Semester Project Report

Neural Networks (AI)

Team Members: David van Wuijkhuijse (s5592968),
Marcus Harald Olof Persson (s5343798),
Richard Frank Harnisch (s5238366)
University of Groningen

November 3, 2025

Abstract

We implemented a feed-forward convolutional neural network from the group up including loss, activations, optimizer, and LR scheduler. We trained and evaluated the model on the German Traffic Sign Recognition Benchmark (GTSRB) dataset with 38 classes. Our model achieved a test accuracy of xx%.

Contents

1 Introduction

With this project, we implemented the 1998 CNN introduced by Lecun et al. (1998) using our own architecture coded from scratch in Python. We then deploy this CNN on the German Traffic Sign Recognition Benchmark (GTSRB), consisting of 51,389 images in 43 classes. However, we only train on 5 of these classes due to resource constraints, as our home-made implementation is significantly slower than optimized libraries like TensorFlow or PyTorch. The dataset contains images of varying sizes and conditions, which we develop a modular suite of preprocessing transforms to handle.

Our main goal with this project is to create the core components of a CNN from scratch, only using lower-level libraries (e.g. Numpy, Pillow) in Python. The motivation behind this decision was to gain further insights into how neural networks work, by understanding and implementing the math behind the components of the CNN. We evaluated model performance using the accuracy metric and confusion matrices. We considered a model having a higher accuracy metric than 20% (random guessing baseline) to be successful.

2 Data

The German Traffic Sign Recognition Benchmark (GTSRB) dataset consists of 51,839 images of traffic signs on German roads, categorized into 43 different classes. The images are taken from dashcam footage of cars driving on the road and thus past the signs. This means that each sign is captured multiple times as the car passes the sign. This lessens the need for data augmentation as each sign is already represented multiple times. Signs are captured in a range of different lighting and weather conditions, as well as from different angles and distances.

Figure 1: The same sign from different distances (and thus sizes) and slightly different angles.

The dataset is provided by Stallkamp et al. (2012) [2] [3] from the Institute for Neuroinformatics at the University of Bochum in Germany. By default, the dataset is split into a training set and a testing set. This was originally for a competition at the International Joint Conference on Neural Networks in 2011. Considering we are (considerably) past the deadline for submission, we have chosen to merge the training and testing sets at download time and then create our own split. This allows us to modify the split ratio as we like.

The images themselves are presented as PPM files with varying resolution and aspect ratio depending on the distance and angle of the sign to the camera. The aspect ratio is mostly close to 1:1. For more detailed analysis of the images sizes see Table 1 and Figure 3. There are three color channels (RGB) and the pixel values are in the range of 0-255. See Figure 4 for the distribution of pixel values. The labels for the images are provided in one master file `labels.csv` for the entire dataset. Each line contains the filename, class ID, and bounding box coordinates for the sign in the image.

Figure 2: Three different signs of the same type exhibiting variation in lighting conditions and angles. For more examples of different types of variation, see Figure 6 in Appendix B.

Statistic	Width (px)	Height (px)	Aspect Ratio (W/H)
Mean	50.76	50.34	1.0056
Std	24.50	23.26	0.0734
Min	25	25	0.3681
25th Percentile	34	35	0.9706
50th Percentile (Median)	43	43	1.0000
75th Percentile	58	58	1.0385
Max	266	232	1.4375

Table 1: Descriptive statistics of image dimensions and aspect ratios in the GTSRB dataset.

We load the data as a numpy array of shape (H, W, C) for each image, where H is height, W is width, and C is the color channel (so three). The labels are loaded as integers corresponding to the class IDs. Before feeding the images to the model, we normalize the pixel values to the range [0, 1] by dividing by 255.

- Source of dataset
- Size and type (tabular, images, time series. . .)
- Key characteristics (with plots or descriptive stats)
- Any preprocessing or cleaning

Data is downloaded and cleaned automatically using the `dataio/gtsrb_download.py` script. This script downloads the zipped dataset (three files: training dataset including labels, test dataset, test dataset labels), merges all images into one folder, and creates one master CSV file with all labels. It then deleted the zip files unless a parameter is set to keep them, as well as cleans the folder structure to remove unnecessary files (dataset text info files) and now empty folders. The data is freely hosted online by the University of Copenhagen Electronic Research Data Archive.

We implement a GTSRB dataset class in `dataio/gtsrb_dataset.py`. This class handles loading the images and labels from disk, applying any specified transforms, and providing access to individual samples. This is meant as a minimal implementation of the PyTorch Dataset class. The dataset provides access to a length attribute (number of samples) and a getitem method to retrieve individual samples by index (returns a tuple of image and label). The dataset class also supports caching of images in memory for faster access during training, as well as multiple worker threads for parallel loading of data. These performance optimizations were implemented later in the project after initial tests showed slow training.

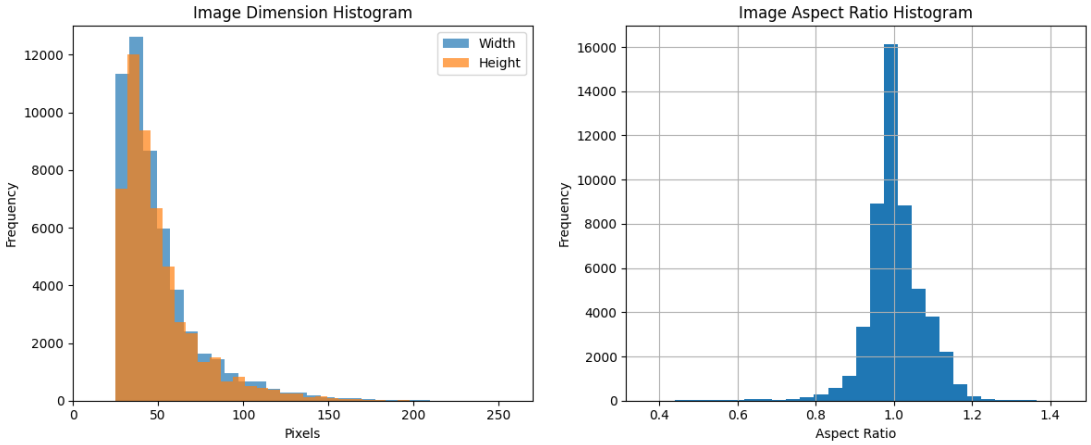

Figure 3: Distribution of image dimensions and aspect ratios in the GTSRB dataset. Most images are around 30-70 pixels in both dimensions, with a long tail towards larger sizes.

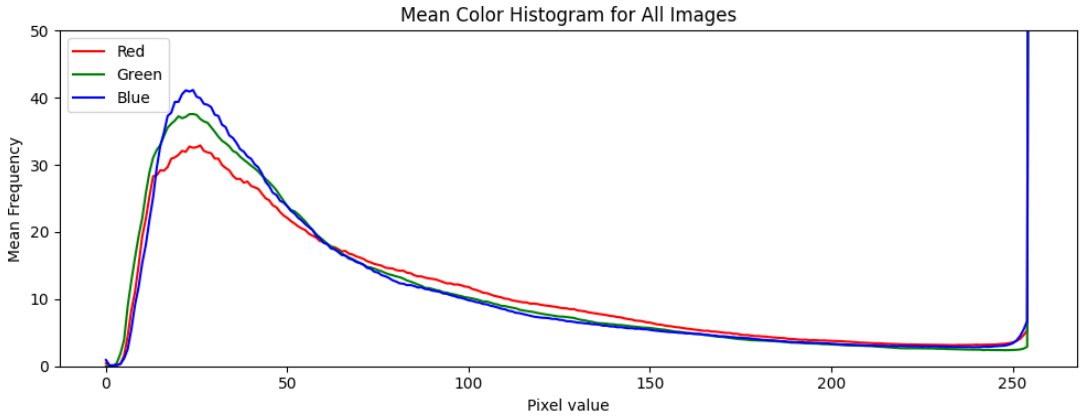

Figure 4: Histogram of pixel values across all images and color channels in the GTSRB dataset. The graph is cut off at frequency=50, but the frequency of flat white is very high due to the reflective nature of signs as well as the common use of the color white, causing (parts of) signs to max out the camera's sensor.

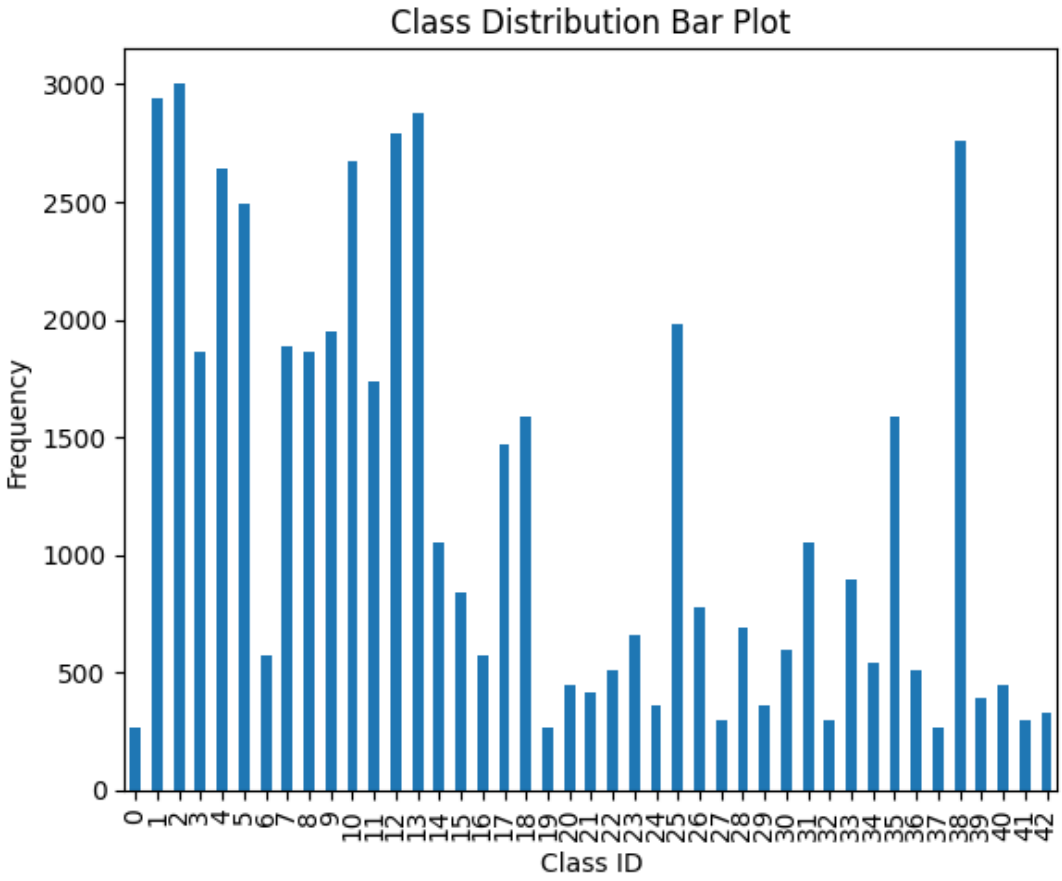

Figure 5: Class distribution of the GTSRB dataset. The dataset is imbalanced, with the smallest class having 270 images and the largest 3000. The mean is 1206 images per class.

3 Methods and Experiments

3.1 Pipeline Overview

To begin, we preprocess the data. For this purpose, we have developed a suite of modular transform classes implemented in `transforms.py`. Each class implements a `__call__` method, allowing instances of these classes to be used as callable functions that apply specific transformations to input images. This design enables easy composition of multiple transformations into a single pipeline using the `ToCompose` class, which sequentially applies a list of transform instances to an image. On top of this, the following classes are implemented:

- `ToResize`: Resizes images to a specified square size using bicubic interpolation. This is so the images can be fed into the model which requires a fixed input size.
- `ToCenterCrop`: Crops a square out of the center of an image to a specified pixel size.
- `ToGrayscale`: Converts RGB images to grayscale using the ITU-R 601-2 luma transform using the formula
$$L = \frac{1}{1000}(299R + 587G + 114B), \tag{1}$$
where R, G, and B are the red, green, and blue pixel values, respectively. This is implemented by the PIL library. We then convert back to RGB by duplicating the grayscale channel three times, as the network expects three channels.
- `ToTensor`: Converts images from numpy arrays to PyTorch tensors and permutes the dimensions from (H, W, C) to (C, H, W). Also converts pixel values from integers (0-255) to floats (0.0-1.0).
- `ToNormalize`: Centers values around zero by subtracting the mean and sets the standard deviation to one by dividing by the standard deviation for each channel. Per-channel mean and standard deviation are computed on the training set and passed as a parameter to the transform.
- `ToRotate`: Applies random rotations within a specified angle range. Rotate function implemented by the PIL library.
- `ToRandomNoise`: Adds Gaussian noise to images with a specified mean and standard deviation. Normal distribution is sampled using numpy.

Data is loaded using the Dataloader class, implemented in `dataio/dataloader.py`. This class handles batching, shuffling, and parallel loading of data using multiple worker threads. When constructing a DataLoader, we specify the dataset to load from, batch size, whether to shuffle the data at the start of each epoch, whether to drop the last batch if it is imcomplete, the seed for shuffling, an optional custom function to collate samples into batches, how many batches to prefetch, and the number of worker threads for parallel loading. The DataLoader provides an iterator interface to loop over batches of data during training. The class also provides a default collating function that stacks images and labels into tensors. The DataLoader uses a thread pool executor to load batches in parallel, improving performance when using multiple workers. This was also implemented after slow initial training. During training we iterate through the dataloader to get new batches of images and labels.

For a diagram of the full data pipeline, see Figure 7 in Appendix B.

3.2 Model Description

- Architecture (MLP, CNN, etc.), with math or pseudocode if needed. - Loss function, optimizer, regularization.

3.3 Training Procedure

- Cross-validation setup, train/test split. - Hyperparameters and how you chose them.

4 Results

- Performance metrics (accuracy, MSE, etc.)
- Learning curves or confusion matrices
- Compare with baseline(s)

5 Discussion

- What worked well and why
- What didn't work, limitations
- Possible improvements or future work

6 Conclusion

Wrap up: summarize objectives, approach, key findings, and lessons learned.

References

[1] Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

[2] Stallkamp, J., Schlipsing, M., Salmen, J., and Igel, C. (2011). The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In *IEEE International Joint Conference on Neural Networks*, pages 1453–1460.

[3] Stallkamp, J., Schlipsing, M., Salmen, J., and Igel, C. (2012). Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*.

A Use of AI Tools

We used LLM tools during both writing the report and coding. The helper of choice was GitHub Copilot using GPT-4.1. During coding, it was used in EDA for plotting as well as to write non-ML boilerplate code, such as converting .ppm images to a numpy array.

During writing, the autocomplete feature was used to speed up formulation of sentences as well as format figures and charts. The ideas and structure of the report are ours.

B Additional Materials

Figure 6: 25 random samples from the dataset.

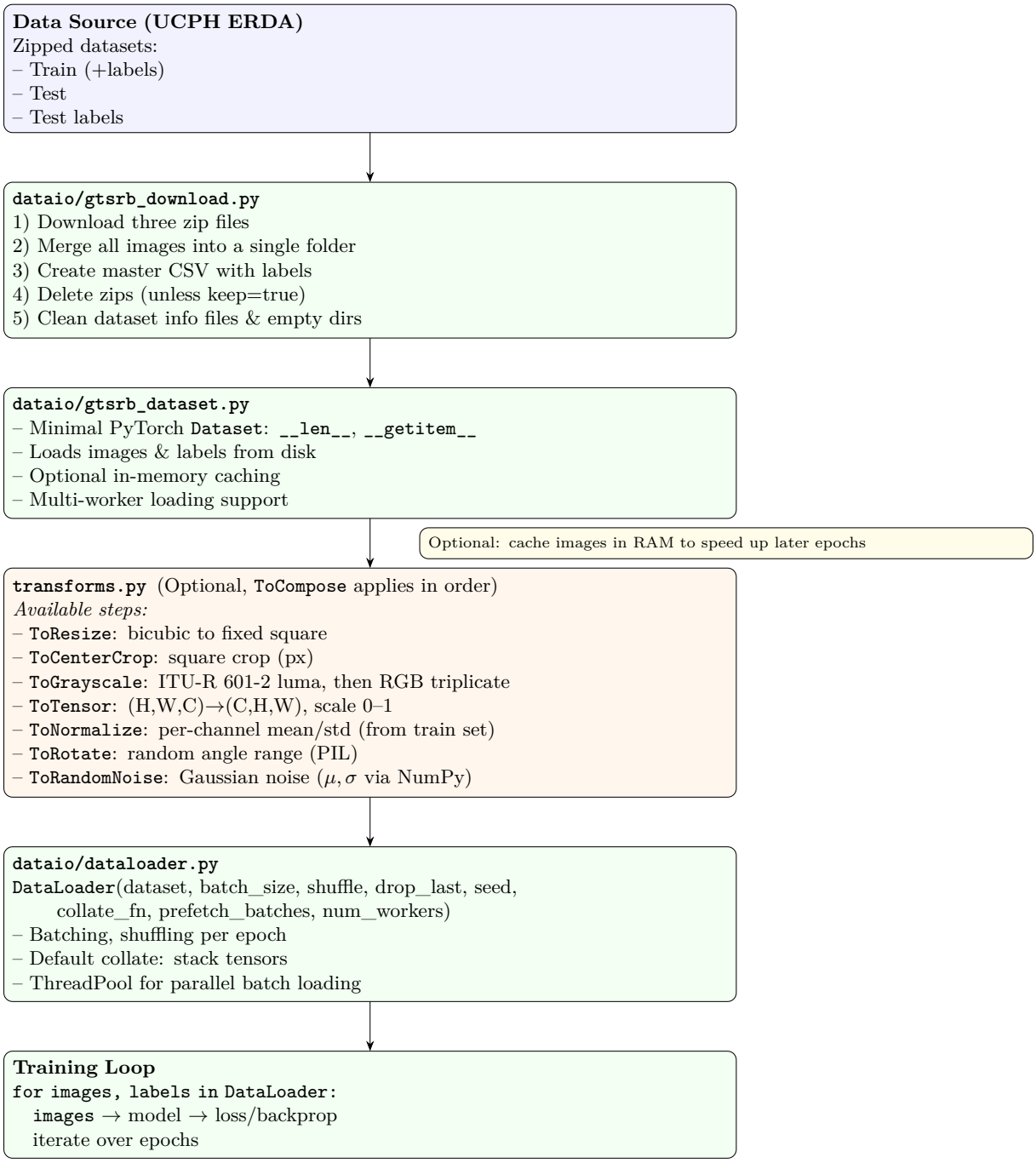

Figure 7: GTSRB data pipeline from download → dataset → transforms → dataloader → training.

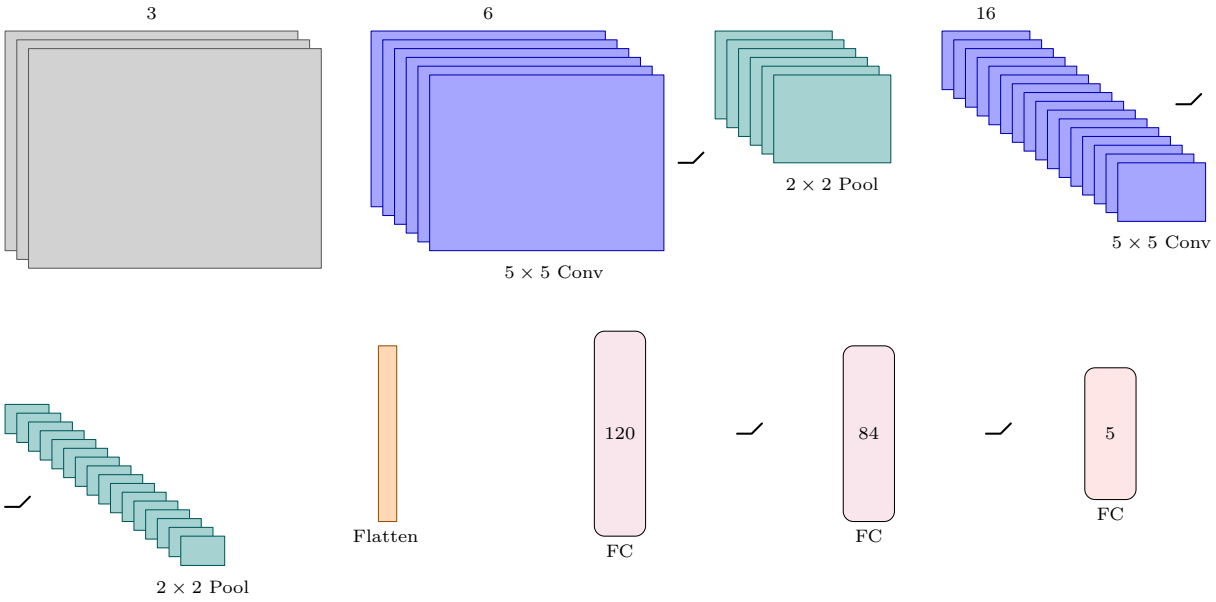

Figure 8: Visual representation of our model architecture. The presence of a small ReLU marker indicates a ReLU activation after that layer.