. An important factor is the design of the reward function. Depending on the task, an agent may need to perform many sub-tasks. Distributing the reward in a granular manner is essential for completing the ultimate task. Additionally, how to present the state and decide which elements of an environment one should leave out to find the optimal actions is critical. Reinforcement learning is covered in great detail in *Reinforcement Learning: An Introduction* [12].

2.1.6 Generalization

The ability we wish to pass on to supervised Machine Learning algorithms is to predict or classify unseen experiences correctly. This ability is denoted by generalization and is what we aim to achieve by creating machine learning frameworks.

The data we use for training a model is known as the training set, and the data we use to evaluate the generalization capability is known as the test set. Optimally, we wish to have little difference between the training and test errors. When the model performs significantly better on the training set, we say that the model is overfitting. On the contrary, when the model performs poorly on the training set, we say that the model is underfitting. When the model is overfitting, it has captured tendencies in the training data that are too specific to generalize. On the other hand, when the model is underfitting, it fails to capture significant characteristics.


(a) An underfitting model, failing to capture the general trend of the problem.

(b) A good tradeoff between underfitting and overfitting. The model captures some specific traits and the general trend.

(c) An overfitting model, capturing overly specific traits.

Figure 2.4: Illustration of model generalization

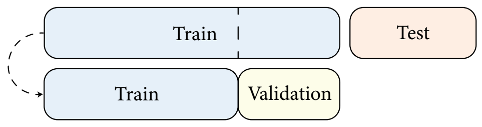
One way to control the model's generalization capability is by adjusting its capacity [13, p. 109]. A model's capacity is an informal definition of how complex a function the model can fit. The capacity of a model can, for example, be adjusted by the hypothesis space. A linear least squares model can be extended in capacity by introducing polynomial terms [13, p. 109]. A low-capacity model may tend to underfit, and a high-capacity model may tend to overfit depending on the complexity of the task.

One must often customize the model to perform well on the task at hand. We can make such customizations to improve test errors. A general term for these customizations is regularization. In this thesis, we will utilize Tikhonov regularization, which is covered in more detail in Section 3.1.4.

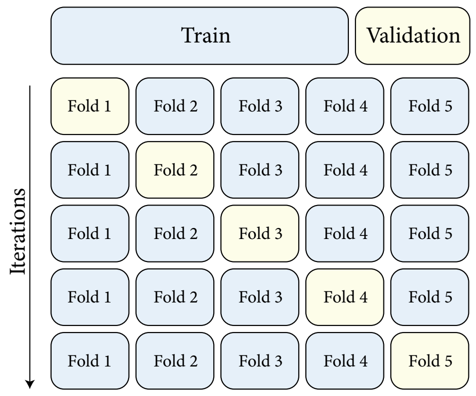
Cross-Validation

To get an accurate evaluation of the generalization ability, we cannot let the test set influence the choice of any model parameters. We can also not only use the training set as guidance, as that would lead to significant overfitting. That is why we construct validation sets out of our training data, as shown in Figure 2.5a. Cross-validation (CV) is randomly constructing validation subsets. The most common cross-validation technique is K-fold cross-validation [13,

p. 119]. K-fold CV is done by creating k subsets, then iterating over the subsets, and each iteration uses the current subset as a validation set and the rest of the data as the training set. Figure 2.5b illustrates the K-fold CV for $k = 5$. Leave-one-out CV (LOOCV) is a particular case of the K-fold validation technique where $k = N$, where N is the number of samples in the dataset.



(a) Illustration of dividing the training data into two subsets without affecting the test data.



(b) Illustration of the K-fold CV, where $k = 5$.

Figure 2.5: Figure displaying techniques for splitting data for validation purposes.

2.2 Optimization

A building block for machine learning is optimization. Given an ML algorithm, one aims to improve the algorithm's performance based on a metric. An objective function $f : \mathbb{R}^n \to \mathbb{R}$ often expresses this metric, and the goal can be to find the value of a variable x^* that minimizes or maximizes this objective function. In machine learning, this objective function is often called a loss function. Minimizing or maximizing an objective function can be equivalent since minimizing $-f$ is the same as maximizing f.

Definition 2.2.1 (Local Minima)**.** x^* is a local minimum of an objective function $f : \mathbb{R}^n \to \mathbb{R}$ defined on some set $S \subset \mathbb{R}^n$, if $\exists$ an $\epsilon > 0$ such that $f(x) \geq f(x^*)\ \forall\, x \in S$ and $\|x - x^*\| < \epsilon$.

Definition 2.2.2 (Global Minima)**.** x^* is a global minimum of an objective function $f : \mathbb{R}^n \to \mathbb{R}$ defined on some set $S \subset \mathbb{R}^n$, if $f(x) \geq f(x^*)\ \forall\, x \in S$.

The minima one finds through optimization can either be local or global. Local and global minima are defined in definitions 2.2.1 and 2.2.2, respectively. The optimal scenario is to find the global minimum, but that is not always possible, especially in deep learning [13, p. 81]. Thus we often stop the optimization process at an acceptable local minimum since there might be several local minima. This is illustrated in Figure 2.6.

2.2.1 Loss Functions

As previously mentioned, loss functions act as an indication of performance for a task. We will briefly introduce some loss functions that are relevant to our thesis.

A typical loss function is the Mean Squared Error (MSE) defined in Definition 2.2.3. This loss function is commonly used in regression problems. It calculates the squared distances between the target and the predicted value. This error measure decreases when the prediction is close to the target value. In computer vision, this metric can be used for comparing images. For two images, $z_1, z_2 \in \mathbb{R}^l$, one computes the square differences between each pixel value in z_1

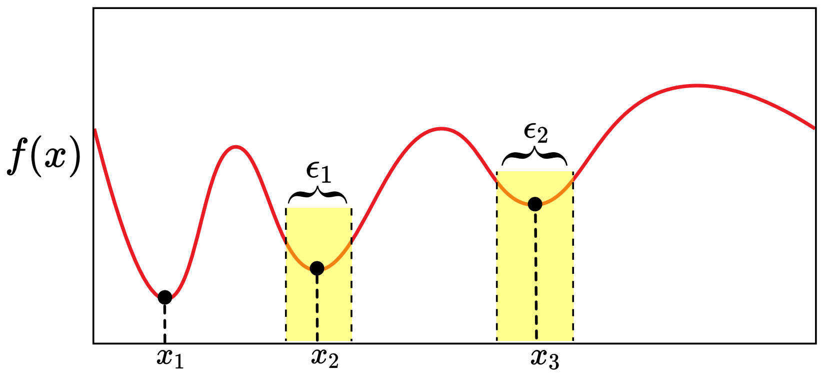


Figure 2.6: Illustration of minima of $f(x)$. The point x_1 corresponds to the global minimum, while x_2 and x_3 are local minima in their ϵ-neighborhoods. In the scenario where the global minimum is not found, one would prefer x_2 over x_3.

and z_2, then computes the mean based on the number of pixels, l. This will give an indication of the degree of similarity between the two images [14].

Definition 2.2.3 (Mean Squared Error)**.** The Mean Squared Error Loss (MSE) is defined as

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2 \tag{2.2}$$

where N equals the amount of samples predicted, y is the true target value, and $\hat{y}$ is the predicted value.

Another loss function worth noting is Cross-Entropy Loss, defined in Definition 2.2.4. This function is often used for classification tasks. This loss function is based upon *Maximum Likelihood Estimation* (MLE), where one wishes to minimize the disparity between the data distribution learned by the model and the actual distribution [13, p. 129]. The Cross-Entropy loss indicates how well the ML model fits the given data distribution. The predicted log probability, $log(\hat{p})$, from a classification model is compared to a binary indicator y.

Definition 2.2.4 (Cross-Entropy Loss)**.** The Cross-Entropy Loss is defined as

$$L_{CE} = -\sum_{i=1}^{G} y_i log(\hat{p}_i) \tag{2.3}$$

where G is the amount of classes in a multinomial classification problem. y_i is a binary indicator of whether the sample in question belongs to class i. $\hat{p}_i$ is the predicted probability of the sample belonging to class i.

2.2.2 Gradient Descent

We need an optimization algorithm to navigate the landscapes of the loss functions. Introduced by Cauchy in 1847 [15], we have the method of gradient descent. This method is the most common neural network optimization technique [16] but can also be used as a general optimization algorithm. The intuition behind gradient descent is to use the first-order derivative of a function. Since the derivative gives information about the slope, we wish to move towards the descending direction, in other words, in the negative direction of the gradient as illustrated in Figure 2.7. Repeatedly taking steps, of size η, in the steepest descent direction until a convergence criterion is reached is the essence of algorithm 1. A criterion can be, for example, when the difference between the new value of x, and the previous value is sufficiently small. The iterative procedure can then be stopped, and a global or a local minimum should have been found.

Algorithm 1 Gradient Descent

```
Require: Gradient of the loss function $\nabla f$
Require: Predefined learning rate $\eta$
Require: Predefined starting point $x_0$
  $x_{prev} \leftarrow x_0$
  while Convergence criterion is not met do
    $x_{new} \leftarrow x_{prev} - \eta \nabla f$
    $x_{prev} \leftarrow x_{new}$
  end while
  $x^* \leftarrow x_{new}$
  return $x^*$
```

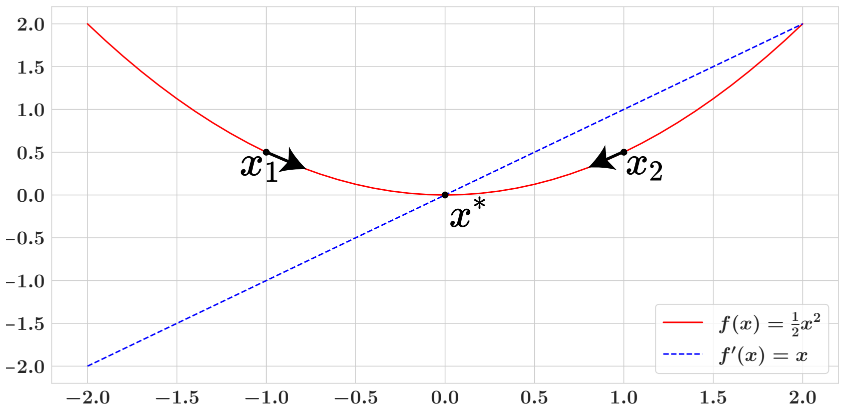


Figure 2.7: Illustration of gradient descent directions at points x_1 and x_2. The goal is to find the optimal point x^*. The gradient at x_1 is negative, thus the direction of steepest descent is towards the right. The gradient at x_2 is positive, thus the direction of steepest descent is towards the left. The figure is inspired by Goodfellow et al. [13, p. 80].

The choice of stepsizes, or learning rates, in gradient-based optimization problems can be cumbersome. An excessive learning rate can lead to overshooting the minima for each step. On the other hand, a diminutive learning rate can lead to slow convergence. Choices of learning rates are illustrated in Figure 2.8. We will not go into the details of selecting stepsize η, or initializing the starting point x_0. Variations of gradient descent used in deep learning will be covered in Section 2.4.3.

2.3 Linear Algebra

We will present some essential concepts from Linear Algebra that are key to this thesis. We begin by introducing one of the fundamental matrix decompositions in machine learning, the Singular Value Decomposition (SVD), and then we move on to the ordinary least squares (OLS).

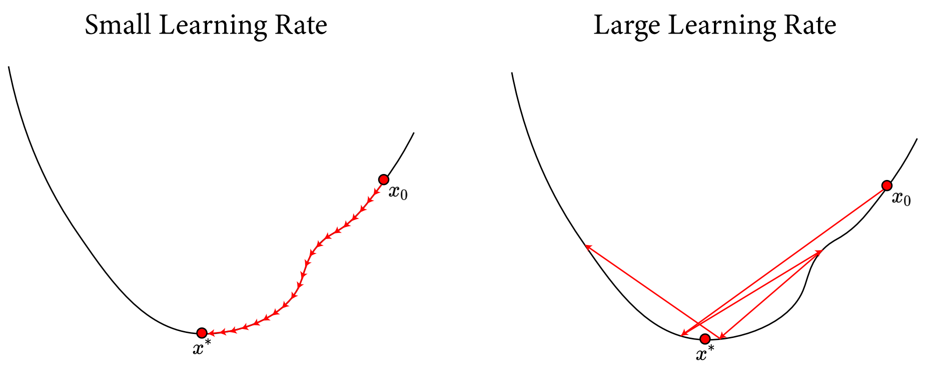


Figure 2.8: Illustration of a small and large learning rate, and its effects on the convergence of the gradient descent algorithm.

2.3.1 Singular Value Decomposition

Definition 2.3.1 (Singular Value Decomposition)**.** The Singular Value Decomposition for any $m \times n$ matrix A, is given as [17, p. 58]:

$$A = U\Sigma V^T \tag{2.4}$$

where U and V are orthogonal matrices with dimensions $m \times m$, and $n \times n$ respectively. Σ is a diagonal $m \times n$ matrix.

$$\Sigma = diag(\sigma_1, \sigma_2, \dots, \sigma_n),\ \sigma_1 \geq \sigma_2 \geq \dots \sigma_n \geq 0$$

The Singular Value Decomposition (SVD) is a type of matrix factorization defined in definition 2.3.1. It provides an orthonormal basis for a matrix's column and row space. A beneficial property of the SVD is that it separates the matrix into parts, which come in order of importance, making it an excellent tool for machine learning.

We can also obtain the rank[1] of a matrix by looking at the non-zero singular values and get a compact representation of the SVD:

$$A_r = U_r\Sigma_r V_r^T = \begin{bmatrix} u_1 & u_2 & \cdots & u_r \end{bmatrix} \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_r \end{bmatrix} \begin{bmatrix} v_1^T \\ v_2^T \\ \vdots \\ v_r^T \end{bmatrix} \tag{2.5}$$

In many practical scenarios, the matrix A is a combination of a low-rank matrix and noise, $A = A_0 + \mathcal{N}$. The noise can be reduced using information from the singular values. The k largest singular values in the matrix Σ carry the most information [17, p. 78]. We can obtain the numerical/effective rank of the matrix by inspecting the singular values and discarding some of them with respect to a certain tolerance. Thus we can approximate any matrix A with the truncated SVD, as defined in definition 2.3.2.

Definition 2.3.2 (Truncated SVD)**.** The truncated Singular Value Decomposition provides an approximation of the original matrix using the k largest singular values:

$$A_k \approx U_k\Sigma_k V_k^T \tag{2.6}$$

[1]The rank of a matrix is the dimension of the vector space spanned by its columns, given as $r \leq min(m, n)$, where m is the number of rows, n is the number of columns.

where $U_k = \begin{bmatrix} u_1 & u_2 & \cdots & u_k \end{bmatrix}$ and $V_k = \begin{bmatrix} v_1 & v_2 & \cdots & v_k \end{bmatrix}$. The diagonal Σ matrix is:

$$\Sigma_k = \begin{bmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_k \end{bmatrix}$$

A helpful property comes from the Eckart-Young theorem, which states that A_k is the best rank k approximation of A [18]. The truncated SVD is highly useful for removing the noise associated with small singular values, and is commonly used in image processing.

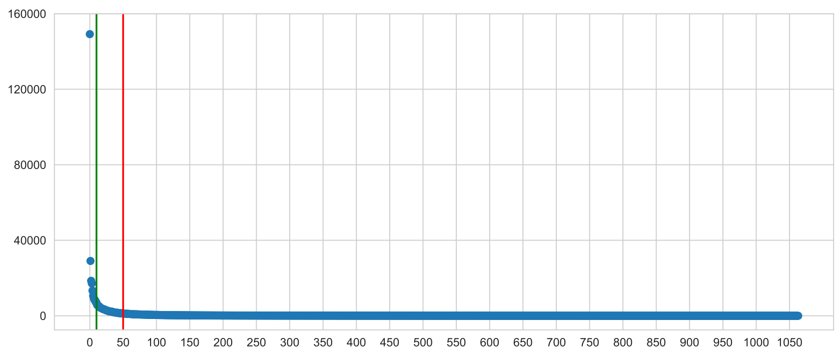


Figure 2.9: Illustration of the singular values from matrix Σ, for a matrix $A \in \mathbb{R}^{1153\times1064}$.

Image Compression

Let us consider a matrix $A \in \mathbb{R}^{1153\times1064}$, where $SVD(A) = U\Sigma V^T$, and observe its' singular values as shown in Figure 2.9. We can see that most singular values are small compared to the first few. We can use a smaller amount of singular values to approximate the matrix and still obtain a pretty good representation. Two cutoff values are shown in the figure, $k = 10$ and $k = 50$. These values are used for the reconstruction presented in Figure 2.10. We can observe that A_{10} is not a good reconstruction and loses most of the specific information but captures the general shape of the image. A_{50} is a relatively good approximation where the compressed image is quite clear. Thus effectively illustrating that the truncated SVD is a handy tool for compressing images and reducing their complexity.

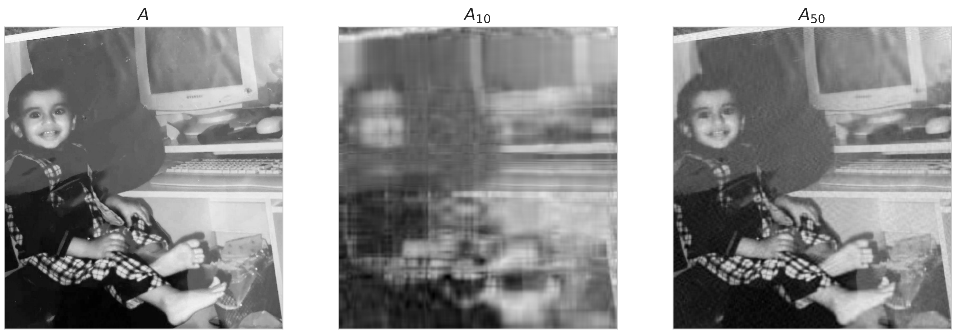


Figure 2.10: Reconstruction of image $A \in \mathbb{R}^{1153\times1064}$, using SVD truncations A_k with $k = 10$ and $k = 50$.

Principal Component Analysis

Principal Component Analysis (PCA) is an unsupervised dimensionality reduction tool. PCA learns a representation of the data with a lower dimensionality which is helpful for many things, for example, visualizing multidimensional data. The representation we obtain consists of variables that are orthogonal to each other.

We can formulate the Principal Component Analysis using the SVD and the critical insight from the Eckart-Young theorem [17, p. 76]. Given a centered matrix $X \in \mathbb{R}^{m \times n}$, and its SVD, $X = U\Sigma V^T$. The columns of the right singular matrix V are known as the principal directions, and the columns of $U\Sigma$ are known as the principal components. Due to the first k singular values carrying the most important information, the first principal component $Xv_1 = \sigma_1 u_1$ explains most of the variance in the dataset. Essentially the PCA learns a linear orthogonal transformation to project data into a lower-dimensional subspace.

The statistical relationship between PCA and SVD is through the squared singular values being the eigenvalues of the sample covariance matrix $\frac{1}{N-1}A^TA$, and the columns of V are the eigenvectors of the covariance matrix [17, p. 76].

We can measure the amount of variance explained by k components using the singular values, which can be valuable in selecting the number of components in various tasks:

$$\frac{\sum_{i=1}^{k} \sigma_i^2}{\sum_{i=1}^{r} \sigma_i^2} \tag{2.7}$$

It is worth noting that the PCA constructs components that are linear combinations of the original variables. The components alone do not provide any information about the original variables, making it difficult to interpret information about the original variables [19].

2.3.2 Least Squares Classifier

We will now present a simple classifier, namely the Ordinary Least Squares (OLS) Classifier. The least squares problem can be seen as an optimization objective. Say we wish to solve the system:

$$Ax = b \tag{2.8}$$

If $b \in Col(A)$, the solution can be found with $x = A^{-1}b$. If that is not the case, we get a minimization problem formulated as such:

$$\|Ax - b\|_2^2 \tag{2.9}$$

We seek the solution $\hat{x}$ such that $A\hat{x} - b$ is as small as possible. Thus in practical scenarios, we often solve the least squares problem with the following normal equations:

$$A^TAx = A^Tb \tag{2.10}$$

The coefficients can then be acquired as such:

$$\hat{x} = (A^TA)^{-1}A^Tb \tag{2.11}$$

Moreover, the predicted values can be acquired by the following equation:

$$\hat{b} = A\hat{x} = A(A^TA)^{-1}A^Tb \tag{2.12}$$

$P = A(A^TA)^{-1}A^T$ is a projection matrix that projects onto the column space of A as shown in Figure 2.11.

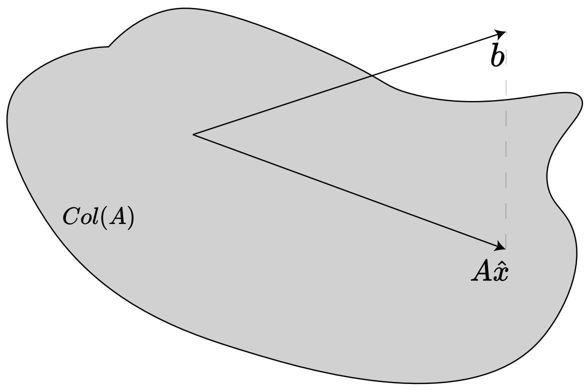


Figure 2.11: Illustration of a least squares projection onto the column space of A.

The least squares method is often used for regression tasks but can be expanded as a classifier. We will look at the multi-class example, where the number of labels exceeds two. The multi-class least squares classifier can be seen as a one versus all method [20, p. 300]. A new dataset is constructed for each label, where the values are replaced with 1 if the sample belongs to the label in question or 0 if it does not. After constructing the dataset of new labels, we proceed with fitting a linear regression model. The classifier can be formulated as such [20, p. 300]:

$$\hat{f}(x) = \underset{i=1,\ldots,G}{\operatorname{argmax}} \tilde{f}_i(x) \tag{2.13}$$

Where there are G labels, and $\tilde{f}_i$ is the least squares fitted model for each label i.

In a practical implementation, each sample would be stored in A with its' features as row vectors. b would be a one-hot encoding of the actual labels. For instance, in the MNIST dataset [21], the matrix A for the training set would be of size 60000×784 where each image 28×28 is flattened. b would be of size 60000×10 since there are 10 labels in the dataset.

2.4 Deep Learning

Deep Learning (DL) is a type of machine learning that uses several layers of mappings to create representations that allow machines to more easily perform supervised, unsupervised, and reinforcement tasks in an automated manner. We will explore the trademark of DL, Artificial Neural Networks (ANNs). The central questions we will analyze are the fundamentals of a neural network, the process of training one, and a particular type of neural network known as a Convolutional Neural Network (CNN) that utilizes convolutional layers.

To understand deep learning, we should first understand the concept of an artificial neuron and the Perceptron. The foundation of deep learning is built on the insights of McCulloch and Pitts, who introduced the first artificial neuron in 1943 [22], inspired by the human brain. In 1958, Frank Rosenblatt extended this idea to the Perceptron [23], and a learning algorithm was included.

2.4.1 The Perceptron

The idea of an artificial neuron, inspired by the biological neuron [22], is to create a thresholded gate. This gate activates when the input exceeds a certain threshold. An activation function models this activation. An artificial neuron acts as a binary supervised classification algorithm. The Perceptron uses a weighted sum of inputs, and a bias passed onto an activation function,

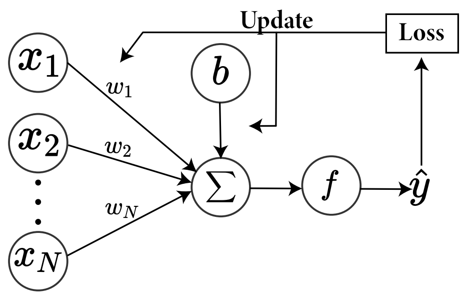


Figure 2.12: Illustration of a Perceptron architecture, prediction process and the learning algorithm. Figure inspired by Raschka [10].

as shown in Equation 2.14. In the Perceptron, the activation function is defined as Equation 2.15. The activation function compresses the input from the weighted sum into an output $\hat{y} \in [0, 1]$, which limits the Perceptron's capability of separating classes to linearly separable problems. The goal of the Perceptron is to find a set of weights that can map the input into a representation that highlights the essential features. These essential features are combined with the activation function to create a linear separation boundary. The architecture of the Perceptron is illustrated in Figure 2.12.

The Perceptron learning algorithm is based upon trial and error [23]. The process is implemented through iterations. After each training sample is processed, the loss is calculated and used to update the model's weights and bias, as shown in Equation 2.16. The η is commonly known as the learning rate, and is the step size for each iteration.

$$\hat{y} = f(o) = f(\sum_{i=1}^{N} x_i + b) \tag{2.14}$$

$$f(o) = \begin{cases} 1 & \text{if } o \geq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.15}$$

$$\begin{aligned} w_i &= w_i + \eta(y_i - \hat{y}_i)x_i \\ b &= b + \eta(y_i - \hat{y}_i) \end{aligned} \tag{2.16}$$

The Multi-Layer Perceptron

While a single Perceptron is limited to learning linear relationships, an extension of the Perceptron known as the Multi-Layer Perceptron (MLP) can also model non-linear relationships. A MLP consists of several Perceptrons chained together in layers where each Perceptron is known as a neuron, as shown in Figure 2.13. The layers of a MLP whose output we do not directly observe are known as hidden layers. It is worth noting that adding layers is not enough for modeling non-linear relationships. One must also choose a suitable activation function which we further discuss in Section 2.4.2. A standard MLP architecture is a feedforward one, where the input is passed in one direction. The feedforward network defines a mapping $\hat{y} = f(x)$, where $f(x)$ can be a chain of an arbitrary number of functions. The architecture displayed in Figure 2.13 is known as a fully connected network, where each neuron in the previous layer is connected to each neuron in the subsequent layer. However, this is not a requirement for MLPs, as we will see with Convolutional Neural Networks.

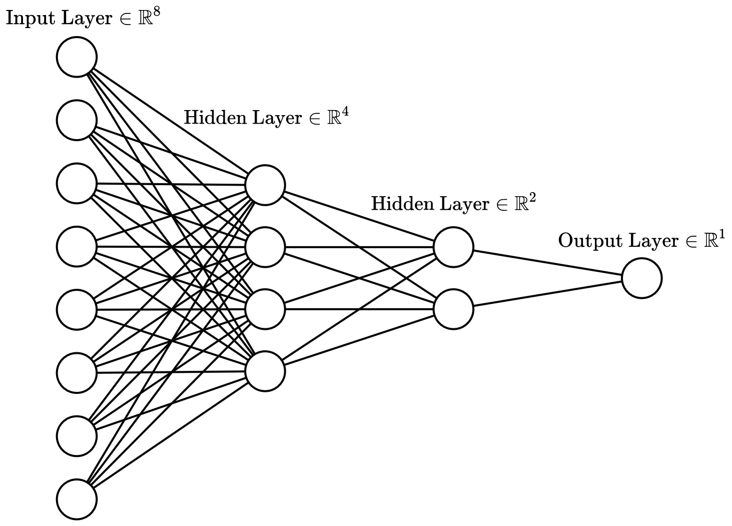

Input Layer $\in \mathbb{R}^8$
Hidden Layer $\in \mathbb{R}^4$
Hidden Layer $\in \mathbb{R}^2$
Output Layer $\in \mathbb{R}^1$

Figure 2.13: A fully connected multi-layer perceptron, with one input layer, two hidden layers, and a single output layer.

2.4.2 Activation Functions

The choice of activation functions in Deep Learning is essential. An activation function determines the output for each layer in a neural network. A combination of activation functions can be seen as a series of switches that may activate based on different features. This series, in the end, creates a combination that represents a specific result. The choice of activation functions can impact accuracy and computation time performance. Trial and error often determine which activation function should be used in the hidden layers [13, p. 186]. The activation functions used in the hidden layers are often the same, and the task at hand determines the activation function used in the output layer. Some common ones are used in Deep Learning, and the ones we will consider are Linear, Step, Sigmoid, and Rectified Linear Unit (ReLU) activation functions. An illustration of the activation functions is given in Figure 2.14.

Linear Activation Function

As defined in Definition 2.4.1, a linear activation is often used in an output layer for regression tasks where one wishes to predict a continuous value [10, p. 37]. If the linear activation function were used in the hidden layers, one could only model linear relationships, regardless of the number of hidden layers. Therefore, in the hidden layers, one uses activation functions with non-linearities to capture a wider variety of relations.

Definition 2.4.1 (Linear Activation Function)**.**

$$f(x) = x \tag{2.17}$$

Step Activation Function

The step Activation function, also known as the Heaviside Activation function [24, p. 131], defined in definition 2.4.2, is used in the Perceptron. This activation function is no longer a regular choice in the hidden layers due to the backpropagation algorithm's usage of gradients. The backpropagation algorithm is discussed in more detail in Section 2.4.3. The gradient of the Step function is not very informative as it is undefined at the origin and zero elsewhere. Additionally, since the output is binary, the information sent out from the hidden layers lacks

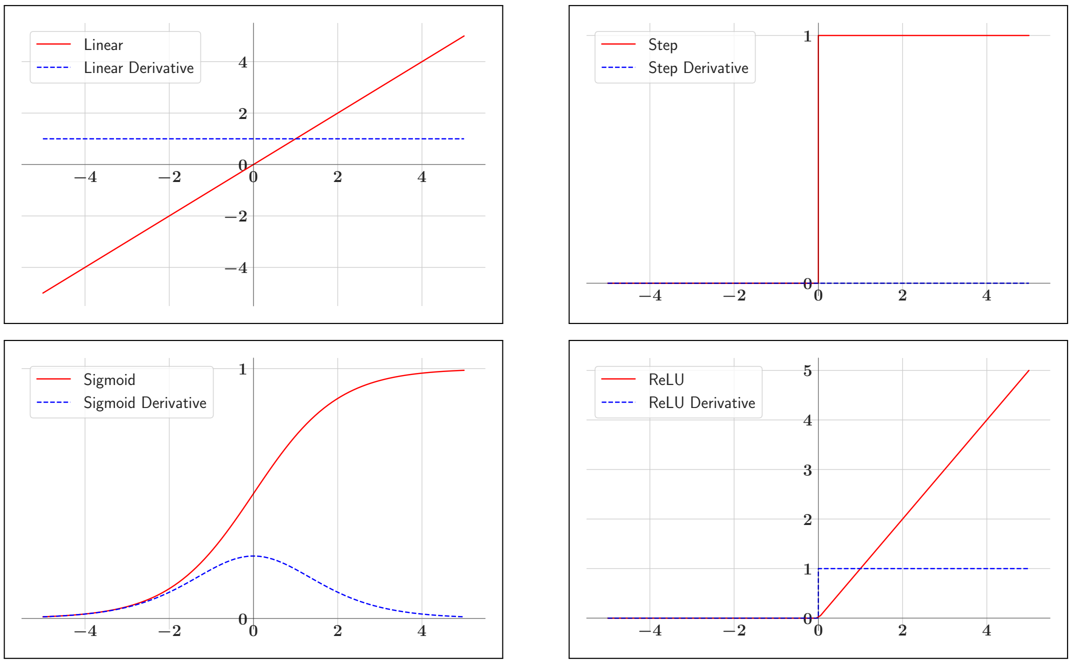


Figure 2.14: Illustration of activation functions and their derivatives

granularity making the training process more difficult due to the information passed between the hidden layers being more difficult to perceive during the optimization process.

Definition 2.4.2 (Step Activation Function)**.**

$$f(x) = \begin{cases} 0, & \text{for } x \leq 0 \\ 1, & \text{for } x > 0 \end{cases} \tag{2.18}$$

Sigmoid Activation Function

The Sigmoid Activation function defined in definition 2.4.3 maps the output to a range of 0 and 1. This activation function offers more granularity than the Step function and gives a probabilistic interpretation [24, p. 234]. The Sigmoid function was commonly used in the hidden layers, but it has the problem of vanishing gradients; the function tends to saturate for large numbers [13, p. 66]. The function becomes insensitive and offers little information about slight changes.

Definition 2.4.3 (Sigmoid Activation Function)**.**

$$f(x) = \frac{1}{1 + e^{-x}} \tag{2.19}$$

Rectified Linear Unit Activaiton Function

The Rectified Linear Unit activation function, defined in definition 2.4.4, is the most commonly used activation function in modern neural networks [25]. It is piecewise linear and introduces a non-linearity while remaining easy to optimize with gradient methods. It does not have the saturation issue as opposed to the Sigmoid activation function. However, it does have an issue with dead neurons, where the weighted input into the neuron is of negative or zero value, such that the gradient becomes zero [25]. The zero gradients lead to the neuron's less likely activation in the optimization process. Variants of the ReLU function have been proposed to alleviate this problem, such as the leaky ReLU [25]. We will not go into the details of the variants of ReLU activation functions.

Definition 2.4.4 (Rectified Linear Unit Activation Function)**.**

$$f(x) = \begin{cases} 0 & \text{for} \quad x \leq 0 \\ x & \text{for} \quad x > 0 \end{cases} \tag{2.20}$$

2.4.3 Training a neural network model

The training process for a deep neural network is done similarly to the Perceptron, by trial and error. The backpropagation algorithm [26] is a way of computing gradients for neural networks consisting of multiple neurons and is the most commonly used algorithm for training neural networks [13, p. 197]. The utilization of gradients to optimize the neural network's parameters is most commonly done by optimizers such as the Stochastic Gradient Descent (SGD) or Adaptive Moment Estimation (Adam).

Backpropagation gradient descent

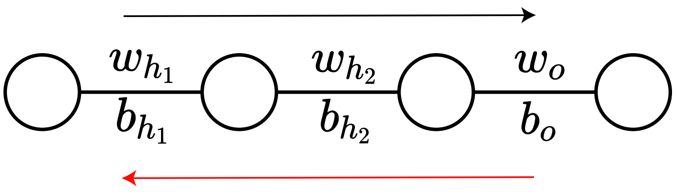


Figure 2.15: A simple neural network with one input neuron, two hidden neurons, and one output neuron. The black arrow illustrating the feedforward direction, and the red arrow indicating the backpropagation direction. Figure inspired by Sanderson [27].

In a deep feedforward neural network consisting of several hidden layers, one can alter several weights and biases to minimize the loss function. When using gradient-based optimization, one might need to calculate the gradient in a very high dimensional space. The backpropagation algorithm offers a way to compute this negative gradient.

A prediction is made for each sample from the data, and an error is calculated from a loss function. To minimize the loss function, one must look at the output layer's result, y, and to alter y, one must alter all the previous layers' weights and biases. To alter the layers between the input and output layer, one must propagate the output error L_o, back to attain the error L_{h_i} for each hidden layer h_i, hence the word backpropagation. The chain rule is essential for the backpropagation algorithm. One must compute the partial derivatives of each step because the output layer can be seen as an extended function composition [10, p. 362].

Using the following notations for the feedforward process [10, p. 363]:

$$\Sigma_h = X_{in} W_h^T + b_h \ (\textbf{Net input of the hidden layer})$$
$$\sigma_h = f(\Sigma_h) \ (\textbf{Activation of the hidden layer, with activation function } f)$$
$$\Sigma_o = \sigma_h W_o^T + b_o \ (\textbf{Net input of the output layer})$$
$$\sigma_o = f(\Sigma_o) \ (\textbf{Activation of the output layer, with activation function } f)$$

And considering the simple neural network in Figure 2.15 with one input feature, two hidden neurons, and one output neuron, the gradient for the output layer using the chain rule is given as [27]:

$$\frac{\partial L}{\partial w_o} = \frac{\partial \Sigma_o}{\partial w_o}\frac{\partial \sigma_o}{\partial \Sigma_o}\frac{\partial L_o}{\partial \sigma_o}$$
$$\frac{\partial L}{\partial b_o} = \frac{\partial \Sigma_o}{\partial b_o}\frac{\partial \sigma_o}{\partial \Sigma_o}\frac{\partial L_o}{\partial \sigma_o}$$

For the previous hidden layer the derivative is given as [27]:

$$\frac{\partial L}{\partial w_{h_2}} = \frac{\partial \Sigma_{h_2}}{\partial w_{h_2}}\frac{\partial \sigma_{h_2}}{\partial \Sigma_{h_2}}\frac{\partial \Sigma_o}{\partial \sigma_{h_2}}\frac{\partial \sigma_o}{\partial \Sigma_o}\frac{\partial L_o}{\partial \sigma_o}$$
$$\frac{\partial L}{\partial b_{h_2}} = \frac{\partial \Sigma_{h_2}}{\partial b_{h_2}}\frac{\partial \sigma_{h_2}}{\partial \Sigma_{h_2}}\frac{\partial \Sigma_o}{\partial \sigma_{h_2}}\frac{\partial \sigma_o}{\partial \Sigma_o}\frac{\partial L_o}{\partial \sigma_o}$$

This gives us the ability to calculate a series of partial derivatives for each layer with respect to the weights and biases, ultimately getting a gradient that can be used with gradient based optimization methods [27]:

$$\nabla L = \begin{bmatrix} \frac{\partial L}{\partial w_{h_1}} \\ \frac{\partial L}{\partial b_{h_1}} \\ \vdots \\ \frac{\partial L}{\partial w_o} \\ \frac{\partial L}{\partial b_o} \end{bmatrix}$$

Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is the extension of gradient descent used to optimize neural networks. ANNs, in general, need a vast amount of data to generalize well, and training on large datasets requires considerable computational effort [28]. Stochastic gradient descent offers a way to alleviate computing the gradient per data sample. For N samples, one must compute the gradient for the whole dataset in backpropagation with standard gradient descent per epoch:

$$\nabla L = \frac{1}{N}\sum_{i=1}^{N} \nabla L_i$$

The key with SGD is using a minibatch of samples $N_{batch} < N$ to approximate each step in the gradient descent method instead of the whole dataset [28]:

$$\nabla L = \frac{1}{N_{batch}}\sum_{i=1}^{N_{batch}} \nabla L_i$$

The minibatch samples are chosen randomly from the whole dataset, thus a model can be fitted with a smaller subset of the data. Larger batches provide more accurate estimates of the gradient [13, p. 272]. The idea is to get a sense of the general descent direction through subsets of the data. A comparison between gradient descent and SGD is shown in Figure 2.16.

To avoid being stuck in local minima, one can utilize SGD with momentum. Momentum uses an exponentially decaying rolling average of gradients [13, p. 288]. In other words, it keeps track of all the previous gradients. It uses them as acceleration, similar to a ball rolling downhill that will overcome smaller hills due to the acceleration gained, and eventually slows down due to friction. An update step of SGD with momentum can be formulated as such [13, p. 288]:

$$
\begin{aligned}
v &= \alpha v - \eta \frac{1}{N_{batch}} \sum_{i=1}^{N_{batch}} \nabla L_i \\
x &= x - v
\end{aligned}
$$

where $\alpha \in [0, 1)$ is the decay rate. Some adaptive versions of the SGD exist, such as Adaptive Moment Estimation (Adam), which is an adaptive method for computing individual learning rates for parameters using the first and second moments of the gradients [29]. It uses the first moments, the moving averages of the gradients similar to SGD with momentum, and the second moments, the uncentered variance of the gradient. The Adam algorithm also includes bias counteractions, ensuring that the moments initialized as zeros are no longer biased towards zero. Adam is computationally efficient and invariant to diagonal rescaling of the gradients. Additionally, it is well suited for vast data, making it a suitable optimization algorithm choice for deep learning [29].

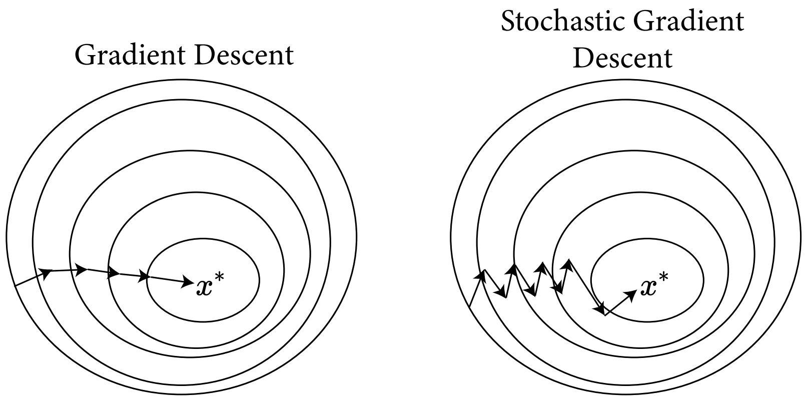


Figure 2.16: A comparison between gradient descent and stochastic gradient descent. SGD uses approximations of the gradients and may require more steps to converge towards the optimum. A trademark of SGD are the jagged descent steps.

2.5 Convolutional Neural Networks

As previously mentioned, Convolutional Neural Networks (CNNs) are considered as the state-of-the-art for solving image classification problems. The origin of CNNs stems from the neocognitron from 1980 [30], which had similar architecture as modern day CNNs but lacked the backpropagation algorithm for supervised training [31]. It was introduced with backpropagation by Lecun et al. in 1989 [32]. CNNs are directly motivated by visual neuroscience[31]. They were not especially popular until their rise of attention due to AlexNet [33] in 2012, which outperformed their competition on a large vision dataset, illustrating that learning convolutional kernels is quite effective for classification purposes. CNNs' use of convolutional and pooling layers and their shared filter banks makes them unique. They differ from regular neural networks in the sense that the convolutional layers replace the standard weighted multiplication in at least one of the layers [13, p. 321]. The convolutional layers are often paired with pooling layers connected through a non-linear transformation, typically obtained by using the ReLU activation function, and most commonly followed by a dense layer, as for the standard MLP.

2.5.1 The idea of convolutional Layers

Convolution can be seen as a linear operation where a convolution kernel W (often called a filter, used interchangeably in this thesis), is applied to input I in terms of a matrix multiplication. This kind of operation is known as a discrete convolution. The kernels are most commonly of smaller dimensions than the input. The operation extracts features that are translation equivariant [2]. For two-dimensional arrays, a discrete convolution is given as follows:

$$Y(i,j) = (I * W)(i,j) = \sum_m \sum_n I(i+m, j+n)W(m,n) \tag{2.21}$$

The two-dimensional discrete convolution process is illustrated in Figure 2.17. The output of the discrete convolution, often referred to as a feature map, can be controlled in terms of size using a process known as padding. By padding an input, one adds an arbitrary amount of pixels surrounding the input such that when a filter window strides through the array, the output dimensions are increased due to the increased dimensions of the input. The commonly used value for the pixels added is zero, known as zero padding. All padding mentioned in this thesis will refer to zero padding. Padding is essential, especially in wide CNNs, where several stages of convolutional kernels are applied. Without padding, the output would shrink each time, and one would have to use smaller kernels each time or be limited to narrower CNNs.

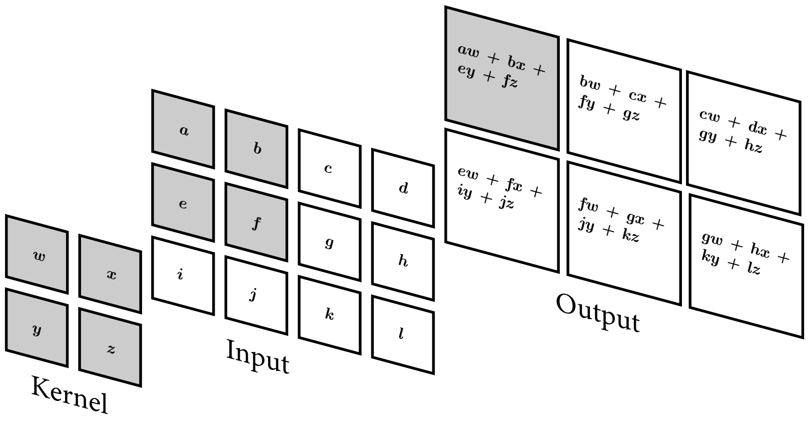


Figure 2.17: Illustration of two-dimensional *valid* convolution process with unit stride. The grey boxes illustrate how the kernel is applied to the upper left area of the matrix. Figure inspired by Goodfellow et al. [13, p. 324].

The traditional padding methods used in deep learning are often referred to as *valid*, *same*, and *full*. A *valid* convolution limits the kernel window to stay within the image and adds no surrounding pixels, and this shrinks the output dimensions each time it is applied. A *same* convolution adds surrounding pixels such that the output dimensions are equal to the input dimensions, which allows one to chain an optional amount of convolutions. However, this can make the pixels at the border underrepresented [13, p. 339]. A *full* convolution is where several surrounding pixels are added such that each pixel is stepped through the same number of times by the kernel window. Goodfellow et al. state that the optimal amount of padding is between valid and same [13, p. 340]. The different kinds of padding are illustrated in Figure 2.18.

Another way to control the output size is through the *stride* parameter. Stride controls the shifting of the kernel window. A unit stride shifts one pixel at a time. Sometimes one might

[2] **Translation equivariance** ensures that if an object in an image is shifted to a different area of the image, the linear operation extracts the same information, but this information is shifted as well.

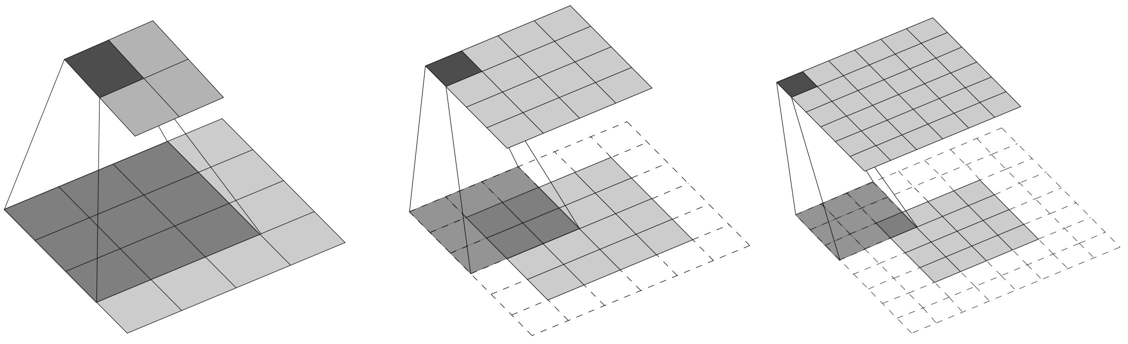

(a) *Valid* convolution. (b) *Same* convolution. (c) *Full* convolution.

Figure 2.18: Illustration of the main padding methods used in deep learning.

want to skip over parts of the image or not have overlapping values for the kernel. This can be obtained by using stride shifts of several pixels.

Kernels

The central selling point of CNNs is that they learn feature extractors in the shape of kernels rather than having experts handcrafting methods for doing so. However, it is no hidden fact that ANNs require a large amount of data to generalize well [34].

There are various ways to initialize these kernels. They are most often initialized through sampling from a distribution. A common one is Xavier initialization as defined in Equation 2.22. Where n_{in} is the number of inputs to the layer, and n_{out} is the number of outputs of the layer. This initialization aims to keep the same activation variance and the backpropagated variance as one moves through a neural network [35]. Xavier initialization was made to combat the slow convergence of neural networks when the weights were randomly initialized or initialized through a normal distribution with fixed standard deviation [35]. Other initialization methods, such as the Kaiming initialization, which is defined in Equation 2.23, is known to be more suitable for the ReLU activation function [36].

$$\mathrm{W} \sim U\left(-\sqrt{\frac{6}{n_{in}+n_{out}}}, \sqrt{\frac{6}{n_{in}+n_{out}}}\right) \tag{2.22}$$

$$\mathrm{W} \sim N\left(0, \frac{2}{n_{in}}\right) \tag{2.23}$$

After the training, a CNN has learned feature extractors that detect edges in different orientations, motifs, and objects. The features extracted are used for classification [31]. An illustration of convolution kernels learned by a simple CNN is shown in Figure 2.19, where some of the kernels learned resemble partial structures of the digits they are trained on.

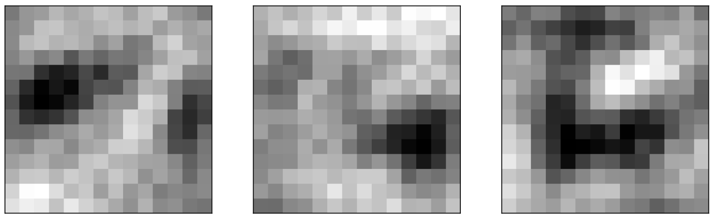
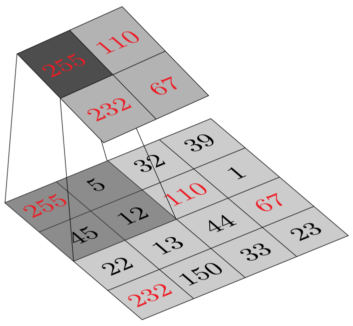
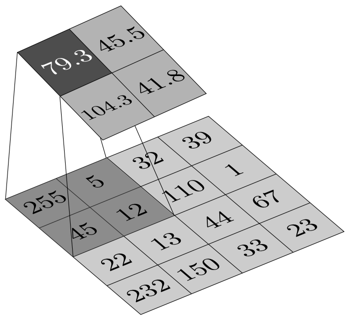


Figure 2.19: Illustration of learned convolution kernels from a CNN with a single convolution layer with 3 kernels. The CNN was trained on the MNIST dataset using 10 epochs.

2.5.2 Pooling Layers



(a) Max-pooling.



(b) Average-pooling.

Figure 2.20: Illustration of two pooling methods with no overlapping. The pooling operation is done with a 2×2 window.

A pooling layer's purpose is to reduce the dimensions of the feature maps produced from convolution and introduce small translation invariance[3]. The essence of a pooling layer is to slide a window through the feature maps with a given stride and perform an operation on each window to reduce the dimensionality. Typical pooling methods are max-pooling [37] and average-pooling. Max-pooling computes the maximum value of a given neighborhood of values in an array, and average-pooling computes the average value. Both pooling methods are illustrated in Figure 2.20.

[3]**Translation invariance** is an attribute that makes translations in an input insignificant. When detecting an object that has been shifted, pooling ensures to some degree that the features extracted are the same from one shift to another.

Chapter 3

Methods

This chapter presents ideas of a new method for supervised image classification. We will go into the details of the reasoning behind the proposed ideas as well as the technical aspects for implementation in comparison to the techniques presented in chapter 2.

3.1 The Binary Patch Convolution idea

Inspired by the success of CNNs and convolution kernels' effectiveness as feature extractors, we propose a method for supervised image classification using binarized patches extracted from a subset of the training images. The idea is to create a filter bank using real images instead of learning the filters through CNNs or other convolution-based methods such as the Convolutional Tsetlin Machine (CTM) [38]. As CTM training process frequently result in (learned) convolution filters that are quite similar to sub-regions of the images associated with the different classes of a classification problem, we question the necessity of learning such convolution kernels. Our goal is to provide a general *Binary Patch Convolution* (BPC) method and explore strategies for selecting images to extract patches acting as effective filters.

3.1.1 Patch Extraction

The framework we propose first consists of extracting patches from a subset of the training images. The patches can be of arbitrary size. Our motivation is to extract patches from different regions of the images to capture a collection of shape fragments reflecting various parts of an object's general shape.

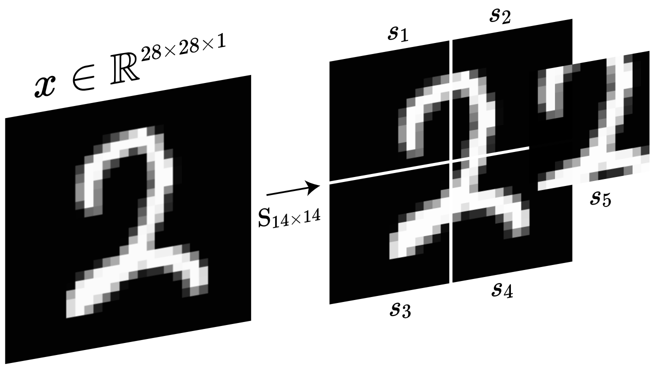


Figure 3.1: Illustration of a patch extraction process for a digit of class 2 from the MNIST dataset [21]. The function S extracts patches of size 14×14 from the input image, including a center patch.

We define a function S : $\mathbb{R}^{m \times n \times 1} \rightarrow \mathbb{R}^{k \times p_1 \times p_2 \times 1}$. Where m is the height of the input matrix, and n is the width. The function returns k patches. S takes an image $x \in \mathbb{R}^{m \times n}$, and the dimensions (p_1, p_2) of the patches as input. The function partitions the image to return a collection of patches of a specified size. The function also includes a patch of the given size from the center of the image, ensuring that any possible general shapes contained in the non-overlapping regions in the center are extracted. The patches extracted from a selection of training images belonging to each group of the classification problem are to be used for subsequently defining a collection of binary convolution filters. An illustration of the patch extraction procedure is shown in Figure 3.1.

3.1.2 The Binarization step

The second stage of the proposed method is to binarize each patch using a common threshold. The idea is to turn the extracted patches into binary representatives of characteristic shapes. Binarizing all the patches also ensures that they capture variations on the same scale when applied in a convolution process.

The binarization is done by replacing all pixel values above or equal to some threshold T with 1 and the rest with -1. The threshold can be chosen empirically through cross-validation or by using *Otsu's method*[39] to each individual patch and taking the average of the resulting threshold values.

Otsu's Method

Otsu's method separates each patch into foreground (+1) and background (-1) pixels according to a threshold value T. The method can be seen as a clustering method for dividing the image into two clusters by iteratively searching for a threshold minimizing the within-class variance with respect to the choice of threshold value T

$$\sigma_w^2(T) = \omega_0(T)\sigma_0^2(T) + \omega_1(T)\sigma_1^2(T) \tag{3.1}$$

$$p_i = \frac{\#\,\text{of pixels of intensity}\, i}{\text{Total number of pixels}} \tag{3.2}$$

$$\begin{aligned} \omega_0 &= \sum_{i=1}^{k} p_i \\ \omega_1 &= \sum_{i=k+1}^{L} p_i, \end{aligned} \tag{3.3}$$

where ω_0 and ω_1 denote the probabilities (fractions) of each class of pixels. ω_0 is computed using pixel intensities with range $[1, k]$, and ω_1 is computed using pixels with range $[k+1, L]$. σ_0^2 and σ_1^2 are the variances of each cluster of pixel intensities, where σ_0^2 is the variance for the pixel intensities with range $[1, k]$, and σ_1^2 is the variance for the pixel intensities with range $[k+1, L]$ [39]. The optimization objective is to minimize Equation 3.1. The method systematically iterates through all possible pixel intensity values of thresholds searching for the T that minimizes the within-class variance defined in Equation 3.1. In a grayscale image, the pixel intensities range from 0 to 255, Figure 3.2 illustrates the binarization process using a threshold obtained from using Otsu's method for an image from the MNIST dataset.

We suggest using Otsu's method on each of the extracted patches to find an ideal threshold, and then use the mean of all these thresholds as compromise for the final patch binarization. The averaging is done to avoid using Otsu's method each time when selecting patches, and by selecting a single threshold, all patches have a mutual definition of foreground and background.

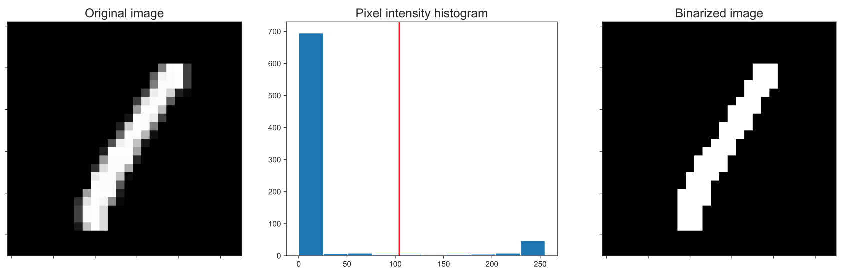


Figure 3.2: Illustration of an image from the MNIST dataset binarized using a threshold. The threshold is obtained from using Otsu's method. The pixel intensity histogram is displayed with a vertical red line that shows the threshold value.

3.1.3 Convolution and Pooling

After binarizing the extracted patches, they are ready to be used as convolution kernels/ filters. We recommend applying these filters in the same fashion as the convolutions described in Section 2.5.1. When the binary patches are applied to images through discrete convolutions with a sliding window, each pixel under the window is multiplied by 1 or -1 and summed. The produced values represent the degree of matching with the shape in the patch. The binary patches match shapes as well as negative space. Every single patch produces one feature map. We obtain a dimensionality reduction by performing max-pooling with a window of the same size as the feature map, with no stride, thus storing the most prominent kernel activations as numerical values for each image and each patch. Simply put, we take the maximum value from each feature map produced by every patch and store it as a feature. Therefore, the number of features extracted by the BPC framework is directly determined by the number of patches. All features produced by every patch applied to one image can be stored as a feature vector for that particular image, which can be passed on to a classifier. The BPC framework is illustrated in Figure 3.3.

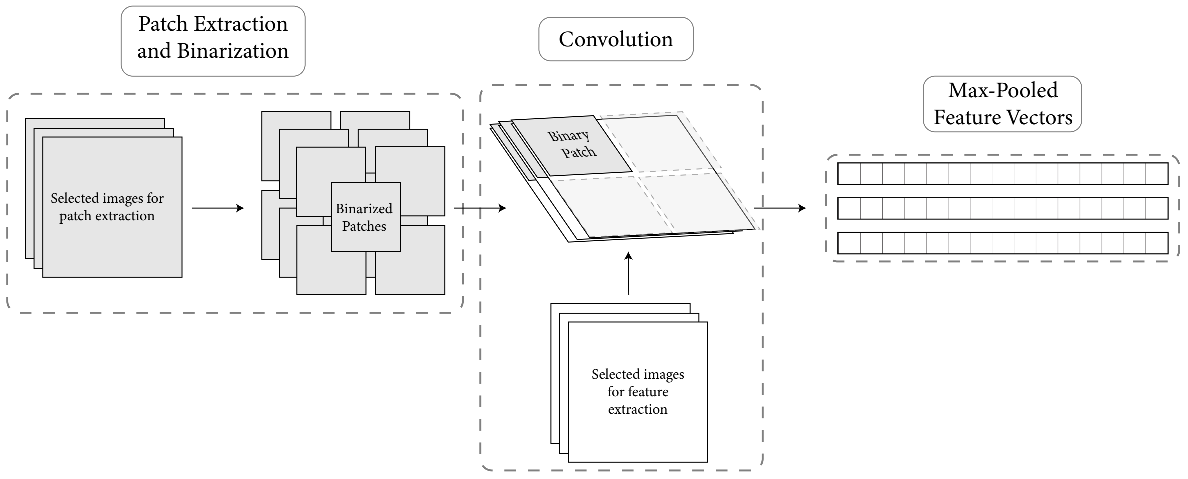

Patch Extraction and Binarization
Selected images for patch extraction
Binarized Patches
Convolution
Binary Patch
Selected images for feature extraction
Max-Pooled Feature Vectors

Figure 3.3: Illustration of the Binary Patch Convolution baseline. Patches are extracted from selected images, the grey rectangles in the figure. The patches are then binarized and used as convolution kernels. Each patch is applied to each target image for feature extraction, which results in a feature map. The feature map is max-pooled to produce a single feature. The number of patches determines the number of features. The figure shows the process for using 15 patches as convolution kernels, applied on 3 images. The result is a set of feature vectors. Each target image has a corresponding feature vector, where each vector has 15 features.

3.1.4 BOLS Classifier

To test the BPC framework's effectiveness in generating feature vectors that are good/efficient for classification, we consider a Tikhonov regularized linear least squares classifier. The regularization parameter is chosen by using the fast algorithm for computing the LOOCV errors, which would otherwise be unfeasible based on an ordinary LOOCV process.

The Tikhonov regularized least squares [40] classifier is an extension of the linear least squares classifier presented in Section 2.3.2, however we will now use a statistical notation, where $Ax = b$ is replaced with $Xb = y$. It is essentially a bi-objective problem where one minimizes the following expression with respect to b:

$$\|Xb - y\|^2 + \lambda\|b\|^2, \tag{3.4}$$

where $\lambda > 0$ is the regularization parameter. We regularize in terms of the L_2 norm by penalizing large weights, and we favor small weights, which leads to less complexity in the model that, in turn, leads to less risk of overfitting.

The optimization problem can be formulated in terms of matrix notation as an OLS problem $Z_\lambda b = y_0$ [41] where

$$Z_\lambda = \begin{bmatrix} X \\ \sqrt{\lambda} I \end{bmatrix}, \quad y_0 = \begin{bmatrix} y \\ 0 \end{bmatrix} \tag{3.5}$$

The associated normal equations are then given as:

$$Z_\lambda^T Z_\lambda b = Z_\lambda^T y_0 \tag{3.6}$$

The unique solution is defined as:

$$b_\lambda = (X^T X + \lambda I)^{-1} X^T y \tag{3.7}$$

It can be simplified using the SVD of $X = U\Sigma V^T$:

$$\begin{aligned} b_\lambda = \left(X^T X + \lambda I\right)^{-1} X^T y &= V\left(\Sigma^T\Sigma + \lambda I\right)^{-1} V^T \left(V\Sigma U^T y\right) \\ &= V\left(\Sigma^T\Sigma + \lambda I\right)^{-1} \Sigma U^T y. \end{aligned} \tag{3.8}$$

Now using the truncated SVD up to rank r:

$$b_\lambda = X_\lambda^\dagger y = V_r\left(\Sigma_r + \lambda\Sigma_r^{-1}\right)^{-1} U_r^T y = V_r c_\lambda \tag{3.9}$$

Equation 3.9 shows that using a linear combination with the right singular vectors, we can obtain the regression coefficients effectively.

We can obtain the fitted values as such:

$$\hat{y}_\lambda = X b_\lambda = X V_r c_\lambda = U_r \Sigma_r c_\lambda = U_r d_\lambda \tag{3.10}$$

For multiple responses $Y \in \mathbb{R}^{N\times G}$, such as in our case for classification where we are using one-hot-encoding for the labels, the regression coefficent matrix is given as such:

$$\mathrm{B}_\lambda = X_\lambda^\dagger Y = V_r C_\lambda \tag{3.11}$$

And the fitted values are given as such:

$$\hat{Y}_\lambda = X\ \mathrm{B}_\lambda = U_r D_\lambda \tag{3.12}$$

C_λ and D_λ correspond to c_λ and d_λ extended to multiple responses.

The computationally fast LOOCV formula can be written with the Predicted Resiudal Sum of Squares (PRESS) for each response $g \in \mathbb{R}^G$, where G is the number of classes, as such, following from the Sherman-Morrison Woodbury formula [41]:

$$PRESS(\lambda)_g = \sum_{i=1}^{N} \frac{(y_i - \hat{y}_{\lambda,i})^2}{(1 - h_{\lambda,i} - 1/N)^2} \tag{3.13}$$

where $\hat{y_{\lambda,i}}$ is the i-th fit of the fits $\hat{y_\lambda} = Xb_\lambda + b_{0,\lambda}$ and $h_{\lambda,i}$ is the i-th entry of h_λ. Which is defined as:

$$h_\lambda = (U_{(r,\lambda)} \odot U_{(r,\lambda)})1 \text{ ($\odot$ is the Hadamard product)} \tag{3.14}$$

Where $U_{(r,\lambda)}$ is obtained from the left singular vectors using the SVD:

$$U_\lambda = Z_\lambda V_r \Sigma_{(r,\lambda)}^{-1} = \begin{bmatrix} XV_r\Sigma_{(r,\lambda)}^{-1} \\ \sqrt{\lambda} I V_r \Sigma_{(r,\lambda)}^{-1} \end{bmatrix} = \begin{bmatrix} U_r \Sigma_r \Sigma_{(r,\lambda)}^{-1} \\ \sqrt{\lambda} V_r \Sigma_{(r,\lambda)}^{-1} \end{bmatrix} \tag{3.15}$$

$$= \begin{bmatrix} U_{(r,\lambda)} \\ \sqrt{\lambda} V_r \Sigma_{(r,\lambda)}^{-1} \end{bmatrix} \text{ where } U_{(r,\lambda)} \overset{\text{def}}{=} U_r \Sigma_r \Sigma_{(r,\lambda)}^{-1}. \tag{3.16}$$

$\Sigma_{(r,\lambda)}$ is a diagonal $r \times r$ matrix with entries $\sqrt{\sigma_i^2 + \lambda}$. Using the equations above allows us to efficiently evaluate any number of λ-values based on a single computation of SVD.

Our choice of the classifier is motivated by the computational efficiency and simplicity of the model. We wish to highlight the BPC framework's effectiveness in extracting good classification features rather than some more sophisticated choice of classifier.

3.1.5 Image Selection

The selection of a subset from the training images for the BPC feature extraction is a crucial step. Similar to how our mind needs examples to relate to things, some examples may be more characteristic than others. We seek an efficient method that can provide us with the the subset of training images from each class that can provide good discrimination between the groups in the predictive sense. Discrimination/generalization ability can be measured by comparing various image selection strategies and the number of kernels/features included to the classification performance. In other words, we aim to find the image selection method that leads to the highest possible classification performance, preferably based on as few kernels as possible. In doing so, we will also learn whether the selection of images matters or that it can be done more or less arbitrarily.

In the following, we propose four sample selection algorithms, *Random Supervised* (RS), *Clustered Random Supervised* (CRS), and *Clustered Kennard-Stone* (C-KS).

Random Supervised

Random Supervised (RS) sample selection is the simple act of dividing the data into groups of each class and randomly selecting p samples from each group. The intention is to sample images from each class such that the patches can act as detectors for each class. This algorithm is proposed to provide a baseline for comparison with other sample selection algorithms. The procedure is presented in Algorithm 2.

Algorithm 2 Random Supervised Sample Selection

```
Require: Samples $X \in \mathbb{R}^{N \times l}$    ▷ $N$ is the number of samples, $l$ is the number of features
Require: Labels $y \in \{1, \dots, G\}$    ▷ Where $G$ is the number of classes
Require: # of samples from each group $p$
  $X_{divided} \leftarrow \{X_{y=1}, \dots, X_{y=G}\}$    ▷ Divided samples
  $X_{selected} \leftarrow \{\}$
  for each $Z \in X_{divided}$ do
      for $j = 1$ to $p$ do
          $X_{selected} \leftarrow X_{selected} \,||\, Random(Z)$    ▷ Append random selection from divided samples
      end for
  end for
  return $X_{selected}$
```

Clustered Random Supervised

Clustered Random Supervised (CRS) sample selection follows the same idea as the RS selection method but has an additional clustering step. After dividing the samples into groups, we perform unsupervised clustering using K-Means to capture variance within groups. Furthermore, we randomly select p samples from each cluster within each label group. The motivation behind this is to gather samples similar to each other into clusters, such that when extracting samples, one gets a wider variety of selections. The CRS procedure is illustrated in Figure 3.4 and presented in Algorithm 3.

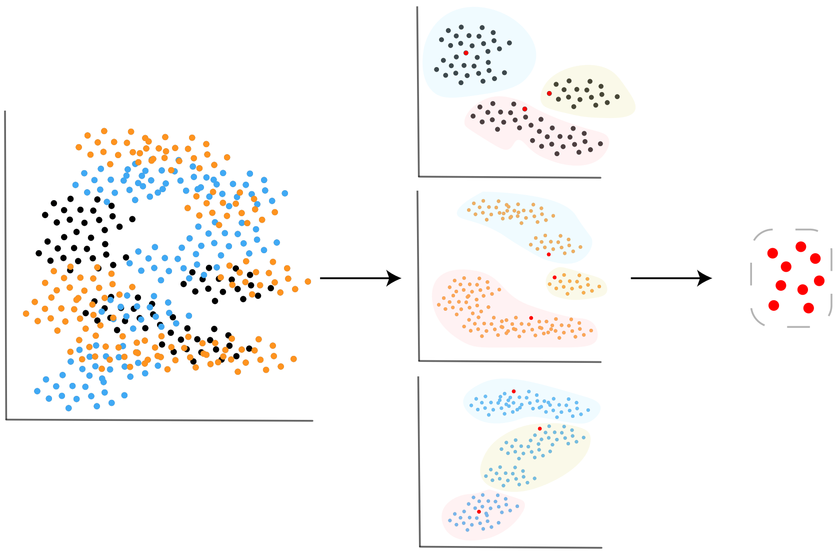


Figure 3.4: Illustration of the Clustered Random Supervised sample selection. The figure illustrates a three class supervised problem, where the samples are first divided into their respective label groups. Then they are divided into clusters, and samples are randomly selected (red points) from each cluster group in each label group.

K-Means clustering [42] , given a set of observations of size N, is done by initializing c cluster centers, otherwise known as centroids, μ. One then assigns each example from the observation to the nearest centroid using a distance metric. One then re-assigns the centroids to the center of all examples assigned to the cluster, i.e., the mean. This procedure is repeated until the samples are no longer assigned to new clusters, a criterion is exceeded, or the number of iterations exceeds the user-defined maximum.

The distance metric we use is the euclidean distance. The optimization problem for K-means is thus minimizing the sum of squared errors [10, p.309]:

$$SSE = \sum_{i=1}^{N}\sum_{j=1}^{c} w_{i,j}\|x_i - \mu_j\|_2^2, \tag{3.17}$$

where $w_{i,j}$ is the binary indicator of whether the sample x_i is in the cluster j.

The cluster centers' initializations are done by a technique known as K-Means++ [43]. The first centroid is drawn uniformly at random from the data. The next center is chosen from the data but with a weighted probability, $\frac{D(x)^2}{\sum_{x\in X} D(x)^2}$. Where $D(x)$ is the shortest distance from a data point to the closest center one has already chosen [43]. The second step is repeated until one has attained the number of clusters chosen by the user. After the initialization, the standard K-means algorithm resumes.

One can use the *elbow method* [44] to determine the number of optimal clusters. We can measure how well the chosen number of clusters divides the data by calculating the inertia. The inertia is the sum of the squared distances of samples to their closest centroid. By computing the inertia with an increasing amount of clusters and then plotting the values, one can look out for a sudden straightening out of the curve, an elbow. The number of clusters at the elbow provides a good indication of an optimal amount of clusters.

Algorithm 3 Clustered Random Supervised Sample Selection

```
Require: Samples $X \in \mathbb{R}^{N\times l}$          ▷ $N$ is the number of samples, $l$ is the number of features
Require: Labels $y \in \{1,\dots,G\}$                  ▷ Where $G$ is the number of classes
Require: # of samples from each cluster $p$
Require: # of clusters for each group $c$
  $X_{divided} \leftarrow \{X_{y=1},\dots,X_{y=G}\}$     ▷ Divided samples
  $X_{selected} \leftarrow \{\}$
  for each $Z \in X_{divided}$ do
      $Clusters \leftarrow$ K-Means$_c(Z)$               ▷ Attain $c$ clusters using K-Means
      for each $C \in Clusters$ do
          for $j = 1$ to $p$ do
              $X_{selected} \leftarrow X_{selected} \,||\, Random(C)$   ▷ Append random selection from current cluster
          end for
      end for
  end for
  return $X_{selected}$
```

Clustered Kennard Stone

Questioning CRS's random selection, due to the random nature and the probability of not collecting diverse samples within clusters, we suggest using a variant of the Kennard-Stone algorithm. The Kennard-Stone [45] sampling method uniformly selects samples from the data. The approach is performed by selecting two samples with the highest distance between themselves, according to a distance metric. These samples are added to a collection of selected samples and removed from the data. The remaining samples are chosen by finding the sample with the furthest distance to already selected samples, and this is repeated until the desired number of samples is chosen.

Algorithm 4 Clustered Kennard-Stone Sample Selection

```
Require: Samples $X \in \mathbb{R}^{N \times l}$    ▷ $N$ is the number of samples, $l$ is the number of features
Require: Labels $y \in \{1, \ldots, G\}$    ▷ Where $G$ is the number of classes
Require: # of samples from each cluster $p$
Require: # of clusters for each group $c$
  $X_{divided} \leftarrow \{X_{y=1}, \ldots, X_{y=G}\}$    ▷ Divided samples
  $X_{selected} \leftarrow \{\}$
  for each $Z \in X_{divided}$ do
      $Clusters \leftarrow$ K-Means$_c(Z)$    ▷ Attain $c$ clusters using K-Means
      for each $C \in Clusters$ do
          $X_{KS} \leftarrow \{\}$
          for $j = 1$ to $p$ do
              $X_{KS} \leftarrow X_{KS} \,||\,$ KS-V$(C)$    ▷ Append Kennard Stone Variant selection from current cluster
          end for
          $X_{selected} \leftarrow X_{selected} \,||\, X_{KS}$    ▷ Append the samples from current cluster to selected samples
      end for
  end for
  return $X_{selected}$
```

We propose a variant of the Kennard-Stone algorithm within clusters (KS-V), which differs only in the first step. First, select the sample farthest from the cluster center within each cluster. The motivation behind the different first step is that if one is only to select one sample from a cluster, the sample should be the most different from the rest of the samples in the cluster. All remaining samples are chosen in the same fashion as the original Kennard-Stone. An illustration for one cluster is displayed in Figure 3.5. The algorithm is presented in Algorithm 4.

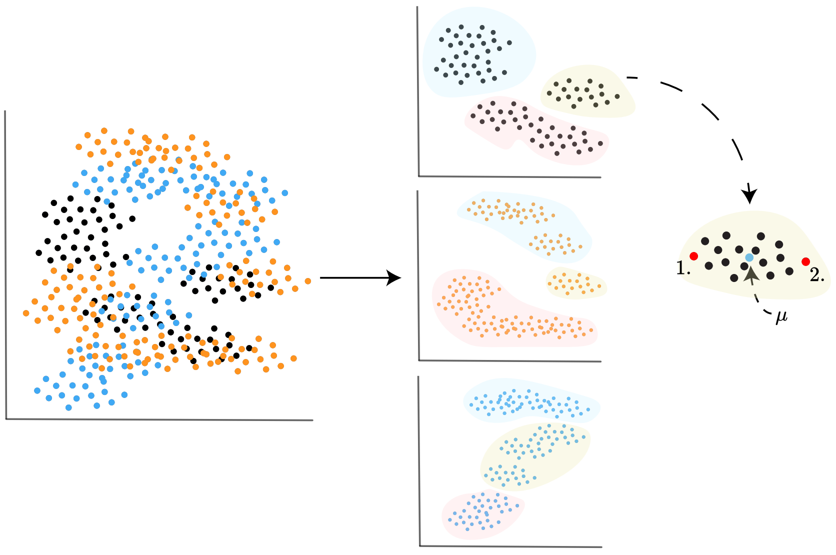


Figure 3.5: Illustration of the Clustered Kennard-Stone (C-KS) sample selection for one cluster. The Figure illustrates a three class supervised problem, where the samples are first divided into their respective label groups. Then they are divided into clusters. We illustrate the method for one cluster, where the two points farthest from the cluster center are selected.

Hu Moments

The previously mentioned methods for selecting samples are not translation, rotation, or scale invariant. Therefore we also want to examine whether the previously mentioned features are relevant for the image selection or whether one can reduce the feature space to a set of moments that can provide information about the images independent of position, size, or orientation.

One way to achieve this is using a classic computer vision technique known as image moments. Image moments are the weighted average of pixel intensities [46]. The raw two-dimensional moments for digital images are defined as such [46]:

$$M_{i,j} = \sum_x \sum_y x^i y^j I(x,y) \text{ where } I \text{ is the pixel intensity at position } x, y. \tag{3.18}$$

With the raw moments for a grayscale image, where the pixel intensity ranges from 0 to 255, weighting each point in the image gives us information about how "heavy" some positions are compared to others. An illustration of weighted images with different combinations of i and j are given in Figure 3.6. One can see that the zeroth order moment, $i, j = 0$, would simply be the area of the image. The first order moment, $i, j = 1$, would provide information about the center of the mass. The digit in Figure 3.6, is heavier at the bottom, and is thus more emphasized at the bottom when weighted with $i, j = 1$.

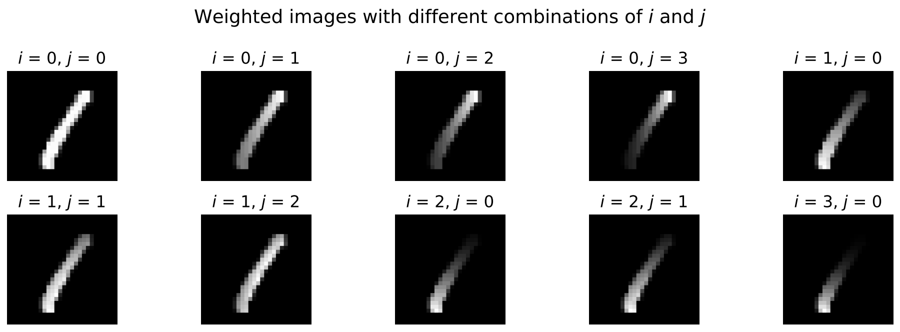


Figure 3.6: Illustration of a weighted digit from the MNIST dataset. The digit is weighted in similar fashion to Equation 3.18, without the summation. The combinations of i, j displayed are the ones used in the set of Hu moments.

The central moments of an image are translation invariant. They are identical to the raw moments but with the mean subtracted. The central moments are defined as such [46]:

$$\mu_{i,j} = \sum_x \sum_y (x - \bar{x})^i (y - \bar{y})^j I(x,y) \tag{3.19}$$

Where

$$\bar{x} = \frac{M_{10}}{M_{00}} \tag{3.20}$$

$$\bar{y} = \frac{M_{01}}{M_{00}} \tag{3.21}$$

To make the central moments invariant to scale, we can normalize them as such [46]:

$$\eta_{ij} = \frac{\mu_{i,j}}{\mu_{00}^{(i+j)/(2+1)}} \tag{3.22}$$

The set of moments proposed by Hu [47] that are invariant to translation, scale, and rotation are defined as follows:

$$I_1 = \eta_{20} + \eta_{02} \tag{3.23}$$

$$I_2 = (\eta_{20} - \eta_{02})^2 + 4\eta_{11}^2 \tag{3.24}$$

$$I_3 = (\eta_{30} - 3\eta_{12})^2 + (3\eta_{21} - \eta_{03})^2 \tag{3.25}$$

$$I_4 = (\eta_{30} + \eta_{12})^2 + (\eta_{21} + \eta_{03})^2 \tag{3.26}$$

$$\begin{aligned} I_5 = {} & (\eta_{30} - 3\eta_{12})(\eta_{30} + \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\right] \\ & + (3\eta_{21} - \eta_{03})(\eta_{21} + \eta_{03})\left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right] \end{aligned} \tag{3.27}$$

$$I_6 = (\eta_{20} - \eta_{02})\left[(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right] + 4\eta_{11}(\eta_{30} + \eta_{12})(\eta_{21} + \eta_{03}) \tag{3.28}$$

$$\begin{aligned} I_7 = {} & (3\eta_{21} - \eta_{03})(\eta_{30} + \eta_{12})\left[(\eta_{30} + \eta_{12})^2 - 3(\eta_{21} + \eta_{03})^2\right] \\ & - (\eta_{30} - 3\eta_{12})(\eta_{21} + \eta_{03})\left[3(\eta_{30} + \eta_{12})^2 - (\eta_{21} + \eta_{03})^2\right] \end{aligned} \tag{3.29}$$

We consider the use of Hu moments to select images for the BPC extraction. However, the moments vary in terms of range. I_0 is not comparable to I_6 [46]. To compensate for this, we use log transformation to make the moments more comparable [46]: $I_i = -sign(I_i)log|I_i|$. The resulting set of Hu moments can more safely be used as feature vectors similar to the original pixels in images. An illustration of the L_2 distance between two images when using the Hu moments as feature vectors is illustrated in Figure 3.7.

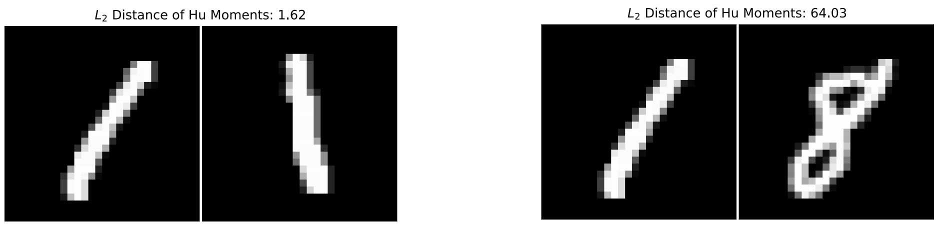

L_2 Distance of Hu Moments: 1.62
L_2 Distance of Hu Moments: 64.03

Figure 3.7: L_2 distances of Hu Moments for two pairs of images from different classes on the MNIST dataset [21].

3.1.6 Feature Map Evaluation

In the following, we propose methods for evaluating the information resulting from a convolution kernel by measuring the information contained in a feature map. This is relevant for the BPC architecture since we are using the idea of class-specific kernels. Our intention with the evaluation methods is not to provide a technique for improving classification accuracy, but rather to provide a sense of which relationships produce the highest amount of information. The purpose of such evaluation is also to enable elimination kernels from different classes to reduce complexity.

We consider Shannon entropy [48] as a measure of information for the feature maps. Shannon entropy quantifies the uncertainty in an information source. The idea is that a more uncertain event contains more information than a certain one. The Shannon entropy is defined in Equation 3.30[48]. Here L is the number of possible pixel values, and x_i denotes the i-th pixel value in a feature map. $P(x_i)$ is the probability of the pixel value in the information source. We use this measurement of information content to emphasize what convolution kernels produce the most information.

$$H(X) = -\sum_{i}^{L} P(x_i) log_2(P(x_i)). \tag{3.30}$$

Furthermore, we can utilize the SVD to measure the numerical rank of the feature maps. The lower the numerical rank, the less critical information is contained in the matrix since one can use fewer dimensions to approximate the real matrix without significant loss of information.

Since we are using kernels extracted from each class in a supervised problem in the BPC approach, we can measure the average information produced by convolution kernels on different targets. This may be useful in practical scenarios where one wishes to quantify the relationship between different classes or measure the amount of produced information by individual kernels.

3.1.7 Related Work

After performing an extensive literature search, we found a method seemingly related to the BPC approach called PCANet [49]. The PCANet also utilizes patches of images. However, it does not use binarization of the kernels. Initially, all the patches are subject to PCA, and the resulting principal components are used as convolution kernels. The authors also provide an alternative approach called LDANet replacing the PCA step with Linear Discriminant Analysis. After the convolution step, they binarize the feature maps and convert the binary numbers to decimals. Thereafter, the feature maps are partitioned into blocks, where a histogram is calculated from each block to be concatenated into a feature vector. These feature vectors are used for building a Support Vector Machine (SVM) classifier.

Our implementation differs from this by not including any linear combination of the original patches but using a subset of patches from each group. We also binarize the feature maps instead of mean-centering them to act as binary activations. Our dimensionality reduction of the feature maps is performed simply by max-pooling rather than binary hashing and histogram computations.

Another related paper by Dosovitskiy et al. [50] suggest using patches to make attention-based transformers computationally feasible for images. A transformer model is a neural network commonly used in natural language processing for modeling sequential data [51]. In the context of sentences, to use transformers, the sentences are embedded into continuous vectors, an additional positional encoding is added to the vector to get the contexts of the words in the sentences, then self-attention is performed. Self-attention computes the pairwise inner product between each feature in the vector to learn how relevant each word is to the other. Using self-attention-based transformers for images would require calculating the relationship between each possible pair of pixels. For an image with n pixels, this would result in $\binom{n}{2}$ computations, and since transformers require a tremendous amount of data to generalize well, this would be impractical to scale. Therefore Dosovitsky et al. [50] suggest performing self-attention between linear projections of patches instead of individual pixels, which allows for faster computation time, thus reducing the time complexity and making transformers feasible to use for images. The authors also state that since they operate on patch-level rather than pixel-level, the projections are more informative since a single pixel may contain less contextual information.

Another related idea using binary weights is called BinaryConnect [52]. The BinaryConnect uses binary weights to reduce the amount of multiply-accumulate operations and replace them with simple accumulations to streamline the training process and save space for the actual deployment of models [52]. The authors propose using binary weights during the forward and backward propagation stages. Using binary weights allows them to replace $\frac{2}{3}$ of the multiplications with addition or subtraction while achieving similar to state-of-the-art accuracy on image classification tasks [52].

3.2 Iterative Classification using SVD Bases

Eldén [8] proposed a classification method using the bases from SVD of feature matrices from each class. The intuition comes from the first singular vectors representing most of the variance inside each group of images. Thus one can compare an unknown image by approximating it in terms of associated singular basis expansions. If one of the singular bases in one group approximates the unknown sample well, one can associate the unknown sample with that particular group. We propose an extension of this classification method using iterative reinforcement as an attempt to capture a general trend in the errors.

3.2.1 Least Squares Classification

The residual between the approximation and the different bases can be presented in the least squares fashion as such [8]:

$$\min_{\alpha} \|z - U_k \alpha\|_2, \tag{3.31}$$

where U_k are the k left singular vectors, and z is the unknown sample. Solving the above Equation for α using the normal equations leads to the following:

$$\alpha = (U_k^T U_k)^{-1} U_k^T z. \tag{3.32}$$

Since $U_k^T U_k = I$, the equation can be simplified to

$$\alpha = U_k^T z, \tag{3.33}$$

and we obtain the residual as

$$\|(I - U_k U_k^T) z\|_2. \tag{3.34}$$

Like the Figure 2.11 showed for the least squares problem, we are projecting the unknown sample and measuring the reconstruction error to the original sample.

The choice of the number of singular basis vectors is analogous to restricting the capacity of a machine learning model. By using fewer vectors, one intends to capture the general features rather than the more specific ones. It is challenging to know beforehand how many singular vectors to include to obtain optimal classification performance. One way to select this is empirically through validation. As illustrated in Figure 3.8, visual inspection of the singular vectors is often a good indication.

3.2.2 Iterative Process

The algorithm proposed by Eldén [8] splits the training samples into groups according to their labels to separately compute the truncated SVD of each group and extract their left singular vectors. Furthermore, as shown in Equation 3.34, one computes the residual for each sample in the test data set with respect to projection onto the subspace spanned by left singular vectors of each group. Finally, one classifies the unknown samples according to the group resulting in the smallest residual.

We have considered an iterative extension of the algorithm. It starts as the original algorithm. However, after performing the first training step, one identifies the misclassified samples of each class in the training data set, and assigns them into a separate subgroup. In other words,

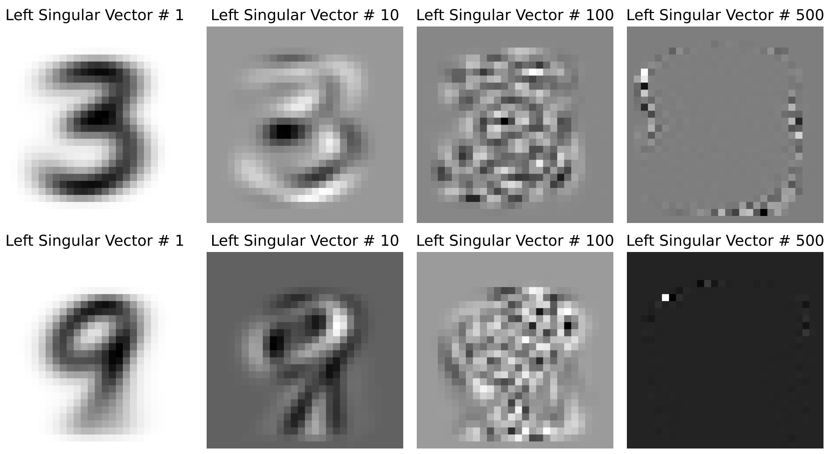

Left Singular Vector # 1
Left Singular Vector # 10
Left Singular Vector # 100
Left Singular Vector # 500
Left Singular Vector # 1
Left Singular Vector # 10
Left Singular Vector # 100
Left Singular Vector # 500

Figure 3.8: Display of left singular vectors for the digits 3 and 9 from the MNIST dataset. We can see that as we progress through the singular vectors more specific variation is captured, where the 500th singular vector looks like noise.

we suggest extending the original class labelling by assigning a new label to the misclassified samples according to which group they originated from. For example, for a problem with 10 labels, and assuming that there are misclassified samples from the first iteration in each group, we can extend the amount of left singular basis vectors:

$$\{U_{k_1}U_{k_2}\dots U_{k_{10}}\} \rightarrow \{U_{k_1}U_{k_2}\dots U_{k_{20}}\} \tag{3.35}$$

This procedure can be repeated as long as misclassified samples are in any group. After the final iteration, one can predict an unknown sample, which can be allocated to a broader number of basis vectors. One must keep track of which label group the misclassified samples were extracted from because when making a final prediction, one must convert the prediction back to the original set of labels.

The intention behind suggesting such an improvement is to capture the error trend by giving the method more granularity in extending the number of bases to which an unknown sample can be approximated for subsequent subgroup classification.

3.3 Within-Task Negative Transfer Minimization

The effects of information transfer between tasks is a well-studied problem in human cognition and are evident in tasks such as learning to drive a new car is easier when one has driven a wider variety of cars earlier, rather than just a single one [53]. Thus, solving a specific task can be supported by knowledge obtained from another domain of similar character. This is often referred to as a positive transfer of knowledge. On the other hand, if knowledge from the other domain negatively impacts the capability to solve a particular task, this is referred to as a negative transfer (of knowledge).

In machine learning, the problem has been studied in transfer learning tasks, where one wants to minimize the unfavorable effects of transferring skills and knowledge from one domain into skills and knowledge in another domain. Wang et al.[54] suggest using a discriminator gate to filter detrimental data from the source to the target by reducing the bias between risks in the source and target data. Deepmind's generalist agent Gato [55] is perhaps the most prominent example of a multitask artificial intelligence that has been trained to complete tasks on various domains including information transfer. It is well known that Gato has experienced negative transfer effects between some of the domains it is intended to handle.

We can therefore postulate that negative transfer within the a supervised learning problem is likely to occur across different pairwise combinations of the groups to be separated.

3.3.1 Within-Task Negative Transfer

Given a supervised learning problem with G labels, one usually considers the whole label set as a G-class classification problem. Depending on the task, learning information about all G labels may result in negative transfer effects. We propose that by reducing the original G classification problem into several smaller C problems, where $C < G$, one can mitigate the effect of the original negative transfer and perhaps even find a subset where a positive transfer effect is evident. The number of combinations can be expressed as such:

$$\binom{G}{C} = \frac{G!}{C!(G-C)!} \tag{3.36}$$

One way of verifying that negative or positive transfer exists is by constructing $\binom{G}{C}$ validation sets corresponding to each subset of the problem. Then one can train a single classifier on the training set consisting of samples corresponding to all labels and train $\binom{G}{C}$ classifiers on the subsets of the training set corresponding to all individual problems. Finally, one can evaluate the full model on all individual validation sets and all individual smaller models on their corresponding validation sets. If the full model performs better on average, there is evidence of positive transfer, and on the other hand, if the individual models perform better on average, there is evidence of negative transfer for the full model.

For instance, for a supervised problem with ten labels, one might consider restricting the problem into considering only the smaller problem of separating only between 5 of the the 10 classes for all possible choices of 5-class problems. This yields a total of $\binom{10}{5} = 252$ distinct 5-class classification problems. One potential way of minimizing the negative transfer is by training all these 252 different local models and performing a majority vote on the samples one wishes to classify. However, this approach introduces additional computational complexity to an already computationally extensive idea. Thus, we can consider a method using previously attained information, akin to a teacher-student architecture [56], utilizing the notion of top-k loss.

3.3.2 Top-k Accuracy Problem Reduction

Given a machine-learning model, one commonly measures the notion of generalization through the top-1 accuracy on the validation set. The top-k accuracy refers to the sample belonging to the top-k predictions. We suggest looking at the top-k accuracy where $k \geq 2$. If one experiences a higher top-k accuracy compared to the top-1 accuracy, one may use this information to improve upon the top-1 accuracy. In other words, if we can, with higher probability, predict that some sample belongs to a subset of the original problem of size k, and we are confident that negative transfer exists within the class labels, we may construct a two-stage classification process.

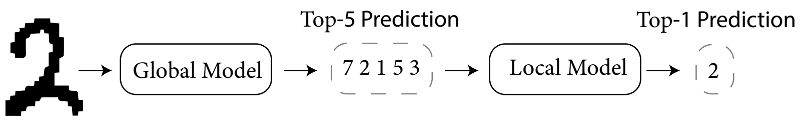


Figure 3.9: Illustration of a two-stage classification process using a global model outputting the top-5 accuracy which is passed on to a local model trained on the combination provided by the global model. The local model outputs the final prediction.

According to this idea, we start with a model for the complete G-class problem, and then we can examine the top-k accuracy to assign each sample to one of $\binom{G}{C}$ problems. One would then train models on the combinations and make top-1 predictions using local models, which we hypothesize minimizes both negative transfer and utilizes information from the top-k accuracy to reduce the loss between the top-k and top-1 accuracy. The two-stage classification process is illustrated in Figure 3.9 for a digit from the MNIST dataset, using the top-5 accuracy from the global model.

Chapter 4

Results

In this section, we will present the results of our proposed methods and provide some brief insights.

4.1 Binary Patch Convolution

We will now explore the BPC method's effectiveness in terms of classification accuracy. We will begin by exploring the techniques presented in Section 3.1.5 for sample selection and test their significance on image selection for patch extraction. Additionally, the framework will be compared against the state-of-the-art solutions and related work on the MNIST and F-MNIST datasets. The MNIST [21] is a well-studied dataset in the machine learning literature and is a standard benchmark for image classification models. The F-MNIST [57] dataset is a drop-in replacement for MNIST and offers a more complex challenge. More details about the datasets can be found in Appendix A: Datasets.

The BPC framework's practical implementation and the supplemental classifier are coded in Python and Julia. The convolutional process is designed to capitalize on the capacity of GPUs using PyTorch [58]; consequently, many convolution kernels can effectively be processed. Additional information about the practical implementation of the BPC framework and the classifier is provided in Appendix B: Practical Implementations and Appendix C: BPC Python Package.

4.1.1 Image Selection

To measure the generalization capability related to the number of images extracted from the sampling algorithms, we evaluate the BPC approach combined with a BOLS classifier on LOOCV samples from both the MNIST and F-MNIST datasets. We compare the accuracies of the different sampling algorithms with an increasing amount of convolution kernels used. We also compare the CV accuracy with the training set accuracy to reveal the degree of overfitting. The ideal image selection algorithm would produce the best result with the least complex model.

In addition to the features extracted through BPC, the original pixels were used as flattened vectors for the classification process. The observed effect of keeping the original pixels in the classification stage is marginal and can be omitted in favor of a narrower feature matrix.

For the sampling algorithms that utilize K-Means clustering as described in Section 3.1.5, the number of clusters was chosen by a quick observation of the elbow method for each label group. We used the same amount of clusters for each group. However, a different amount for each group could lead to better results.

The selection of patch sizes was chosen through cross-validation, and omitting the center patch had the most significant adverse effect on accuracy. The threshold for the binarization was chosen using Otsu binarization on the training images and extracting the mean of all the thresholds computed. Experiments showed that the classification accuracy is not particularly sensitive to the choice of binarization threshold.

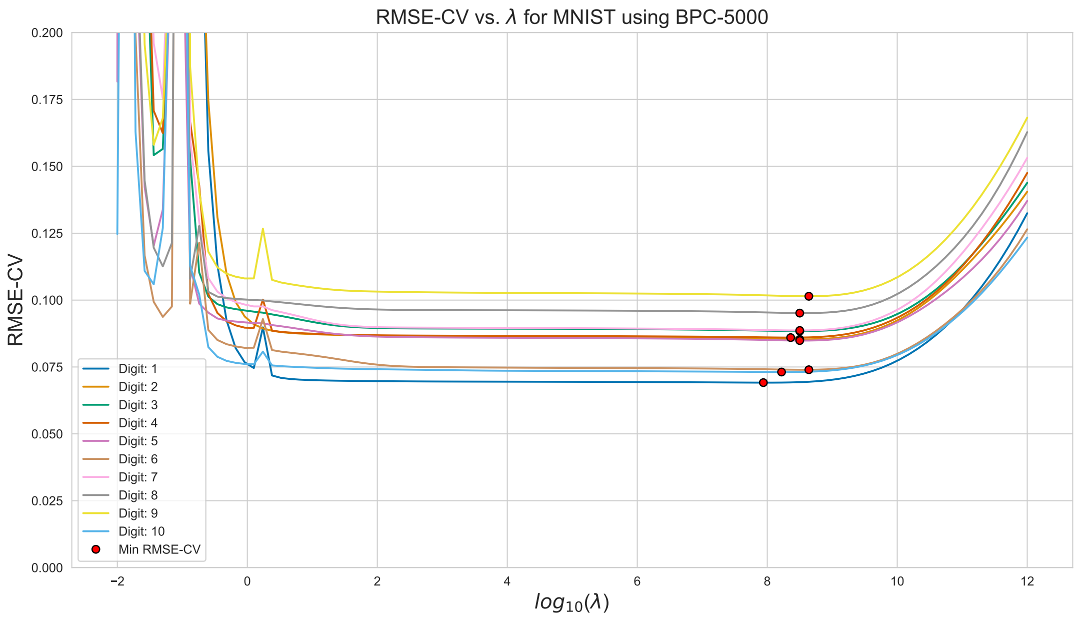


Figure 4.1: Illustration of the Root Mean Square Error for each regularization parameter λ in the interval $[10^{-2}, 10^{12}]$ from the LOOCV process. The LOOCV was performed on the MNIST dataset using 5000 features extracted by the BPC method.

For the classifier, we considered 101 regularization parameter values between 10^{-2} and 10^{12}. Figure 4.1 shows the Root Mean Square Error (RMSE) vs. the regularization parameter values from the LOOCV on the MNIST dataset using 5000 features extracted by the BPC method. The RMSE-CV for each group g out of all G groups is given as following:

$$\text{RMSE-CV}_g = \sqrt{\frac{PRESS(\lambda)_g}{N}} \tag{4.1}$$

Where N refers to the number of samples, and PRESS is defined in Equation 3.13. The regularization parameters λ for each digit group are chosen with respect to the minimum RMSE-CV value, as shown in Figure 4.1.

A summary of the parameter values used for evaluating the sample selection algorithms is shown in Table 4.1.

	MNIST	**F-MNIST**
Bin. threshold	95	84
# Of Clusters	5	5
# Of Patches	5	5
Patch Size	14×14	14×14

Table 4.1: Summary of the validation setup for the BPC approach for both the MNIST and F-MNIST dataset.

With each sampling algorithm for the MNIST and F-MNIST, the results from increasing the number of kernels are displayed in Tables 4.2 and 4.3, respectively. We can infer from the results that increasing the number of kernels carries the most impact on accuracy. Even with

Table 4.2: MNIST Train and LOOCV Accuracy comparison using different sample selection strategies. The 0 kernel model is the base BOLS using only the original pixels.

	# Kernels	0	500	1000	5000	10000	20000	25000	30000
RS	MNIST CV (%)	85.26	97.96	98.42	99.02	99.15	99.26	99.30	99.33
	MNIST Train (%)	85.72	98.12	98.66	99.43	99.67	99.81	99.84	99.85
CRS	MNIST CV (%)	85.26	98.05	98.43	99.05	99.22	99.31	99.31	99.33
	MNIST Train (%)	85.72	98.21	98.66	99.46	99.67	99.80	99.84	99.85
C-KS	MNIST CV (%)	85.26	98.04	98.43	99.15	99.25	99.34	99.34	99.39
	MNIST Train (%)	85.72	98.23	98.70	99.52	99.71	99.82	99.86	99.88

a few convolution kernels, we see a significant increase in accuracy for both datasets. When using a small number of clustering kernels, the choice of sampling algorithm is less influential. However, with the increase of kernels, we can see that the C-KS method performs the best. It is also interesting that RS performs similarly to the CRS method when the number of convolution kernels is large. The effects of pre-clustering at some point become minimal when using random sampling. However, with the C-KS method, we still maintain increased accuracy. We discuss additional sampling selection algorithms in Section 5.1.

Table 4.3: F-MNIST Train and LOOCV Accuracy comparison using different sample selection strategies. The 0 kernel model is the base BOLS using only the original pixels.

	# Kernels	0	500	1000	5000	10000	20000	25000	30000
RS	F-MNIST CV (%)	82.22	87.76	88.49	90.14	90.70	91.16	91.30	91.33
	F-MNIST Train (%)	82.79	88.44	89.49	92.13	93.26	94.27	94.73	95.03
CRS	F-MNIST CV (%)	82.22	87.64	88.34	90.16	90.65	91.24	91.30	91.34
	F-MNIST Train (%)	82.79	88.34	89.29	92.14	93.17	94.50	94.87	94.99
C-KS	F-MNIST CV (%)	82.22	87.64	88.74	90.42	90.92	91.37	91.53	91.59
	F-MNIST Train (%)	82.79	88.33	89.58	92.43	93.52	94.53	95.03	95.24

We also present the results of only using Hu moments as features for selecting images (as described in Section 3.1.5) using CRS and C-KS for the MNIST dataset in Table 4.4. We experienced that using only Hu moments with euclidean distance clustering leads to a similar accuracy for the MNIST data compared with the original features, especially for the CRS method. However, we saw that only using the Hu Moments with an increased amount of kernels with the C-KS method leads to slightly reduced accuracy. This indicates that when the amount of kernels is high using the C-KS method, one needs to capture features dependent on position, size, and orientation, to gain that extra accuracy boost. We experienced similar results for the F-MNIST dataset.

Table 4.4: Hu Moments MNIST Train and CV Accuracy

	# Kernels	500	1000	5000	10000	20000	25000	30000
CRS (Hu)	MNIST CV (%)	97.99	98.43	99.03	99.21	99.29	99.31	99.33
	MNIST Train (%)	98.19	98.69	99.43	99.67	99.80	99.82	99.86
C-KS (Hu)	MNIST CV (%)	98.06	98.43	99.06	99.20	99.29	99.31	99.33
	MNIST Train (%)	98.24	98.67	99.47	99.67	99.81	99.82	99.86

4.1.2 Test-set Comparison

After seeing that the C-KS method performs the best compared to the other sampling algorithms, we kept expanding the number of selected images for patch extraction, basically increasing the number of convolution kernels with the intention to halt when we stopped seeing a gain in the CV accuracy. We had to discontinue increasing the number of convolution kernels at one point, because we were limited by some challenges concerning computing the SVD of large matrices, which we elaborate on in section 5.1. Therefore, we ended up with models including 61700 and 61210 kernels for MNIST and F-MNIST, respectively. Using the same configuration as presented in Table 4.1, we evaluated the models we ended up with on the test sets of both MNIST and F-MNIST.

We limit the comparisons to methods not using data augmentation, as our approach does not utilize any augmentation strategy for the datasets. Data augmentation is a proven way of achieving even higher accuracy on both the MNIST and F-MNIST datasets [21] [57]; thus, we deemed it inaccurate to compare our framework against methods that utilize augmentation. However, future testing of the BPC framework's accuracy with artificial data could be interesting. The prediction results on the test sets from both datasets of size 10000 are presented in Table 4.5, and relevant and state-of-the-art solutions are compared. Table 4.5 also shows the CV accuracy based on the training dataset. This gives a nice illustration of the degree of agreement (in terms of estimated generalization ability) between the validation part used for model selection and the associated test set predictions.

Model	**MNIST**	**F-MNIST**
SVM [38]	98.57	89.7
Random Forest [59]	97.3	81.6
Basic CNN [59]	99.06	90.7
BinaryConnect [52]	98.99	-
PreActResNet-18 [60]	99.56	92.00
CTM [38]	99.33	91.18
PCANet [49]	99.38	-
BPC	99.50	91.47
BPC (CV)	99.43	91.97

Table 4.5: Accuracy comparison of BPC framework on test sets of MNIST and F-MNIST against relevant methods. The amount of convolution kernels are provided for BPC. The CV accuracy for BPC is provided as well.

The results indicate that the BPC framework is quite competitive for 2D gray-scale images. The BPC framework compares favorably against the other methods, even outperforming related work such as PCANet and CTM and outperforming a basic CNN quite significantly. However, it should be noted that PCANet uses 8 filters in the feature extraction stage. We discuss this in more detail in Section 5.1.

The BPC framework does not perform better than an 18-layer residual neural network but comes quite close regarding classification accuracy. The key to surpassing such a network might lie in increasing the convolution kernels to an even higher amount, since we are only using approximately 12% of the training datasets to extract patches. Or by extending the framework to several stages. We discuss our attempts at extending BPC to a two-stage process in Section 5.1.

4.1.3 Feature Map Evaluation

We proposed two alternative methods for evaluating the information content of a feature map in Section 3.1.6. We will display the results from 10 simulations of collecting convolution kernels from each digit group in the MNIST dataset and applying these kernels to randomly selected images from each digit group (including the digit group the filters were collected from). It was computationally infeasible to store a large number of feature maps. Thus we proceeded to run simulations by collecting random images instead. We label the digit group the kernels were collected from as the source and the group they were applied upon as targets. The result from collecting both the Shannon entropy and the numerical rank is displayed in Figure 4.2.

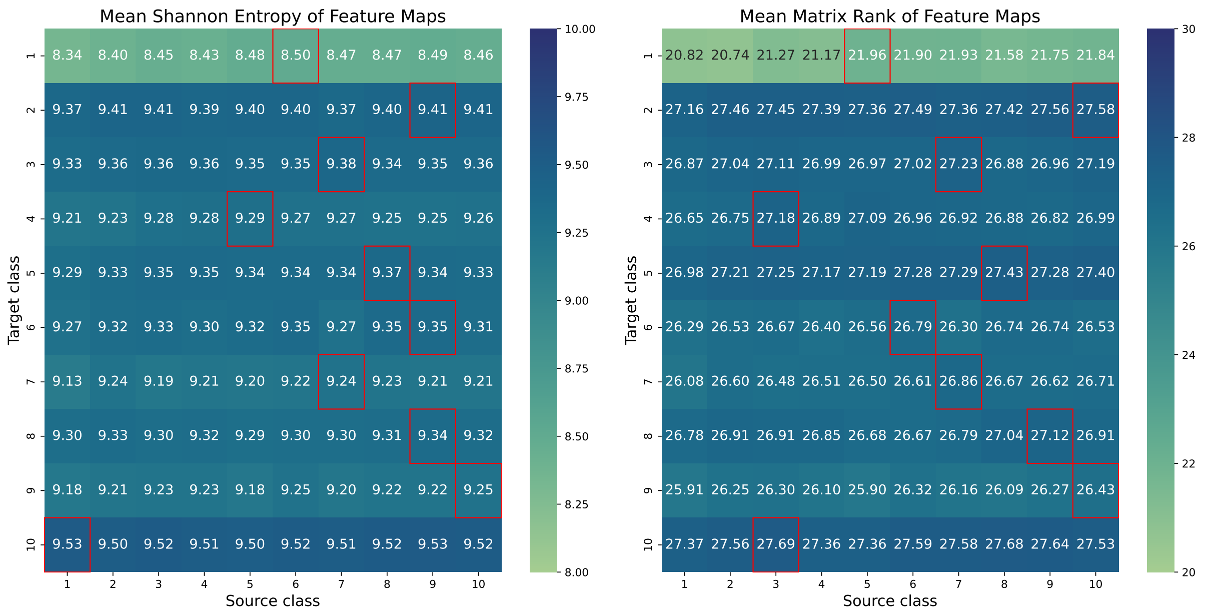


Figure 4.2: Heatmaps of information content contained in the feature maps produced by the BPC framework on the MNIST dataset. The boxes marked with red are the maximums for each target class.

The heatmaps created by measuring the information content indicate what combinations of sources and targets produce the most information. The measured information by either Shannon entropy or the numerical rank is similar. Generally, the information produced by having the digit group 1 as the target is less. This could be due to the smaller size of digit 1, meaning that the patches produced are more sparse than the other digits.

However, when the patches from digit 1 are used as the source, they resulted in the largest Shannon entropy when digit 0 (group 10) is the target, perhaps the detection of negative space is vital for the information content of digit 0. These measurements' effect on classification accuracy require a more thorough investigation outside the scope/time-frame of this thesis. (Such an investigation could be conducted using information theory for pruning convolution kernels, see our discussion on the topic in Section 5.1.3.)

4.2 Iterative Classification using SVD Bases

Here we present the results from the iterative singular value bases classification algorithm, as presented in Section 3.2, evaluated on the MNIST dataset [21]. The algorithm is coded in Python, mainly using libraries such as Numpy [61] for the mathematical calculations and Numba [62] for parallelization of the calculation of residuals. Additional details are provided in Appendix B: Practical Implementations.

4.2.1 Selection of Components

We begin by selecting the number of bases/components for the least squares classification. We construct a validation set and compute the accuracy with increasing bases. The results are shown in Figure 4.3.

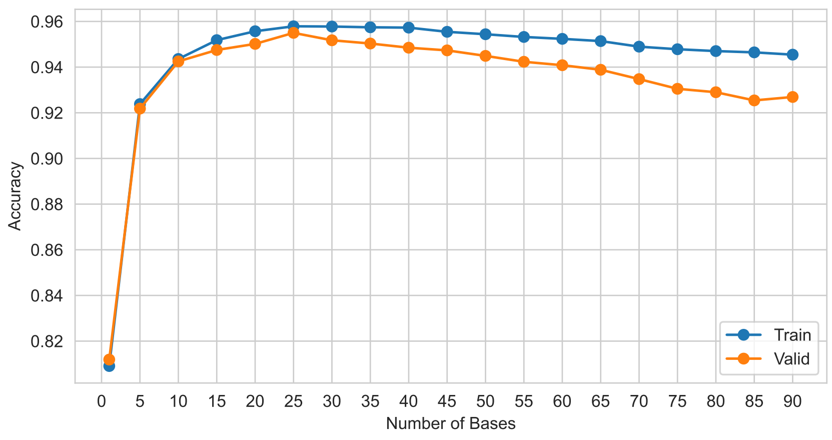


Figure 4.3: Plot of accuracies on training and validation data of the MNIST data, using classification by singular value bases, with an increasing amount of bases.

We see almost no overfitting before reaching 30 components, indicating that the optimal amount of components is 25. After 30 components, as seen in Figure 3.8, too specific variation is captured by the singular vectors that do not generalize well to the validation set. Thus for our testing with the iterative method, we chose to use 25 components.

We included three iterations of reinforcement, and the results are displayed in Figure 4.4. The reason for stopping at three iterations is because the training set had almost no more misclassified samples within the groups (hence, an additional iteration would be rather ineffective). We see that the classification algorithm quickly starts overfitting the training data, with just minor improvements for the validation dataset. In other words, the captured trend in the error of the training dataset does not generalize well to the validation data. The results indicate that more than capturing errors in the training set may be needed to improve the generalization capability. We discuss this aspect in more detail in 5.2. We also compared the predictions of iterative method for the test set to conclude that it gave only a minor improvement from the base algorithm suggested by Élden[8].

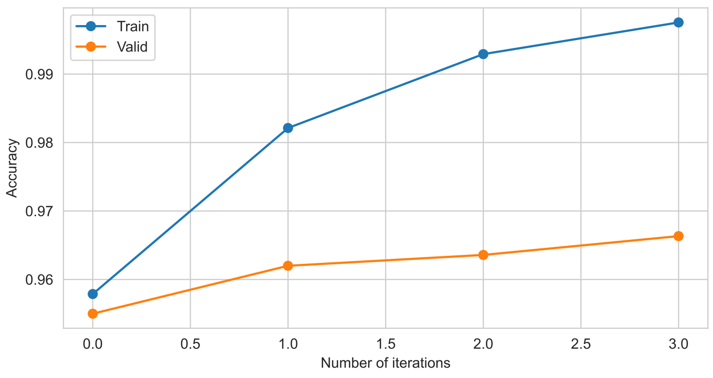


Figure 4.4: Illustration of the effect of iterative reinforcement steps on the singular value bases classification algorithm, evaluated on the MNIST dataset.

4.3 Within-Task Negative Transfer Minimization

We briefly described the concept of within-task negative transfer minimization in Section 3.3. The results we present here refer to classification models for the MNIST dataset [21]. We observed similar results on the F-MNIST dataset [57] as well. The algorithms are coded in both Python and Julia. We have used the BOLS classifier approach described in Section 3.1.4 to evaluate the negative transfer effect of original pixels using fast LOOCV. Furthermore, we also investigate the effect of negative transfer on the features extracted by the BPC idea. Finally, we conclude the effectiveness of the negative transfer minimization idea based on the test data predictions.

4.3.1 Detection of Negative Transfer

We begin by exploring the presence of negative transfer classicifation models using the original pixels of the MNIST dataset as features. We first split the original training data into new training- and validation sets. Thereafter, we divide the training and validation data into $\binom{10}{5} = 252$ subsets corresponding to every possible 5-class sub problem. The reason for choosing 5-class sub problems (compared to any other combination of g-class sub problems) was that we experienced significantly higher top-5 accuracy compared to the top-1 accuracy, therefore the 5-class sub problems can be combined with the top-k problem reduction technique presented in Section 3.3.2.

First, we train a global model on the complete training set (60000 samples), and all the local models associated with the subset partitions. Thereafter, we evaluate the global model on each of the 252 validation sets associated with the 5-class sub problems to compare to all the sub-problem validation results. The goal of this comparison is to explore if there is either a positive or negative transfer effect in the global model with respect to the sub-problem models. The results are shown in Figure 4.5.

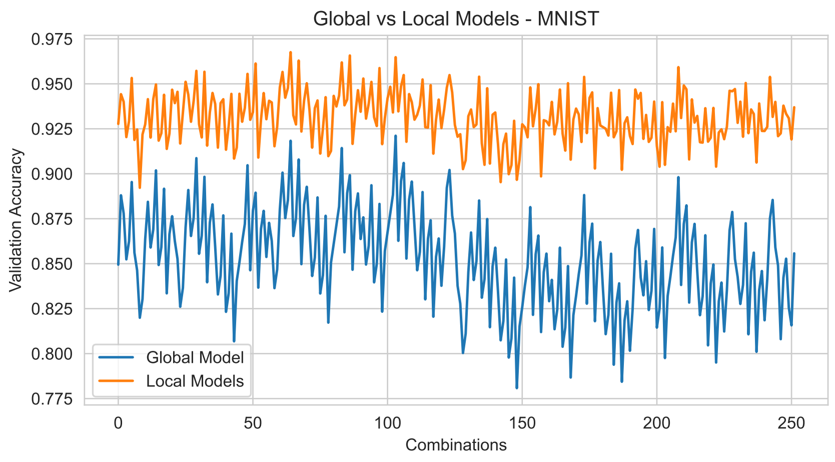


Figure 4.5: Comparison of accuracies between a global and local models on the 252 different problems on the MNIST dataset.

The results indicate that there is a negative transfer present within the class groups. The local models perform consistently better than the global model when validated on the same data. Another interesting observation is that the single global and the multiple local models tend to follow a similar pattern in validation accuracy, indicating that some sub-problems are inherently more difficult than others, which can be seen in Figure 4.7.

4.3.2 Combination with BPC

Here, we explore whether models based on the features extracted by the BPC has any effect on the negative transfer we saw for the modelling based on the original pixels as features. The below results are generated by using BPC with 5000 convolution kernels (as an increased amount would be infeasible due to the computational time needed for fitting 252 different models). For the local models, we only extract patches from images belonging to the sub-problem, i.e., for a sub-problem such as $\{2, 3, 4, 5, 7\}$, we do not select patches from the groups $\{1, 6, 8, 9, 10\}$. The results are presented in Figure 4.6.

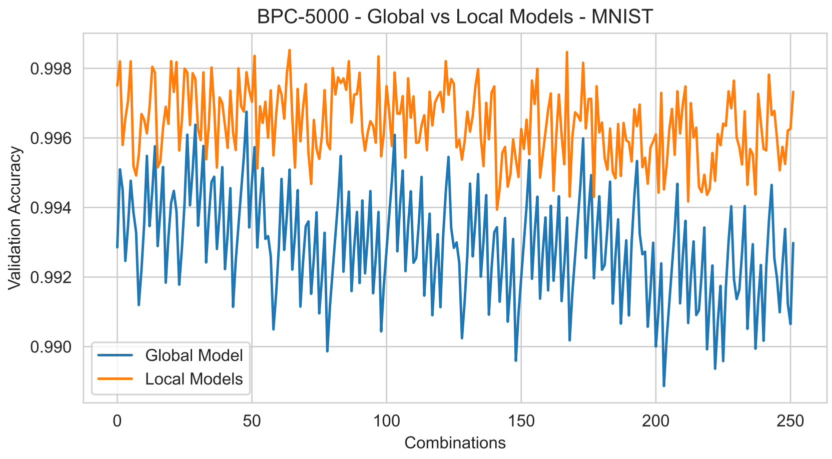


Figure 4.6: Comparison of accuracies between a global and local models on the 252 different problems on the MNIST dataset with features extracted using the BPC framework.

We again see that some (but much less than above) negative transfer of information is present, and it follows the same trend as the original problem (note the considerably higher accuracies obtained due to the BPC features). We can also see once again that some sub-problems are inherently more difficult as shown in Figure 4.7.

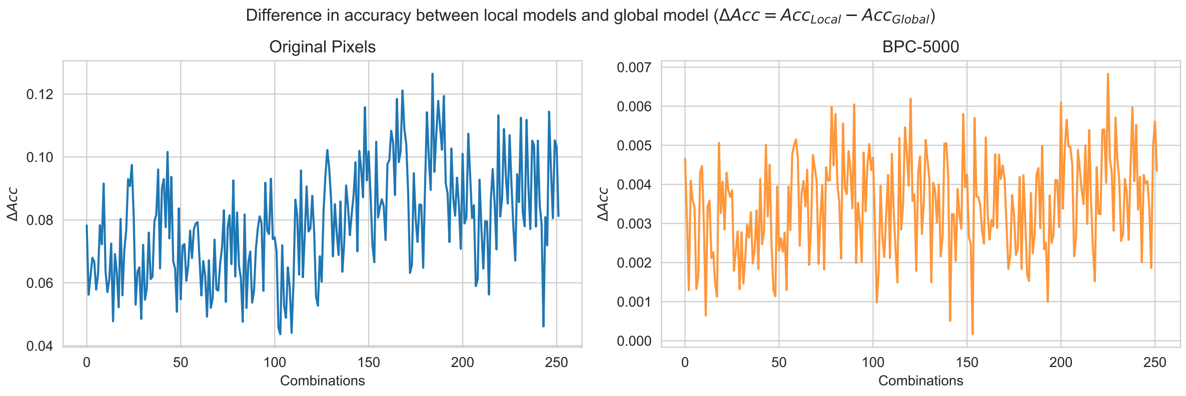


Figure 4.7: Differences in accuracies between local and globals models on the 252 different problems on the MNIST dataset. The plot shows the differences for the models used with the original pixels, and with the features extracted with BPC using 5000 convolution kernels.

4.3.3 Negative Transfer Minimization

We also checked whether the top-5 accuracy was higher than the top-1 accuracy obtained from cross-validation on the features extracted by the BPC method, as this was one of the conditions we stated for suggesting the top-k accuracy in two stage problem reduction as described in

3.3.2. We experienced that the top-5 accuracy was 99.9% in comparison to the top-1 accuracy of 99.15%. With both conditions satisfied, we used the top-5 prediction from a model trained on the whole dataset (global) to select the sub-problem/-model appropriate for making the final assignment of a sample as shown in Figure 3.9. The prediction made by the global model, determined which local model we would use to make a prediction for an unknown sample, leading to a two-stage classification process. We evaluated this method on the MNIST test set samples and compared it to a majority-voted version.

The majority-voted version refers to using all the 252 different local models to predict an unknown sample, then performing a majority vote on the predictions. By doing so one can determine what label to assign to an unknown sample. The majority-voted version introduced even more computational time to an already complex problem. However, we carried on to verify whether an assignment made by the top-5 prediction from a global model, then passed on to a local model corresponding to the top-5 prediction (namely the two-stage approach), led to better or worse results than the majority-voted version.

The results from using the two-stage top-5 problem reduction and the majority voting approach on the MNIST test set are displayed in Table 4.6. We have also included a global model, which refers to using a BOLS model trained on the whole dataset using BPC with 5000 convolution kernels for comparison.

Global Model	**Two-stage Local Models**	**Majority Vote Local Models**
99.13	99.31	99.40

Table 4.6: Accuracy (%) results from methods to minimize negative transfer on the MNIST test set with features extracted using BPC with 5000 kernels. The global model refers to the BOLS model trained on the whole dataset, the two-stage local models refer to using the top-5 accuracy prediction from a global model to make assignments to local models, and the majority vote local models refer to using 252 local models to make a prediction and performing a majority vote.

We can achieve better accuracy by using local models trained on every possible 5-class sub-problem than a global model. The majority-voted version performs the best, most likely due to the error made by the first stage in the top-5 problem reduction approach. The majority-voted version approach is more time-consuming than the two-stage approach since each local model must make a prediction on each unknown sample. On the other hand, the two-stage approach using the top-5 accuracy from a global model requires only a prediction to be made twice for each unknown sample. Therefore, the two-stage top-5 problem reduction offers a good tradeoff if time is of priority. Both approaches scale poorly when the original problem is large, and there are many classes. If we had considered a problem where $G = 20$, the number of subproblems would increase dramatically to $\binom{20}{5} = 15504$, rendering our approaches infeasible to use. Nonetheless, the results illustrate the effectiveness of being aware of the negative transfer of information within a global model, which motivates finding better methods to reduce the negative transfer.

Chapter 5

Discussion

In this section, we will discuss the proposed methods and the results obtained, the challenges faced, failed attempts, and pointers to future work. The discussion will be based on the motivations of this thesis and the insights obtained from the work performed.

5.1 The BPC approach

With the aspiration of achieving Convolutional Neural Network performance without explicitly using said networks, we searched for a method utilizing training image patches for doing convolutions. We based our intuition on CNNs' use of shared weights, they simply ask the same questions through convolution to each sample, and these questions produce corresponding responses from each sample to be classified. In other words, multiple characteristic responses are extracted from individual samples depending on their origin class. We have presented a method that alleviates the learning procedure of kernels in a CNN by simply extracting them from existing images. The motivation behind this is using the notion of relatable examples. We also saw that CNNs and other convolutional-based methods, such as CTM [38], create kernels that resemble fragments of the image subject to classification. Thus, we came to the idea that we could use patches of images as "questions" by applying them as convolution kernels. The fact that the BPC approach maintains a direct relationship between the convolution kernels and original images is probably an important part of why it seems to work so well.

A primary reasons for the BPC method's effectiveness is the binarization process. Performing convolution using non-binary patches leads to drastically reduced performance of the classification step. Alternatively, one can apply mean-centering such that the patches perform activations on the same intensity levels. However, binary patches are superior to the mean-centered patches. We think that with binarized patches, characteristic shapes in the images are matched better. In contrast, the shapes typically vary in intensity in the non-binarized patches (mean-centered patches included). Thus without binarization, the low intensity parts of the patch shapes may not be sufficiently influential in the convolution process causing an unwanted bias in the feature extraction. A similar effect will be caused by the most high intensity pixels.

Although the choice of binarization threshold was not very critical on the classification accuracy, Otsu's method provided much better results compared to simply using a threshold of 0. The choice of threshold values was more critical for the F-MNIST dataset, indicating that detection of more complex shapes benefits more from a carefully chosen binarization threshold. Another possible reason for the F-MNIST being more sensitive concerning the threshold is that the objects in the dataset are portrayed using a wider variety of pixel intensities, as compared to the MNIST dataset, which has a more economical use of pixel intensities. This can be seen in Figures 6.1 and 6.3.

Before observing the results from different sample selection algorithms with the BPC framework, we suspected that image selection for designing the binarized patches would be crucial, but our results indicate that quantity is more important than refined selection. However, when the number of selected images is low, it seems to be slightly better to use more sophisticated sampling algorithms than just randomly selecting some images.

As the number of selected images increases, we saw a diminishing effect of using complex sampling algorithms, but this was only true up to a certain number. After reaching a certain number of convolution kernels, one needs to capture more unique shapes to reduce the misclassifications. For this purpose, the C-KS method seemed effective, as it finds the most diverse samples within clusters compared to the CRS method. Similarly, we saw that by replacing the original pixels with the set of Hu moments in the sample selection process, odd shapes seem to be required for better classification.

We saw a similar effect using the Principal Variable Selection method (Prinvar) [63], which takes the PCA as a voting function to find original variables representing most of the data's variance. The Prinvar technique can be used to find images that account for most of the variance in different sample directions by considering each image as a flattened column vector. We experienced that the Prinvar technique produced patches of performance slightly poorer than patches from randomly selected images. This indicates that more is required than samples accounting for the main sample space variation to achieve patches resulting in the best classification accuracy. One needs to include atypical samples as they can contribute relevant pattern information that most samples do not contain. We also experimented by using the PCA on patches extracted from each group. Using the components generated by PCA from each label group as convolution kernels gave poorer results than the Prinvar technique. This may be due to the effect of PCA smoothing over some local details in the images important to differentiate between some groups.

We also tried using a reinforcement strategy to assist the sampling algorithms with choosing images to extract patches from. This was done by running multiple iterations of the BPC framework with a classifier and observing the misclassified images. These images would be chosen and extracted patches from to expand the initial filter bank. The reinforcement step was an effective way of improving classification accuracy on both the validation- and training set, as opposed to the similar iterative method for classification using SVD bases; this did not overfit the training data completely. However, it was only effective when the number of kernels was initially low and gave no significant improvements when the already chosen number of images was high.

The BPC approach for feature extraction was paired with a relatively weak BOLS (linear) classifier, with the motivation of doing fast model selection by the efficient LOOCV to emphasize the effectiveness of the BPC feature extraction. There is just a slight difference between the LOOCV and test-set accuracies for the MNIST dataset, indicating that the extracted features are efficient for classification of new samples (and not only suited for prediction on the training data). This is in contrast to the iterative singular value bases classification method, that overfits excessively on the training set. However, for the F-MNIST dataset, the difference between the LOOCV and test-set accuracy is more apparent, which could be due to the more complex nature of the problem (requiring more investigation in terms of regularizing the classifier).

The results also showed that the BPC framework is competitive, outperforming many comparable methods, which was one of the conditions we imposed in our research. Along with the binarization of the patches, the large number of convolution kernels contributes to the BPC's competitive ability. However, at one point, we faced constraints regarding the number of convolution kernels we could utilize. The matrices of features obtained from including many convolution kernels were dense and of corresponding high dimensions. When exceeding the size of 60000×62000, we experienced that computing the SVD of the feature data matrix became impossible with standard software.

5.1.1 Issues regarding the SVD-computations

One of the internal techniques used in most programming languages for computing the SVD is by divide and conquer, mainly through a library such as Intel MKL or OpenBlas that calls on the Fortran function *gesdd* [64]. Another technique for computing the SVD is QR iteration through the Fortran function *gesvd* [65]. The divide and conquer method is the most commonly used because it is significantly faster on large matrices but less precise in terms of numerical accuracy [66] compared to the QR iteration method. On the other hand, the QR iteration method is shown to be significantly slower for large matrices [66].

When we exceeded matrices of size 60000×62000, the results from computing the SVD using the divide and conquer method were entirely inaccurate, and the singular values became unreliable, with some being zeroed out in particular intervals. We believe that since our matrices are dense and of high numerical rank, the divide-and-conquer method could not produce numerically accurate enough results. We also tested out computing the SVD of our large matrices with the QR iteration method, where we were able to attain sensible values for the singular values. However, this method resulted in a dramatic increase of runtime, taking several days to of computations compared to a couple of hours for the divide and conquer algorithm.

To remedy this problem, we also considered the randomized SVD (RSVD) [67]. RSVD uses randomized algorithms to approximate matrix decompositions, which allows one to efficiently compute the SVD of matrices where the numerical rank is much smaller than the matrix dimensions. The RSVD requires an initial guess of the numerical rank of the matrix, as it computes the truncated SVD up to some chosen number of components. Through the RSVD, we were able to compute the truncated SVD up to 25000 components. However, when using the approximated truncated SVD, we experienced a slight loss in CV accuracy for the training data. When attempting to approximate the SVD with a larger number of components, we were limited by the memory capacity of the computer. We conclude that the RSVD is probably more suited for large sparse matrices with a small numerical rank. It is nonetheless an interesting alternative for future explorations based on hardware with more memory available.

Another approach to computing the SVD for large matrices that we looked into was the out-of-core solutions based on the MapReduce architectures [68]. These use a distributed approach to solve the SVD through QR iterations, where the intermediate arrays are stored on the disk rather than in the random access memory. However, these methods introduced a significant overhead. Due to the size of the arrays, most of the time was spent on copy-write actions on the disk. We could not compute the SVD using said methods, and we gave up when the runtime exceeded several days.

One could avoid the SVD problem through large-scale machine-learning models using distributed techniques, but these are beyond the scope of this thesis.

5.1.2 Issues regarding the feature map information content

We also provided two methods for measuring the information contained in the feature maps produced by the patches. Due to the direct relationship between the patches and original images, we can use this information to understand the relationship between different images and classes. In practical scenarios, such as medical tasks, this could potentially reveal certain relationships between images that would be difficult to observe through methods using CNNs. Additionally, this ability could also help in collecting data. Suppose one can see that certain images captured with a specific type of camera produce more information than the ones that do not. In that case, one could prioritize collecting data with said camera.

Our focus has not been on hyperparameter optimization for the BPC framework, and we have only considered a small variety of patch sizes and pooling methods. In our experience, smaller patch sizes compared to the ones we used, 14×14, harmed the classification accuracy, which is in contrast with modern CNNs where a common choice of kernel is 3×3 or 5×5 [33].

As one can observe from training CNNs, the convolution kernels learned resemble shapes, but are not entirely sharp in terms of detail, therefore they may require the use of smaller kernels to capture smaller finer details, compared to our method which uses original images that are sharp in detail. The reason for the BPC method not performing well with smaller patches, could be due to the minor patch sizes not capturing general sub-shapes but rather too specific details. In other words, a too small patch size may not contain enough contextual information to aid with feature extraction. However, this would require more experiments to verify. We also experimented with omitting the center patch, which was quite detrimental to the classification accuracy. Capturing some overlapping shapes seems to be essential when extracting features, which might also be due to some contextual information being contained in the overlapping shapes. Furthermore, we also experimented with replacing the max-pooling operation with an average-pooling, which negatively affected the classification accuracy. Capturing the average activations of the kernels is not the key, as it might smoothen out the most noteworthy activations that seem essential to capture. We also tested max-pooling with unit stride over smaller regions of the feature maps. However, this lead to more features being produced per patch, which restricts the amount of patches to be used as convolution kernels. Our experiments showed that it is more beneficial to use more patches where they produce a single feature per patch, compared to using fewer patches producing more features.

Furthermore, we tested out adding depth to the BPC framework by introducing stages. One would repeat the procedure on the feature maps produced by the first iteration of BPC. This is done by performing sample extraction among the feature maps produced and extracting binarized patches, which would be applied through convolution on the feature maps. We tested out a depth of 2 stages where we experienced a significant loss in accuracy, possibly due to being limited to a lower amount of convolution kernels in the first stage since the number of feature maps increases quite extensively. With a large number of kernels in the first stage, the second stage becomes infeasible to compute. This contrasts with PCANet [49], which can utilize several stages since the filters are linear combinations of patches, therefore they can use a lower amount of patches to obtain good accuracy in the first stage. However, their two-stage classification does not significantly improve accuracy for the tasks we have considered in this thesis.

5.1.3 Future Work

One direction to explore is to extend the filter bank produced by the binarized patches by performing Independent Component Analysis (ICA) on the patches. ICA [69] is a blind source separation technique. While PCA finds components that explain the variance of the data in different directions, the ICA finds independent components. It has been shown by Bell et al.[70] that the independent components of an ensemble of natural scenes are edge filters, and aiding the BPC framework with edge detectors could be vital for classification, as it has been shown that CNNs perform edge detections to some degree [31].

It has also been shown that using the numerical rank of the feature maps is an effective method for reducing the size of a convolutional neural network without a significant loss in classification accuracy [71]. This is done by removing the convolution kernels in a CNN where the feature maps contain low amount of information. A similar method could be utilized in our framework to remove filters from each class that do not produce meaningful information. One could build a metric based on the Shannon entropy and the numerical rank, compared to only using the numerical rank. This metric could be defined as such:

$$FI = \alpha M(x) + \zeta H(x) \tag{5.1}$$

Where M is the numerical rank of the feature map x, and α is the corresponding weight. H is the Shannon entropy of the feature map, and ζ the corresponding weight. The weights on the numerical rank and the Shannon entropy would have to be decided through empirical testing.

Another direction to explore is the effect of supplementing the training set with artificial data. As previously mentioned, data augmentation is a proven way of improving the accuracy of both MNIST and F-MNIST for CNNs. It would be interesting to see if that is also the case for the BPC framework. Since we are not explicitly learning filters from the data but extracting them from the images, we could directly measure the influence the patches from augmented data have on classification accuracy by looking at the feature importance in classification models, such as the regression coefficients in the BOLS classifier. By looking at the sign of the coefficients we could determine whether there was a positive or negative correlation between the patches. One could also use methods that generate synthetic data based on the metric 5.1, such that we could tailor kernels to produce the most information on complex cases.

We have not in this thesis considered feature-importance as a topic; however, the BPC framework's attractive property is that the convolution kernels have a straightforward relation to the original images. By considering the importance of a feature, we can learn which patches are most important for classification purposes. More complex approaches for determining which patches to use in the classification process can be based on feature importance. This could be performed with the classifier we used in this thesis, namely the BOLS classifier, by looking at the regression coefficients' magnitude. Other feature-importance selection tools, such as a tree-based feature selection method, might be worth exploring. A possible method for selecting patches based on feature importance is by running several iterations of randomly selected patches with the BOLS classifier, and extracting a certain amount of patches that correspond to a number of the largest regression coefficients.

Nevertheless, another direction to explore is with alternative convolution and pooling techniques. As previously mentioned, the convolution process is quite fast due to the utilization of GPU. However, convolution in the frequency domain could be a promising direction for the BPC framework. An image can be expressed in the frequency domain by the fast Fourier transform (FFT) [72], in other words, the Fourier transform allows one to view an image from an entirely different viewpoint [73, p. 9]. Convolution in the frequency domain offers a faster approach without significant accuracy loss [74]. The frequency domain convolution can be obtained from the *convolution theorem* [74]:

$$\mathcal{F}(x * w) = \mathcal{F}(x) \odot \mathcal{F}(w) \tag{5.2}$$

Where $*$ is convolution in the spatial domain (the regular domain we perform convolution in with the BPC method), x the given image, and w the convolution kernel. As we can see from the convolution theorem, in the frequency domain the convolution can be easily performed with the Hadamard product, allowing for a speedup by replacing the sliding window approach of the discrete convolution. The use of convolution in the frequency domain has not catched on to the mainstream, despite the fact that there have been papers such as by Mathieu et al. [75] that show that one can speed up the training process for CNNs by performing convolution in the frequency domain. We speculate that, one of the reasons for this is the added complexity. Since non-linearities such as the ReLU function cannot be perfomed in the frequency domain [75], one must compute the inverse fast Fourier transform per iteration of training of the feature maps, which introduces a significant overhead. Since the BPC framework requires no learning, one would only have to compute the Fourier transform through the fast Fourier transform once for the samples, and once for the convolution kernels, essentially getting rid of the overhead introduced if one were to use CNNs. Additionally, pooling in the frequency domain, denoted *spectral pooling* [74], is known to retain more spatial information while reducing the data size by the same amount as a max-pooling operation in the spatial domain, and might offer an improvement over our use of max-pooling.

Finally, another future work is to extend the BPC framework to more than grayscale images. We theorize that one would have to extract patches from each channel from selected images and correspondingly convolve them on the target images' channel from which they originate similar to CNNs for multi-channel data [33]. We suspect that for binarization to work, the binarization threshold would have to be determined for each channel. Due to the larger amount of channels, one will be limited in terms of how many convolution kernels one can use per channel, and it would be interesting to see if the BPC framework still performs well due to this restriction.

5.2 The Iterative Classification approach using SVD Bases

Our motivation for implementing iterative classification using the SVD bases was to offer the method more granularity and capture the trend in error. We saw that with each iteration, the training accuracy increased significantly. However, the validation accuracy was only increasing slightly. We speculate that we captured the training set error rather than a generalizable error trend.

We learned from this less successful approach that each image's error is different, and predicting a certain kind of error is challenging. We experienced the same challenge when trying a method including both a Bayesian Neural Network (BNN) and a Variational Autoencoder (VAE).

The latter is a neural network that is trained to copy its input to its output [13, p. 493]. The encoding part of the autoencoder can be seen as a compression step. Hence, the atoencoder learns a reduced representation of the original input, often referred to as the latent space, and the decoding part of the autoencoder learns to decompress the compressed input. A Variational Autoencoder includes additional regularizations making it easier to sample from the latent space to generate new representations.

Motivated by the BNN's ability to quantify uncertainty due to the distribution of weights in each layer rather than point estimates in contrast to NNs, we wanted to use this uncertainty in combination with VAE to generate a corrected version of the image. The idea was to produce a new representation of misclassified images with lower uncertainty using an auxiliary BNN when optimizing over the latent space of a VAE. After a literature search, we discovered a similar implementation by Antorán et al. [76]. This implementation was not used for classification purposes but rather for creating an explainable reconstruction. We were curious if reconstructed images would be easier to classify due to their lower uncertainty representations. However, we experienced that the reconstructions were actually harder to classify correctly. We also attempted to use the reconstructed images as supplemental features to the original pixels, which had no significant effect on the classification accuracy.

Finally, we experienced that capturing or correcting errors in the pixel domain did not work out as an efficient strategy. This is probably because each reconstructed image was different in its own way. There was no generalizable trend in the reconstruction of different image groups. One possible reason for this could be that we constantly perform extrapolation for new samples in a high-dimensional space such as images [77]. Thus the errors will be different from the ones in the training data. In contrast, the BPC-convolutions are effective because they extract features that seem general and characteristic for each class of a classification problem.

5.3 The Within-Task Negative Transfer Minimization approach

Pondering about the effects of negative transfer in human cognition and the notion of negative transfer in transfer learning in machine learning problems lead us to think about negative transfer within an original multiclass classification problem. We set out to find a way to reduce the amount of negative transfer in such problems, as well as a method for detecting the presence of negative transfer. We were able to observe an evident negative transfer in the datasets we were working with, and that locally trained models seemed to perform better through training models on sub-combinations of the original problem.

One drawback of the proposed method is that one cannot beforehand know the extent of subproblems leading to only a restricted amount of negative transfer. In this thesis, we tried reducing a 10-class problem to many 5-class problems. (Reducing the original problem to several 6-class problems might have yielded even better results.) Considering the two stage process as guidance can be useful, when a top-k accuracy is significantly higher than the top-1 accuracy. However, this does not ensure identification of the best subproblem and would have to be verified empirically.

The difference in accuracy between the majority voting and the two stage top-k accuracy problem reduction was more evident than we first realized. However, it seems sensible due to the error from top-k accuracy being passed on to the assignment of the subproblems. The majority voting, on the other hand, performs better, but as stated earlier, is a much slower approach. Top-k accuracy problem reduction is a better alternative if time is of the essence.

Our approach scales poorly when the number of original classes become large, and is impractical to deploy in real-world applications. Our inclusion is merely meant to provide an intuition about negative transfer and motivate more research. By providing such an approach, we wish to make practitioners in Machine Learning and AI more aware of the potential challenging structure of the original problem.

5.3.1 Future Work

Wang et al. [54] suggest that divergence between the joint distributions for the source, $P_s(Y|X_s)$, and the target task, $P_t(Y|X_t)$, in a transfer learning problem causes negative transfer. The authors propose a discriminator gate based on the Generative Adversarial Network (GAN) [78] and Domain-Adversarial Neural Network (DANN) [79] architectures. Their idea is to filter out the source data that does not contribute positively to the target task. In essence, they provide a mechanism for aligning distributions of features from two tasks. They train two separate feature extractors on source and target data. They train a discriminator and a classifier, where the discriminator predicts which domain the input belongs to, either the source or the target. The DANN finds a tradeoff between domain invariant features and aids the classifier through training [79].

However, the approach by Wang et al. [54] and the DANN implementation [79] only considers minimizing negative transfer when the source and target come from two different datasets. For future work, one could use such implementations within an original problem where there is no data coming from two datasets. This could be done in a one vs. all approach for each label to filter out the data that adversely affect the classification accuracy. Such an approach could also be combined with the BPC framework since the features extracted by BPC did not eliminate negative transfer, only reduce its effects.

To improve the two stage top-k problem minimization, one could utilize models tailored to reduce the top-k loss in the first stage, such as a top-k multiclass SVM [80]. Furthermore, to avoid having local models in the second stage, one could have an additional global model that can model the joint distribution of $P(Y|X, \hat{Y}_k)$, where $\hat{Y}_k$ are the top-k predictions. Essentially a secondary classifier that can use the top-k predictions as additional features. This approach could be combined with the BPC framework to attain an even higher top-k classification accuracy resulting in a reduced error transfer in a multi-stage classification pipeline.

Chapter 6

Conclusions

The objective of this thesis was to explore the possibilities of attaining CNN quality performance without using CNNs. We took inspiration from convolution-based models and experimented with notions inspired by human intelligence, such as the concept of reinforcement and negative transfer.

The most important results are obtained from the Binary Patch Convolution (BPC) approach, using binarized patches from existing training images as convolution kernels. The framework presented does not require any learning of the convolution kernels besides the binarization. We have included unsupervised learning techniques to assist in the sample selection for images to be used for patch extraction.

Our results indicate that with the BPC approach we have managed to obtain a competitive method that maintains a direct connection to the original images while obtaining classification accuracy on 2D grayscale images as compared to state-of-the-art CNN models. We have also provided methods for measuring information content, which may be helpful in practical scenarios where inferring information about the convolution kernels' relations to discrimination between different groups is desired.

Furthermore, we have also explored methods attempting to capitalize on prediction errors using reinforcement without notable success. The attempts have provided us with useful insights and an enhanced basis for intuition on how to approach solving problems of high dimensions, as well as illustrating that capturing a generalizable trend in error is complicated.

Moreover, we have analyzed the concept of negative transfer within original tasks and how one could proceed to minimize the effect of negative transfer. Our proposed approaches are functional but infeasible in terms of computation time for practical scenarios. However, they provide us with valuable insight, namely that the concept of negative transfer should extend beyond transfer learning. We have shown that negative transfer can be inherent in the original task and should be focused on.

We have also discussed the limitations and challenges of our proposed approaches to emphasize the constraints and motivate more research to come up with solutions. Ultimately, we have provided justified ideas for future work pointing out directions to improve and develop the BPC into a framework that could merge with the conceptual ideas of negative transfer minimization.

Finally, we believe that the main ideas of this thesis has showcased how taking inspiration from CNN modelling to alleviate their use can lead to competitive methods in terms of classification accuracy, and using concepts from human cognition can be valuable, especially in providing intuition and insight.

Bibliography

[1] Marvin Minsky. *The Society of Mind.* 1st ed. USA: Simon and Schuster, Inc., 1986. ISBN: 978-0671657130.

[2] Hao Dong et al. “Automatic Brain Tumor Detection and Segmentation Using U-Net Based Fully Convolutional Networks”. In: *ArXiv* abs/1705.03820 (2017).

[3] Yann LeCun et al. “Object Recognition with Gradient-Based Learning”. In: *Shape, Contour and Grouping in Computer Vision.* 1999.

[4] Computer Vision Machine Learning Team. *An On-device Deep Neural Network for Face Detection.* Nov. 2017. URL: `https://machinelearning.apple.com/research/face-detection`.

[5] Chenyi Chen et al. “DeepDriving: Learning Affordance for Direct Perception in Autonomous Driving”. In: *2015 IEEE International Conference on Computer Vision (ICCV)* (2015), pp. 2722–2730.

[6] Randall Balestriero and Richard Baraniuk. “A Spline Theory of Deep Learning”. In: *Proceedings of the 35th International Conference on Machine Learning.* Ed. by Jennifer Dy and Andreas Krause. Vol. 80. Proceedings of Machine Learning Research. PMLR, July 2018, pp. 374–383. URL: `https://proceedings.mlr.press/v80/balestriero18b.html`.

[7] Kwan Ho Ryan Chan et al. “ReduNet: A White-box Deep Network from the Principle of Maximizing Rate Reduction”. In: *CoRR* abs/2105.10446 (2021). arXiv: `2105.10446`. URL: `https://arxiv.org/abs/2105.10446`.

[8] Lars Eldén. *Matrix Methods in Data Mining and Pattern Recognition.* Society for Industrial and Applied Mathematics, 2007. DOI: `10.1137/1.9780898718867`. eprint: `https://epubs.siam.org/doi/pdf/10.1137/1.9780898718867`. URL: `https://epubs.siam.org/doi/abs/10.1137/1.9780898718867`.

[9] T.M. Mitchell. *Machine Learning.* McGraw-Hill International Editions. McGraw-Hill, 1997. ISBN: 9780071154673. URL: `https://books.google.no/books?id=EoYBngEACAAJ`.

[10] Sebastian Raschka and Vahid Mirjalili. *Python Machine Learning, 3rd Ed.* 3rd ed. Birmingham, UK: Packt Publishing, 2019. ISBN: 978-1789955750.

[11] Jianfeng Wang and Thomas Lukasiewicz. *Rethinking Bayesian Deep Learning Methods for Semi-Supervised Volumetric Medical Image Segmentation.* 2022. DOI: `10.48550/ARXIV.2206.09293`. URL: `https://arxiv.org/abs/2206.09293`.

[12] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* Bradford Books, 2018. ISBN: 978-0262039246.

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

[14] Zhou Wang and Alan Conrad Bovik. “A universal image quality index”. In: *IEEE Signal Processing Letters* 9 (2002), pp. 81–84.

[15] A. Cauchy. *Méthode générale pour la résolution des systèmes d’équations simultanées.* 1847.

[16] Sebastian Ruder. *An overview of gradient descent optimization algorithms.* 2016. DOI: `10.48550/ARXIV.1609.04747`. URL: `https://arxiv.org/abs/1609.04747`.

[17] Gilbert Strang. *Linear Algebra and Learning from Data.* Wellesley-Cambridge Press, 2019. ISBN: 978-06921963-8-0.

[18] Carl Eckart and G. Marion Young. “The approximation of one matrix by another of lower rank”. In: *Psychometrika* 1 (1936), pp. 211–218.

[19] George P. McCabe. “Principal Variables”. In: *Technometrics* 26.2 (1984), pp. 137–144. ISSN: 00401706. URL: `http://www.jstor.org/stable/1268108`.

[20] Stephen Boyd and Lieven Vandenberghe. *Introduction to Applied Linear Algebra: Vectors, Matrices, and Least Squares.* Cambridge University Press, 2018. ISBN: 978-1316518960.

[21] Yann LeCun and Corinna Cortes. *The mnist database of handwritten digits.* URL: `http://yann.lecun.com/exdb/mnist/` (visited on 01/05/2022).

[22] Warren S. McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *Bulletin of Mathematical Biology* 52 (1990), pp. 99–115.

[23] Frank Rosenblatt. “The perceptron: a probabilistic model for information storage and organization in the brain.” In: *Psychological review* 65 6 (1958), pp. 386–408.

[24] C.M. Bishop et al. *Neural Networks for Pattern Recognition.* Advanced Texts in Econometrics. Clarendon Press, 1995. ISBN: 9780198538646. URL: `https://books.google.no/books?id=T0S0BgAAQBAJ`.

[25] Bing Xu et al. “Empirical Evaluation of Rectified Activations in Convolutional Network”. In: *ArXiv* abs/1505.00853 (2015).

[26] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. “Learning representations by back-propagating errors”. In: *Nature* 323 (1986), pp. 533–536.

[27] Grant Sanderson. *Backpropagation Calculus.* Nov. 2017. URL: `https://www.3blue1brown.com/lessons/backpropagation-calculus`.

[28] Léon Bottou. “Large-Scale Machine Learning with Stochastic Gradient Descent”. In: *COMPSTAT.* 2010.

[29] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* 2014. DOI: `10.48550/ARXIV.1412.6980`. URL: `https://arxiv.org/abs/1412.6980`.

[30] Kunihiko Fukushima. “Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position”. In: *Biological Cybernetics* 36 (2004), pp. 193–202.

[31] Yann LeCun, Y. Bengio, and Geoffrey Hinton. “Deep Learning”. In: *Nature* 521 (May 2015), pp. 436–44. DOI: `10.1038/nature14539`.

[32] Y. LeCun et al. “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Computation* 1.4 (1989), pp. 541–551. DOI: `10.1162/neco.1989.1.4.541`.

[33] Olga Russakovsky et al. “ImageNet Large Scale Visual Recognition Challenge”. In: *International Journal of Computer Vision* 115 (2015), pp. 211–252.

[34] Dhruv Mahajan et al. *Exploring the Limits of Weakly Supervised Pretraining.* 2018. DOI: `10.48550/ARXIV.1805.00932`. URL: `https://arxiv.org/abs/1805.00932`.

[35] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *AISTATS*. 2010.

[36] Kaiming He et al. *Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification.* 2015. DOI: `10.48550/ARXIV.1502.01852`. URL: `https://arxiv.org/abs/1502.01852`.

[37] Yi-Tong Zhou and Rama Chellappa. “Computation of optical flow using a neural network”. In: *IEEE 1988 International Conference on Neural Networks* (1988), 71–78 vol.2.

[38] Ole-Christoffer Granmo et al. “The Convolutional Tsetlin Machine”. In: *arXiv preprint arXiv:1905.09688* (2019).

[39] Nobuyuki Otsu. “A threshold selection method from gray level histograms”. In: *IEEE Transactions on Systems, Man, and Cybernetics* 9 (1979), pp. 62–66.

[40] Andrey Nikolayevich Tikhonov. *On the solution of ill-posed problems and the method of regularization.* 1963.

[41] Joakim Skogholt. *Efficient model selection in the Tikhonov Regularization framework and pre-processing of spectroscopic data.* Ås, 2019.

[42] J. MacQueen. “Some methods for classification and analysis of multivariate observations”. In: 1967.

[43] David Arthur and Sergei Vassilvitskii. “k-means++: the advantages of careful seeding”. In: *SODA ’07*. 2007.

[44] Robert L. Thorndike. “Who belongs in the family?” In: *Psychometrika* 18 (1953), pp. 267–276.

[45] R. W. Kennard and L. A. Stone. “Computer Aided Design of Experiments”. In: *Technometrics* 11.1 (1969), pp. 137–148. DOI: `10.1080/00401706.1969.10490666`. eprint: `https://www.tandfonline.com/doi/pdf/10.1080/00401706.1969.10490666`. URL: `https://www.tandfonline.com/doi/abs/10.1080/00401706.1969.10490666`.

[46] Krutika Bapat Satya Mallick. *Shape Matching using Hu Moments (C++/Python.* Dec. 2018. URL: `https://learnopencv.com/shape-matching-using-hu-moments-c-python/`.

[47] Ming-Kuei Hu. “Visual pattern recognition by moment invariants”. In: *IRE Trans. Inf. Theory* 8 (1962), pp. 179–187.

[48] Claude E. Shannon. “A mathematical theory of communication”. In: *Bell Syst. Tech. J.* 27 (2001), pp. 623–656.

[49] Tsung-Han Chan et al. “PCANet: A Simple Deep Learning Baseline for Image Classification?” In: *IEEE Transactions on Image Processing* 24.12 (Dec. 2015), pp. 5017–5032. DOI: `10.1109/tip.2015.2475625`. URL: `https://doi.org/10.1109`.

[50] Alexey Dosovitskiy et al. *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.* 2020. DOI: `10.48550/ARXIV.2010.11929`. URL: `https://arxiv.org/abs/2010.11929`.

[51] Ashish Vaswani et al. *Attention Is All You Need.* 2017. DOI: `10.48550/ARXIV.1706.03762`. URL: `https://arxiv.org/abs/1706.03762`.

[52] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. *BinaryConnect: Training Deep Neural Networks with binary weights during propagations.* 2015. DOI: `10.48550/ARXIV.1511.00363`. URL: `https://arxiv.org/abs/1511.00363`.

[53] Thomas J. Shuell. “Cognitive Conceptions of Learning”. In: *Review of Educational Research* 56.4 (1986), pp. 411–436. ISSN: 00346543, 19351046. URL: `http://www.jstor.org/stable/1170340` (visited on 11/03/2022).

[54] Zirui Wang et al. *Characterizing and Avoiding Negative Transfer*. 2018. DOI: 10.48550/ARXIV.1811.09751. URL: https://arxiv.org/abs/1811.09751.

[55] Scott Reed et al. *A Generalist Agent*. 2022. DOI: 10.48550/ARXIV.2205.06175. URL: https://arxiv.org/abs/2205.06175.

[56] Qizhe Xie et al. "Self-Training With Noisy Student Improves ImageNet Classification". In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)* (2020), pp. 10684–10695.

[57] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. Aug. 28, 2017. arXiv: cs.LG/1708.07747 [cs.LG].

[58] Adam Paszke et al. "Automatic differentiation in PyTorch". In: (2017).

[59] Yitao Liang and Guy Van den Broeck. "Learning Logistic Circuits". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 33.01 (July 2019), pp. 4277–4286. DOI: 10.1609/aaai.v33i01.33014277. URL: https://ojs.aaai.org/index.php/AAAI/article/view/4336.

[60] Tarin Clanuwat et al. "Deep Learning for Classical Japanese Literature". In: *ArXiv* abs/1812.01718 (2018).

[61] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[62] Siu Kwan Lam, Antoine Pitrou, and Stanley Seibert. "Numba: A LLVM-Based Python JIT Compiler". In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. LLVM '15. Austin, Texas: Association for Computing Machinery, 2015. ISBN: 9781450340052. DOI: 10.1145/2833157.2833162. URL: https://doi.org/10.1145/2833157.2833162.

[63] Ulf Geir Indahl. *Principal Variable Selection, Lecture 9*. 2021.

[64] *Developer Reference - gesdd*. URL: https://www.intel.com/content/www/us/en/develop/documentation/onemkl-developer-reference-fortran/top/lapack-routines/lapack-least-squares-and-eigenvalue-problem/lapack-least-squares-eigenvalue-problem-driver/singular-value-decomposition-lapack-driver/gesdd.html.

[65] *Developer Reference - gesvd*. URL: https://www.intel.com/content/www/us/en/develop/documentation/onemkl-developer-reference-c/top/lapack-routines/lapack-least-squares-and-eigenvalue-problem/lapack-least-squares-eigenvalue-problem-driver/singular-value-decomposition-lapack-driver/gesvd.html.

[66] James W. Demmel. *Applied Numerical Linear Algebra*. Society for Industrial and Applied Mathematics, 1997. DOI: 10.1137/1.9781611971446. eprint: https://epubs.siam.org/doi/pdf/10.1137/1.9781611971446. URL: https://epubs.siam.org/doi/abs/10.1137/1.9781611971446.

[67] N. Halko, P. G. Martinsson, and J. A. Tropp. "Finding Structure with Randomness: Probabilistic Algorithms for Constructing Approximate Matrix Decompositions". In: *SIAM Review* 53.2 (2011), pp. 217–288. DOI: 10.1137/090771806. eprint: https://doi.org/10.1137/090771806. URL: https://doi.org/10.1137/090771806.

[68] Dask Development Team. *Dask: Library for dynamic task scheduling*. 2016. URL: https://dask.org.

[69] Seungjin Choi. "Independent Component Analysis". In: *Handbook of Natural Computing*. 2009.

[70] Anthony J. Bell and Terrence J. Sejnowski. “The “independent components” of natural scenes are edge filters”. In: *Vision Research* 37.23 (1997), pp. 3327–3338. ISSN: 0042-6989. DOI: https://doi.org/10.1016/S0042-6989(97)00121-1. URL: https://www.sciencedirect.com/science/article/pii/S0042698997001211.

[71] Mingbao Lin et al. *HRank: Filter Pruning using High-Rank Feature Map.* 2020. DOI: 10.48550/ARXIV.2002.10179. URL: https://arxiv.org/abs/2002.10179.

[72] James W. Cooley and John W. Tukey. “An algorithm for the machine calculation of complex Fourier series”. In: *Mathematics of Computation* 19 (1965), pp. 297–301.

[73] E. Oran Brigham. *Fast Fourier Transform and Its Applications.* Pearson, 1988. ISBN: 978-0133075052.

[74] Oren Rippel, Jasper Snoek, and Ryan P. Adams. *Spectral Representations for Convolutional Neural Networks.* 2015. DOI: 10.48550/ARXIV.1506.03767. URL: https://arxiv.org/abs/1506.03767.

[75] Michael Mathieu, Mikael Henaff, and Yann LeCun. *Fast Training of Convolutional Networks through FFTs.* 2013. DOI: 10.48550/ARXIV.1312.5851. URL: https://arxiv.org/abs/1312.5851.

[76] Javier Antorán et al. *Getting a CLUE: A Method for Explaining Uncertainty Estimates.* 2020. DOI: 10.48550/ARXIV.2006.06848. URL: https://arxiv.org/abs/2006.06848.

[77] Randall Balestriero, Jerome Pesenti, and Yann LeCun. *Learning in High Dimension Always Amounts to Extrapolation.* 2021. DOI: 10.48550/ARXIV.2110.09485. URL: https://arxiv.org/abs/2110.09485.

[78] Ian J. Goodfellow et al. *Generative Adversarial Networks.* 2014. DOI: 10.48550/ARXIV.1406.2661. URL: https://arxiv.org/abs/1406.2661.

[79] Yaroslav Ganin et al. “Domain-Adversarial Training of Neural Networks”. In: *J. Mach. Learn. Res.* 2016.

[80] Maksim Lapin, Matthias Hein, and Bernt Schiele. *Top-k Multiclass SVM.* 2015. DOI: 10.48550/ARXIV.1511.06683. URL: https://arxiv.org/abs/1511.06683.

[81] JuliaPy Development Team. *PyJulia.* URL: https://github.com/JuliaPy/pyjulia.

[82] Amir Arfan. *Thesis.* 2022. URL: https://github.com/amirarfan/Thesis/tree/28d81d77dfa4780d4289558f52dbaceaae424725.

[83] Python Software Foundation. *Python Package Index, PyPi.* 2022. URL: https://pypi.org/.

[84] Amir Arfan. *Alpha BPC Python Package Commit: 59b40be.* 2022. URL: https://github.com/amirarfan/alpha_bpc/tree/59b40be341e0f1380649690d88e3024b90ddd6ef.

[85] Amir Arfan. *Alpha BPC Python Package.* 2022. URL: https://github.com/amirarfan/alpha_bpc/.

Appendices

Appendix A: Datasets

In this appendix, we will present the datasets used in this thesis for testing our methods.

MNIST

The MNIST dataset [21], often referred to as a database, consists of grayscale images of handwritten digits. It is a well-studied problem in the machine-learning world, thus motivating our usage of this dataset. By testing our methods on the MNIST dataset, we can easily compare them against state-of-the-art methods. The MNIST dataset is also a relatively easy problem to solve using CNNs; consequently, the minimum criterion for our proposed methods is to achieve good performance on this dataset.

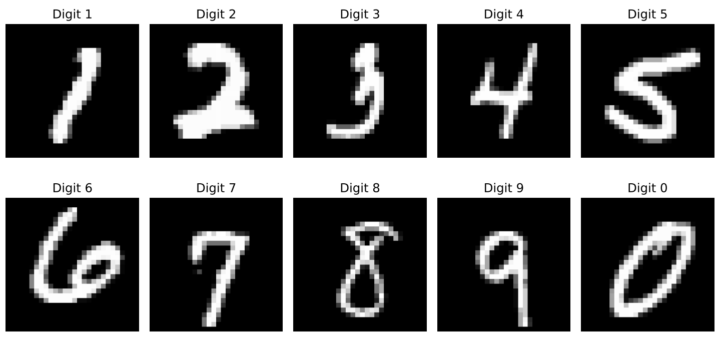


Figure 6.1: Randomly selected digits from each group in the MNIST dataset.

Figure 6.1 showcases randomly selected images from each digit group. The labels range from 1 to 10, where label 10 corresponds to the number 0. The dataset is pre-divided into a training set and a test set. The training set consists of 60 000 images, and the test set consists of 10 000 images. The distribution of classes in the training set is illustrated in Figure 6.2, showing an almost equal distribution. Therefore, one does not need to use any upsampling or downsampling to balance out the distribution of classes.

F-MNIST

The Fashion-MNIST dataset (F-MNIST) [57], motivated by the ease of use of the MNIST database, provides a more complex drop-in replacement for MNIST. The dataset consists of different types of grayscale images of clothing. It is a more challenging problem, with a human performance of only 83.5% accuracy [60]. Therefore, we deemed it necessary to test our methods on the F-MNIST since it provides a more formidable challenge than MNIST.

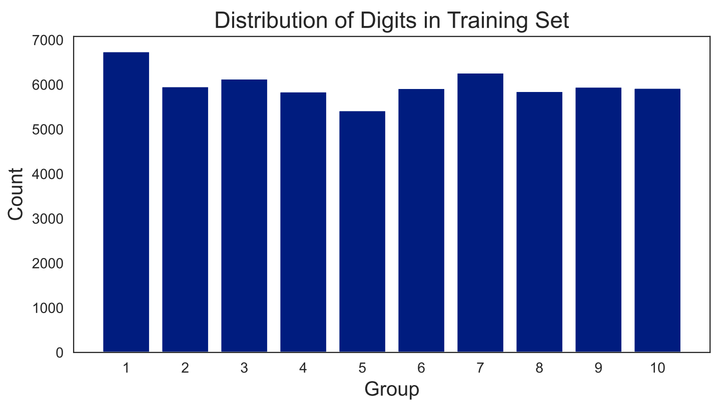


Figure 6.2: Distribution of each digit group in the training dataset for MNIST.

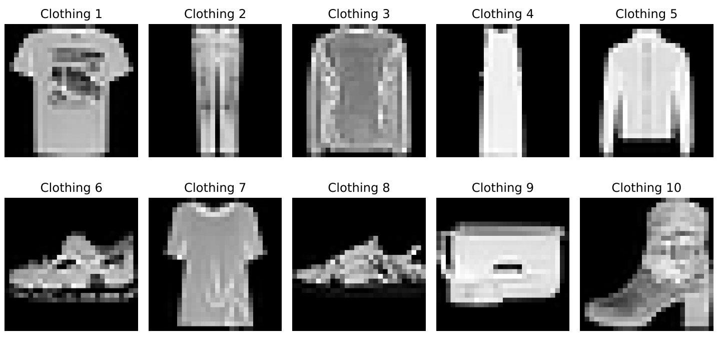


Figure 6.3: Randomly selected clothing from each group in the F-MNIST dataset.

Figure 6.3 illustrates randomly selected images from each clothing group. The groups range from 1 to 10, where table 6.1 provides information about each group. The dataset, similar to MNIST, is pre-divided and consists of a training set of 60 000 images and a test set of 10 000 images. The distribution of each class is illustrated in Figure 6.4, showing an equal distribution.

Label	Clothing Type
1	T-Shirt
2	Trouser
3	Pullover
4	Dress
5	Coat
6	Sandals
7	Shirt
8	Sneaker
9	Bag
10	Ankle Boots

Table 6.1: Different types of clothing groups in the F-MNIST dataset.

Appendix B: Practical Implementations

This appendix contains information about the practical implementations of the proposed methods in this thesis. We will cover what hardware was used for developing the source. Additionally,

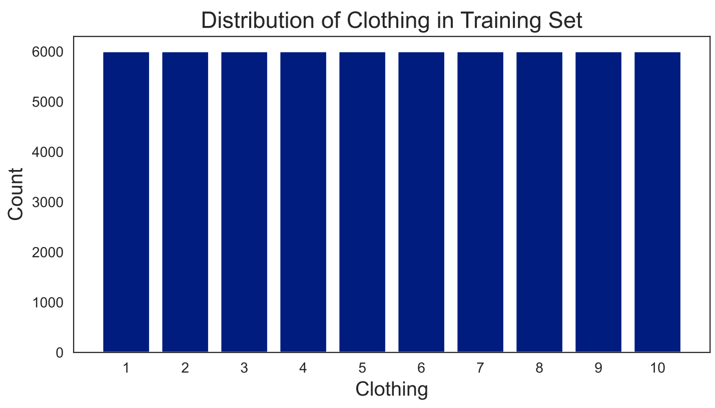


Figure 6.4: Distribution of each clothing group in the training dataset for F-MNIST.

we will present Python and Julia packages for implementing this thesis's source code. Links to relevant source code and the GitHub repository will be presented.

Hardware Specifications

The hardware specifications of the setup used for testing all implementations are given in table 6.2.

GPU	RTX 3080 12 GB
CPU	Intel I7-12700K
RAM	32 GB
OS	Windows 11 64 Bit

Table 6.2: Hardware specifications used for implementing source code

Python and Julia Packages

A combination of Python and Julia was used. Python was chosen for ease of use due to a wide variety of packages. PyTorch [58] ensured that convolution operations performed in this thesis could efficiently be coded using GPUs. Julia was used for the faster code runtime and vectorized arithmetics. We connected these two languages through the PyJulia package [81], which provides an interface to call on Julia functions within Python code. A summary of relevant packages for this thesis is given in tables 6.3 and 6.4.

Github Files

Appendix C: BPC Python Package

Understanding that the source code can be challenging to navigate through and to motivate experimenting with the BPC framework, we have developed an alpha version of BPC as a Python package.

The package is currently only limited to work for grayscale images and requires a GPU to use for the convolution process. It has also only been tested on a limited amount of computers, and we cannot guarantee the compatibility of the package with all systems.

Python	**3.9.12**
Package Name	**Version**
NumPy	1.21.6
SciPy	1.9.3
PyJuila	0.5.7
PyTorch	1.12.1
Scikit-Learn	1.1.3
Scikit-Image	0.19.2
Matplotlib	3.5.2
Numba	0.55.1
Pandas	1.5.1
Seaborn	0.12.0
OpenCV-python	4.6.0

Table 6.3: Information about the Python distribution and packages.

Julia	**1.8.2**
Package Name	**Version**
MKL	0.6.0
VMLS	0.4.0
FLoops	0.2.1
NPZ	0.4.2
PyCall	1.94.1

Table 6.4: Information about the Julia distribution and packages.

Name	**Description**	**Commit Hash**
TregsYLooCV.jl	Implementation of BOLS with fast LOOCV	fe5c9f8
cks.py	Implementation of Clustered Kennard-Stone Variant sample selection	cdd4df4
crs.py	Implementation of Clustered Random Supervised sample selection	cdd4df4
rs.py	Implementation of Random Supervised sample selection	cdd4df4
extract_filters.py	Function for extracting patches and binarizing them. Optional center patch, and optional threshold	cdd4df4
convolve_filts.py	Batched implementation for convolving patches onto images using GPU	cdd4df4
hu_moments.py	Vectorized implementation for calculating Hu moments of images	cdd4df4
MNIST_RESULTS/	Folder containing source code to produce results from using BPC and MOLS on the MNIST dataset with vaious sampling strategies	28d81d7
FMNIST_RESULTS/	Folder containing source code to produce results from using BPC and BOLS on the F-MNIST dataset with various sampling strategies	cbb2cf0
cks_61700/	Folder containing pre-stored BOLS model for the MNIST dataset using 61700 filters selected using C-KS	5cc626f
cks_61210/	Folder containing pre-stored BOLS model for the F-MNIST dataset using 61210 filters selected using C-KS	cbb2cf0
isvbase.py	Implementation of iterative classification using SVD bases	c5ee964
i-svdbases/	Folder containing source code for the results obtained using the iterative classification method	c5ee964
mnist_cks_cl_valid_neg.py	Source code for testing out negative transfer on training and validation data for MNIST	d9765fc
mnist_cks_neg.py	Source code for negative transfer minimization on original test data and BPC data for MNIST	28d81d7
Repository	Github repository containing source code for methods and results in this thesis	28d81d7

Table 6.5: Collection of relevant files for this thesis along with their description, commit hash and permalinks to access them in the GitHub repository [82].

Installation

The package has been published to PyPi [83] for ease of installation and can be installed as such:

```
python -m pip install alpha-bpc==0.1.2
```

If the installed package does not work as intended, the package can be cloned from the following GitHub repositories:

- Alpha BPC - Commit: 59b40be [84] (Corresponds to the version of the repository as of writing this thesis)
- Alpha BPC [85] (Corresponds to the active development version of the repository)

Usage

First, to use the BPC class, one must import the class from the alpha_bpc package, as such:

```
from alpha_bpc.BPC import BPC
```

By calling `help(BPC)`, one can get an overview of the parameters needed to initialize the BPC class. During the initialization phase, one can choose between three sample selection algorithms, C-KS, CRS, and RS. The `fit` function will then use the selected sample selection algorithms to select the number of images specified and extract binarized patches for a chosen threshold, effectively storing the convolution kernels. The `fit_transform` function will find a set of kernels and return the convolved training data horizontally stacked with the original training data. The `transform` function will convolve filters with provided data. It requires the `fit` function to be called first.

Examples

A set of examples are available for both MNIST and F-MNIST in the examples directory in the BPC GitHub repository [84]. Here is an example of using the BPC package with the F-MNIST dataset:

```python
"""Example of using BPC for feature extraction on MNIST dataset.
ImageSelectionMethod = 1: C-KS
ImageSelectionMethod = 2: CRS
ImageSelectionMethod = 3: RS
"""

# Remember to set PythonPath = ".." in the Run/Debug Configuration

from alpha_bpc.BPC import BPC
from alpha_bpc.utils.load_fmnist import load_fmnist as lf


def main():
    x_train, y_train = lf("../data/fmnist", kind="train")

    y_train = (
        y_train.copy() + 1
    )  # Since our framework operates on 1-10, and not 0-9
    num_clusters = 5
    num_imgs = 4

    BPC_extractor = BPC(
        x_train,
        y_train,
        num_clusters=num_clusters,
        verbose=True,
        image_selection_method=1,
    )

    x_train_convolved = BPC_extractor.fit_transform(num_imgs)
    print(x_train_convolved.shape)


if __name__ == "__main__":
    main()
```