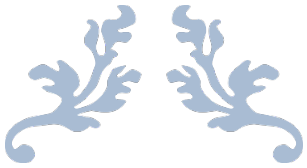

AUTOMATED TRAFFIC SIGN CLASSIFCATION

MAY 30, 2023

MRWAN ALHANDI

Table of Contents

Problem Description

In this assignment the task is to create an image recognition machine learning algorithm. In this case, the application will be street sign recognition. This problem has always been a novel training exercise for learning machine learning due to its simplistic nature, although it also has many real world applications.

This problem represents itself in machine learning as a classification task. In classification, given inputs are 'sorted' into one of a set of outputs based on several features. This differs from regression tasks, where the aim is to predict a numeric value based on some features. In the implementation of classification problems, however, regression is often used internally to measure the 'confidence' of something belonging to a particular class.

Classification machine learning models are often the product of many regression algorithms combined and passed through additional steps such as 'activation'. Activation refers to determining the class of a given input or set of inputs based on the output of a numeric function. Activation functions are integral to classification machine learning and often have a strong effect on the final model. Multiple different activation functions will be used in this report.

Street sign recognition is a novel problem in that it is intuitive to understand yet can be made quite challenging to solve. Part of the complexity of the problem comes from the likely realistic applications of street sign recognition algorithms. For example, classification models run on data that has been taken directly from an input (i.e a camera) is much harder to accurately classify than data that has gone through complicated pre-processing. This pre-processing is also a part of the pipeline of creating a machine learning model and will be addressed in this report. Additionally, the computational complexity of a model must be factored in when considering real worl applications. For example, a highly complex model would be more difficult to run quickly in applications like an autonomous car, and thus it may be more optimal for a model to be less complex and more efficient, even if that meant sacrificing other areas.

Solution Overview

The first step to solving the problem is understanding the datasets being used. In practice, this involved loading the images and displaying a few to understand the form the data was in. Label mappings also had to be created to understand the distribution of the data. In the case of the shape recognition task, Figure 1 shows the counts and relative distribution of each class in the dataset. The distribution is uneven, with the 'round' class containing by far the most samples, and the 'hex' class containing by far the least. This could produce some issues, where models may bias to the classes with higher representation in the dataset. Because of this, feature importance is something that will need to be considered and monitored to ensure that overfitting to some classes does not occur. In the case of the sign recognition task, the distribution is much more even, however the 'warning' and 'stop' classes still have significantly different sample sizes to the rest (Figure 2).

Baseline Models

Many different classification models were considered to solve the problem. These include decision tree, random forest, KNN and neural networks. Each of these models has slightly different implementations and advantages in different areas. For example, decision tree models are often the least complex and thus the most computationally efficient, however this can lead to lower accuracy. Similarly, KNN models can also be computationally inefficient due to the requirement to both store and make calculations based on the entire dataset for every prediction, although they do not require a training phase so can be quick to create. These pros and cons for each model type were considered, and ultimately Decision Tree was chosen as the baseline model for the shape classification task. Decision tree was chosen due to the relative simplicity of both the implementation and the computations required to train the model, making it a good candidate for a baseline. The shape classification task was handled first given it is the simpler of the two problems, thus the sign classification will likely benefit from insights gained in the first task.

Shape Classification

Before the model development can begin, pre-processing needs to take place. This happens by reading in all the images sorted based on their class, and creating a label mapping that can be better interpreted by the model (rather than the long string labels that are human readable). The shape of the images is also manipulated to ensure that the sklearn can understand the contents. Without this step, the model would either fail to train or train on other aspects of the input files that are not the desired inputs. Finally, the input images are normalized (a form of scaling, adapted to be done on images) to ensure that a standard [0,255] distribution of pixel values is present. Finally, the data is split into training and testing subsets. A 20% test set is used given the small sample size of some classes. If a larger test set size was used, then these classes could have so few samples that the model would not be able to generalize to them at all.

Once the data is prepared, the Decision Tree model can be fit. For the baseline model, a novel value of 10 is used for the max_depth hyper parameter, which essentially controls the complexity of the tree. Greater depths can result in higher specificity and thus performance, but also increase the complexity of the model. To measure the performance of the model, accuracy_score from sklearn is used. Accuracy score represents the proportion of correctly predicted samples compared to the total number of samples. That is, a higher accuracy score represents fewer errors in the model. This decision tree implementation achieves a score of around 85% which is very good for the baseline model. This does suggest that 15% of samples are being incorrectly classified, however, which does suggest further investigation is needed. By plotting the confusion matrix for this model (Figure 3) it's observed that the 'triangle', 'round' and 'square' classes are often the most miss-classified. The 'hex' class shows a relatively low number of incorrect classifications, although this can be attributed to the proportionally low number of 'hex' class samples in the datasets.

Sign Classification

Based on the performance of the Decision Tree model for the shape classification task, a Feed-Forward Neural Network was decided as the baseline model for the sign classification task. This is due to the increased complexity of classifying the type of a sign rather than just a shape. This increased complexity comes from the greater number of parameters that must be considered, i.e multiple signs of the same shape can have different types. Also, there are many more output classes in this problem, and thus a more sophisticated model was deemed necessary. Neural Networks also have the property that feature importance / selection is implicitly handled by the model weights, and less important features can be entirely dropped out of the model. A Feed-Forward Neural Network was chosen specifically because of the relative simplicity of the implementation. As one of the most basic neural network types, the FFNN consists of only full connected layers without any grouping or filtering. This does have some drawbacks, such as less understanding of higher-dimension data sets but does allow for much easier creation and training of the model.

For the baseline neural network, a FFNN model with two hidden layers was used. Using two hidden layers was a novel choice to allow the model to generate some complexity while ensuring it would not become too complicated / overfit. A learning rate of 0.001 was used at this stage. These hyperparameters are novel baseline choices for this model and do represent areas for possible improvement / refining in the future. A validation split of 20% was also used for this model, which is handled implicitly by the training step. Overall, a FFNN with roughly 55,000 parameters was used for this task.

Figure 4 displays the training and validation loss of the FFNN approach. The FFNN model achieves a sparse categorical accuracy on the validation data of around 90%, which again is a good baseline. The validation accuracy score must be used here because the training accuracy score is not indicative of the generalized performance of the model. The training accuracy score does give some insight, however, as it can ensure that the model is not overfitting by achieving a 100% accuracy on the training data.

Improved Models

The first method used to improve the performance of both models was implementing a Convolutional Neural Network (CNN). This differs from a FFNN in that not all layers are fully connected to each other, and in this case multiple pooling layers were used along with a series of connected layers. This does increase the complexity of the neural network implementation which must be monitored during development. The results from these implementations showed a significant increase over the decision tree implementation and a slight increase over the FFNN performance, with a score of around 92%.

The improved scores do represent an improvement, however there are still flaws in the models. The sign detection task was chosen for further development due to the higher complexity and slightly poorer performance than the shape detection task, showing there was more room for improvement. By plotting the confusion matrix for the sign detection CNN model (Figure 5), there are some flaws with the ‘warning’ and ‘noentry’ sign classes being miscategorised often by the

model. This details a flaw in the evaluation metric used, sparse categorical accuracy, as this metric is less suited for applications with a large difference in sample classes. This is supported by the fact that these two sign classes are the most represented in the dataset.

Hyperparameter Tuning, Regularisation and Dropout

Three techniques used to improve the performance of the CNN model were hyperparameter tuning, regularization and dropout. Each of these techniques serves a unique purpose in improving the performance of the model in a different way.

Hyperparameter tuning is the most significant change introduced to improve the model performance. Hyperparameters refer to the novel values used in the models that are not directly inputs from the dataset. In this case, the hyperparameters being tuned are 'learning rate' and 'epochs'. Learning rate refers to the scale of the changes made by the model when fitting to a dataset. A higher learning rate can lead to a model finding optimal values faster, but can also cause the model to over-correct for errors and slow down training. Similarly, a lower learning rate will be more precise but is also much slower and may not be able to reach optimal values. The learning rate is tuned to find the optimal value that allows a model to fit correctly and efficiently. The number of epochs is also tuned to ensure the model has appropriate time to train effectively but is still efficient and does not begin too overfit. If left to run for too long, the model eventually will be overfit to the training set and will lose and generalisation it may have gained.

In our model, hyperparameter tuning was conducted using Grid Search, with multiple possible values for learning rate and epochs defined. The grid search then asynchronously trains models using different combinations of these hyperparameters and determines the most optimal combination based on the validation sparse categorical accuracy. Once the optimal combination is found, the optimal model can be extracted and used to predict on the test set.

One issue with hyperparameter tuning in this way is that it can lead to overfitting or disproportional feature importance. One strong way to combat this is to introduce regularisation. Regularisation involves penalising the model loss function based on the weights of the model to encourage a model with lower weights. This is done to reduce model complexity, but also to ensure that no single feature(s) are too important and control the outcome of the predictions too heavily. In practice, L2 regularisation was implemented on the fully connected layers of the CNN model. L2 regularisation refers to when the loss penalty is equal to the sum of the squares of the model weights, which is done to place higher importance on the larger weights of the model. Using regularisation reduces the complexity of the model which thus reduces the tendencies to overfit because the model will have a more equally distributed weighting on each input.

Dropout is another way that overfitting can be avoided. Dropout refers to randomly setting the model weights for some nodes to zero, effectively removing them from the model. This is done to achieve the simplest model possible because nodes that are dropped out but do not have an effect on the performance of the model represent less important data than can possible be completely ignored. In our model two dropout layers with a dropout chance of 0.3 were used, meaning each

layer will drop out roughly 30% of the nodes. This helps to reduce overfitting by reducing the number of nodes and therefor complexity and did lead to an increase in model performance.

After performing these improvements, the CNN model produces the loss graph seen in Figure 6. This graph does look distinctly different to the others in this report, and this is because of the hyperparameter tuning and dropout introduced. Both techniques create noise in the dataset which manifests as a less consistent loss curve in the final model. It is important to note than in the model, the training loss and validation loss curves do decrease and importantly converge on each other. This decrease in both loss curves suggests that the model is training but not overfitting, and the convergence suggests that the model would generalise well onto unseen data. The final model produced a sparse categorical accuracy score on the validation set of around 97%, which is a notable increase over the other models.

Areas for Improvement

Although many techniques were applied to increase the performance of the models in this report, there are still areas that could be improved upon. The first of these is implementing a better hyperparameter tuning process. There are more hyperparameters that could be tuned, including dropout rate and batch size. Due to technical limitations these were not tuned in the report. Increasing the number and scope of hyperparameters being tuned increases the training time exponentially, but ideally all these parameters would be tuned for an optimal model. In addition to this, implementing an early stopping protocol could increase the performance of the model by allowing the final model to predict in a state where overfitting is at a minimum, if overfitting does begin to occur.

Another factor to consider is that the evaluation metric used, sparse categorical accuracy, can be not ideal for situations with a large difference in class quantities. This is because this metric can be biased towards the majority class, which was observed in the untrained CNN model's confusion matrix (Figure 5).

Ultimate Judgement

The final model for both shape and sign type detection were a Convolutional Neural Network. In the case of sign type detection, hyperparameter tuning and regularisation were also applied to increase the performance and generalisation of the model. This final model was measured to be the highest performing both with direct performance measures and intuitive features. The final model had the highest sparse categorical accuracy score on the validation set which was the main performance metric used for this report. This model also had a lower or similar training categorical accuracy than some of the earlier models used. This is a good sign for the generalisation of the model, as it suggests that the model is not overfitting to the training set or weighting certain over-represented features too heavily. Because of these factors, the Convolution Neural Network model with hyperparameter tuning (learning rate and epochs), regularisation and dropout used is the best performing model and likely would be the most appropriate for a real-world application.

Garbage in, Garbage out

The models discussed in this report show high accuracy on the validation set and appear well-balanced between not overfitting and not underfitting. However, the real question is about their practical use. When we tested these models with real-world traffic sign images, they failed to correctly predict any signs. This brings us to a fundamental issue: if the input data isn't good enough, the output won't be either. This is the garbage in, garbage out principle: flawed data going in leads to flawed results, algorithms, and business decisions. For instance, if a self-driving car's decision-making algorithm is trained on data of traffic collected during the day, it wouldn't be prudent to deploy it on roads at night. Extending this further, if such an algorithm is trained in an environment with cars driven by humans, its performance on roads dominated by other self-driving cars would be questionable. Beyond the autonomous driving example, the "garbage in" side of the equation can manifest in various forms—such as incorrectly entered data, poorly packaged data, and data collected incorrectly.

While the BelgiumTS dataset is excellent for testing model capabilities and for academic research, it falls short for real-world applications. Several challenges arise when using real-world images. For one, the model is trained on low-quality 28x28 images, which is quite different from the high-resolution images captured by modern cameras, typically 2048x1536 pixels or close with three colour channels, whereas our model requires single-channel images. Additionally, the dataset images are taken under uniform lighting conditions and from direct angles, which is rarely the case outside of a controlled setting.

Why is high-quality and accessible data foundational? If you're basing business decisions on dashboards or the results of online experiments, you need to have the right data. On the machine learning side, we are entering what Andrei Karpathy, director of AI at Tesla, dubs the Software 2.0 era, a new paradigm for software where machine learning and AI require less focus on writing code and more on configuring, selecting inputs, and iterating through data to create higher-level models that learn from the data we give them. In this new world, data has become a first-class citizen, where computation becomes increasingly probabilistic and programs no longer perform the same action every time they are run. Here, the model and the data specification become more important than the code itself. Therefore, the training data used doesn't adequately prepare the model for the complexities of real-life applications [1].

Real World Applications

The models created in this report have many applications in the real world, both currently and prospectively, but one of the most prevalent is in autonomous cars. These cars often use an array of cameras to detect street signs and directions on the road to aid their navigation. Given this, they must have very sophisticated models to recognise the contents of the sign to make sure the car can act accordingly. In addition to the accuracy of the model, the efficiency is also very important. Sometimes these vehicles are required to make very fast decisions or take in a lot of information

at once (i.e multiple signs on a single post, or conflicting signs that are more obvious to experienced human drivers). This requirement is just as important as the accuracy because there is little benefit to having a 100% accurate recognition model if the results are always available 500ms too late.

This requirement suggests the need for a measurement of a model's efficiency, or how fast a model can make predictions. While this metric can be hard to measure given it depends highly on the hardware running the model, one absolute metric that can be measured is the FLOPS of a model. FLOPS stands for Floating point Operations Per Second and represents the number of mathematical calculations required per second to run a model. This can be interpreted intuitively, as a more complicated model requires more operations, which is thus more difficult / inefficient to run. However, this can also be directly calculated, although this can be quite complicated. The formula for calculating the FLOPS required in a fully connected Neural Network layer is:

FLOPs = 2 x I x O

Where I is the number of input nodes to that layer, and O is the number of output nodes.

Let's consider two small 2x2 matrices A and B, then we have C as an output matrix:

$$C = \begin{matrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{matrix}$$

Where:

$$c_{11} = a_{11}b_{11} + a_{12}b_{21}$$
$$c_{12} = a_{11}b_{12} + a_{12}b_{22}$$
$$c_{21} = a_{21}b_{11} + a_{22}b_{22}$$

Floating Point Operations (FLOPs) can be calculated by counting the number of multiplications and additions performed.

For each element in the resulting matrix C, there are 2 multiplications and 1 addition and there are four elements in the matrix, so total flops:

Total Multiplications = 2 (for each element) ×4 (number of elements) = 8
Total Additions = 1 (for each element) × 4 (number of elements) = 4
Total Additions = 1 (for each element) × 4 (number of elements) = 4

Therefore, the total FLOPs for multiplying two 2x2 matrices is:
Total FLOPs= Total Multiplications + Total Additions = 8 + 4 = 12
So, multiplying two 2x2 matrices involves 12 floating-point operations.

This calculation can be directly applied to our models to evaluate their efficiency, allowing us to choose the most effective model for real-world applications, such as autonomous cars, where both speed and accuracy are critical. By measuring the FLOPs, we can ensure our models are not only accurate but also capable of making rapid decisions essential for real-time navigation.

Conclusion

In conclusion, the models developed in this report demonstrate significant potential for real-world applications, particularly in the domain of autonomous vehicles. The ability of these models to accurately recognize street signs is essential for ensuring the safety and efficiency of autonomous navigation systems. However, accuracy alone is not sufficient; the efficiency of the models, measured in FLOPs, is equally critical. This dual focus on accuracy and efficiency ensures that the models can make rapid decisions in real-time scenarios, a necessity for autonomous cars navigating complex environments with multiple and potentially conflicting signs.

Throughout this project, we implemented and evaluated various classification models, including decision trees, feed-forward neural networks (FFNN), and convolutional neural networks (CNN). We began by creating baseline models and progressively improved them through techniques such as hyperparameter tuning, regularization, and dropout. These efforts led to significant improvements in model performance, culminating in a highly accurate and efficient CNN model.

Moreover, this study underscores the fundamental principle of "garbage in, garbage out." Our real-world tests revealed that despite high accuracy in controlled settings, the models struggled with real-world images due to differences in resolution, lighting, and angle. This highlights the importance of using high-quality, representative datasets for training to ensure the models can generalize well to real-world conditions. By prioritizing both the computational efficiency and the quality of training data, we can develop robust models that not only perform well in theoretical evaluations but also excel in practical, real-world applications. This comprehensive approach sets a strong foundation for further advancements in machine learning applications within the transportation industry.

Reference

The Unreasonable Importance of Data Preparation. O'Reilly Radar. Available at: https://www.oreilly.com/radar/the-unreasonable-importance-of-data-preparation/ [Accessed 30 May 2024].

Appendix

Figure 1 – Class Distribution for sign shape

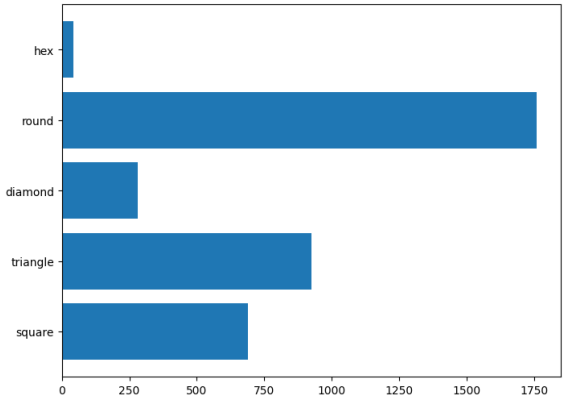

Figure 2 – Class distribution for sign type

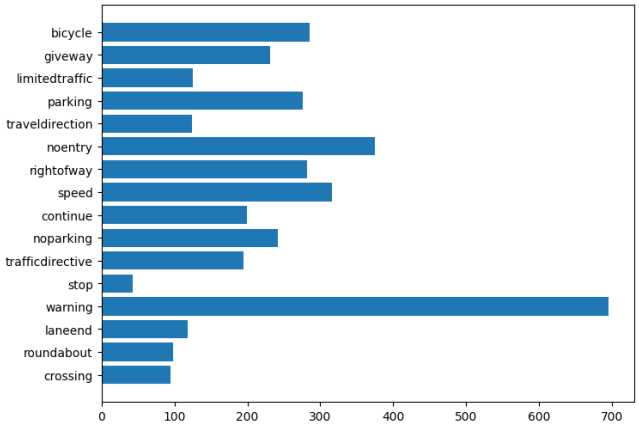

Figure 3 – Decision Tree baseline model confusion matrix

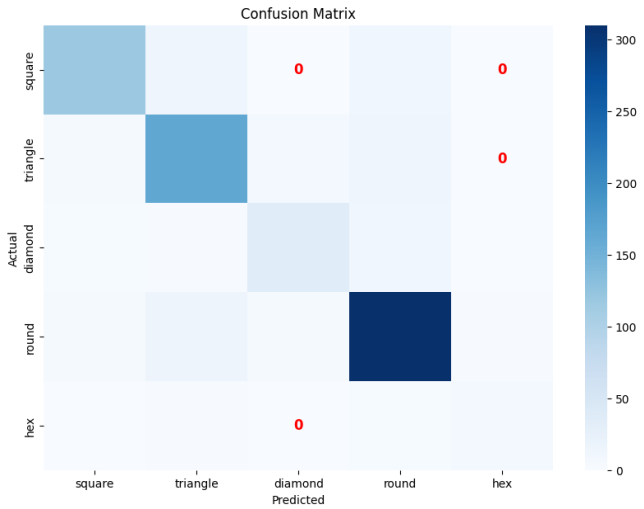

Figure 4 – Sign type FFNN loss graph

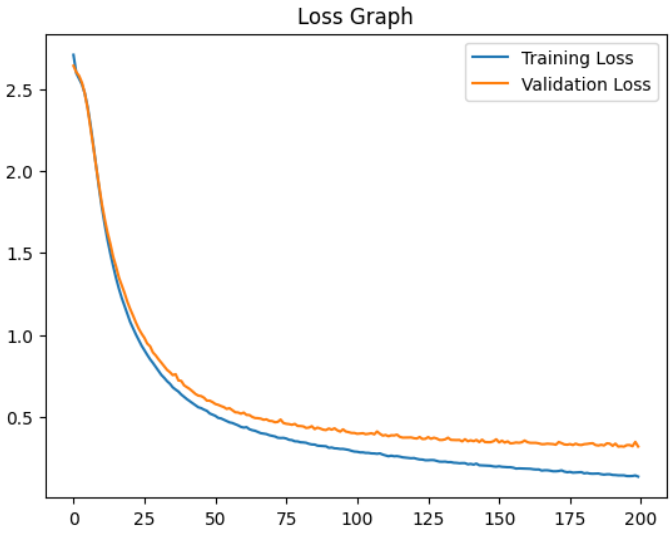

Figure 5- Sign type CNN confusion matrix

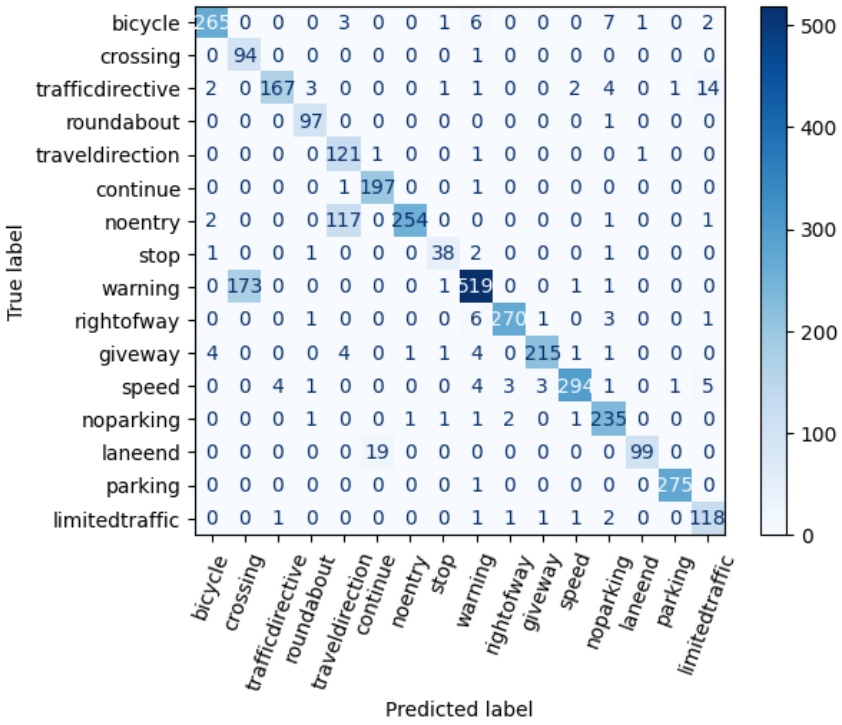

Figure 6 - Sign type tuned CNN loss graph