SCHOOL OF MECHATRONIC SYSTEMS ENGINEERING
SIMON FRASER UNIVERSITY

MSE491 D100 Applications of Machine Learning in Mechatronic Systems

Final Project – Traffic Sign Identification

March 23, 2021

Gurpreet Singh Marwaha	301367972
Chris Kurian Vattathichirayil	301328090

Introduction

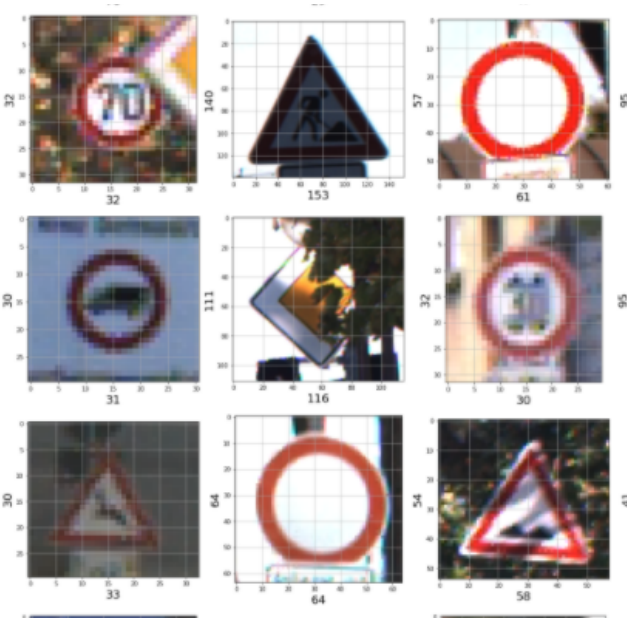
In this project we will be categorizing traffic sign images into one of 43 categories. The dataset used will be the GTSRB - German Traffic Sign Recognition Benchmark found on GTSRB - German Traffic Sign Recognition Benchmark | Kaggle. competition held at IJCNN 2011
One area of autonomous vehicles and driver assistance research is identification of various road information using cameras. One of the important functionality is recognition of traffic road signs. The traffic signs dataset is used for a Single-image, multi-class classification problem 43 classes, more than 50,000 real life images in total.

Libraries used in this project include, PIL, numpy, pandas, matplotlib.pyplot, cv2, tensorflow, tensorflow_datasets, tensorflow, tensorflow.keras, random, os, and sklearn.model_selection. The library used to make the neural networks will be Keras API. The Keras API includes all features to make a customized neural network.The algorithms we will be using in this project are Artificial Neural Networks(ANN) and Convolutional Neural Networks. The reason behind implementing ANN is that its ability to detect the complex non-linear relationships and in case of CNN we know that this is the best Deep learning technique to implement for image processing as its ability to apply appropriate filters which in turn helps to capture the temporal dependencies and also this algorithm reduces the number of parameters involved to produce the better fitting image of the selected data.

In this project we will be training data by using both ANN and CNN with different parameters like filters,density etc. The reason behind using different algorithms is to showcase which one fits best for implementing in case of training images, the comparison between the algorithms is done by comparing the obtained accuracies.

Preprocessing

The dataset folder is saved in the D drive and contains three .csv files and three folders for Test, Train and Meta. The .csv files contain information such as the image path, the size, We begin by an output of randomly selected test images and their dimensions.



The next step is to retrieve images from the Train folder, resizing and appending the data for each image into a numpy array using a nested for loop. The first for loop iterates for each subfolder for each class from 0 to 42, and the second for loop iterates for each image within the specified class folder and appends the image and label information to separate arrays.

The shape of the image array and labels array is (39209, 30, 30, 3) and (39209,) where 39209 is the total number of images in all folders, 30x30 is the dimension of the images and 3 is the number of colour channels. Each element in the array has three numbers that tell the RGB values of the pixel with a total of 900 pixels for each image. The labels for each image are a number from 0 to 42 denoting the class of traffic sign.

```
print(data.shape, labels.shape)
print(data)
print(labels)
```

```
(39209, 30, 30, 3) (39209,)
[[[[ 75  78  80]
```

Using the numpy arrays of images and labels, the elements within the array are split into training and testing data with an 80/20 split of the array of all data within the Train folder. The Test folder will be used for validation at a later step. We can see that the total of 39209 images have been split into 31367 and 7842 images for training and testing respectively. The X_train and X_test are for the images and Y_train, Y_test are for labels.

```
print(X_train.shape, X_test.shape, Y_train.shape, Y_test.shape)
print(X_train)
```

```
(31367, 30, 30, 3) (7842, 30, 30, 3) (31367,) (7842,)
[[[[ 30  28  28]
```

Before making our neural network model, we must use One Hot encoding of the label arrays. This form of encoding is recommended for categorical datasets because instead of an integer value as the output,

there is an array the length of total categories as the output for each label where all elements except one are zero. Depending on the value of each label, the corresponding position in the array is a 1. Using the following code this is achieved.
Y_train = to_categorical(Y_train,classes)
Y_test = to_categorical(Y_test,classes)

```
Y_train = to_categorical(Y_train,classes)
Y_test = to_categorical(Y_test,classes)
print(Y_train)
```

```
[[0. 0. 0. ... 0. 0. 0.]
```

One-hot encoding is required for use in a neural network because the output of the network will be an array with the number of elements equal to the total classes or categories in this case, 43.

Methods

We will be building neural networks using Keras, explaining the functionality of each of the options and the objective will be to discuss which settings make for an efficient model that can be trained with highest accuracy while reducing training time. Comparisons of several different ANN's and CNN's with different numbers of layers and settings will be summarized in the tables to see what factors have a positive effect on reducing training time and increasing accuracy.

Artificial Neural Network (2 layers)

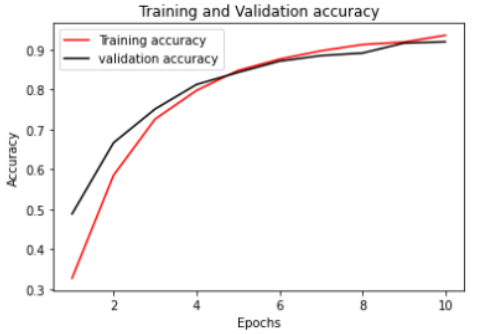
For an artificial neural network we only use Dense and flattening layers. Passing in an input shape the size of X_train to a fully connected Dense layer of size 5 <n<45 step size 5. The output of the first Dense layer is flattened and the second Dense layer output is to 43 outputs.

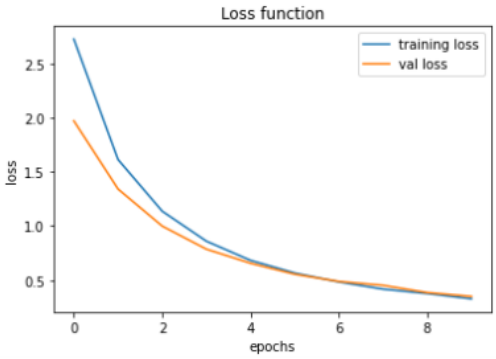
```
ANNmodel_2layer = Sequential()
ANNmodel_2layer.add(Dense(20, activation='relu', input_shape=X_train.shape[1:]))
ANNmodel_2layer.add(Flatten())
ANNmodel_2layer.add(Dense(43, activation='softmax'))

#Compilation of the model
ANNmodel_2layer.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

epochs = 10
history = ANNmodel_2layer.fit(X_train, Y_train, batch_size=1000, epochs=epochs, validation_data=(X_test, Y_test))
ANNmodel_2layer.save("ANNmodel_2layer.h5")
```

Number of Dense	Total time (s)	Val_accuracy
3	21 (10 epochs)	0.7993
3	41 (20 epochs)	0.8518
5	20 (10 epochs)	0.7211
15	38 (10 epochs)	0.8957

25	71 (10 epochs)	0.9152
35	102 (10 epochs)	0.9027
45	128 (10 epochs)	0.9193

From the table we can see that the increasing the number of units in the first dense layer has the effect of converging the model to the maximum accuracy in less epochs. With only three units as the output dimension, the accuracy of the model is around 80%, at 25 units, the accuracy of the model is maximized at 91.93 % within the two layer approach. It was observed that any extra units have no substantial effect on accuracy. This is because the identifiable features in the dataset using a Dense layer converges to a constant value.





Convolutional Neural Network

Two layer CNN:

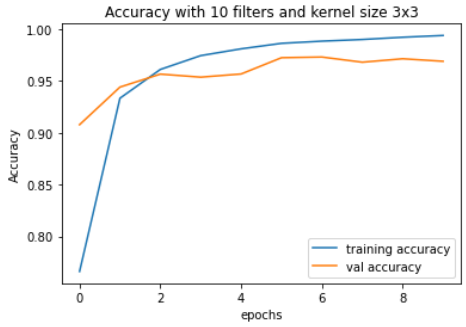
In a convolutional neural network, the number of filters is the dimensionality of the output space and kernel size specifies the height and width of the convolution window. The second layer remains as a Dense layer with output dimension 43 which is required for sorting the categories.

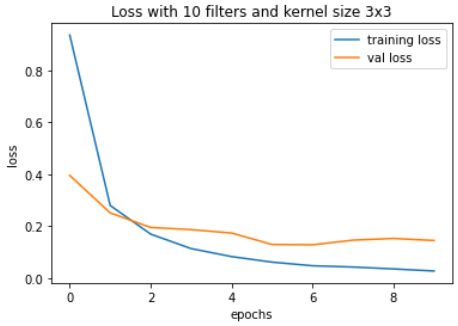
```
CNNmodel_2layer = Sequential()
CNNmodel_2layer.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=X_train.shape[1:]))
CNNmodel_2layer.add(Flatten())
CNNmodel_2layer.add(Dense(43, activation='softmax'))
#Compilation of the model
CNNmodel_2layer.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
epochs = 10
history = CNNmodel_2layer.fit(X_train, Y_train, batch_size=32, epochs=epochs, validation_data=(X_test, Y_test))
CNNmodel_2layer.save("CNNmodel_2layer.h5")
```

Number of Filters	Kernel size	Total time (s) (10 epochs)	Val_accuracy
1	3,3	72	91.24
1	5,5	120	91.57
2	5,5	121	94.62

3	3,3	82	95.08
3	5,5	143	96.20
5	5,5	128	97.07
10	3,3	102	97.69
10	5,5	121	97.37
15	3,3	147	97.51
20	5,5	184	97.49

From the table we can see that increasing the Kernel size may or may not increase the training accuracy but always increases training time. The optimum kernel window size was found to be three, and the increasing the number of filters past 10 had negligible effect on the validation accuracy. The best accuracy was a kernel size of 3,3 window size and 10 filters in total yielding an accuracy of 97.49% accuracy.

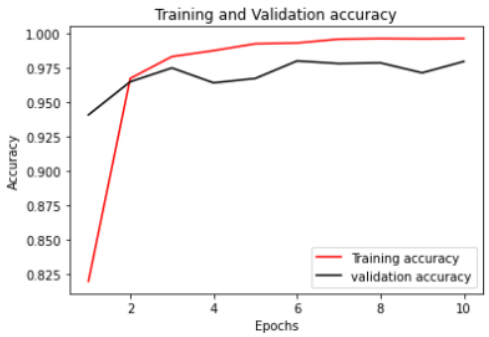


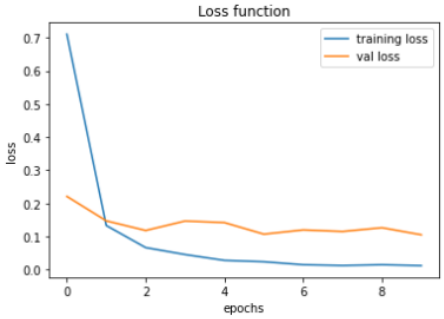


Three layer CNN:

By combining the optimum dense layer with 25 units with the optimum convolutional layer of 10 filters and kernel size three, we are able to create a three layer model with an even higher validation accuracy of 98.13%.

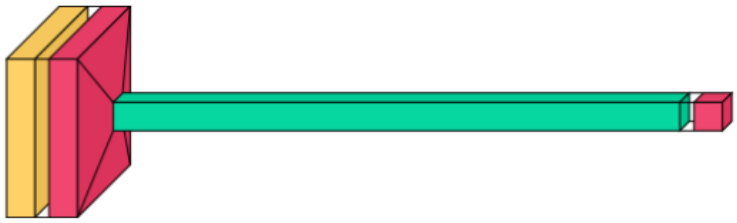
```
CNNmodel_3layer = Sequential()
CNNmodel_3layer.add(Conv2D(filters=10, kernel_size=(3,3), activation='relu', input_shape=X_train.shape[1:]))
CNNmodel_3layer.add(Dense(25, activation='relu', input_shape=X_train.shape[1:]))
CNNmodel_3layer.add(Flatten())
CNNmodel_3layer.add(Dense(43, activation='softmax'))
#Compilation of the model
CNNmodel_3layer.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
epochs = 10
history = CNNmodel_3layer.fit(X_train, Y_train, batch_size=32, epochs=epochs, validation_data=(X_test, Y_test))
CNNmodel_3layer.save("CNNmodel_2layer.h5")
```






```
In [36]: import visualkeras
         visualkeras.layered_view(CNNmodel_3layer)
```

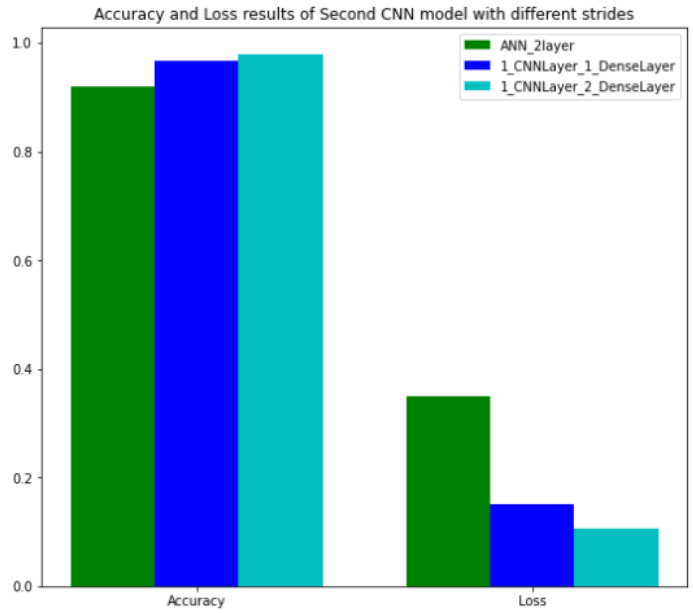
Out[36]:

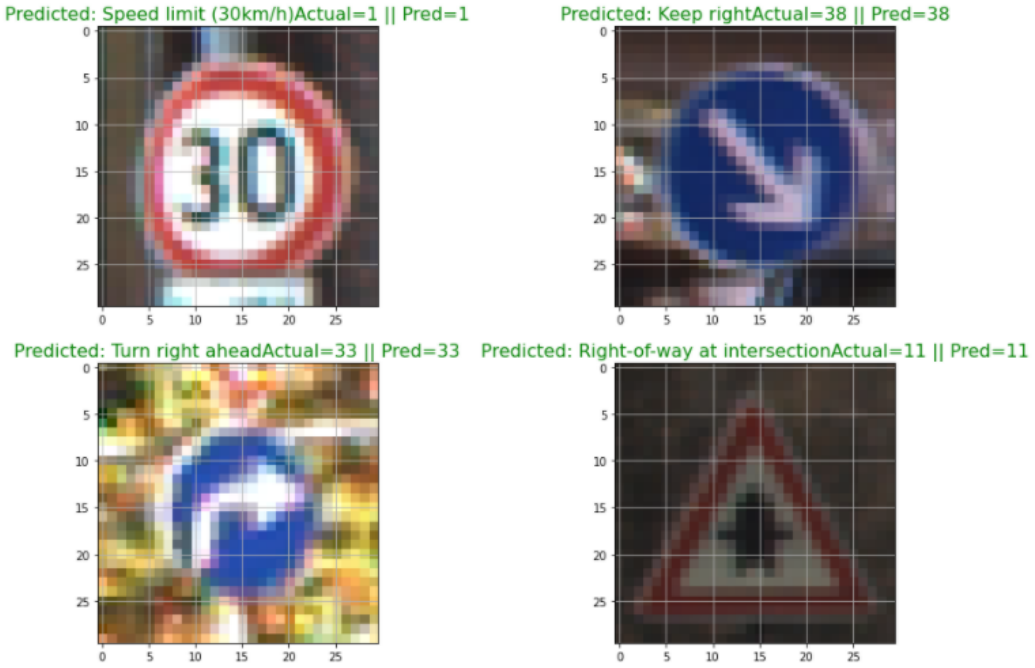


The three layers of 2D convolution, and two dense layers can be seen by the visual keras layered view. The coloured blocks represent the dimensions of the data arrays. Observe that after the flatten function, the length of the array increases represented by the green block.

Results and Discussions

From running three different configurations of neural networks we found the optimized network configuration that achieves the highest accuracy. The first network was a simple two layered artificial neural network where we tested for the effect that the output dimension of the first Dense layer had on the validation accuracy within ten epochs. The best output dimension was found to be 25 units resulting in a prediction accuracy of 91.93% . The second network was a two layered neural network with the first layer being a 2D convolutional layer and the second layer being a Dense layer. The best configuration for the convolutional layer was found to be 10 filters and a kernel size of 3 resulting in a prediction accuracy of 97.49%. The final model was to combine three layers with one 2D convolutional layer and two Dense layers of output dimensions of 25 and 43. This final model achieved an accuracy of 98.13%. The summary of the accuracies of the different networks are shown in the bar graph.





After saving the final trained model, the classes are labeled and randomized images from the test dataset are run through the neural network and the predicted vs actual classifications are shown at the bottom of each image. The reason for the 98.13% accuracy is accounted for due to some images being too dark to detect any features as shown by the image in the first row third column.

Conclusion

To sum up, the idea behind selecting this project “Traffic Sign Identification” is that it can be implemented in autonomous vehicles to reduce the number of accidents on the roads. The death rates due to road crashes are rising by huges numbers every year so using self-driving cars trained by implementing machine learning algorithms with much greater accuracy would help to reduce the amount of road accidents. In this project we used two different techniques named ANN and CNN with parameters to train the selected dataset of traffic signals.. When we implemented the ANN algorithm we obtained the accuracy of the training data to be 91.93% on the other hand in case of CNN using single layer we obtained the accuracy of 97.49% which was way greater than the accuracy obtained in ANN algorithm then in order to improve the accuracy we again implemented CNN but with 2 layers and we obtained the accuracy to be 98.13% which was far greater than the one obtained with 1 layer using CNN. So used the model with higher accuracy to predict the traffic signals in our project. Overall this project was fun, the algorithms learnt during the lectures and while performing the labs has helped us a lot in achieving our desired goal in this project.

References

Dense layer (keras.io)

Adam (keras.io)

Layer activation functions (keras.io)

Keras layers API

Losses (keras.io)

Accuracy metrics (keras.io)

Python Project on Traffic Signs Recognition with 95% Accuracy using CNN & Keras - DataFlair (data-flair.training)

GTSRB - German Traffic Sign Recognition Benchmark | Kaggle