

**SCHOOL OF COMPUTER TECHNOLOGY**

AASD 4014 Deep Learning II

Project 1: Object Classification

By: VietKhoi (James) Hoang (101350362)

Jaideep Prasad (101323133)

**Problem Statement:**

For this project, our goal is to create a machine learning classification model which classifies images. The model is to be a convolutional neural network (CNN) that will be optimized based on model accuracy. Based on these requirements, we will make a classifier for common road signs.

**Dataset:**

The dataset we used was the famous German Traffic Sign Recognition Benchmark (GTSRB) dataset that could be found on Kaggle [[1]](https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign) or the official site [[2]](https://benchmark.ini.rub.de/gtsrb_dataset.html). The dataset contains 43 classes with 39.2 thousand training images and 12.6 thousand testing images. Image size varies between 15x15 and 250x250 pixels and are not necessarily always square images. The signs in the images themselves are not always centered in the frame of the image. While the dataset does come with annotations in the form of an excel file for both the train and test data, this will be ignored for the purposes of this project. It is important to remember that the dataset is not balanced as shown in the image below [[3]](https://medium.com/@thomastracey/recognizing-traffic-signs-with-cnns-23a4ac66f7a7).

Chart, histogram

Description automatically generated

[[3]](https://medium.com/@thomastracey/recognizing-traffic-signs-with-cnns-23a4ac66f7a7) Graph 1: Number of Samples per Class

**Machine Learning Model:**

Before starting on the model there were a few issues we had to consider. First, we needed to load an even amount of train images from each class, so our model is exposed to each class proportionally. This is important when looking at confusion matrices later to narrow down issues with our model as we do not want a poor preforming model to stem from non-diverse data. This also prevents a feature dominating over other features. Second, we wanted to have 5 test cases where we used 200, 1000 and all train images as well as mixture of 200 train 300 augmented images and 1000 train 1000 augmented images. This is so that we have an abundance of data at our disposal, as the more high-quality data we have the better our model will learn all the different classes.

For the first problem we wrote a custom function called load\_train that requested the number of images the user wanted. If they wanted all the training images (denoted as -1), if not the function would divide the number of requested images with the number of total classes to calculate the number of images requested per class. From here we check the number of images each class has. If the class did not contain enough images, we simply return all the images in the class. If the class contained enough images, we randomly picked the required number of images from the class. While extracting each image, we define the resolution of each image to deal with the varying image sizes.

Text

Description automatically generated

Figure 1: load\_train code

For the second problem, we created a custom function called imageGen that would create augmented images. We used the ImageDataGenerator function from Keras to produce the augmented images based on parameters we set such as zoom, sheer, width and height shifts, rotation, etc.

Text

Description automatically generated

Figure 2: imageGen code

With all the main issues sorted out and the code created, it was time to create the CNN. Our final CNN included 3 sets of 2D convolution + 2D average pooling layers where the filter size dropped from 11x11 to 7x7 to 3x3, a flatten layer, 4 dense layers the first 2 dense layers were accompanied by a dropout layer which would drop 20 percent of the current neurons in each dense layer.

Text

Description automatically generated

Figure 3: Final CNN architecture

**Results:**

Our first CNN was tested on the 200-training data. It had 3 sets of 2D convolution + 2D average pooling layers where the filter size dropped from 11x11 to 7x7 to 3x3. The ideal for this was to capture general characteristics in the beginning and then start to look at more specific details later. After the 3 sets on convolution and pooling layers, we added a flatten and 7 dense layers. There were 7 dense layers to slowly reduce the 9000 neurons to 43 neurons at a slow rate to prevent any major loss of data however, we noticed that the performance of this model was unideal. Our best epoch had a validation accuracy of 0.2561.

Text

Description automatically generated

Figure 4: First CNN architecture

|  |  |
| --- | --- |
|  |  |

Figure 5: First CNN architecture with 200 training images

From here we reduced the number of dense layers in the CNN as well as adding dropout layers and normalizing the pixel intensities. This seemed to increase the model performance on the 200-image dataset and so we decided to benchmark this new model, let us call it version 2, with all the other datasets.

|  |  |  |
| --- | --- | --- |
| Dataset | Dataset code | Validation Accuracy |
| For 200 training images: | 200t | Best val acc = 0.4143 |
| For 200 training and 300 augmented images: | 200t 300a | Best val acc = 0.2213 |
| For 1000 training images: | 1000t | Best val acc = 0.7382 |
| For 1000 training and 1000 augmented images: | 1000t 1000a | Best val acc = 0.5972 |
| For all training images: | -1t | Best val acc = 0.9527 |

Text

Description automatically generated

Figure 6: Second CNN architecture

|  |  |
| --- | --- |
|  |  |

Figure 7: Second CNN architecture with 200 training images

|  |  |
| --- | --- |
|  |  |

Figure 8: Second CNN architecture with 200 training and 300 augmented images

|  |  |
| --- | --- |
|  |  |

Figure 9: Second CNN architecture with 1000 training images

|  |  |
| --- | --- |
|  |  |

Figure 10: Second CNN architecture with 1000 training and 1000 augmented images

|  |  |
| --- | --- |
|  |  |

Figure 11: Second CNN architecture with all training images

Being satisfied with the second CNN model, we started looking into optimizing the model by changing the following listed below and comparing them to the base models above. Outside of the second CNN architecture, it uses 40 filers per Conv2D, sheer and rotation and an image size of 150x150. All the models run on 30 epochs except for -1t which runs on 10 epochs.

|  |  |  |
| --- | --- | --- |
| Changes | Dataset code | Validation Accuracy |
| No sheer, no rotation | 200t 300a | Best val acc = 0.2373 |
| Sheer but no rotation | 200t 300a | Best val acc = 0.2594 |
| No sheer but rotation | 200t 300a | Best val acc = 0.2318 |
| Non-normalized pixels | 1000t | Best val acc = 0.0594 |
| Image size of 100x100 | 1000t | Best val acc = 0.8143 |
| Image size of 200x200 | 1000t | Best val acc = 0.6509 |
| 20 filters per Conv2D | 1000t | Best val acc = 0.7392 |
| 60 filters per Conv2D | 1000t | Best val acc = 0.7546 |
| Max pooling | 1000t | Best val acc = 0.7080 |
| Sigmoid activation layers | 1000t | Best val acc = 0.0594 |

Looking at all these test cases, it seems like sheer, no rotation, normalized pixel intensities, smaller image size, and more filters will give us the best model yet. Let us test that model on 1000t 1000a and see how it goes. This model will have 60 filters, an image size of 100x100, sheer and no rotation, normalized pixel intensities, and the rest will be the same as the second CNN architecture.

|  |  |
| --- | --- |
|  |  |

Figure 12: Third CNN architecture with 1000 training and 1000 augmented images

The Third CNN model seems to be horrible with an accuracy of 0.0523. We are not too sure where we went wrong here but we will use our second CNN model and our final model.

**Appendix**

[1]

<https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign>

[2]

<https://benchmark.ini.rub.de/gtsrb_dataset.html>

[3] Graph 1: Number of Samples per Class

<https://medium.com/@thomastracey/recognizing-traffic-signs-with-cnns-23a4ac66f7a7>

Chart, histogram

Description automatically generated

Figure 1: load\_train code

Text

Description automatically generated

Figure 2: imageGen code

Text

Description automatically generated

Figure 3: Final CNN architecture

Text

Description automatically generated

Figure 4: First CNN architecture

Text

Description automatically generated

Figure 5: First CNN architecture with 200 training images

|  |  |
| --- | --- |
| Chart, line chart  Description automatically generated | Chart, line chart  Description automatically generated |

Figure 6: Second CNN architecture

Text

Description automatically generated

Figure 7: Second CNN architecture with 200 training images

|  |  |
| --- | --- |
| Chart, line chart  Description automatically generated | Chart, line chart, histogram  Description automatically generated |

Figure 8: Second CNN architecture with 200 training and 300 augmented images

|  |  |
| --- | --- |
| Chart, line chart  Description automatically generated | Chart, line chart  Description automatically generated |

Figure 9: Second CNN architecture with 1000 training images

|  |  |
| --- | --- |
| Chart, line chart  Description automatically generated | Chart, line chart  Description automatically generated |

Figure 10: Second CNN architecture with 1000 training and 1000 augmented images

|  |  |
| --- | --- |
| Chart, line chart  Description automatically generated | Chart, line chart, histogram  Description automatically generated |

Figure 11: Second CNN architecture with all training images

|  |  |
| --- | --- |
| Chart, line chart  Description automatically generated | Chart, line chart  Description automatically generated |

Figure 12: Third CNN architecture with 1000 training and 1000 augmented images

|  |  |
| --- | --- |
| Chart, line chart, histogram  Description automatically generated | Chart  Description automatically generated |