Classifying Handwritten Letters of The Latin Alphabet with Machine Learning

Elias Myllymäki and Rasmus Pouta

September 2025

1 Introduction

In this paper, we explore the performance of machine learning in recognizing and classifying handwritten latin alphabetical characters. Classification of visual data is an increasingly important task in modern society with technologies such as optical character recognition (OCR), which is, for example, used to convert physical books to digitized versions, relying on accurate methods for classifying characters from image data. Machine learning plays an important role in developing these technologies by enabling the analysis of documents under suboptimal conditions such as poor handwriting or unusual fonts[1].

This report focuses on the formulation of the problem and the selection of a suitable machine learning method to analyze the image data. The dataset used in this project will be presented, as well as the process of feature selection and data preprocessing. The properties of a suitable model and loss function will be discussed with the process of model validation, but analysis of the validation and training errors is beyond the scope of this report. The topic of this project was inspired by a student project titled "A Machine Learning Approach to Classifying Bangla Handwritten Characters" from 2023 found on the CS-C3240 Machine Learning course page[2].

2 Problem formulation

The problem of classifying handwritten alphabetical characters can be formalised as a supervised machine learning problem where image data of characters is used to train a machine learning model to classify handwritten characters accurately. Training, test, and confirmation data used in this project was sourced from a publicly available dataset containing images of handwritten letters of the latin alphabet[3]. The dataset contains 300x300 pixel grayscale images of handwritten latin alphabetic characters, categorized with 26 labels (one for each letter), which can be represented as vectors where each element of the vector contains the color value of a pixel. A value in the inclusive whole

number interval [0, 255] is used to represent the color of a pixel in a grayscale image.

Thus, each of the datapoints represents a 300^2-dimensional vector containing the full color data of a single grayscale image. However, due to computational limitations and properties of the images, which will be discussed in the later sections, image resizing and PCA is conducted to reduce the dimension of the data to 30. Therefore, the machine learning model uses data with 30 feature variables. These datapoints lack a direct interpretation, but they can be considered a projection of the vectors containing color data of the resized images onto a 30-dimensional hyperplane.

As the project aims to implement a model for classifying latin alphabetic characters, the labels of the model represent each letter of the latin alphabet. Hence, the model has 26 labels with one label for each letter. The labeled training data teaches the model to differentiate between different characters and to accurately label given characters based on the color data of the image.

3 Methods

3.1 The dataset

3.2 Data preprocessing and feature selection

3.3 Model

In this paper we used two models, a multilayer perceptron (MLP) and TOINEN MALLI. They are explained in more depth below.

3.4 Multilayer perceptron (MLP)

Our MLP consists of four dense layers, an input layer, a hidden layer, another hidden layer and an output layer with 30, 128, 64 and 26 neurons each.

3.5 Loss function

We used the cross entropy loss function **??** because it has historically been a great choice for classification tasks and it is computationally efficient.

3.6 Measuring performance

We compared the performance of different loss functions for both models. A full list is included in the code itself. The list included the sigmoid function, RELU, leaky RELU and others. The performance was measured by the total loss of the very last iteration of said activation function, lower loss meaning better performance in classification. Some of these loss functions themselves have a learnable parameter for example the PRELU function expressed

as PRELU$(x) = \max(0, x) + a \cdot \min(0, x)$ has the learned parameter a [4] . Practically the measurement was done via the use of a simple for loop which iterated over each activation function.

In addition, we also iterated over different optimizers to see how well they perform.

We did not iterate over different neural network architectures since that would have required a very large amount of computation power to make a search over different amount of layers and neurons in addition to our search over all the activation and loss functions and optimizers.

3.7 Model validation?

A subset of the dataset with 200 images of each character was used for training, since the original dataset is massive and, due to practical limitations of computation power, processing all of these entries is infeasible for the purposes of this student project.

The file actually contains many different datasets, but we used a subset of the "train" directory. This directory contains directories corresponding to each character of the latin alphabet, and we used the alphabetically sorted 200 files from each one of these subdirectories for training. Each training entry in this dataset is a 300x300 grayscale picture of a handwritten character and the corresponding label.

Validation was done via the use of the "validation" directory from the original ZIP-file. As before, we narrowed our use of this dataset to 200 files per one character of the english alphabet and used these to determine the accuracy of the trained model.

References

[1] What is optical character recognition (ocr)? URL `https://www.ibm.com/think/topics/optical-character-recognition`.

[2] Cs-c3240 - machine learning. URL `https://mycourses.aalto.fi/course/section.php?id=271289`.

[3] Srivastava, S. Alphabets dataset (300x300). URL `https://www.kaggle.com/datasets/sankalpsrivastava26/capital-alphabets-28x28/data`.

[4] Pytorch prelu. URL `https://docs.pytorch.org/docs/stable/generated/torch.nn.PReLU.htmln`.