Traffic Sign Classification

Hasan Isgandarli
Department of Process Automation Engineering
Baku Higher Oil School
Baku, Azerbaijan
hesenisgenderli999@gmail.com

Elmaddin Mahmudov
Department of Process Automation Engineering
Baku Higher Oil School
Baku, Azerbaijan
elmaddin.mahmudov.std@bhos.edu.az

I. Introduction

The goal of this project is to achieve classification of different traffic signs, as given in the Traffic Sign Dataset from Kaggle.

All the code written for this project, along with the weights of the trained models are uploaded to the GitHub repository that we created.

II. Tasks

A. Data Preprocessing

Upon the initial inspection of the dataset, one can clearly see that every image in the dataset has a duplicate, which has `"_1_"` in the file name. The script `dataset_removeduplicates.py` handles the removal of these images.

Next, we can inspect the `TEST` directory to see that certain classes have no test images, making it impossible to test our model's performance on these classes, and therefore we agreed that there is no point in including them in the training batch. The script `dataset_removezerotestcases.py` handles removing these images from the dataset. The removed classes were `9, 18, 19, 33`, reducing our total class count from 58 down to 54.

Next, checking the class statistics, one can see that there is a significant class imbalance. In order to handle this, we have run the script `dataset_augmentate.py`, which applies random rotation and perspective transform to the images. The class created to augmentate images also includes a deaugmentation method, which basically restores the dataset to the original state. Figures 1 and 2

Finally, the data for the training was not split into training and validation batches. The script `dataset_split.py` handles this issue, and performs a 80-20 split.

B. Augmentation Techniques

The augmentation techniques that could be used for this dataset was extremely limited. We could only do rotation up to a small degree, and a small amount of perspective transformation, due to the nature of the dataset. Traffic signs can be misinterpreted if flipped or rotated too much, either become completely unrecognizable, or be mistaken for a completely different traffic sign. We would not want to misclassify a "turn left" sign for a "turn right" sign.

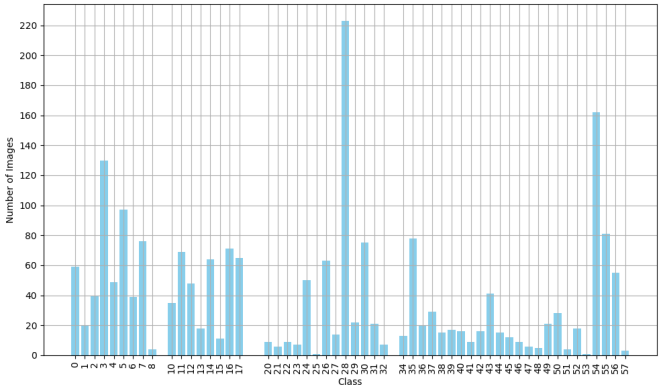

Fig. 1. Class distribution before augmentation

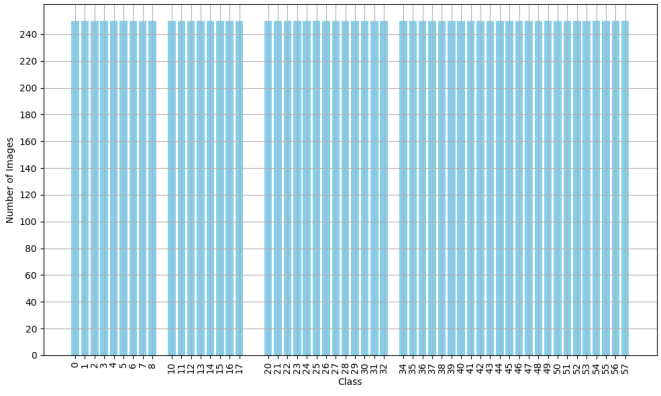

Fig. 2. Class distribution after augmentation

C. Define CNN architectures

The feature extractors from ResNet18 and VGG16 were used in the `ResnetModel.py` and `VGGModel.py` files in respective classes. We modified the classifier of the two architectures by adding a linear layer at the end that has the input shape similar to the model's original output shape, and the output shape that corresponds to the number of our classes.

D. Difference between architectures

VGG16 is a simple architechture with convolutional and max-pooling layers. As simple it is, it also has a vanishing gradient problem, which ResNet18 solves by implementing skip

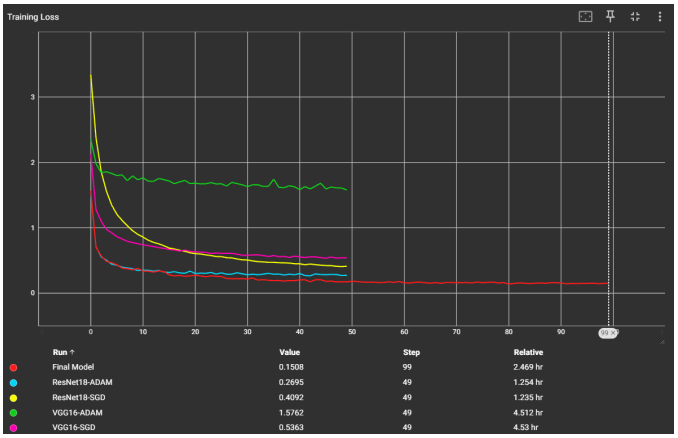

Fig. 3. Training Loss

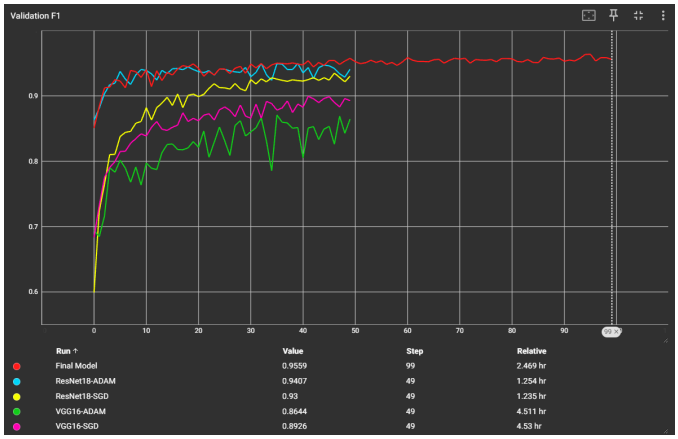

Fig. 4. Validation F1 Score

connections or shortcuts, which allow the gradients to flow more directly in the training phase. ResNet18 also achieves a better and more efficient performance in general.

E. Difference between optimizers

SGD (Stochastic Gradient Descent) is a traditional optimization algorithm that updates model parameters by computing the gradient of the loss function. Usually converges slower. Adam (Adaptive Moment Estimation) on the other hand, is an algorithm that computes individual adaptive learning rates for different parameters. Combines the advantages of AdaGrad and RMSProp, and that is why it is most commonly used in Neural Networks. Converges significantly faster than SGD.

F. Train the models

The loss function was taken as Cross-Entropy loss function, and optimizers were SGD and Adam. Learning rates for all cases were set to `0.001` and the training was run for 50 epochs. According to the theory, we are expecting the ResNet18 model with Adam optimizer to converge fast, which is the case, as seen from the plots.

After the graphical evaluation of performance, we also added the `ReduceLROnPlateau` callback from PyTorch with patience of 5 epochs and factor of 0.7 to the best performing model, and ran it for 100 epochs for a better result. Figures 3, 4 and 5 show the Training Loss, Validation F1 Score and Validation Loss metrics for 5 different model configurations.

Looking at the metrics in the graphs, one can say that the best performing backbones are all ResNet18. F1 scores of the models on training, validation and test batches are given in the Table I

The reason for inclusion of the ReduceLROnPlateau was because we observed that all models stop improving after some time, and the losses just start oscillating. This is usually due to the fact that the learning rate was set too much, and we should decrease it when we start going back and forth around the global minimum. Additionally, Table I reveals that although the models perform good on training and validation data,

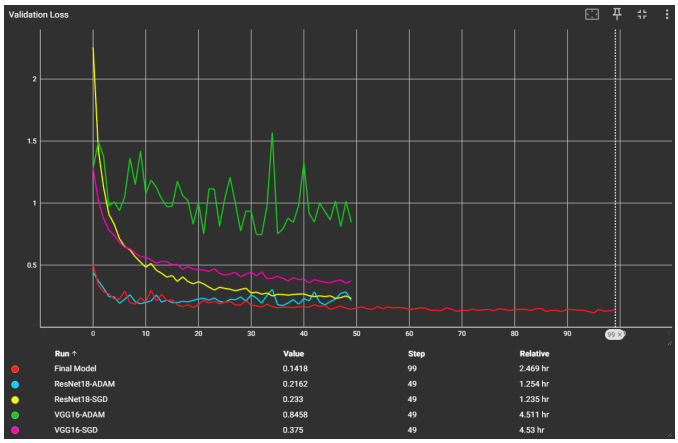

Fig. 5. Validation Loss

their F1 scores on the unseen Test data is not that promising. This is probably due to the fact that the dataset is not in a good condition. The augmentation of classes with 1 training image results in overfitting on that image, regardless of the quality and variety of augmentations. It is always preferred that the dataset itself already has enough training images, and augmentation only adds a small bit for balancing.

III. Conclusion

In this project, we took the challenge of traffic sign classification using two different CNN architectures. For feature extraction from the images, we used the ResNet18 and VGG16 architectures, adapting their classifiers for our specific

TABLE I
Model Comparison

Model	Optimizer	Epochs	Train F1	Val F1	Test F1
VGG16	SGD	50	0.9169	0.8926	0.4845
VGG16	Adam	50	0.8896	0.8644	0.4122
ResNet18	SGD	50	0.9481	0.93	0.5777
ResNet18	Adam	50	0.9708	0.9407	0.6008
ResNet18	Adam	100	0.9828	0.9559	0.6409

classification task by modifying the final fully connected layers. We trained these models using the Cross-Entropy loss function with two different optimizers, SGD and Adam, over 50 epochs. Then additionally, we picked the best performing model, improved it with ReduceLROnPlateau, and achieved even better result.

Analysis of the models' performances shows that ResNet18 outperformed VGG16. However, while our models display good performance on training and validation data, their performance on unseen test data was less impressive. This disparity may stem from the limitations of our dataset, which is rather poorly

In conclusion, we did our best with the given dataset, but the results on the test batch was not as impressive as the results on the training and validation batches, which may correlate to the models overfitting to some extent. If there was more time for the project, a good consideration could have been collection of more data for the given classes.