PROGRESS REPORT: INTELLIGENT TRAFFIC SIGN CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS FOR AUTONOMOUS AND ASSISTIVE DRIVING

Audrey Wang
Student# 1010173955
audreyxiwei.wang@mail.utoronto.ca

Judie Du
Student# 1009981407
judie.du@mail.utoronto.ca

Yiling Fan
Student# 1010161352
yiling.fan@mail.utoronto.ca

Yinyin(Eva) Zhang
Student# 1010477761
yinyin.zhang@mail.utoronto.ca

Colab Link:

https://colab.research.google.com/drive/1YtmGzW_bqlVXa0NJK2dj7S9FL4_WSwkQ?usp=sharing

ABSTRACT

This project presents an intelligent traffic sign recognition system designed to support both autonomous and assistive driving technologies. Using the German Traffic Sign Recognition Benchmark (GTSRB) dataset, we implement and compare two classification approaches: a baseline model using Histogram of Oriented Gradients (HOG) combined with a linear Support Vector Machine (SVM), and a primary model, Convolutional Neural Network (CNN), tailored for real-time image analysis. The baseline model offers interpretability and low complexity, while the CNN leverages deep learning to automatically extract spatial and semantic features for improved accuracy and generalization.

To date, we have completed data preprocessing (including resizing, splitting, and augmentation), initial baseline training, and CNN development and evaluation. This report outlines our model architecture, training setup, early evaluation results, and current challenges, including augmentation overhead and limited hyperparameter tuning. Our next steps involve experimenting with transfer learning, improving the model's generalization, and conducting final evaluations. Overall, this report summarizes our progress and outlines the work ahead to complete the project. —-Total Pages: 9

1 PROJECT DESCRIPTION

1.1 MOTIVATIONS AND GOAL

Whether in everyday driving or autonomous navigation, traffic signs play a crucial role in ensuring safety and order on the roads. They convey essential information — from speed limits to warnings and regulations — that drivers must interpret quickly and accurately, as any failure can lead to serious consequences. With the increasing adoption of autonomous vehicles and advanced driver assistance systems (ADAS), the ability to recognize and respond to these signs without human input has become more important than ever. This growing need has motivated us to develop an intelligent traffic sign recognition system capable of reliably identifying and classifying signs from images.

The goal of our project is to build a model that can effectively analyze images and determine the specific type of traffic sign depicted (i.e., stop sign, speed limit indicator, or warning symbol). Such a system could support not only autonomous driving but also assistive technologies for human drivers, such as real-time alerts for overlooked signs.

1.2 The Role of Deep Learning

Deep learning offers a compelling solution to this task due to its proven strengths in image classification. Convolutional Neural Networks (CNNs), in particular, are well-suited for capturing the spatial patterns, shapes, and color features that distinguish different traffic signs. By learning hierarchical representations directly from raw pixel data, CNNs can adapt to variations in lighting, angle, background noise, and more, all of which are common in real-world driving environments. This adaptability and generalization capability make deep learning especially suitable for traffic sign recognition, where consistency and high accuracy are essential for safety.

1.3 Overview of the Model

Our proposed traffic sign classification model is built using a custom CNN designed to balance performance and computational efficiency. The model takes RGB images from the German Traffic Sign Recognition Benchmark (Stallkamp et al., 2011), resized to 32×32 pixels, and processes them through a sequence of convolutional layers to extract spatial and semantic features. Batch normalization is used to stabilize training, dropout is applied to reduce overfitting, and data augmentation is incorporated to address class imbalance. The resulting feature maps are flattened and passed through fully connected layers to classify each image into one of 43 traffic sign categories. Figure 1 below presents the pipeline of our model.

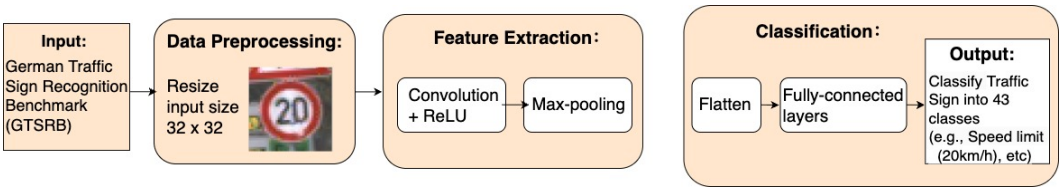

Figure 1: Traffic sign recognition model. Image: GTSRB(Stallkamp et al., 2011)

2 Individual Contributions and Responsibilities

2.1 Ways of Working

Our team maintains effective collaboration through scheduled meetings, ongoing communication, and clear code management practices. We meet every Tuesday at 11 a.m. EDT via our WeChat/Discord group channel to review progress, assign tasks, and discuss technical issues. These weekly syncs ensure alignment on deadlines and allow us to resolve challenges in real time. Outside of meetings, team members stay in touch through WeChat or Discord, and are expected to respond within 24 hours to maintain steady progress.

Before distributing deliverables, the team held a planning session to review the evaluation criteria and project guidelines. This ensured everyone understood the expectations, allowing us to divide tasks based on individual skillsets and availability. Workload distribution has remained transparent, and any major changes to responsibilities are discussed and agreed upon as a team. If a team member is unavailable for a key task, the rest of the team is prepared to step in and provide support as needed.. This helps ensure the project stays on track and reduces the risk of delays due to unexpected issues.

To manage collaborative coding, we use a shared Google Colab notebook with clearly assigned sections for each member. This structure allows us to work in parallel without overwriting each other's code. All team members are expected to document their work with clear comments, especially when introducing assumptions or modifying model architecture. Before merging significant changes, all

members are expected to review the updates. Major changes are preceded by group discussion and informal peer review to ensure compatibility.

2.2 Project Plan and Work Distribution

Table 1 below outlines the project plan, including task descriptions, responsible members, deadlines, type, and current progress status.

Table 1: Project Plan Timeline and Work Distribution

Task	**Responsible Member(s)**	**Deadline**	**Deadline Type**	**Status**
Brainstorm & Idea Generation	All	June 6	Internal	Completed
Background Research & Literature Review	Judie	June 9	Internal	Completed
Data Collection & Research (GTSRB sourcing)	All	June 9	Internal	Completed
Flowchart & Illustration Generation	Eva	June 12	Internal	Completed
Data Preprocessing in Google Colab (resize, relabel, split, augmentation, etc)	Yiling	June 12	Internal	Completed
Research on Baseline Model & Initial Implementation	Audrey	June 12	Internal	Completed
Project Proposal	All	June 13	External	Completed
Updating Baseline Model after Project Proposal & Final Implementation	Yiling, Judie	July 7	Internal	Completed
Building and Training Primary Model	Audrey, Eva	July 9	Internal	Completed
Primary Model Initial Testing & Evaluation (obtain quantitative and qualitative result.)	All	July 9	Internal	Completed
Progress Report	All	July 11	Internal	Completed
Improving the Model by Implementing Transfer Learning and Training Further	All	July 21	External	Incomplete
Final Testing & Evaluation of Model on New Data	All	July 25	Internal	Incomplete
Final Presentation Slides	All	August 4	Internal	Incomplete
Final Presentation Video	All	August 6	Internal	Incomplete
Final Report	All	August 15	External	Incomplete

3 Data Processing

3.1 Image Resizing & Dataset Split

The dataset used in this project was sourced from the official German Traffic Sign Recognition Benchmark (Johannes Stallkamp, 2019), a publicly available archive created by Christian Igel at the University of Copenhagen. We extracted the pre-organized training subset from the source archive, which contains 43 class folders with approximately 1000–2000 images each (over 50000 total). Since none of the 43 traffic-sign classes contain extensive text—only simple shapes and short legends—a 32×32 pixel crop is sufficient to preserve all necessary detail while enabling efficient CNN training. Using a fixed random seed (25) for reproducibility, we shuffle each class and split into 70% train / 15% validation / 15% test. A 70/15/15 split gives the model enough data (70%) to learn robust features, retains a separate validation set (15%) for hyperparameter tuning without peeking at test results, and holds out an unbiased test set (15%) for final evaluation. By stratifying each class equally into these proportions, we ensure balanced representation, maintain reproducibility, and keep training, validation, and testing pipelines fully isolated. The images are then organized into three directories:

`train/ val/ test/`

Each containing 43 subfolders (one per class), enabling straightforward loading via `torchvision.datasets.ImageFolder`.

3.2 Addressing Class Imbalance via Augmentation

Initial analysis revealed a wide class distribution (some classes ∼2000 samples, others ∼200) as seen in Figure 2.

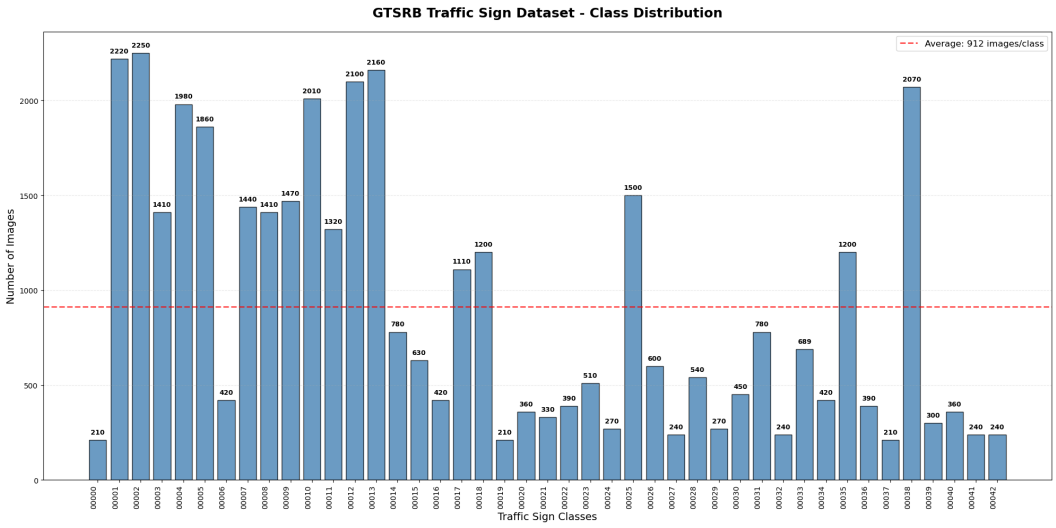

Figure 2: Class distribution of the GTSRB training dataset

To enforce a near-uniform distribution (each class = 1575 images) during training, we apply on-the-fly augmentation only to the `train/` set. Our augmentation pipeline in `traffic_sign_recognition.py` is:

```
augment_transform = transforms.Compose([
    transforms.Resize((32,32)),
    transforms.RandomHorizontalFlip(),      % left/right invariance
    transforms.RandomRotation(15),          % tilt robustness
    transforms.RandomAffine(0, translate=(0.1,0.1)),
```

```
    transforms.ColorJitter(0.2, 0.2),       % brightness & contrast
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485,0.456,0.406],
        std=[0.229,0.224,0.225]
    )
])
```

Horizontal flips simulate mirror invariance, rotations handle tilted camera angles, affine shifts mimic framing shifts, and colour jitter compensates for lighting variations.

Figure 3 below illustrates the data processing pipeline.

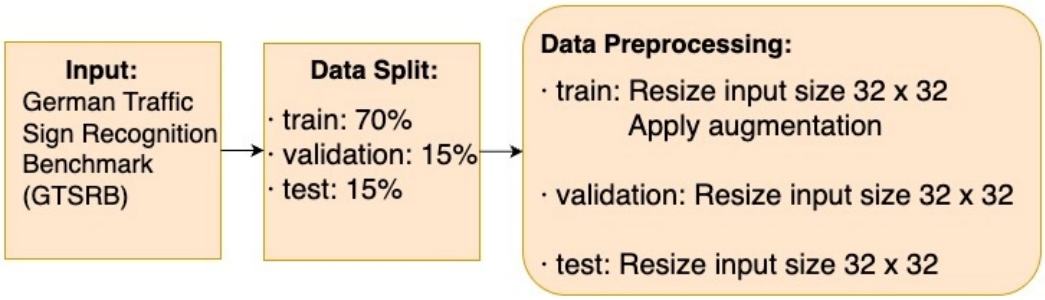

Figure 3: Data processing

3.3 Sample Visualization & Processing Challenges

To verify integrity, we plot one random sample per class in a 7×7 grid using a snippet that selects the first index for each label from `train_data.samples` and displays them with Matplotlib. This grid (Figure 4) confirms coverage of all 43 classes and that augmentations preserve key features.

Figure 4: One Sample Image From Each Class

We encountered a few practical challenges during data processing. Copying tens of thousands of small image files into our split directories proved to be a significant disk I/O bottleneck, which we alleviated by batching filesystem operations and skipping redundant writes. Tuning our augmentation parameters also required caution: we needed to carefully limit the range of rotations and translations to avoid cropping out critical sign elements, striking the right balance between diversity and legibility. On-the-fly augmentation further increased GPU memory and data-loading demands, so we enabled `pin_memory = True` in our DataLoaders and chose a moderate batch size of 68 to maintain smooth throughput. Additionally, because Google Colab enforces strict GPU usage limits and doesn't persist large datasets across sessions, our team migrated to local runtime environments. This meant each member had to store a full copy of the 50 000+ image dataset on their own machine and update the data path in the code to their local directory whenever they ran experiments.

As for future plan, to validate on truly new data, we could collect around 500 traffic-sign images around campus (and/or use the BelgianTSRB benchmark), resize them to 32×32, and run them through our trained model without any further fine-tuning. That will give us an unbiased sense of how well our system generalizes to different sign designs, lighting conditions, and backgrounds.

4 Baseline Model

Our classical baseline uses a Histogram-of-Oriented-Gradients (HOG) feature extractor paired with a linear Support Vector Machine (SVM). Each input image is first denormalized and converted from tensor to NumPy format, restoring the RGB pixel values to the [0, 255] range. HOG features are extracted independently for each of the three RGB channels, then concatenated to form a single flattened feature vector. These descriptors are passed into a linear SVM classifier trained using scikit-learn's LinearSVC, requiring minimal tuning and no GPU acceleration.

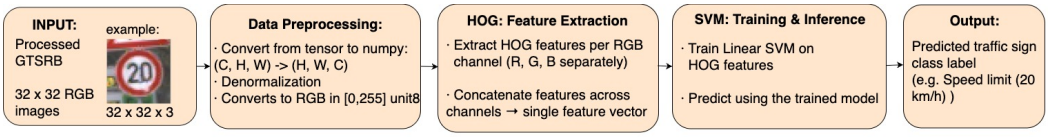

Figure 5: Baseline Model

Quantitatively, the HOG+SVM model achieves an overall classification accuracy of 89.1% on the GTSRB test set (Figure 6). The macro-averaged F1-score is 0.87, with particularly strong performance on clearly distinguishable signs as shown in the Classification Report (Figure 7). On the validation set, the model reaches a similar accuracy of 89.5%, suggesting good generalization. However, the model underperforms on less frequent or visually ambiguous classes, such as Class0 (Speed limit (20km/h)) and Class5 (Speed limit (80km/h)), as seen in the confusion matrix (Figure 8).

```
Extracting training features...
Training: 100%|| 2117/2117 [01:13<00:00, 28.74it/s]
Extracting validation features...
Validation: 100%|| 184/184 [00:06<00:00, 30.59it/s]
Training SVM...

RESULTS
==================
 **Training Accuracy**:   0.598 (59.8%)
 **Validation Accuracy**: 0.895 (89.5%)
 **Training Samples**:    67,725
 **Validation Samples**:  5,872
 **Classes in Train**:    43/43
 **Classes in Val**:      43/43

Testing baseline model...
Testing: 100%|| 185/185 [00:05<00:00, 30.89it/s]**Test Accuracy**: 0.891 (89.1%)
```

Figure 6: Baseline: Validation and Test Accuracy

Visual inspection of the confusion matrix highlights consistent misclassification between semantically similar signs, particularly among speed limit signs (e.g., 30 km/h vs. 50 km/h) and warning triangles with similar shapes. These signs share color schemes and general structure, but differ in small text or icons, which HOG struggles to differentiate. This suggests a limitation in HOG's reliance on local gradient patterns and lack of contextual understanding.

Classification Report:

	precision	recall	f1-score	support
0	0.92	0.38	0.53	32
1	0.77	0.79	0.78	333
2	0.81	0.84	0.83	338
3	0.85	0.83	0.84	213
4	0.94	0.97	0.95	297
5	0.78	0.71	0.74	279
6	0.97	0.97	0.97	63
7	0.81	0.80	0.81	217
8	0.81	0.79	0.80	213
9	0.92	0.88	0.90	221
10	0.96	0.98	0.97	302
11	0.82	0.92	0.87	199
12	0.99	0.99	0.99	315
13	1.00	1.00	1.00	324
14	1.00	0.97	0.98	117
15	0.81	0.97	0.88	95
16	0.90	0.95	0.92	63
17	0.97	1.00	0.98	167
18	0.88	0.97	0.92	180
19	0.88	0.66	0.75	32
20	0.77	0.75	0.76	55
21	0.90	0.69	0.78	51
22	0.85	0.78	0.81	59
23	0.91	0.87	0.89	77
24	0.81	0.71	0.75	41
25	0.85	0.93	0.89	225
26	0.86	0.84	0.85	90
27	0.89	0.69	0.78	36
28	0.90	0.69	0.78	81
29	0.91	0.76	0.83	41
30	0.82	0.75	0.78	68
31	0.86	0.98	0.92	117
32	0.95	0.97	0.96	36
33	0.94	0.97	0.95	104
34	0.95	0.89	0.92	63
35	0.97	0.99	0.98	180
36	0.93	0.95	0.94	59
37	0.96	0.84	0.90	32
38	0.97	0.97	0.97	311
39	0.92	0.98	0.95	45
40	0.94	0.82	0.87	55
41	0.89	0.92	0.90	36
42	0.94	0.92	0.93	36
accuracy			0.89	5898
macro avg	0.89	0.86	0.87	5898
weighted avg	0.89	0.89	0.89	5898

Figure 7: Baseline: Per-Class Precision, Recall, and F1 Scores

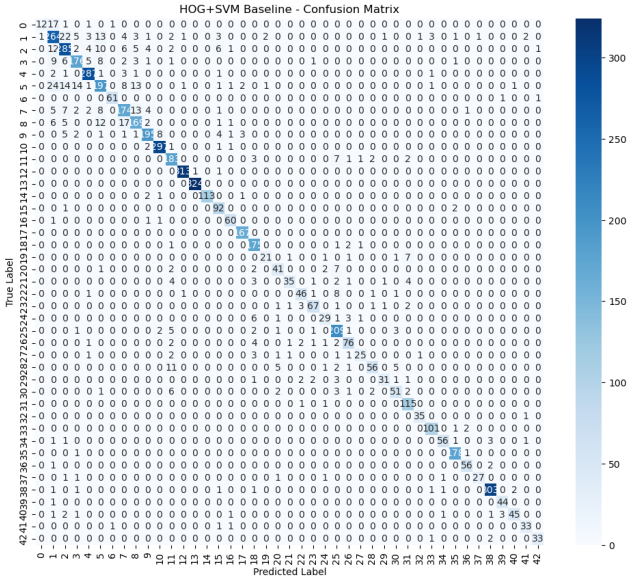

Figure 8: Baseline: Confusion Matrix

Despite its simplicity, the baseline serves as a useful performance floor and a benchmark to measure improvements from learned representations. Its primary strength lies in its ability to classify images with strong edge definitions and clear geometric features. However, the model lacks robustness to variations in lighting, occlusion, and intra-class similarity. These challenges motivate the use of deep convolutional architectures, which are better suited for learning hierarchical, task-specific features directly from data.

5 Primary Model

5.1 Model Architecture & Training Setup

Our primary classifier is a three-block convolutional neural network designed to balance learning capacity with computational efficiency. Each block applies a 3×3 convolution (with padding to preserve spatial resolution), followed by a ReLU activation and a 2×2 max-pooling operation. The first block maps the 3-channel input to 32 feature maps, the second expands 32→64, and the third 64→128. This progression lets the network first capture simple edge and corner information, then increasingly abstract higher-level shapes and patterns needed to tell apart 43 different traffic-sign classes.

After the third pooling layer, the feature map is 128×4×4 in size, which we flatten into a 2048-dimensional vector. That vector passes through a 256-unit fully connected layer (with ReLU and 50% dropout for regularization), and finally a 43-unit output layer (one neuron per class) whose scores are fed into a cross-entropy loss. We train using the Adam optimizer (learning rate = 0.001),

chosen for its fast convergence and adaptive per-parameter updates, and use cross-entropy because it directly penalizes incorrect class probabilities in multi-class settings.

We selected this architecture for three main reasons: first, using three small conv–pool blocks yields a sufficiently large receptive field to cover most of the 32×32 input without inflating the parameter count, which keeps both training and inference rapid even on a dataset exceeding 50,000 images; second, stacking multiple 3×3 filters is far more parameter-efficient than employing larger kernels, and the successive pooling layers support a hierarchical learning process that evolves from detecting simple edges to recognizing complex sign shapes; and third, incorporating dropout in the fully connected layer provides essential regularization against overfitting—particularly for rarer classes—while max-pooling inherently supplies translational invariance, improving the model's robustness to small shifts and rotations in the appearance of traffic signs.

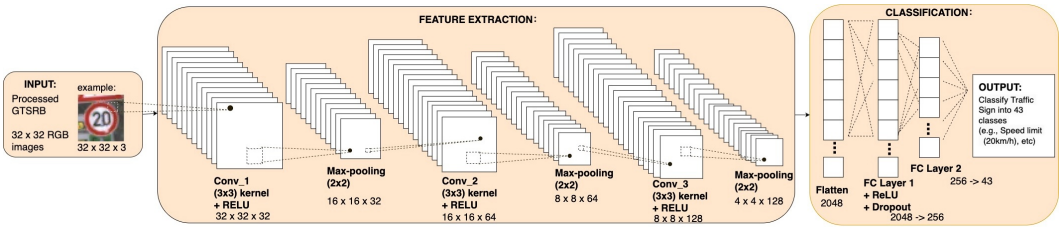

Figure 9: Primary Model

5.2 Training & Validation Performance

We trained for 20 epochs on a 70/15/15 train/val/test split with batch size 68 and learning rate 0.001. Key metrics:

$$\begin{aligned} \text{Epoch 1:} \quad & \mathcal{L} = 2.5460, \quad \text{Train Acc} = 58.3\%, \quad \text{Val Acc} = 82.9\%, \\ \text{Epoch 10:} \quad & \mathcal{L} = 0.5832, \quad \text{Train Acc} = 89.3\%, \quad \text{Val Acc} = 98.9\%, \\ \text{Epoch 20:} \quad & \mathcal{L} = 0.4015, \quad \text{Train Acc} = 94.9\%, \quad \text{Val Acc} = 98.9\%. \end{aligned}$$

As shown in Figure 10, validation accuracy rises swiftly alongside training accuracy, plateauing at $\approx 98.9\%$ by epoch 10. The final model also achieves $\sim 98.9\%$ on the held-out test set, indicating strong generalization and no extreme over- or under-fitting.

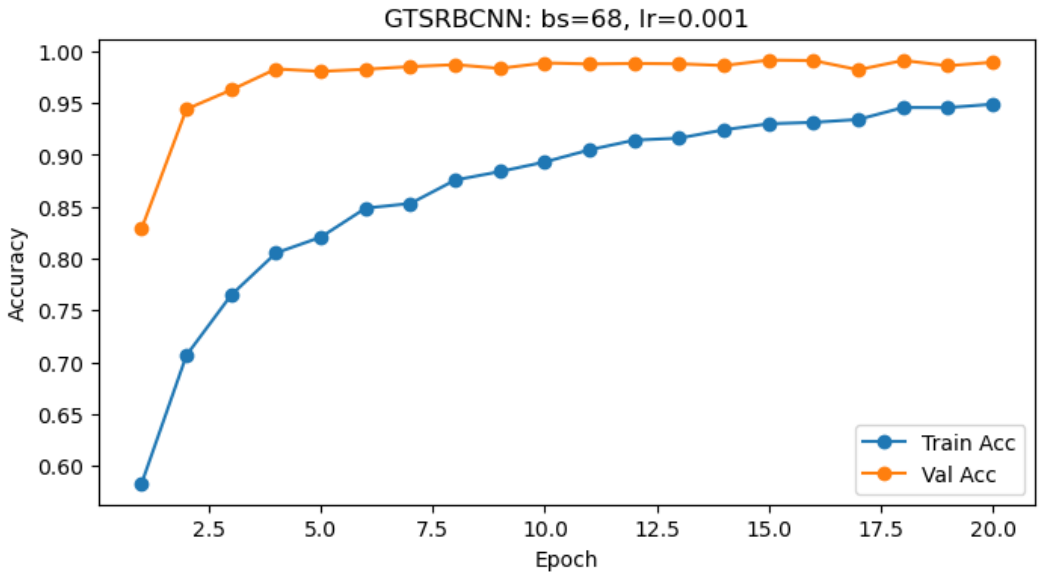

Figure 10: Training and Validation Accuracy of Primary Model

5.3 Challenges & Limitations

Our three-block CNN's shallow design potentially did not fully captured the fine-grained visual cues required for successful classification, making it difficult to distinguish between 43 subtle distinct traffic-sign classes, especially those with almost similar shapes or colors. Furthermore, we have not yet to use transfer learning by fine-tuning a pretrained backbone (such as ResNet on ImageNet), which would have hastened training convergence and given the network richer, more generalizable feature representations. Every end-to-end training cycle with approximately 68000 photos takes many hours on our current hardware, which significantly restricts our capacity to systematically study key hyperparameters, including learning rates, dropout probability, batch sizes, and optimizer methods.

Additionally, to improve resilience and reduce class imbalance, we use as needed data augmentation (random translations, rotations, and brightness jitter). However, this method adds another 30 to 50% to per-epoch calculation, which further strains our compute budget. When combined, these elements lead to less-optimized models, slower development iterations, and ultimately poorer classification accuracy on the most challenging sign pairings.

6 Conclusion and Next Steps

So far, we have developed a complete traffic sign recognition system, including dataset preprocessing, augmentation to address class imbalance, and the implementation of two classification models: a classical HOG+SVM baseline and a tailored CNN as our primary model. The baseline model provided a solid starting point with 89.5% validation accuracy, while the primary model significantly improved performance, achieving a high accuracy of $\tilde{9}8.5\%$. These results demonstrate the effectiveness of deep learning in handling complex image classification tasks like traffic sign recognition.

In future work, we plan to enhance our model by incorporating transfer learning with pretrained architectures, exploring learning rate schedulers, and using smaller proxy datasets to accelerate hyperparameter tuning. With these improvements, we aim to increase both accuracy and generalization of our model, as well as addressing current limitations.

Although this is a course project, our work aims to contribute meaningfully to the broader goal of safer autonomous and assistive driving through reliable traffic sign recognition.We are confident in our current progress and are on track to complete the remaining tasks according to our project plan in the coming weeks.

References

Jan Salmen Shivank Patel Johannes Stallkamp, Marc Schlipsing. German traffic sign recognition benchmark gtsrb, 05 2019. URL https://sid.erda.dk/public/archives/daaeac0d7ce1152aea9b61d9f1e19370/published-archive.html.

Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The German Traffic Sign Recognition Benchmark: A multi-class classification competition. In *IEEE International Joint Conference on Neural Networks*, pp. 1453–1460, 2011. URL https://ieeexplore.ieee.org/document/6033395?arnumber=6033395.

1

[1]The reference list is organized using iclr2022_conference format.