1. **Results**

As said in II.iii, many different CNN architectures were implemented. Within those configurations, also different preprocessing steps were applied. The process of reaching a final model included a vast trial and error methodology, always following relevant literature.

The first step taken in training was to implement a basic model with two convolutional layers each followed by a max pooling layer plus two final dense layers following a flatten layer. This simple architecture could give the authors an idea of which would be the starting point. In this try the authors worked only with a fraction of the available data in order to preserve

After looking at the first results the authors rapidly identified large amounts of overfitting (figure 1). Due to the amount of overfitting experienced the authors were forced to tackle that issue, having found two main ways of doing it: adding dropout layers (before the dense layers) and using data augmentation.

Chart, line chart

Description automatically generated

Figure 1 - Base Model vs Base + Dropout vs Base + Dropout + Data Augmentation

As seen in figure 1, none of the models really outperformed the others. It’s also fair to say that the worst performance belonged to the model where data augmentation was applied. Since no great improvements were achieved the authors decided that using all data for training could be one good solution.

Considering what is said in II.iii about training images series, the authors were aware that it could pose some issues regarding generalization capability of the model, therefore, the dropout layer was kept. In the first models trained using all data two models were tested in order to decide which would be the best choice for the dropout rate. The values used in this testing were 0.3 and 0.9 (Figure 2).

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Figure 2 – Base Model with Dropout Rate equal to 0.3 and 0.9

After a brief look at the results obtained using all data, the authors quickly found that using either one of the two values a significant part of the overfitting would disappear.

After getting these results, using class weights in training was tested. However, the results obtained didn’t differentiate significantly. Having that into consideration, it was decided that the weighting could be a final step to take when a final architecture was found.

Moreover, and with the objective of increasing performance on the validation set, the authors decided to test the model using one more convolutional layer.

The configuration of the model at this point is shown in Figure 3.

Table

Description automatically generated

Figure 2 – Model Architecture After Adding 1 Convolutional Layer

In addition to the third convolutional layer added, that as mentioned above was used to increase performance, also grayscale was applied. The latter was introduced as a new attempt to tackle overfitting.

From Figure 3, the reader can conclude that the two additions applied in this step, one more convolutional layer and grayscale, resulted mainly in an increased model performance (overtaking the 90% validation accuracy line) but the overfitting levels remained almost unchanged. Even though the loss values for the model without grayscale are more convincing, the authors still traded that for a slightly larger overall validation accuracy, keeping grayscale for the next steps.

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Figure 3 – Model with 3 Convolutional Layers with and without Grayscale

Having achieved an increase in performance, the priority turned to reduce the amount of overfitting. The first idea to reduce overfitting would be to introduce more dropout layers in the current architecture. In order to find which were the optimal values to use in this step the authors tried two similar configurations in which the only difference resided on the dropout rates. In each of the models a dropout layer was introduced after all convolutional layers. The combination of dropout values (from the first convolutional layer to the last) in the model with lower dropout rates was *0.15 – 0.15 – 0.1 – 0.3.* As for the one with higher dropout rates it was *0.25 – 0.25 – 0.25 – 0.5.*

In Figure 4, the reader is provided with each model’s accuracies*.* In this case, the choice of model became easier with great trade-offs to be decided on. It is clear that in terms of validation and training accuracy, the model with lower dropout rates outperformed its opponent after 50 epochs.

Chart, line chart

Description automatically generated

Figure 4 – Higher vs Lower Dropout Rates

From Figure 4 it also gets perceptible that at this point the overfitting issue seemed to be taken care of. That being said the next step would consider either improving accuracy, in order to reach values closer to 95% or improving performance in terms of time. Since times wasn’t being an issue so far, the authors wanted to check the result of adding a fourth convolutional layer to the model. Being aware that this could also result in increased amount of overfitting, a dropout layer was also added to this architecture. Two variants were tried in which the main difference was the number of filters used. The first had 32 – 64 – 64 - 128 filter structure (from first to last layer). The second one had two 128 filters layers instead of having two 64 filters layers, so it looked like 32 – 64 – 128 – 128. Unfortunately for the authors, the result was not positive. Although the second four convolutional layers model (2x128) quickly reached convergence in terms of accuracy (achieving values closer to the current best model), the loss values soared, which was not a good indicator. Therefore, the author remained with the previous best model, with three convolutional layers and lower dropout rates in each dropout layer.

The results of the last testing can be seen in Figure 5.

A picture containing chart

Description automatically generated

Chart, line chart

Description automatically generated

Figure 5 – Two Model with 4 Convolutional Layers Against the Older One

Getting back to our older model, as said in II.iii, also image normalization was applied as a preprocessing step. Using ‘featurewise\_center’ and ‘featurewise\_std\_normalization’, the authors tried to make the range of distribution of feature values similar between features. In order to evaluate if the goal would be reached, two exact same models were test. The only difference between one and the other was that one of them had image normalization as a preprocessing step, whilst the other did not. In Figure 6 the reader can find the results of such experiment. From the graphics there isn’t again a real trade-off. The model with image normalization outperformed the counterpart both in training and validation accuracies (this one remained constantly over 90% accuracy on the validation set) and also it has much more encouraging loss values. Since the difference in the experiment was clear, the author opted for adopting image normalization as a preprocessing step for future experiments.

Chart

Description automatically generated

Chart, line chart

Description automatically generated

Figure 6 – Testing a Model with and without Image Normalization

In this stage the model was reaching a plateau, where no great improvement was taking place. In these cases, the way to overcome it is by making larger changes to the model rather than simply adding layers one by one or come up with new preprocessing techniques. Considering that, it made sense at this stage to draw a similar model but with some changes across the whole architecture. It would make sense to keep the three convolutional layers, however some parameters could be changed, such as the size of the filters and important dropout layers (for example, the first dropout layer is important since it is applied directly over the input convolutional layer). The model that came up had the following architecture:

Insert Architecture of Final Model

Results of it …

1. **Comparisson to Other People’s Results**

The problem addressed by the authors is not something entirely new. In section [Introduction Section] it is mentioned that this type of model can be used by self-driving cars or systems that identify the traffic rules in certain locations.

In order to generally classify the model, it gets pertinent to compare this model’s performance with previous ones.

The first comparable model developed [1] in the past achieved an accuracy of 94.7% on the test set.

Other model designed to correctly identify traffic signs can be found in [2]. This one having reached a score of 99% on the test dataset.

A third source [3] also reached values near 100%. This project although, resorted to several object detection systems, rather than just simples Convolutional Neural Networks.

[1] <https://towardsdatascience.com/traffic-sign-recognition-using-deep-neural-networks-6abdb51d8b70>

[2] <https://towardsdatascience.com/traffic-sign-detection-using-convolutional-neural-network-660fb32fe90e>

[3] Evaluation of Deep Neural Networks for traffic sign detection systems, Álvaro Arcos-García, Juan A. álvarez-García, Luis M. Soria-Morillo