Image classification on the German Traffic Sign Recognition Benchmark

Tiaan Stals[1]

[1] *University of Sydney, Camperdown, Sydney*

November 9, 2022

We build a general classifier trained for the German Traffic Sign Recognition Benchmark (GTSRB) using convolution neural networks (CNNs). We experiment with two model based on the AlexNet and utilise PyTorch's transforms modules to augment our dataset. Our final model has an overall accuracy of 97.9% as measured on the test set, with the lowest class scoring an accuracy of 75%.

1 Introduction

It is very likely that in the 21st century autonomous vehicles will move from the realm of science fiction into reality. One of the key impediments to achieving this goal is that autonomous vehicles will need to be able to recognise, interpret and act on the rules of road safety all across the world (Lim et al., 2017). Doing so will require a strong ability to detect and classify road signs. This paper focuses on a generalised algorithm to classify road signs from images where they have been detected. This will be achieved by use of the *GTSRB* dataset. The *GTSRB* is a dataset of around 50,000 images of traffic signs labelled into 13 categories (Stallkamp et al., 2011).

1.1 Dataset Exploration

The training data set consists of 39209 images arranged into 13 classes with relatively significant class imbalance as can be seen in Figure 1. The images contain relatively significant variation in terms of input resolution, brightness, motion blur, color conditions and other factors. Figure 2 demonstrates just one dimension of this variation - variation in resolution. One of the key decision to make revolves around selecting a standard resolution: some images have input resolution as low as 15×15 whilst others are as large as 256×256.

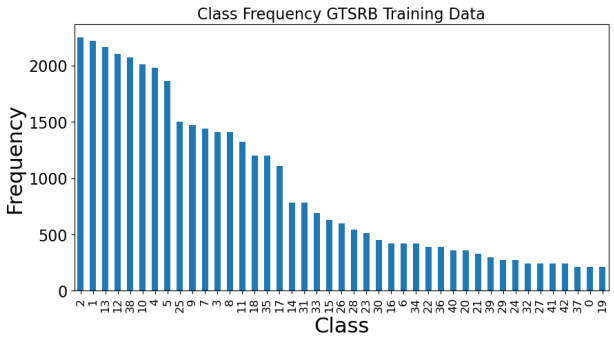

Figure 1: *Frequency distribution of different classes in training set*

1.2 Approach

This project seeks to build a robust multi-label classifier able to classify traffic signs as evaluated by the performance on the test dataset. A number of approaches have been utilised for image classification such as neural networks and support vector machines (Yuan et al., 2014). Traditionally, image classification was handled as a two step process: first features were extracted using feature descriptors, and these were fed as input into

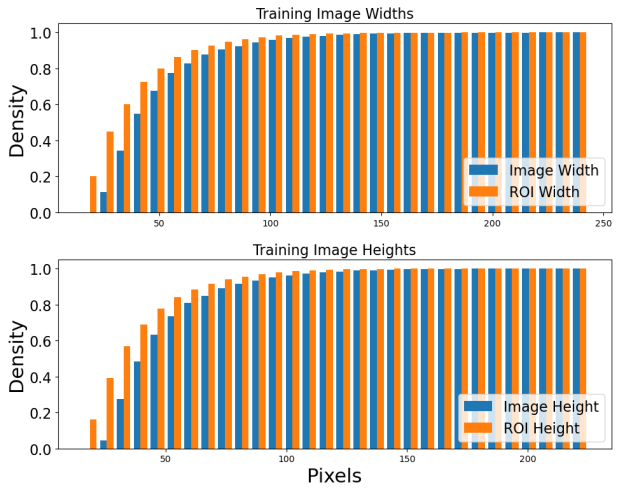

Figure 2: *CDF for image width and height, as well as region of interest (ROI)*

the classification task (Rawat and Wang, 2017). Due to the time consuming nature of rich feature extraction required by SVM's, less progress has been made in developing image classification algorithms using SVMs.

The multi layered nonlinear processing capacity of CNN's have made them the leading architecture for image classification, particularly since Krizhevsky et al. (2012) used a deep CNN to achieve first place in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC).

2 Methodology

2.1 Pre-Processing

Initially we attempted to apply histogram equalisation to both training and test images based on the work by Team ISDIA in Stallkamp et al. (2011), but due to less optimal results this approach was abandoned in favour of data augmentation as described in 2.2 (Stallkamp et al., 2011). As is described in 2.3, we made use of two different input images sizes. The first was based on a slightly smaller version of AlexNet, with resolution 112×112. The second was the original AlexNet size of 227×227.

2.2 Data Augmentation

We augmented the original dataset using various transformations from PyTorch's `tranforms` module. Our approach was to create additional augmented datasets and combine these augmented datasets with our original dataset - and use the combination as our training data. In total we created 5 augmented data sets in addition to the original images, leading to a total of 235254 images for training. These augmentations were:

- Histogram Equalisation:
  `tranforms.RandomEqualize(p=1)`
- Rotation:
  `tranforms.RandomRotation(degrees=15)`
- Perspective Distortion:
  `tranforms.RandomPerspective(distortion scale=0.1)`
- Auto Augmenting
  using `IMAGENET` and `CIFAR10` policies

The autoaugmenting is a useful feature of the PyTorch package. It automatically generates augmented images from a given search space. We utilised pretrained auto-augment policies (Cubuk et al., 2018). An example of the types of augemented images this package generates can be seen in Figure 3.

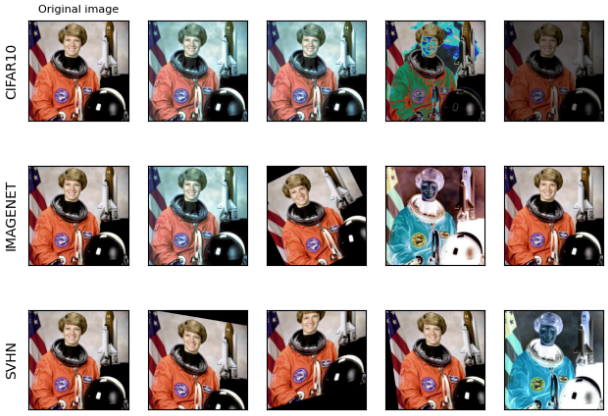

Figure 3: *Example augemnted images for `CIFAR10`, `IMAGENET` and `SVHN` policies*

2.3 Convolutional Neural Network

Our neural network processing layer is based on AlexNet designed by Alex Krizhevsky in 2011 using deep convolutional neural networks (CNN's) implemented PyTorch (Krizhevsky, Sutskever, and Hinton, 2017; Paszke et al., 2017).

AlexNet consists of 8 layers; 5 convolutional layers and 3 fully connected layers. It makes use of the Rectified Linear Units (ReLU) for the activation function, and uses a 50% dropout for the fully connected layers. The basic structure can be seen in Figure 4. The input image is fed into a convolutional layer, which serves as a feature extractor. This is performed by operating with a kernel on the input image, to produce a transformed version of the image except the values in the kernel matrix are not specified, but instead are changed through the learning process. The output of each convolutional layer is input into a max pooling operation, which is a down sampling operation to extract key features. The output of the max pooling layer is multiplied by the $ReLU$ function, which allows it to introduce non-linearities. Whilst we are able to change parameters to these functions such as stride and kernel size, we choose to not vary these parameters as this

would lead to too many experiments and lies outside the scope of this paper.

The AlexNet network has 5 such convolutional layers followed by 3 fully connected layers leading to an output of 43 classes in our case. The fully connected layers do as the name specifies, the connect each output from the previous layer of nodes to a new layer of nodes. Here the AlexNet also utilises the $ReLU$ activation function.

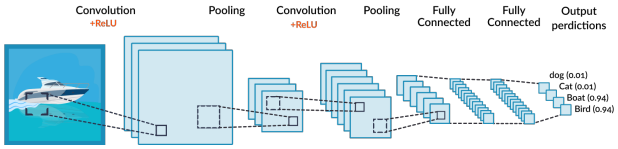

Figure 4: *Simplified schematic showing CNN structure (Choudari, 2020)*

We used 2 slightly different versions of the AlexNet for our experiments. One took in an input of 112×112 images and the other took 227×227. These were sourced from two online tutorials (Krishnamurthy, 2020; Ahmed, 2022).

2.3.1 MODEL 1: 112x112 Input Size

MODEL 1 utilised the following hyperparameters:

- Learning rate: 0.001
- Batch Size: 256
- Epochs: 10

These hyper parameters were found to perform well, and within 10 epochs the model had a very high accuracy on the validation set, and any further training risked over fitting.

2.3.2 MODEL 2: 227x227 Input Size

MODEL 1 utilised the following hyperparameters:

- Learning rate: 0.002
- Batch Size: 256
- Epochs: 20

2.4 Loss function

We made use of a cross-entropy loss function to calculate the loss for the neural network. This is a common loss function for classification tasks. Whilst the exact machinery of the cross entropy loss function is outside the scope of this paper, the basic premise is that the function measures the performance of a classification model which outputs a predicted probability for a class. We made use of a weighted class loss function in one of our experiments. This was done as an attempt to reduce some of the issues of class imbalance in our training dataset. The idea is that when we are calculating losses, for training classes the loss is weighted by proportionally higher amounts for underrepresented classes. The weight for the loss of samples from a particular class are:

$$w_i = \frac{\#\ samples}{\#samples \times \#\ of\ samples\ in\ class\ i} \tag{1}$$

2.5 Training

Our training was completed on a Google Cloud Deep Learning Virtual Machine with a NVIDIA T4 GPU. PyTorch was able to make use of the GPU using the CUDA driver.

For each training run, if the run included augmented data, this was first generated and a complete dataset for training was created. The training dataset was split into 80:20 training:validation subsets which could be used to train the model. We would begin with a randomly initialised model, which we would train in batches of 256 images. For each batch the loss was calculated and added to the running loss for the epoch. Initially we began with 5 epochs but we later found the loss function only plateaus near 10 epochs, as can be seen in Figure 5.

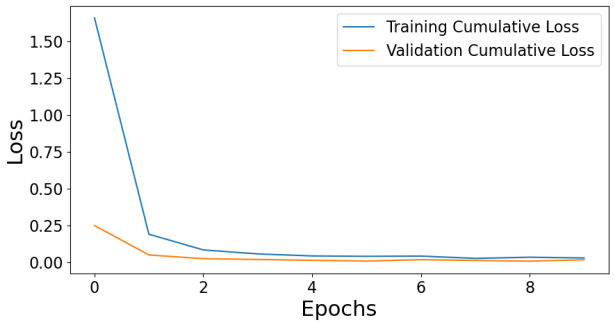

Figure 5: *Example of how loss changed for a training run*

2.6 Experiments

Our experiments were constructed as follows. We ran two batches of experiments, using the two different versions of AlexNet. First we would begin by training a model using no augmented data only the original images in the training data set. This will set our baseline, against which we will compare the effects of various augmentations and adaptations. The different scenarios we will test are as follows:

- Augmenting with histogram equalised, scale distorted and rotated images
- Implementing a class weighted loss function
- Augmenting with AutoAugmented Images using the `IMAGENET` and `CIFAR10` policies

We would then record the performance of each model and then we could make comparisons between the two different verions of AlexNet.

3 Experimental Setup

For each scenario, we will train our model and assess its accuracy on the test dataset. Accuracy will be compared using overall accuracy, mean class accuracy and a distribution of accuracies for different classes will also be inspected.

3.1 Overall Accuracy (OA)

Overall accuracy will be evaluated as:

$$OA(\%) = \frac{\#\ correctly\ predicted\ samples}{\#\ of\ test\ samples} \tag{2}$$

3.2 Mean Class Accuracy (MCA)

In order to account for distortions due to variations in class size, the mean class accuracy will weight the accuracy of each class equally

$$MCA(\%) = \frac{\sum_i\ class\ accuracy\ in\ class\ i}{\#\ of\ classes} \tag{3}$$

Where the class accuracy for a class is similar to the OA but limited to images from that class

3.3 Lowest Class Accuracy (LCA)

Given the application to autonomous driving, our model cannot score too poorly on any given class. If it does, this could lead to very bad outcomes, for example mistaking a speed limit for a pedestrian crossing. For this reason we will also assess the accuracy of the lowest class for our different experiments

4 Results

4.1 AlexNet with 112x112 input images

As we can see in Table 1, this model performed relatively well without any fine tuning, but we could make several improvements with fine tuning. Firstly we note that using our class weighted loss function, the LCA increased significantly from the base model. By far the most effective method to improve the model is to increase the size of the dataset with augmentation. We notice that Augmented (1) had higher performance using the non weighted loss function than when we added the weighted class loss function, but its LCA decreased. The downside of the augmentation approach is that the model took increased GPU time in order to complete a training run. With this in mind, we now move to testing using our different model of AlexNet.

4.2 AlexNet with 227x227 input images

Whilst the 112×112 AlexNet had 15,063,891 trainable parameters, the model with 227×227 input images had 58,460,267 training parameters which meant training took significantly longer. To remedy this we utilised 2 GPU's and processed our data in parallel using the `DataParallel` module. This introduced a few more complexities, and the learning rate was not significantly faster. Two models were trained, the base model and the class weighted model. Given the lower performance of these models (Table 2) and the time consuming nature of the training process - we decided not to pursue training further models with the augmented data. Particularly the very low performance of the Class Weighted Model indicates we may have made an error implementing the `DataParallel` module.

Table 1: *Summary of results for different experiments for the* 112×112 *AlexNet. Augmented (1) uses rotations, scale distortion and histogram equalisation whilst Augmented (2) uses these three in addition to augmentation using the* `IMAGNET` *and* `CIFAR10` *AutoAugmentation*

Experimental Setup	OA	MCA	LCA
Base	92.7%	88.9%	46.6%
Augmented (1)	97.2%	96.0%	65.0%
Class Weighted Loss	93.3%	91.5%	56.0%
Augmented + Class Weighted Loss	94.8%	94.0%	67.3%
Augmented (2)	97.9%	96.2%	75.0%

Table 2: *Summary of results for different experiments for the* 227×227 *AlexNet*

Experimental Setup	OA	MCA	LCA
Base	85.8%	76.1%	0.0%
Class Weighted Loss	36.6%	33.0%	0.0%

4.3 LUV Color Space

Some authors have explored the optimal colour space for CNN models and found that LUV may be a suitable alternative to RGB (Reddy, Singh, and Uttam, 2017). To test this we attempted to compare the performance of our base model for the 112×112 AlexNet but trained and evaluated on images in the LUV colour space. This was performed using the `Kornia` `rgb_to_luv` function. The result is shown in Table 3. As we can see, this lead to an improvement of nearly 1% in OA to our base model trained in the RGB color space. However there was a reduction in the LCA which was not ideal.

Table 3: 112×112 *AlexNet with colour conversion to LUV space*

Experimental Setup	OA	MCA	LCA
Base	92.7%	88.9%	46.6%
LUV	93.8%	89.8%	35%

4.4 LUV and Augmentations

With these learnings in mind, we attempted to build a final model to see if it could outperform the Augmented (2) model from 4.1. In this model we made use of all the aforementioned Data Augmentation techniques, this time also including a data set which was AutoAugmented using the `SVHN` policy. We also converted all images to the LUV color space. This resulted in a dataset containing 274,463 images. This model trained for 10 epochs using the hyperparameter settings as described in 2.3.1. The result is seen in Table 4. As we can see, this model did not outperform the Augmented (2) model.

Table 4: *Final model of the* 112×112 *AlexNet with colour conversion to LUV space and several augmentation techniques*

Experimental Setup	OA	MCA	LCA
Augmented (2)	97.9%	96.2%	75.0%
LUV & Augmentation	97.8%	95.9%	55.0%

5 Conclusion

We have successfully built a classifier which is able to classify images in the *GTSRB* dataset with $\mathbf{97.9}\%$ overall accuracy, and a lowest class accuracy of $\mathbf{75}\%$. Our classifier is based on the popular implementation of a CNN known as AlexNet. We have relied on primarily data augmentation through PyTorch's `AutoAugment` feature but we have also tried many hyperparameter settings and tested the effects of different network sizes. Finally, we have tested the use of a different colour space, which a relatively significant increase in performance.

5.1 Future Work

There remains some avenues which can lead to further improvements of our model. Some more complex pre-processing could be attempted, such as Zero Component Analysis, Mean Normalisation and Standardisation (Pal and Sudeep, 2016). Recent models which are successor models to the AlexNet could also be implemented and evaluated such as GoogLeNet and ResNet. Finally, with more work we can obtain a deeper understanding of the methods needed to construct these networks, and begin to experiment with variations of the aforementioned models for the *GTSRB* dataset.

Bibliography

Ahmed, Nouman (2022). *Writing AlexNet from Scratch in PyTorch*. `https://blog.paperspace.com/alexnet-pytorch/`.

Choudari, Pratik (2020). *Understanding "convolution" operations in CNN*. `https://medium.com/analytics-vidhya/understanding-convolution-operations-in-cnn-1914045816d4`.

Cubuk, Ekin Dogus et al. (2018). "AutoAugment: Learning Augmentation Policies from Data". In: *CoRR* abs/1805.09501. arXiv: `1805.09501`. URL: `http://arxiv.org/abs/1805.09501`.

Krishnamurthy, Surak (2020). *TSR$_{PyTorch}$*. `https://github.com/surajmurthy/TSR_PyTorch`.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey Hinton (2017). "ImageNet classification with deep convolutional neural networks". eng. In: *Communications of the ACM* 60.6, pp. 84–90. ISSN: 0001-0782.

Lim, Kwangyong et al. (2017). "Real-time traffic sign recognition based on a general purpose GPU and deep-learning". eng. In: *PloS one* 12.3, e0173317–e0173317. ISSN: 1932-6203.

Pal, Kuntal Kumar and K. S Sudeep (2016). "Preprocessing for image classification by convolutional neural networks". eng. In: *2016 IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*. IEEE, pp. 1778–1781. ISBN: 9781509007745.

Paszke, Adam et al. (2017). "Automatic differentiation in PyTorch". In.

Rawat, Waseem and Zenghui Wang (2017). "Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review". eng. In: *Neural computation* 29.9, pp. 2352–2449. ISSN: 0899-7667.

Reddy, K. Sumanth, Upasna Singh, and Prakash K Uttam (2017). "Effect of image colourspace on performance of convolution neural networks". eng. In: *2017 2nd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*. IEEE, pp. 2001–2005. ISBN: 9781509037049.

Stallkamp, J et al. (2011). "The German Traffic Sign Recognition Benchmark: A multi-class classification competition". eng. In: *The 2011 International Joint Conference on Neural Networks*. IEEE, pp. 1453–1460. ISBN: 1424496357.

Yuan, Xue et al. (2014). "Robust Traffic Sign Recognition Based on Color Global and Local Oriented Edge Magnitude Patterns". eng. In: *IEEE transactions on intelligent transportation systems* 15.4, pp. 1466–1477. ISSN: 1524-9050.