Traffic Sign Classification Using Convolutional Neural Networks (CNNs)

Authors

Sahir Doshi and Yiran (Elaine) Wang

Introduction

Image classification is a machine learning task that is crucial to many systems in the modern day. Its use is employed in a plethora of areas, such as medical imaging, facial recognition, and more. One area where there has been particular focus and scrutiny is in the emerging field of autonomous electric vehicles (EVs). The idea of a self-driving car has been rooted in pop culture for a long time. As technology advanced, what was once a theory became reality. Today, EVs have carved a significant share in both the automotive and technology industries.

For our project, we wanted to explore the traffic sign detection system in autonomous EVs. We believed this application of image classification was relevant and challenging. After learning about image classification fundamentals through our assignment in which we classified dogs and cats, we were inspired to see exactly how well we could fine-tune a model to detect traffic signs correctly. Specifically, the goal of our project was to deduce which preprocessing function(s) were most impactful on a base traffic sign detection model, measuring impactfulness by model accuracy.

Methodology

Our methodology can broadly be categorized into these sections:

1) Data Setup
2) Model 1 - CNN with Data Augmentation Only
3) Model 2 - CNN with Grayscale Preprocessing Only
4) Model 3 - CNN with Grayscale Preprocessing and Data Augmentation
5) Model 4 - CNN with Grayscale Preprocessing and Data Augmentation with Random Noise

6) Model 5 - CNN with Grayscale Preprocessing and Data Augmentation with Random Noise and Fill Mode
7) Model 6 - CNN with Transfer Learning using VGG16 Network

Data Set Up

Our data set up involved many steps, including importing packages, loading in the data, and training hyperparameters. Broadly speaking, we imported six main packages: NumPy, Matplotlib, CV2, OS, Pandas, Random, and TensorFlow Keras. After loading in the traffic sign labels (.CSV), we set up a handful of hyperparameters such as the test, train, and validation ratios and the image dimensions. Next, we loaded in traffic sign images, processing 58 classes. We manually searched for classes that are labeled "unknown" and inserted the names of these traffic signs. Finally, we set up a few more hyperparameters regarding the number of samples, steps per epoch, and the number of epochs, and these were also tuned based on later performances.

We calculate the number of images for training the model by applying the specified training ratio to the total dataset, which allows us to easily adjust the size of the training set. The number of steps per epoch is then determined based on the number of training samples divided by the batch size.Also, by using np.ceil, we ensure that every training sample is used, including those in the final batch, which might be smaller than the batch size. This approach means that we only need to fine-tune the training ratio, and the model will automatically handle the rest, maximizing the use of our data for training.

In data splitting, we also set the "shuffle = True" which shuffles the data before each training epoch, to ensure that the model does not learn anything from the order of the samples.

In addition to the above set up, the loss function has been set to categorical cross entropy for all models for multiple class categories. For all models, both the test score and test accuracy were determined to deduce the magnitudes of the model's loss function and the model's performance. The target variable datasets (train, validation, and test) were all one-hot encoded. Lastly, all models were built with the following architecture in a specific order:

- 3 Convolutional Layers
- 2 Pooling Layers
- 2 Dropout Layers (both with rates of 0.5 is used to help prevent overfitting)

- Flatten Layer
- 2 Dense Layers
- Output Layer

Moreover, we also experimented with various techniques to further enhance our models' performance. We explored image pre-processing techniques like lighting normalization and histogram equalization to make our traffic sign images less affected by changes in lighting, to mimic those caused by different weather conditions. These attempts did not yield the desired improvement in accuracy, possibly due to the models becoming over-sensitive to subtle lighting changes or it is possible that it lost the critical information in shadows and highlights. But all these experiments provided us with deeper insights into the balance of feature preservation and normalization.

Results

We explored how different data preprocessing and augmentation techniques influence the performance and generalization abilities of image classification systems. Each variation of the models was specifically designed to assess certain facets of model training and to enhance overall performance. The motivation behind experimenting with a range of models was to methodically analyze each technique's contribution to improving model accuracy, managing various data variations, and ensuring robustness and efficiency.

Model 1 - CNN with Data Augmentation Only

The aim for this model was to understand how data augmentation impacted the accuracy of a model. Data augmentation artificially expands by generating variations of the existing images, assisting in the improvement of model generalization. Our augmentations included shifts in width, height, zoom, shear, and rotation. After augmenting the data, the data was run through the preprocessing function to a standardized size and normalizing its pixel values for it can be fed into the model by resizing it.

Model 2 - CNN with Grayscale Only

The aim for this model was to understand how adding grayscale to data preprocessing impacted the accuracy of a model. Grayscale has a number of impacts, including dimensionality reduction (RGB → black and white), feature simplification (greater focus on

the intensity of lightness/darkness), and higher model training efficiency (less input per pixel). After data partitioning, the grayscale preprocessing function was applied to all datasets (train, validation, test), and then the model was built.

Model 3 - CNN with Grayscale and Data Augmentation

After establishing models using grayscale and data augmentation individually, we were curious to see the compounded effects on a model. Post-data partition, the data was first augmented as before, and then preprocessed using grayscale.

Model 4 - CNN with Grayscale and Data Augmentation with Random Noise

To explore how image classification models can grow their robustness, we decided to add random noise to the grayscale-augmented model. By training with random noise, the model should be able to generalize. It also helps with reducing overfitting and simulates real-world scenarios better.

Model 5 - CNN with Grayscale and Data Augmentation with Random Noise and Fill Mode

Additionally to Model 4, we tried to apply a parameter of autofill in ImageDataGenerator. When data is augmented, there may be new, unfilled pixels (i.e., the image is rotated 90 degrees). To quickly fill in these pixels without altering the original image, we added the fill_mode parameter. This parameter was set to nearest, which fills a pixel with the value of that pixel's closest neighbor.

Model 6 - CNN with Transfer Learning using VGG16 Network

In the last model we tried, we used VGG16 as the base model for transfer learning. This base is popular for image classifications and has learned a robust hierarchy of features, which can be beneficial for similar problems with different datasets. We froze the base layers and applied custom layers which are flatten layers followed by dense layers on the top of the model. We leveraged the pre-trained features to handle the complexities of image recognition while adapting to the specifics of our own dataset. The accuracy was about 99%, which proves that this approach is effective when we adapt this model to our task. Notably, in this model, the accuracies on both the training and validation sets showed consistency, underscoring the model's robust generalization capabilities.

Some result using the model of Transfer Learning:

Summary of Model Results

Model	*Test Score*	*Test Accuracy*
CNN with Data Augmentation	0.1905	0.9330
CNN with Grayscale	0.0293	0.9928
CNN with Grayscale and Data Augmentation	0.1748	0.9545
CNN with Grayscale and Data Augmentation with Random Noise	0.2316	0.9306
CNN with Grayscale and Data Augmentation with Random Noise and Fill Mode	0.24042	0.93301
CNN with Transfer Learning using the VGG16 Network	0.0343	0.9900

It is worth mentioning that when comparing the model that employed data augmentation alone with the one that combined data augmentation and grayscale preprocessing, the latter shows an

improvement. This enhancement can largely be attributed to the grayscale preprocessing, which simplifies the images by reducing them to a single intensity channel.

Conclusion

Our most accurate model was a CNN with Grayscale Preprocessing. The grayscale function reduces the complexity of the model by downgrading the number of dimensions from three (RGB) to two (black and white). The elimination of color lets the model narrow its focus on more relevant features, such as traffic sign shapes and textures. Furthermore, color images can vary in quality and accuracy depending on lighting and weather conditions. If a CNN were to be heavily dependent on color for image classification, its performance could be negatively impacted if given a clear dataset. Furthermore, our CNN utilizes two dense layers instead of one, enhancing its ability to learn. The first dense layer captures high-level features, and the second layer transforms these into final outputs. This division allows the model to refine its understanding through progressive steps, promoting better learning and generalization.

Potential Limitations: While the use of grayscale images does influence feature representation, the high performance of our CNN might be due to various uncovered factors. The observation of unusually high validation accuracy compared to training accuracy can be attributed to several potential issues. One possibility is an imbalanced dataset, where the model predominantly learns to predict the majority class. Alternatively, the simplicity of the dataset might lead to easier training, potentially causing the model to overfit narrowly to the training data. These factors necessitate a cautious interpretation of the model's performance metrics. However, when applying transfer learning with the VGG16 network, we observed consistent accuracy between training and validation scores. This consistency indicates robust model generalization, mitigating concerns of overfitting and validating the effective adaptation of the model to our dataset. Such factors underline the necessity for a cautious interpretation of the model's performance metrics, ensuring that we acknowledge both the strengths and limitations of our approach.

In conclusion, we believe that our project has only scratched the surface of image classification models. There are a plethora of models we can add to our report given the number of hyperparameters involved in convolutional neural networks. This project served as an enriching experience to understand autonomous electric vehicles at the technical level, and it is always inspiring to see concepts learned in the classroom be applied in the real world.