GTSRB Traffic Sign Classification CNN

Alonso Bravo Villanueva - Fresno State - CSCI 167

Goal

Traffic Sign Recognition is becoming increasingly important with the increasing electric and modern vehicles and their autonomous/driver assistance features. Our goal is to achieve the highest possible accuracy on the German Traffic Sign Recognition Benchmark (GTSRB) dataset.

Dataset Exploration

The dataset used was the German Traffic Sign Recognition Benchmark Dataset from Kaggle. This is a multi-class image classification dataset. It contains a number of 58 classes, in my case I reduced it to 30 classes.

Training and Testing sets

The training set contains 2607 images categorized into 30 traffic sign classes. The testing set contains 1200 images used for evaluation.

Feature Analysis

The images are inputted in RGB format, but normalized between pixel values 0 to 1. This helps for consistency and convergence when training. The labels are converted into one-hot encoded vectors to achieve model compatibility. The class names are simply a short description of the traffic signs, example: “No U-turn”. The bar chart below illustrates the number of images per class, showcasing class imbalances.

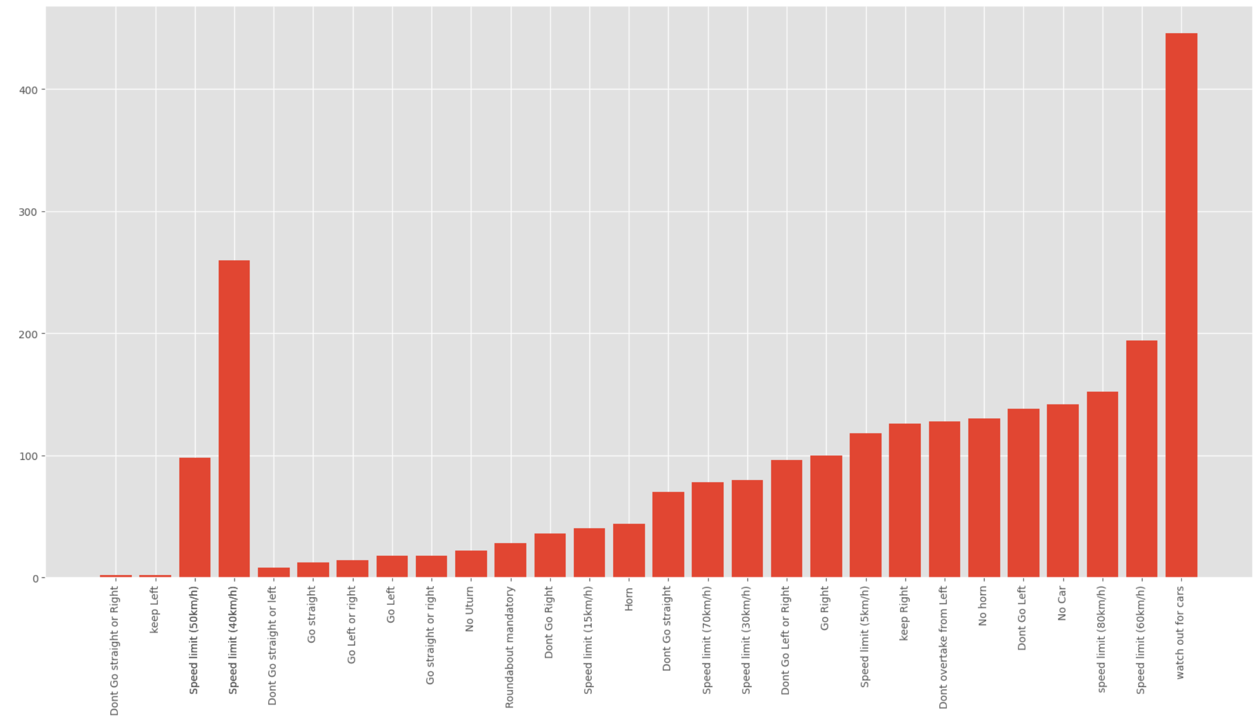


Image Preprocessing

All of the images are resized to 32x32 dimensions. Pixel values are normalized to a range of 0 to 1 for consistent scaling. Labels are mapped to their corresponding categories and are one-hot encoded.

Training Data Diversity

We introduce several variables of diversity. Images are randomly rotated up to 10 degrees in either direction. Images are randomly zoomed up to 15%. Images are randomly shifted up to 10% horizontally and vertically. Images are not mirrored as that will change the integrity of the data.

Model Creation

I chose to go with a Sequential Convolutional Neural Network as it is the best for dealing with image data. This project is a multi-class image classification project, in which a CNN outperforms all other types of deep learning models. It is also a sequential linear model. I used Tensorflow and Keras as the two main technologies for this project.

Convolutional Layers (4 Layers)

This model extracts low level features such as edges and textures. The conv2D layers contain filters: 16, 32, 64 and 128 along with the ReLU activation function.

Pooling Layers (2 Layers)

MaxPooling is used to reduce spatial dimensionality.

Batch Normalization Layers (3 Layers)

These layers stabilize learning and they accelerate convergence.

Dense Layers (2 Layers)

Fully connected layer with 512 Neurons used for feature combination.

Flatten Layer

Converts the multidimensional outputs so far, into a 1D layer. Helps with retaining chanel information.

Dropout Layer

Dropout layer containing a dropout rate of 50% for regularization.

Output Layer

Another Dense layer with 30 neurons to match the number of classes in my dataset. It uses the softmax activation function to compute probabilities for all 30 classes.

Total Number of layers

Thirteen total layers are used to train this model.

Model Summary

This model has a total number of 5,260.604 parameters. It contains 1,753,086 total trainable parameters and only 1,344 non-trainable parameters.

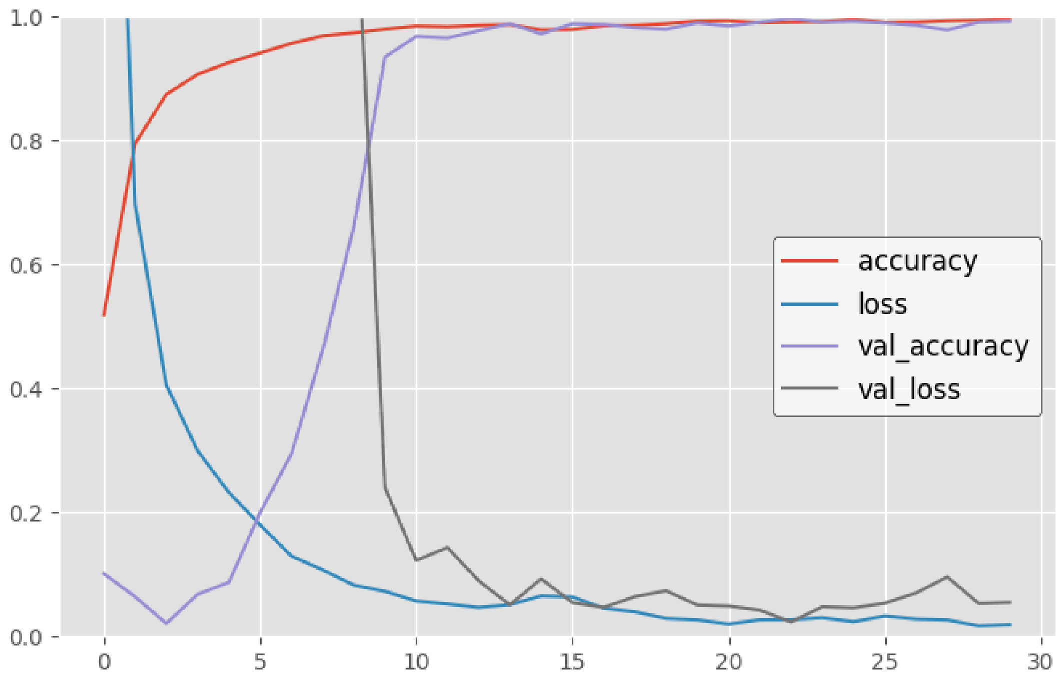
Hyperparameter Exploration

Several hyperparameters were fine tuned when optimizing this model and its performance. The optimizer used is Adam, the best learning rate was 0.001. These two hyperparameters balanced the convergence speed and stability. We used a batch size of 32 to ensure sufficient gradient updates without overpowering the memory of the system. The model underwent 30 epochs of training. 30 epochs was plenty for training, more would cause overfitting.

Early versions of the model gave me an average result of 66% accuracy. After tuning the hyperparameters I was able to achieve an average of 81% accuracy. Plenty of overfitting was minimized with dropout and batch normalization.

Results

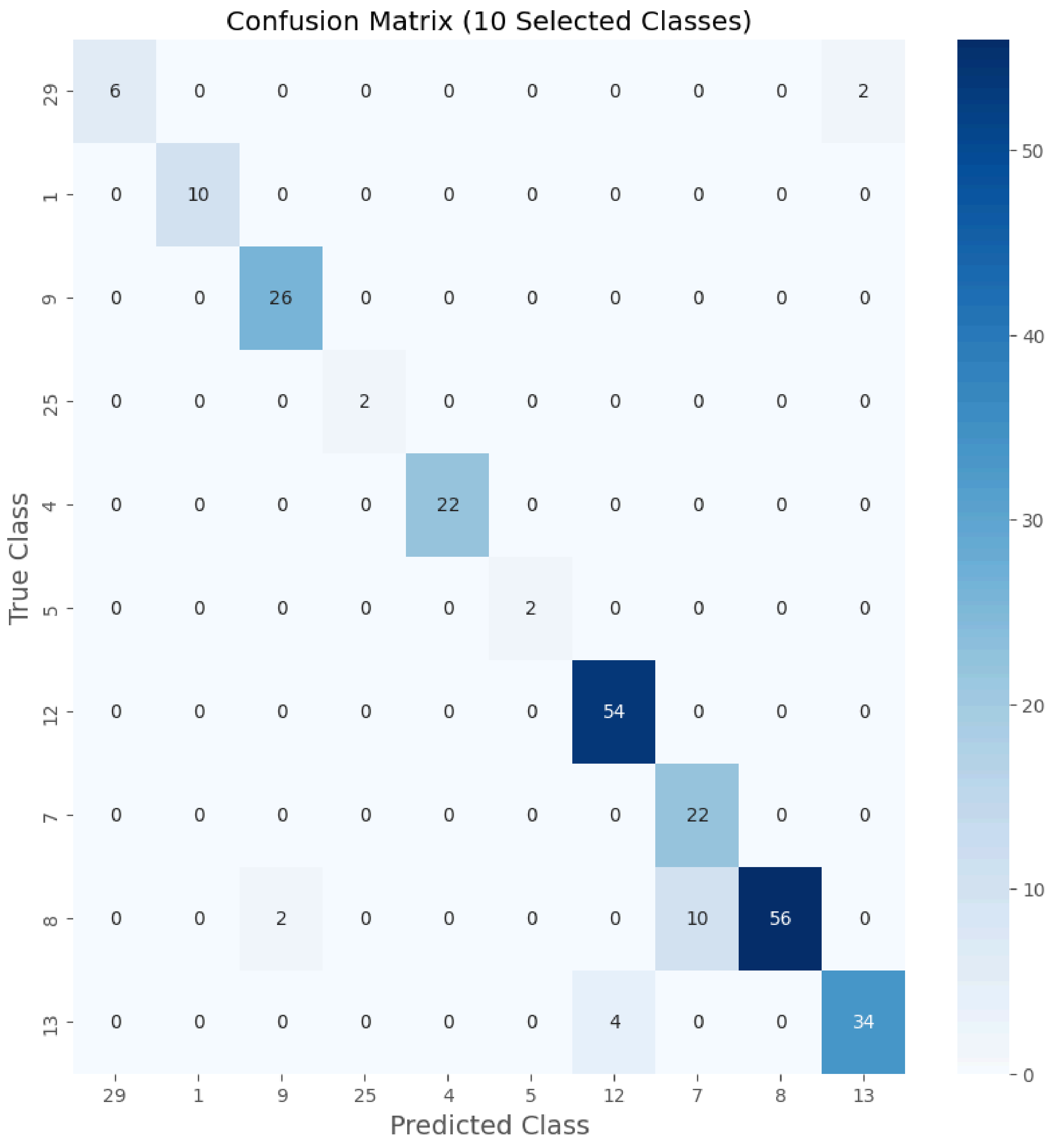
This model underwent comprehensive training and testing phases. During its training and validation phase, the model received a validation accuracy of ~99%. Its validation loss was only about 0.042. In the testing phase, it received an average of 81% accuracy.



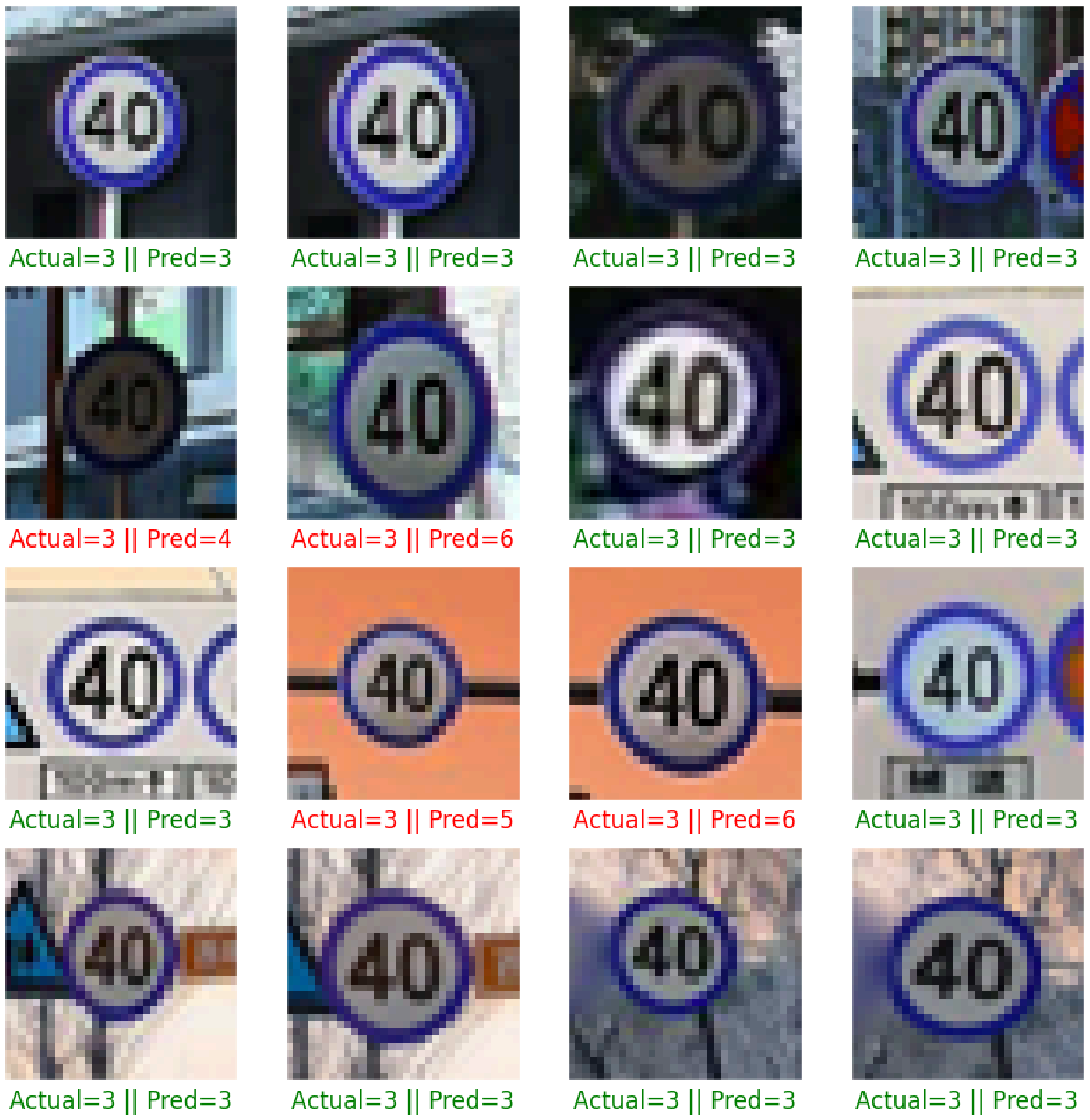
Averages

Its weighted average accuracy was higher due to the inconsistency in images per class. Class 28 labeled “Watch out for cars” contained almost 500 images, while the second largest class only contained about 255 images. These two classes are outliers in the amount of data they contained, all other 28 classes remain under 200 images.

	Precision	**Recall**	**F1-Score**
Macro Average	77%	80%	0.74
Weighted Average	88%	81%	0.81



Confusion Matrix of 10 Random classes.



Here we have 16 predictions in which the green is true and the red is false.

Proof of Concept Implementation

I created a python script that takes the trained model and my laptops webcam to predict signs. I then took some images from the testing data and printed them out on paper. This does not record metrics, it is simply a proof of concept. I notice it works so long the image is taking up the entire frame. I there is background objects, it tends to not be accurate.

Areas of Improvement

A more robust model can be used in autonomous vehicles for autopilot features. It can also be used in modern vehicles for driving assistance. I believe the best area for improvement is in the data and image pre-processing. I believe there are better datasets than the GTSRB. Having a more robust or pre-trained model such as the ResNet can in combination with a high quality and consistent dataset can greatly improve feature extraction and accuracy. 80% accuracy is unsafe when you are in a vehicle making decisions for you.