Traffic signs classification by Mathias Walter Nilsen – Student number: 182435

Intro:

Utilize machine learning to correctly classify traffic signs into their corresponding categories based on images of traffic signs. Something that could be utilized for self-driving cars, such that they may identify and act upon information related to a traffic sign.

Primary models chosen for this task are CNN(Convolutional Neural Network) and MLP(Multi-Layer Perceptron). As both models can represent neurons that can be used to detect objects, like traffic signs. The dataset is from https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign.

Dataset description:

The dataset contains the following: metadata folder consisting of pictures for each sign class. Test and Train folders containing traffic sign pictures utilized in training and testing the model. And csv files containing additional data correlating to each of the folder pictures respectively.
Each training picture class contains varying images of the traffic sign being blurred, zoomed in/out, different brightness, altered angels and some are somewhat obscured. Allowing the models to train on more realistic images which in turn helps to better generalize the models and add robustness to the model. There are total of 39 209 pictures for training the models.

Images utilized for testing are mostly poor of quality, obscured and tilted. Which resembles the real world, making the test results more reliable when testing. In addition to being unseen data, as it is introduced after the models have been fitted on training data. There are in total 12 631 test images.

Utilized libraries:

keras	Utilized for creating and compiling a CNN.
data_loading	Custom library for loading the images from the dataset.
sklearn	For accuracy, score and other statistical metrics.
joblib	For serializing and deserializing python objects.
numpy	For various arrays operations.
matplotlib	For plotting graphs.
pandas	For handling data and storing that into dataframes.
Pillow	For converting images to grayscale from RGB channels
Scipy	Scientific and technical computing.
tensorflow	Machine learning library, wrapped around by keras.

Utilized functions:

Keras:

to_categorical()	Encodes a class vector to a binary class matrix
Sequential()	Instantiate a sequential model in keras
Sequential().add()	Add layer object to model
Sequential().compile()	Compiles the model with given layers
Sequential().fit()	Train model for n epochs using supplied data.
Sequential().predict()	Predicts class probabilities given data
Sequential().save()	Serialize the model for later usage.
Keras.models.load_model()	Loads a saved keras model.
Sequential().summary()	Prints a summary of the model architecture and vital features.

Numpy:

argmax()	Returns maximum indices along an axis

Sklearn:

train_test_split()	Shuffles and splits features and labels into two separate datasets according to supplied ratio.
classification_report()	Returns a summary of various scores based from ground truth and predicted classes.
StandardScaler()	Standardize features by removing the mean and scaling to unit variance. Utilized to reduce the impact from varying values of individual pixels.
confusion_matrix	Compute confusion matrix to evaluate the accuracy of a classification.
accuracy_score	Accuracy classification score for the MLP model.

matplotlib.pyplot:

figure()	Creates a figure
plot()	Plot data on figure
title()	Set figure title
xlabel()	Set label on x-axis
ylabel()	Set label on y-axis
show()	Display all figures
legend()	Place legend on axes
scatter()	Create scatter plot
colorbar()	Add colorbar

Joblib:

Dump()	Store python object
load()	Load stored python object

Pandas:

pd.set_option	Set custom options for pandas
pd.DataFrame()	Create dataframe from matrix

concurrent.futures:

ThreadPoolExecutor()	Generate executor with threads
map()	Map function to arguments

Convolutional Neural Network(CNN) Analysis:

Final architecture for this model where developed through incremental testing. The very first (cnn_model_1.h5) only utilized four basic layers: Conv2D, MaxPool2D, Flatten and Dense. Where Conv2D utilized 8 filters with a 3x3 kernel_size and ReLu activation function. MaxPool2D used pool_size of 2x2 and Dense consisting of 43 neurons, one neuron per class as it is the last layer, using SoftMax as its activation function. And loss function being categorical cross entropy.

This architecture received an average f1-score of 0.80 and contained a total of 67 467 trainable parameters. Making it a decent and small model for classifying the test data, though there are potential for improvement.

Kernel size determines how fine grained the extracted features are. Where smaller size yields finer granularity features, leading to higher accuracy of the model. Though due to the image size being 30x30 in conjunction to many of the pixels having similar neighbors in the first layer, it gained better results by increasing the first conv layer kernel size from 3x3 to 5x5. As this allows for detecting more generalized features present in the image early on, leading to higher accuracy on test data.

Though kernel sizes in deeper part of the model should be as fine grained as possible. As this allows the model to extract more descriptive features than that of a 5x5 kernel size. Therefore, the rest of the model utilizes a kernel size of 3x3 in its deeper layers.

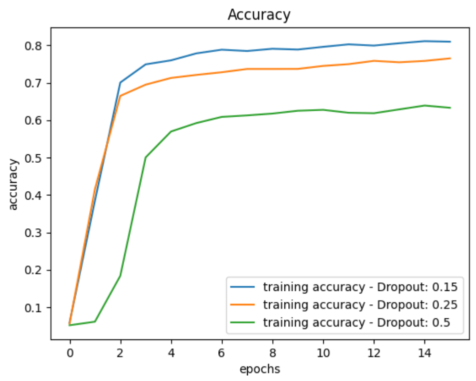


Figure 2 - Accuracy related to early dropout layer

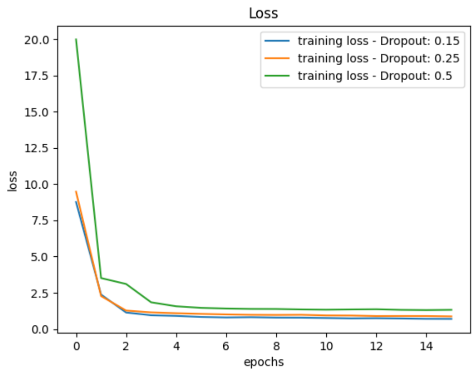


Figure 1 - Loss related to early dropout layer

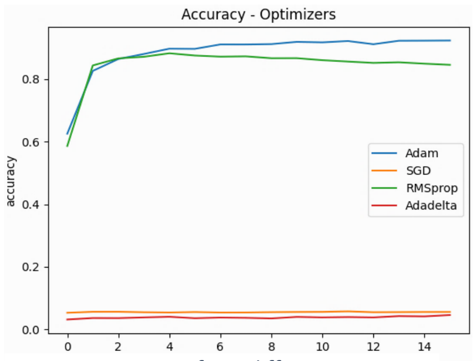


Figure 3 - Accuracy from different optimizers

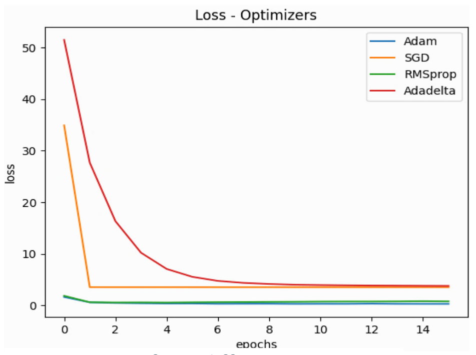


Figure 4 - Loss from different optimizers

Increasing number of filters per layer allowed for additional features to be detected. Something that led to increased correct classifications. Though also increased computational time per conv2d layer.

Adding an early dropout layer allows for increased generalization and robustness of the network. Something that was confirmed by adding this layer after the first MaxPooling layer, turning of 25% random neurons. Which increased accuracy for the model. Any higher percentage and it caused lowered accuracy, whilst any lower, and the network lost robustness, yielding lower accuracy on the test data. Though it received higher accuracy per epoch, during the training phase. As illustrated in *figure 1*.

It is observed that loss values for initial epochs are higher with the usage of an early dropout layer, though is quickly mitigated after the initial training phase. In addition to this the initial loss shows some correlation to the dropout percentage. Where the higher the dropout value, the higher the initial loss is. Thereby requiring further epochs in order to level out, than that of a lower dropout value, as shown in *figure 2*.

MaxPooling serves the purpose of preserving important features from neighboring pixels whilst removing redundant pixels, making it useful for the earlier part of the network. As if used too much would reduce the image dimensions to much, yielding a lower accuracy result.

Placing an additional MaxPooling layer after the last two stacked Conv2D layers, allows for making extracted features more present. Something that also reduces number of total trainable parameters and removes further redundant pixels. Focusing on features.

Optimizers allows the model to converge faster and more accurately, minimizing loss of the cost function. Among the most popular optimizers are SGD, RMSprop, Adadelta and Adams, where Adams is the most popular and the one that faired best. As illustrated in *figures 3 and 4*.

The others converged either on a higher minima, resulting in a lower accuracy score or created unstable loss gradients causing them to vanish. The vanishing gradients problem is somewhat present with Adam. Though heavily mitigated with the usage of ReLu activation functions, combined with Keras utilizing mini batches during model fitting.

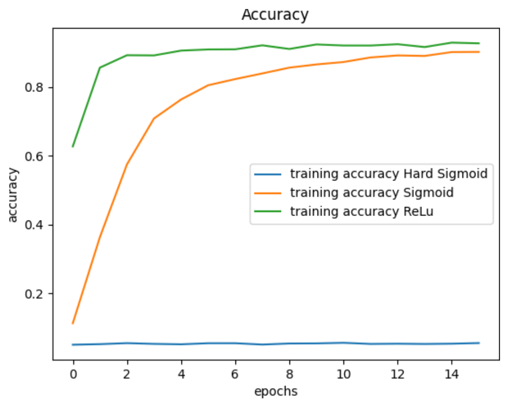


Figure 5 - Accuracy using different activation functions

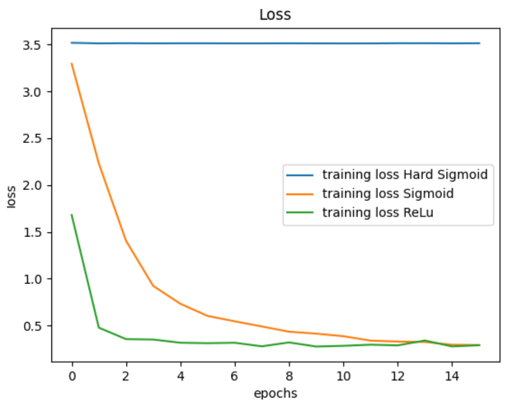
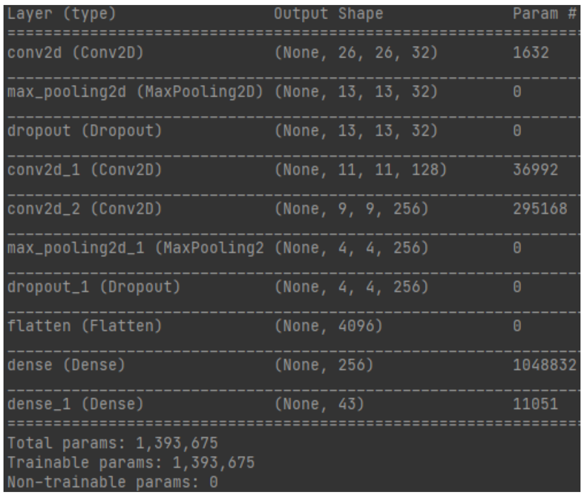


Figure 6 - Loss using different activation functions

Utilizing ReLu allows for turning off pixels that won't contribute or don't matter for result after each convolution and dense layer. Altering to other activation functions led to increased loss for all epochs and a lower average accuracy as well reduced f1-score for several classes. Though it is observed that the sigmoid activation function, albeit it came close to the same accuracy as ReLu, it did reach an upper limit lower than ReLu. Making ReLu the better choice.



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        1632
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
dropout (Dropout)            (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 128)       36992
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 9, 9, 256)         295168
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 256)         0
_________________________________________________________________
dropout_1 (Dropout)          (None, 4, 4, 256)         0
_________________________________________________________________
flatten (Flatten)            (None, 4096)              0
_________________________________________________________________
dense (Dense)                (None, 256)               1048832
_________________________________________________________________
dense_1 (Dense)              (None, 43)                11051
=================================================================
Total params: 1,393,675
Trainable params: 1,393,675
Non-trainable params: 0
```

Figure 7 - Final model summary of layers and parameters

Robustness for CNN is achieved via dropout layers combined with resizing the image to 30x30 and converting it to grayscale. Something that also reduces the number of channels per image as well as number of parameters. Which in turn increases performance.

Further robustness could be achieved by creating new images by altering the originals via things like flip, blur or zooming or other filters. Though this has already been done as the images are taken from a sequence of shoots of a traffic sign, at varying distance, resolution and brightness.

Figure 7 showcases the final architecture and corresponding summary of the model. Which contains details of various layers as well as an overview of trainable parameters pertaining to the model and individual layers.

Multilayer Perceptron(MLP) Analysis:

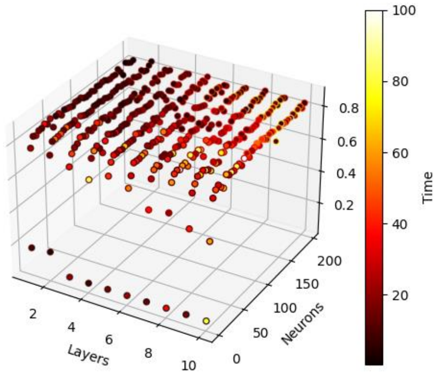


Figure 8 - Illustrates correlation between number of layers, neurons, process time and accuracy of the mode

By varying number of layers and neurons per layer, it is observed that this model reaches an upper accuracy limit of around 0.82 as shown in *figure 8.* Where increasing layers and neurons led to increased training time.

All pixels are utilized making the model both computational and memory expensive as it must hold and compute all neurons. Which in turn may or may not be redundant as not all neurons are utilized nor needed for the model to correctly classify a traffic sign. Another observation is that more neurons utilized, yields higher accuracy. Though may be redundant as not all neurons are utilized or needed.

Increasing the number of layers, allows the mode to represent convex regions. Thereby allowing one to separate between the various classes in some high dimensional space, leading to higher accuracy. Though too many may have the opposite effect, causing miss classifications, lowering the accuracy. In addition to reducing computational speed as more layers require more computational power.

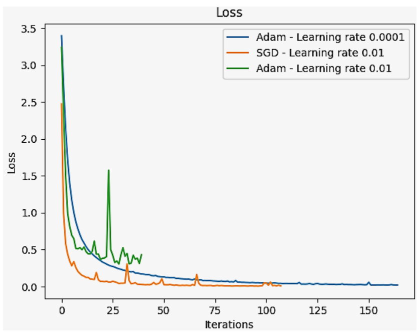


Figure 9 - Loss per iteration with optimizers Adam and SGD

Therefore, the most optimal configuration that where found was with three layers: (150, 110, 43). Which received an average f1-score of 0.82, making it a decent model for classifying traffic signs.

There are some solvers to choose from. Here Adam and SGD where chosen, due to them being the most popular. It is observed that Adam requires additional iterations compared to SGD for it to converge on a smaller loss value than SGD. As is evident in *figure 9*.

Another observation from *figure 9*, is that Adam works best with a smaller learning rate than SGD. Which means it is an algorithm that converges more slowly towards a lower minima. Making it analogous to being like a heavy ball, slowly converging towards the lowest point of a slope.

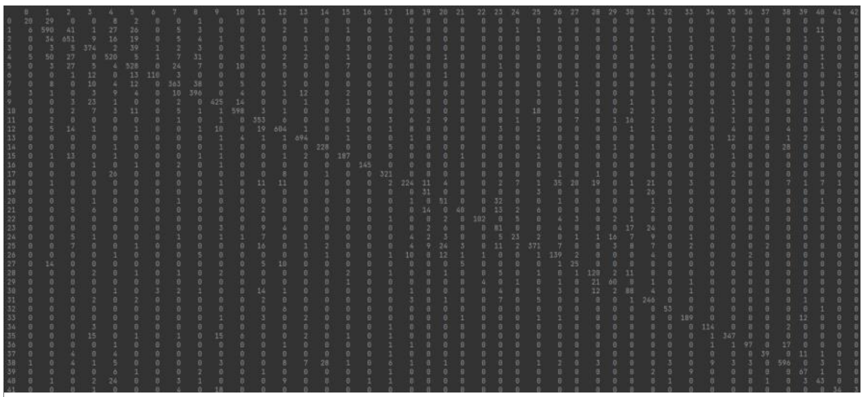


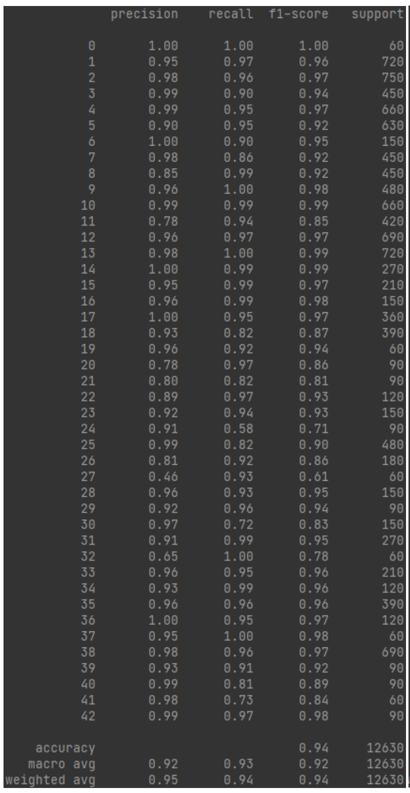
Figure 10 - Confusion matrix from classifying the test data using the MLP model

Analysis of the confusion matrix reveals that miss classification mostly stems from traffic signs that are similar both in shape and other features. Like for example the different speed signs. In addition to the model being locally dependent on individual pixels instead of features for traffic sign identification.

MLP VS CNN Conclusion:

Based on the precision, recall and f1-score from the two models it is observed that *CNN* faired the best out of the two models, as is evident in *figures 11 and 12.*

Classes with lower score from the model *MLP(figure 12)* stems mostly from localized features. Causing the model to not properly classify the object as it may not be statically located in all images. Thereby requiring a larger amount of training data.

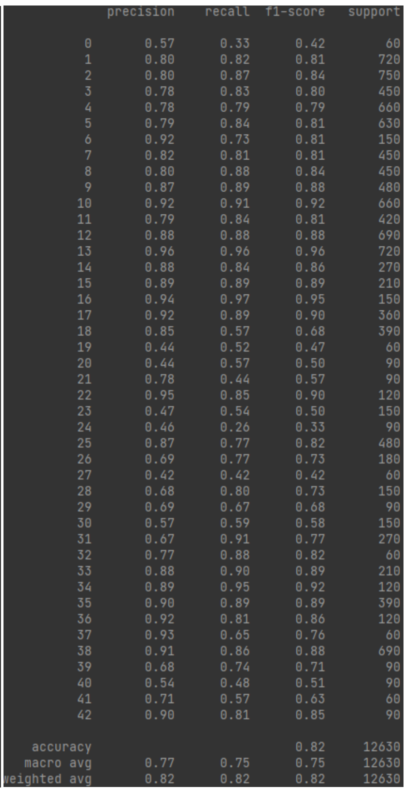


```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        60
           1       0.95      0.97      0.96       720
           2       0.98      0.96      0.97       750
           3       0.99      0.90      0.94       450
           4       0.99      0.95      0.97       660
           5       0.90      0.95      0.92       630
           6       1.00      0.90      0.95       150
           7       0.98      0.86      0.92       450
           8       0.85      0.99      0.92       450
           9       0.96      1.00      0.98       480
          10       0.99      0.99      0.99       660
          11       0.78      0.94      0.85       420
          12       0.96      0.97      0.97       690
          13       0.98      1.00      0.99       720
          14       1.00      0.99      0.99       270
          15       0.95      0.99      0.97       210
          16       0.96      0.99      0.98       150
          17       1.00      0.95      0.97       360
          18       0.93      0.82      0.87       390
          19       0.96      0.92      0.94        60
          20       0.78      0.97      0.86        90
          21       0.80      0.82      0.81        90
          22       0.89      0.97      0.93       120
          23       0.92      0.94      0.93       150
          24       0.91      0.58      0.71        90
          25       0.99      0.82      0.90       480
          26       0.81      0.92      0.86       180
          27       0.46      0.93      0.61        60
          28       0.96      0.93      0.95       150
          29       0.92      0.96      0.94        90
          30       0.97      0.72      0.83       150
          31       0.91      0.99      0.95       270
          32       0.65      1.00      0.78        60
          33       0.96      0.95      0.96       210
          34       0.93      0.99      0.96       120
          35       0.96      0.96      0.96       390
          36       1.00      0.95      0.97       120
          37       0.95      1.00      0.98        60
          38       0.98      0.96      0.97       690
          39       0.93      0.91      0.92        90
          40       0.99      0.81      0.89        90
          41       0.98      0.73      0.84        60
          42       0.99      0.97      0.98        90

    accuracy                           0.94     12630
   macro avg       0.92      0.93      0.92     12630
weighted avg       0.95      0.94      0.94     12630
```

Figure 11 - CNN classification report



```
              precision    recall  f1-score   support

           0       0.57      0.33      0.42        60
           1       0.80      0.82      0.81       720
           2       0.80      0.87      0.84       750
           3       0.78      0.83      0.80       450
           4       0.78      0.79      0.79       660
           5       0.79      0.84      0.81       630
           6       0.92      0.73      0.81       150
           7       0.82      0.81      0.81       450
           8       0.80      0.88      0.84       450
           9       0.87      0.89      0.88       480
          10       0.92      0.91      0.92       660
          11       0.79      0.84      0.81       420
          12       0.88      0.88      0.88       690
          13       0.96      0.96      0.96       720
          14       0.88      0.84      0.86       270
          15       0.89      0.89      0.89       210
          16       0.94      0.97      0.95       150
          17       0.92      0.89      0.90       360
          18       0.85      0.57      0.68       390
          19       0.44      0.52      0.47        60
          20       0.44      0.57      0.50        90
          21       0.78      0.44      0.57        90
          22       0.95      0.85      0.90       120
          23       0.47      0.54      0.50       150
          24       0.46      0.26      0.33        90
          25       0.87      0.77      0.82       480
          26       0.69      0.77      0.73       180
          27       0.42      0.42      0.42        60
          28       0.68      0.80      0.73       150
          29       0.69      0.67      0.68        90
          30       0.57      0.59      0.58       150
          31       0.67      0.91      0.77       270
          32       0.77      0.88      0.82        60
          33       0.88      0.90      0.89       210
          34       0.89      0.95      0.92       120
          35       0.90      0.89      0.89       390
          36       0.92      0.81      0.86       120
          37       0.93      0.65      0.76        60
          38       0.91      0.86      0.88       690
          39       0.68      0.74      0.71        90
          40       0.54      0.48      0.51        90
          41       0.71      0.57      0.63        60
          42       0.90      0.81      0.85        90

    accuracy                           0.82     12630
   macro avg       0.77      0.75      0.75     12630
weighted avg       0.82      0.82      0.82     12630
```

Figure 12 - MLP classification report

In CNN the classes with lower score are mainly caused by a lack of images used for training. In conjunction with that several of the traffic signs share many similar features. Improving those scores could done by adding several new image data, all with varying data augmentations, in conjunction to making the model deeper. Thereby allowing the model to focus on both additional features and complex features, present in a traffic sign.

Another difference between the two models is that *MLP* will learn different interpretations on something that is possibly the same. Making the model wasteful with regards to memory. In contrast to this, in *CNN*, the kernel weights depend on the kernel size. Which combined with pooling allows the model to focus on spatial features and not on local features. Something that reduces both computational speed and memory usage, in contrast to *MLP*.

In addition to this, CNN considers spatial invariance in images which reduces the amount of required training data. Something that *MLP* cannot. Which makes it a bad fit for pictures taken during drive time, and thereby an unfit model for this type of classification. It is also worth noting that in practice, the classification would not necessarily consider a single image, but several, taken in sequence. And utilize top voting amongst those to make the final classification.