Transfer Learning Based Traffic Sign Classification Using VGG16 Network

Amy Reidy

Machine Learning COMP09012, Institute of Technology Sligo

Abstract

As vehicles become more autonomous, there is an increasing demand in the automobile industry for driver assistance systems that can identify and classify road signs quickly and accurately. This project aims to create a classifier, using a pre-trained convolutional neural network, that can effectively classify images from the German Traffic Sign Recognition Benchmark dataset. The convolutional base of the popular VGG16 model was used for feature extraction while the top fully connected layers were retrained with the GTSRB dataset and the hyperparameters were fine-tuned to further improve the model. The results show that the best performing model in this experiment achieved an accuracy of 82.98%. As the dataset is quite imbalanced, future work could improve on this result by augmenting the images in the smaller classes to create a more balanced dataset.

Keywords: Deep Learning, Convolutional Neural Networks, Image Recognition, Transfer Learning, VGG16.

1 Introduction

Automatic traffic sign recognition is a challenging real-world image classification problem that is becoming even more prominent as the demand for driving assistance systems increases. The goal of this project was to use a pre-trained neural network as a starting point to create a model that can accurately classify traffic signs. Convolutional neural networks (CNNs) are often used for image recognition and classification as they outperform standard artificial neural networks in tasks like these. However, creating a CNN from scratch requires a great deal of computational power and time, and it needs to be trained on a very large dataset to reduce overfitting. Thus, this project used a machine learning method called transfer learning that allows the use of a pre-existing deep learning model that was trained on a huge dataset for our own classification of a smaller, similar dataset. This project used a very popular state of the art image classification model called VGG16. This network was proposed by Simonyan and Zisserman (2014) from Visual Geometry Group in the paper "Very Deep Convolutional Networks for Large-scale Image Recognition", and it was one of the winners in the 2014 ImageNet Large-Scale Visual Recognition Challenge (ILSVR).

2 The Dataset and Pre-processing

The dataset used for this project is called the German Traffic Sign Recognition Benchmark, and it was used in a multi-class, single image classification challenge of the same name at the 2011 International Joint Conference on Neural Networks (IJCNN) (Stallkamp et al., 2011). It contains over 50,000 images of traffic signs, including more than 12,500 images for testing. There are 43 classes of traffic signs, and the dataset is quite imbalanced, with the largest class having around 2000 images and the smallest classes only having about 200 images. The images vary in size and as they were taken in real life conditions, the quality of the images vary too due to weather conditions, blurring, poor illumination, etc.

The main training data directory was randomly split into two subset directories: 80% of the images were assigned to training and 20% to validation. Originally, a Keras data generator was used to split the data into training and validation subsets, however this led to more overfitting compared to manually dividing the data into two directories with a coded function. The images were pre-processed using the Keras VGG16 preprocess_input function. This function converts inputs from RGB to BGR and then zero-centres each colour channel with respect to the ImageNet dataset (which the VGG16 network was originally trained on), thus transforming the images into the format that the network requires. A data generator was used to resize the data into 224 x 224 pixels.

3 Model Architecture

CNNs get their name from convolution, which in deep learning is a linear operation where a filter is applied to an input and results in an activation (Goodfellow et al., 2016). CNNs contain an input layer, one or more hidden layers (that can be convolutional, pooling or fully connected), and an output layer. The convolutional (conv.) layers use filters to extract features from the input to create a feature map. The purpose of the pooling layers is to reduce the number of parameters of the input tensor and move from high resolution data to lower resolution information. And the fully connected layers are feed forward neural networks that form the last few layers in the network.

The VGG16 network is comprised of 13 conv. layers with 5 max pooling layers, and 3 fully connected layers at the end – the '16' in the name refers to the 16 layers with weights (Simonyan and Zisserman, 2014). The conv. layers are stacked into 5 conv. blocks, and each block is followed by a max pooling layer. The conv. layers use 3 x 3 kernel size filters with a stride of 1 (meaning the filter shifts by 1 pixel at a time). The pooling is carried out by a 2 x 2 window with a stride of 2, and the max output for each window is reported. VGG16 is a very popular model to use for transfer learning as the fully connected layers generalize well to other classifications. However, VGG16's fully connected layers were replaced for this project and only the conv. base of the network was used (these layers were frozen during training, so that they retained the original weights from the ImageNet dataset).

This project focused on training the new classification layers of the network. After the last max pooling layer, the data was flattened into a one-dimensional array, then fed into a fully connected dense layer with 512 neurons. The rectified linear unit, or ReLU activation was used in this layer. ReLU is also utilized in each conv. layer, and it is a non-linear activation function that returns 0 if it receives any negative input, and if the value is positive, it returns the value back (Agarap, 2018). The softmax activation function was applied to the second and final fully connected layer. This function is a generalization of logistic regression that can be used for multi-class classification, and it outputs probabilistic values for each of the 43 target classes in the GTSRB. Additional dropout layers were added to the network to reduce overfitting in later models during the experiment (see Section 4. for more information), and the architecture of the top performing model is summarized in Figure 1.

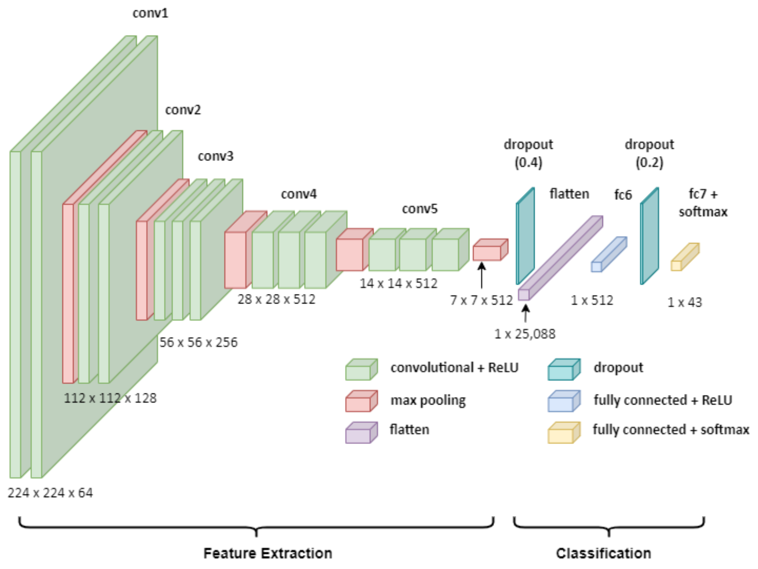

Figure 1: Architecture of Best Performing Model.

4 Model Training and Tuning

CNNs require lots of computational power (even when using pre-trained networks), so it was necessary to use a graphics processing unit (GPU) via Google Colab, and the models were implemented using Keras, which is a high-level, deep learning API that acts as an interface for Tensorflow, an open-source machine learning library.

As there is an imbalance of classes in the dataset, the main priority of training the classification layers was to avoid overfitting to create a better generalized model. The first technique to prevent overfitting during training was the use of early stopping checkpoints. This method works by monitoring a certain performance metric and stopping the training once that metric stops improving. For this project, the validation loss was monitored and after the loss had not decreased for 3 epochs, training ceased, and the best weights were restored. Categorical cross entropy was used as the loss function; this function returns the divergence between the predicted distribution of the target classes and the true distribution. To minimize the loss efficiently, the models were optimized with the Adam algorithm, which is a very popular adaptive learning rate method used in deep learning, and training started with a learning rate of 0.001.

The batch size (the number of training examples used in one iteration) was the first parameter that was tuned during training. After each batch, gradients are imputed by the optimizer and the weights are updated accordingly. The first models were trained with a range of batch sizes (15, 32, 64, 128 and 256). The most accurate results were obtained with a batch size of 256, so this size was used for all subsequent models. Next, a regularization method called dropout was used to reduce overfitting in the fully connected layers. Dropout layers work by randomly de-activating some filters in each step of the training, which forces the model to learn off only the filters that are left activated (Srivastava et al., 2014). First, a 20% dropout layer was applied directly before the softmax layer, and then in later models, a 40% dropout layer was also added before the flatten layer. For the last two models, the learning rate was reduced to 0.0001, and for the final model, class weights were added to try to further decrease the test loss, and this meant that classes with less images had a greater weight in the loss function.

5 Results

	Learning Rate	Dropout	Class Weights	Val Loss	Val Accuracy	Test Loss	Test Accuracy	Run Time
A	0.001	No	No	0.083	0.9754	0.9619	0.7972	7min 44s
B	0.001	1 layer	No	0.0904	0.9709	1.0602	0.8095	11min 17s
C	0.001	2 layers	No	0.0762	0.9786	1.1467	0.8249	17min 19s
D	0.0001	2 layers	No	0.0424	0.9885	1.1591	0.8205	35min 56s
E	**0.0001**	**2 layers**	**Yes**	**0.0348**	**0.988**	**0.8987**	**0.8298**	**34min 17s**

Table 1: Results of Top 5 Models

The results of the top five performing models are summarized in Table 1. Model E was the top performing model, and it achieved a test accuracy of 82.98% and a test loss of 0.8987. This model had two dropout layers (40% dropout before the flatten layer and 20% dropout before the softmax layer), and it was trained with a learning rate of 0.0001 and a dictionary of class weights (to penalize over-represented classes). Figure 2 shows the accuracy and loss results during the training and validation of Model E. As early stopping checkpoints were used, training stopped after epoch 28 and the weights from

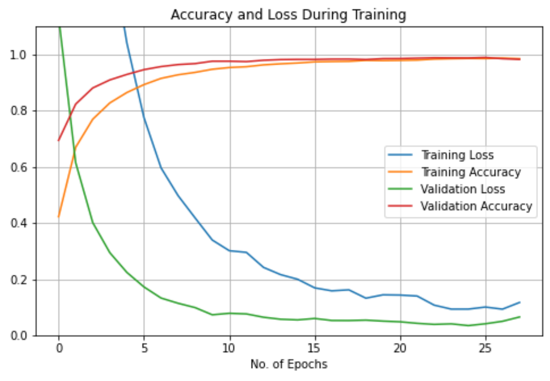

Figure 2: Loss and Accuracy for Model E Training

epoch 25 were restored as the validation loss started to increase after this iteration. We can also see from the graph that the training loss was quite high for the early epochs, more so than in other models, and this is likely due to the additional dropout layers and class weights. However, the validation curves for loss and accuracy indicate that the model generalized quite well.

6 Discussion

The results show that quite a high degree of accuracy can be achieved in a very short amount of time and with a relatively small dataset by using transfer learning. While the top 5 models have slightly different learning configurations and network architectures, these models varied more in training time than in accuracy. Adding dropout layers seemed to slightly increase the accuracy, while decreasing the learning rate had minimal effect on the accuracy. Adding class weights barely improved the accuracy but did reduce the test loss. However, even for the best performing model there is still a significant difference between the performance of the validation set and the test set, which indicates that the model has overfitted. To further improve the model's accuracy at predicting all the classes, future work could use data augmentation for just the smaller classes to generate more data and thus balance out the dataset. Another alternative would be to try to further tune the network, perhaps by unfreezing some of the convolutional layers and retraining them with the GTSRB dataset, or to experiment with another type of pre-trained CNN architecture. For example, Lin et al. (2019) achieved 99.18 % of recognition accuracy of the Belgium Traffic Sign Database using transfer learning with Google's Inception v3 model.

7 Conclusions

This project demonstrates how transfer learning can be used with a pre-trained CNN architecture, such as VGG16, to classify a smaller dataset quickly and easily. The best model in this project achieved 83% test accuracy, however the difference between the test and validation accuracy suggests that the model has overfitted, and this accuracy is still quite low for a critical task like correctly classifying road signs. For driving assistance systems, even the smallest errors can put lives at risk, and so it is recommended that future studies apply data augmentation to the minority classes in GTSRB to create a more balanced dataset and a more accurate model.

References

[Agarap, 2018] Agarap, A. F. (2018). *Deep Learning using Rectified Linear Units (ReLU)*. arXiv preprint arXiv:1803.08375.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning.* MIT Press, ISBN: 9780262337434. `http://www.deeplearningbook.org`.

[Kingma & Ba, 2014] Kingma, D. P., & Ba, J. (2014). *Adam: A Method for Stochastic Optimization*. arXiv preprint arXiv:1412.6980.

[Lin et al., 2019] Lin, C., Li, L., Luo, W., Wang, K. C., & Guo, J. (2019). *Transfer Learning Based Traffic Sign Recognition Using Inception-V3 Model*. Periodica Polytechnica Transportation Engineering, 47(3), 242-250.

[Simonyan and Zisserman, 2014] Simonyan, K., & Zisserman, A. (2014). *Very Deep Convolutional Networks for Large-scale Image Recognition.* arXiv preprint arXiv:1409.1556.

[Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). *Dropout: A Simple Way to Prevent Neural Networks from Overfitting*. The Journal of Machine Learning Research, 15(1), 1929-1958.

[Stallkamp et al., 2011] Stallkamp J., Schlipsing M., Salmen J. & Igel C. (2011). *The German Traffic Sign Recognition Benchmark: A Multi-class Classification Competition.* The 2011 International Joint Conference on Neural Networks, pp. 1453-1460, doi: 10.1109/IJCNN.2011.6033395.