MTH 5320

Neural Networks

Project 2: Traffic Sign Classification

Sagar Jagtap

ID: 903935334

Contents

Table of Figures

Table of Tables

1) Objective:

The main goal of this project is to perform Traffic Sign Classification from cropped images of traffic signs such speed limits, stop sign, and right turn ahead to name a few. This sort of classification is performed very commonly in the autonomous vehicle industry. The aim of this project is to implement Convolution Neural Network based deep learning algorithms, benchmark their performance, and produce a high performing model.

2) Dataset:

The dataset contains images was sourced from Kaggle;

https://www.kaggle.com/valentynsichkar/traffic-signs-preprocessed

The dataset contains 51839 unique cropped RGB images of traffic signs stored in pickle file format. The images are split into three sets: train set with 34799 images, validation set with 4410 images, and test set with 12360 images. The dataset also contains images stored in different files with augmented images with different normalizations. The dataset also contains bounding box information for all images of the traffic signs.

ClassId	SignName
0	Speed limit (20km/h)
1	Speed limit (30km/h)
2	Speed limit (50km/h)
3	Speed limit (60km/h)
.	.
.	.
.	.
40	Roundabout mandatory
41	End of no passing
42	End of no passing by vehicles over 3.5 metric tons

3) Data Preprocessing:

3.1) Data Distribution

The distribution of train, validation, and test sets are visualized below. The distribution for train set is not uniform as there is a large difference in number of images available for some of the classes. For instance, there are 2010 images for class Id 2, but only 180 images for class Id 0. Although there is a big disparity between the number of images for each class in the train set, the distribution is also mimicked by data in validation set and test set. Therefore, data

augmentation might help train a less biased model, but its benefit may not reflect on test or validation set accuracies.

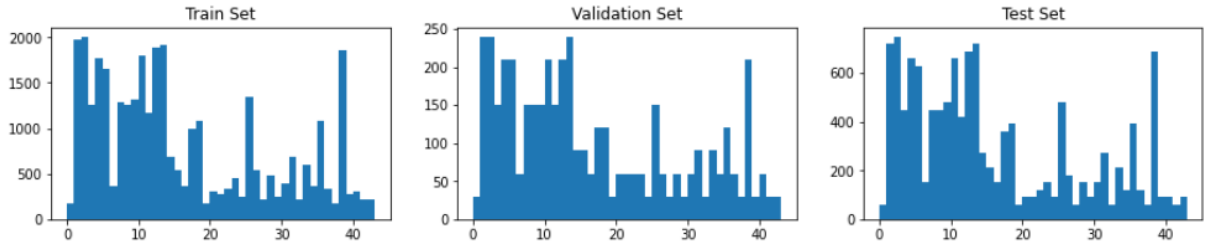

Figure 1. Histogram of image data distribution

3.2) Bounding Boxes

Random images from the dataset along with bounding boxes are plotted in the image below. The bounding boxes however only sometimes correctly demarcates traffic sign in the image properly. In some of the images below the bounding boxes also contain large portions of the image that are outside the traffic signs, and sometimes cuts out the edges of a sign.

Figure 2. Train Set with bounding boxes

Since the images are already cropped with the traffic signs at their center, there is no need to crop the images again the with bounding boxes given. For rest of this project these bounding boxes were ignored because the images are cropped well, and the bounding boxes provided are inaccurate.

3.3) Data Augmentation

Data augmentation was performed to equalize distribution of images in the train set to avoid model bias towards some class of data. A few augmentations were tested using some custom functions and, using the ImageDataGenerator() function from the keras library.

The image below compares the original image on the left, with the same image augmented for brightness and rotation using custom functions (in the middle) and using keras on the right.

Figure 3. Comparison of augmentation using custom functions and keras

The following images compare augmented data with nearest-fill (left) and zero-fill (right) options. The sharp edges of the zero-fill option could affect our CNN models negatively. This was also found to be the case when both the augments were tested as train set for LeNet CNN.

Figure 4. Augmentation with nearest fill (left) and zero-fill (right)

Following combinations of train set data was tested on a LeNet to see how much effect each of them could have on model performance. Each model was run three times to ensure that no output was a result of a bad minima encountered by chance.

	Using custom	Using keras	Rotation	Brightness	Height shift	Width shift	Zoom	Nearest fill	Zero fill	Normalization
Unequalized Distribution										Not performed
Unequalized Distribution										Divide by 255
Augment 1		✓	✓	✓				✓		Divide by 255
Augment 2		✓	✓	✓	✓	✓			✓	Divide by 255
Augment 3		✓	✓	✓	✓	✓		✓		Divide by 255
Augment 4	✓		✓	✓					✓	Divide by 255
Augment 5		✓	✓	✓				✓		Standard
Augment 6		✓	✓	✓	✓	✓	✓	✓		Divide by 255

Table 1. Augments tested

The results for each of the above testes are presented below.

Data	Validation Set Accuracy (Runs:1,2,3)	Test Set Accuracy (Runs:1,2,3)
Unequalized	92.5, 90.5, 94.4	91, 92, 92
Unequalized and normalized	95.58, 95, 93.42	94, 94, 93
Augment 1	96, 95, 94.26	93, 94, 93
Augment 2	93, 92, 91	90, 93, 91
Augment 3	92, 92.5, 9.4	91, 93, 93
Augment 4	95, 94, 94.7	94, 92, 93
Augment 5	94, 92, 94.7	92, 92, 93
Augment 6	89.6, 90.14, 89.66	90, 90, 89

Figure 5. Comparison of performance of different train data

Inferring from the results above, normalized train set with non-uniform distribution performs equally well as the distribution equalized with augment 1 configuration. This is could be because the same initial distribution is consistent with the distribution of validation and test sets. For this project, the data prepared with augment 1 will be used for further training.

3.4) Train, Validation and Test Data

After applying *Augment 1* (Table 1), the augmented train set was combined with the train initial images, shuffled, normalized, and stored as a pickle file. The augments were performed using keras ImageDataGenerator function and is illustrated in the code snippet below.

```
# create image data augmentation generator
datagen = ImageDataGenerator(rotation_range=13,brightness_range=[0.2,1.8])

# Create augmented images
imageGen = []
labelGen = []
for num in range(0,43,1):
    if np.size(np.where(train_labels==num),axis=1) < maxSize:
        print('[INFO] Augementing images for label: ' + '[' + str(num) + '] ' + labels['SignName'][num])
        indices = np.where(train_labels==num)
        # train_labels[train_labels.tolist().index(num):train_labels.tolist().index(num)+np.size(indices,axis=1)]

        # Prepare data to sample
        data = train_images[train_labels.tolist().index(num):train_labels.tolist().index(num)+np.size(indices,axis=1)]
        samples = data
        # prepare iterator
        it = datagen.flow(samples, batch_size=1)
        # generate batch of images

        count = 0
        # Generate images to equilize histogram
        for epoch in range(maxSize - np.size(np.where(train_labels==num),axis=1)):
            count += 1
            batch = it.next()
            imageGen.append(np.squeeze(batch.astype('uint8')))
            labelGen.append(num)
```

Figure 6. Generate augmented image set for histogram equalization.

Each time the test and validation sets are loaded, they are normalized using the same normalization method used for train set. The following code shows the implementation of this.

```
# Load augmented data images
data = pickle.load(open(DATA_DIR + 'data_aug_1.pickle', 'rb'))

# Extract Train Images
train_images = data['images']
train_labels = data['labels']

# Load validation and test files
valid = pickle.load(open(DATA_DIR + 'valid.pickle', 'rb'))
test = pickle.load(open(DATA_DIR + 'test.pickle', 'rb'))
labels = pd.read_csv(DATA_DIR + 'label_names.csv')

# Extract validation and test image and label sets
valid_images = valid['features']
valid_labels = valid['labels']
test_images = test['features']
test_labels = test['labels']

# Normalize validation and test set
valid_images = preprocess_data(valid_images,norm_255=True)
test_images = preprocess_data(test_images,norm_255=True)
```

Figure 7. Load Train, Validation, and Test sets

The distribution of the data used for train, validation, and test is illustrated below.

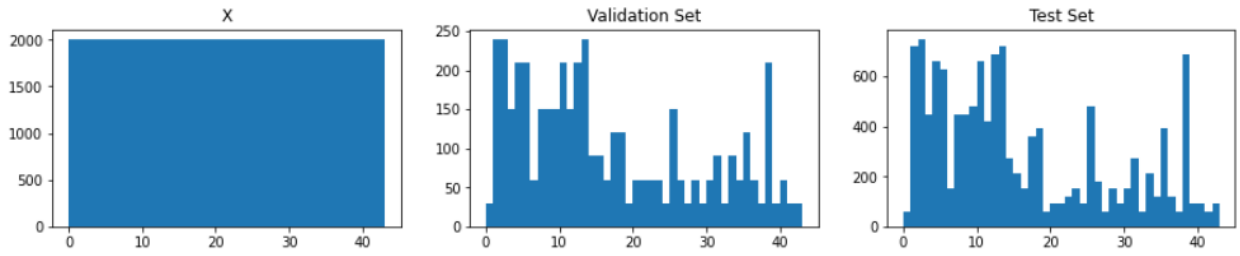

Figure 8. Distribution of X (train), validation, and test set

4. Benchmarking CNN's

A variety of Convolutional Neural Network implementations were tested. The networks ranged from shallow CNN's such as LeNet to modern deep learning networks such as ResNets, and DenseNets. Initially each network was trained three times to ensure that it can output consistent performance. However, when it came to training deeper networks due to computational complexity they were only trained once.

The deeper networks like ResNets and DenseNets were trained using weights from ImageNet as a start point and then the networks were fine-tunned.

Convolutional Neural Networks	**Train Set**			**Validation Set**			**Test Set**		
	Run: 1	2	3	1	2	3	1	2	3
LeNet	99.16	99.43	98.97	90.23	90.27	90.23	90	90	90
MiniVGG	99.83	99.90	99.84	96.28	97.28	97.28	95	97	97
VGG16	99.16	99.34	99.47	93.83	97.01	95.94	93	96	95
VGG19	98.32	96.53	97.48	97.48	96.62	94.85	95	94	94
MiniGoogLeNet	99.89	99.90	99.90	92.22	91.7	92.74	91.87	92	91.95
ResNet	98.86	-	-	82.97	-	-	81.3	-	-
ResNet50	02.75	-	-	00.68	-	-	-	-	-
ResNet50 (CR)	83.45	-	-	78.28	-	-	75	-	-
ResNet50 (TF)	82.17	-	-	76.17	-	-	74	-	-
ResNet101 (TF)	81.98	-	-	73.08	-	-	75	-	-
DenseNet121(TF)	93.94	-	-	87.48	-	-	66	-	-
DenseNet201(TF)	95.26	-	-	80.37	-	-	80	-	-

Table 2. Benchmarking results. (TF = Transfer Learning, CR = Cyclic cosine learning rate)

5. Ensemble

An ensemble of 5 models was trained for LeNet and MiniVGG Net models. Three ensemble tests were performed for each model, all with different train set configuration.

- Ensemble 1: Trained with Augment 1 generated data.
- Ensemble 2: Trained with Original (Pre-augmentation) data, unnormalized, augmenting it in the model.
- Ensemble 3: Trained with Original (Pre-augmentation) data, 255 normalized, augmenting it in the model.

The results of model averaging are presented in the table below.

	Ensemble	Model 1 (Train, Validation)	Model 2	Model 3	Model 4	Model 5	Test
	1	99.35, 90.09	99.34, 92.40	99.45, 91.13	99.21, 91.75	99.51, 92.00	93
LeNET	2	96.77, 85.28	97.37, 90.48	96.80, 91.32	96.83, 85.74	97.78, 88.71	93
	3	95.80, 7.94	97.29, 4.24	97.41, 4.04	97.85, 4.99	97.03, 8.71	05
	1	99.89, 96.33	99.80, 97.26	99.50, 97.28	99.80, 96.17	99.79, 96.58	97
MiniVGGNET	2	98.74, 96.39	99.07, 96.87	98.84, 94.13	99.30, 96.94	99.00, 94.83	97
	3	98.92, 7.55	98.75, 3.47	99.18, 9.68	98.71, 4.76	98.47, 7.39	06

Figure 9. Ensemble (Model Averaging; Train, Validation and Test accuracy)

The ensemble test performed servers two purposes. The first it confirms MiniVGG Net is the model to choose for this project because it returns the highest accuracy. And second it also confirms that augment 1 is a good choice to proceed with since it returned higher validation accuracy on most of the models used for ensemble.

The ensembles however do not seem very promising in improving the base performance of the model as both LeNet and MiniVGGNet models trained without the ensembles returned equivalent results.

Comparing classification report between both ensembled and individual model did not seem to improve overall network accuracies by any significant amount either. Even if it did so, this ensemble is very expensive to train several final models as it will likely take a very long time to train each model before averaging them.

6. Model Selection

Based on the results presented above, MiniVGG was chosen as the CNN architecture to train models for this project. The reason being that the network has shown to achieve highest test and validation accuracies, and the model may be potentially overfitting the data with consistent ~99.90% train set accuracy. Therefore, there might be scope for improvement with hyperparameter tunning.

Results of tests on Mini-VGGNet and VGG16Net were almost identical. But Mini-VGG Net consistently gave slightly higher test set and validation results. Following table shows the final network architecture of our Mini-VGG Net after hyperparameter tunning (in the next section).

Input Image (32,32,3)
Conv2D(32,32,64)
Conv2D(32,32,128)
Max Pooling(30,30,128)
Conv2D(30,30,256)
Conv2D(30,30,512)
Max Pooling(28,28,512)
Dense(128)
Dense(43)

Table 3. Mini-VGG Network Architecture used for the final model.

7. Hyperparameter Tunning

Once the Mini-VGG Net was selected for the project, hyperparameter tuning was performed for the following hyperparameters.

- Optimizers
- Weight Initialization
- Kernel Sizes
- Convolution Layer Sizes
- Loss Function
- Dense Layer Size
- Pool and stride size
- Dropout for dense layer
- Dropout for pooling layers
- Kernel Regularization
- Activity Regularization

During tuning, ensemble method was used to get multiple results for most hyperparameter inputs.

7.1) Optimizers

The Results on optimizer tests with default parameters are presented in the table below.

Optimizers	Train Set Accuracy	Validation Set Accuracy	Test Set Accuracy
Adam	99.66	97.19	96
SGD(0.01)	94.73	91.75	91
RMSprop	99.54	96.39	95
Adadelta	05.04	09.07	06
Adagrad	69.96	71.20	70
Adamax	99.32	94.60	94

Adam and RMSprop optimizers showed the best results with default parameters. Based on the above results Adam optimizer was selected for further tuning.

7.2) Weight Initialization

Following weight initializations were tested for all the layers in the network.

- Random Normal
- Random Uniform
- Truncated Normal
- Zeros
- Ones
- Glorot Normal
- Glorot Uniform
- Orthogonal
- Variance Scaling

Accuracy scores of train, validation, and test sets obtained for respective initialization methods can be seen in the table below. Based on the results of this test Random Uniform weight initialization method chosen.

Initializers	Train Set Accuracy	Validation Set Accuracy	Test Set Accuracy
Random Normal	99.8	96.26	96
Random Uniform	99.72	96.92	96
Truncated Normal	99.81	96.87	96
Zeros	2.63	1.36	1
Ones	5.8	2.2	0.02
Glorot Normal	98.76	96.26	95
Glorot Uniform	99.79	95.31	96
Orthogonal	99.90	95.85	96
Variance Scaling	99.84	96.37	95

Table 4. Weight initialization results

7.3) Activation Function

A list activation functions consisting of ReLU, Softmax, Tanh, and Sigmoid activation functions were tested on the output layer. The following Figure illustrates the train and validation set accuracy for each of these activations applied to the output layer.

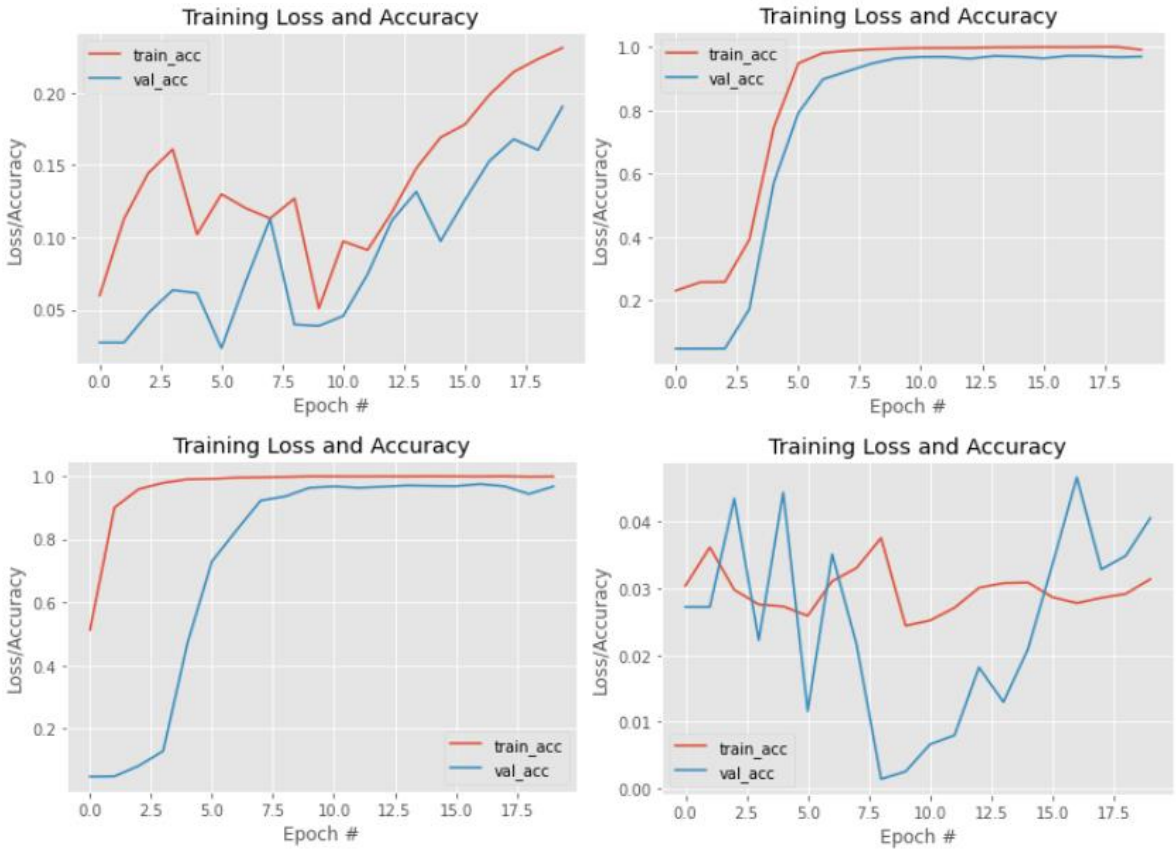

Figure 10. Different output layer activation functions (Top left: ReLU activation, bottom left: Softmax activation, top right: Sigmoid activation, bottom right: Tanh activation)

Softmax activation was chosen because it returned the highest accuracies and it also seems to begin fitting to train set much earlier.

Although, ReLU should be the best fit for all initial layers, different activation functions were also tested on all layers. The following figure illustrates the results of these tests.

Figure 11. Testing activation on all layers (Top left: ReLU, bottom left: Sigmoid, top right: Sigmoid, bottom right: Tanh)

Similar results can be seen from the second test; however it must be noted that the networks that were able to fit in this second test did so slower than when ReLU was applied to all initial layers and Softmax to the final.

Therefore, picking ReLU activation function for all initial layers and Softmax activation function for the output layer.

7.4) Kernel Sizes

Combination of kernel sizes ranging from (3,3) to (11,11) was tested on all four convolution layers in the Mini-VGG Net model. The results of kernel tuning are presented in the table below.

Kernel Sizes	Train Set Accuracy	Validation Set Accuracy	Test Set Accuracy
All (3,3)	99.90	96.53	97
All (5,5)	99.50	96.78	95
All (7,7)	99.83	96.26	95
All (9,9)	99.75	92.09	92
All (11,11)	99.63	92	91
(3, 3), (5, 5), (5, 5), (5, 5)	99.81	96.8	95
(3, 3), (5, 5), (7, 7), (11, 11)	99.92	96.69	96
(5, 5), (5, 5), (3, 3), (3, 3)	99.60	95.35	95
(3, 3), (5, 5), (5, 5), (7, 7)	99.99	97.6	96
(5, 5), (7, 7), (5, 5), (3, 3)	99.18	94.72	94
(3, 3), (5, 5), (3, 3), (3, 3)	99.92	97.78	96
(3, 3), (5, 5), (5, 5), (3, 3)	99.77	91.70	91
(5, 5), (3, 3), (3, 3), (3, 3)	99.86	95.94	96

Table 5. Kernel Size comparision

Based on the results above following kernel size combination was selected.

- Convolution layer 1: (3,3)
- Convolution layer 2: (5,5)
- Convolution layer 3: (3,3)
- Convolution layer 4: (3,3)

7.5) Convolution Layer Size

Combination of convolution layer sizes from the list: [32, 64, 128, 256, 512] was tested on all four convolution layers in the Mini-VGG Net model. The results of convolution layer size tuning are presented in the table below.

Covolution Layer Sizes	Train Set Accuracy	Validation Set Accuracy	Test Set Accuracy
All 32	99.66	97.01	96
All 64	99.86	96.51	97
All 128	100.00	98.46	97
All 256	99.99	98.14	97
32, 32, 64, 32	99.78	97.10	96
32, 32, 64, 64	99.91	97.23	97
32, 64, 128, 256	100.00	98.03	97
128, 256, 512, 512	98.73	89.61	89
64, 64, 32, 32	99.83	97.01	96
64, 128, 256, 512	100.00	98.53	97

Table 6. Convolution layer size

Layer size combination of (all 128), (32, 64, 128, 256), and (64, 128, 256, 512) are almost identical in performance. Selecting (64, 128, 256, 512) because it has the highest validation set accuracy, even though it is by a very small margin.

7.6) Dense Layer Size

Since Mini-VGG Net has one dense layer before the output layer, sizes of only that layer can vary as the output layer size must be fixed to the number of labels. Results of dense layer test are presented in the table below.

Dense Layer	Train Set Accuracy	Validation Set Accuracy	Test Set Accuracy	Max. Val. Accuracy
64	99.91	96.64	97	98.10
100	100.00	98.87	99	99.07
128	100.00	99.05	99	99.07
175	100.00	99.07	98	99.07
225	99.98	98.62	98	99.07
256	100.00	98.50	98	98.59
300	100.00	98.73	98	98.73
450	100.00	98.14	97	98.37
500	100.00	98.28	98	98.28
512	100.00	98.66	97	9.66
600	99.94	96.39	97	97.35

Table 7. Dense layer size

Based on the results of this tuning, a dense layer size of 128 was chosen. It provided one of the highest accuracies with only 4194432 number of parameters to train compared to 7373025 parameters for dense layer of size 225, and 5734575 parameters for dense layer of size 175.

7.7) Pool and stride

A variety of combination of pool size and stride size was tested. The following tables shows training results for P1, S1 and P2,S2 (max Pooling and stride length) sizes for both the pooling layers.

[P1,P2,S1,S2]	Train Set Accuracy	Validation Set Accuracy	Test Set Accuracy	Max. Val. Accuracy
[(2, 2), (2, 2), (1, 1), (1, 1)]	100.00	98.87	99	98.87
[(2, 2), (2, 2), (2, 2), (2, 2)]	100.00	98.98	98	99.18
[(2, 2), (2, 2), (3, 3), (3, 3)]	100.00	98.16	98	98.64
[(2, 2), (2, 2), (4, 4), (4, 4)]	99.09	95.51	93	97.96
[(3, 3), (3, 3), (1, 1), (1, 1)]	100.00	99.23	99	99.23
[(3, 3), (3, 3), (2, 2), (2, 2)]	100.00	99.52	99	99.52
[(3, 3), (3, 3), (3, 3), (3, 3)]	99.88	94.97	95	99.05
[(3, 3), (3, 3), (4, 4), (4, 4)]	99.95	96.98	97	97.57
[(4, 4), (4, 4), (1, 1), (1, 1)]	100.00	99.32	99	99.39
[(4, 4), (4, 4), (2, 2), (2, 2)]	100.00	99.14	99	99.27
[(4, 4), (4, 4), (3, 3), (3, 3)]	99.75	97.44	97	98.62
[(4, 4), (4, 4), (4, 4), (4, 4)]	99.96	95.98	95	98.75

Table 8. Pool and stride size

Pool size (3,3) and stride length (1,1) were selected for both pool layers because it gave the one of the highest test and validation accuracies.

7.8) Loss Function

Three loss function were tested. Cross-entropy based loss functions performed the best and the training results are presented in the table below. Categorical cross-entropy, which is also the default loss function used by keras for multiclass classification, was taken.

Loss Function	Train Set Accuracy	Validation Set Accuracy	Test Set Accuracy
Mean Squared Error	97.99	78.19	78
Categorical Crossentropy	100.00	98.53	98
Binary Crossentropy	100.00	98.37	98

Table 9. Loss Functions

7.9) Dense layer dropout

Training for dropout values while changing the dropout rate was performed for dense layer and pooling layers. This section deals with dropout layers for the dense layer only. The table below presents the training results for a range of dropout values.

Dropout value	Train Set Accuracy	Validation Set Accuracy	Test Set Accuracy	Max. Val. Accuracy
0	100.00	98.41	99	98.49
0.2	100.00	99.48	99	99.50
0.4	99.90	97.87	97	99.18
0.5	100.00	99.09	99	99.46
0.6	100.00	99.30	99	99.30
0.7	99.87	98.03	98	98.21
0.8	99.41	98.73	99	99.55
0.9	84.29	97.91	98	99.23

Table 10. Dense layer dropout

All values except for 0.4 seem to be are above 98% test set accuracy. However, this seems like a one-off instance since training with 0.4 dropout rate multiple time reveals that it can consistently output the highest performance. This result is presented in the table below.

Dropout value	Train Set Accuracy	Validation Set Accuracy	Test Set Accuracy	Max. Val. Accuracy
0.4	100.00	99.30	99	99.34
0.4	100.00	99.30	99	99.41
0.4	100.00	98.75	98	99.27

Table 11. Dense layer dropout 0.4 retrained.

7.10) Pool layer dropouts

This section deals with dropout layers for the pooling layers. The two tables below present the training results for a range of dropout values for the first (top) and the second (bottom) pooling layers, respectively.

Pool 1 dropout value	Train Set Accuracy	Validation Set Accuracy	Test Set Accuracy	Max. Val. Accuracy
0	100.00	99.02	99	99.02
0.1	100.00	99.30	99	99.37
0.25	100.00	99.21	99	99.41
0.35	100.00	98.78	99	98.78
0.5	100.00	99.34	99	99.34
0.6	100.00	99.30	99	99.32
0.75	100.00	99.23	99	99.34
0.8	100.00	98.78	99	98.80
0.9	99.88	96.58	97	98.48

Table 12. Pool 1 dropout values

Pool 2 dropout value	Train Set Accuracy	Validation Set Accuracy	Test Set Accuracy	Max. Val. Accuracy
0	100.00	99.41	99	99.46
0.1	100.00	98.89	99	99.00
0.25	100.00	99.16	99	99.23
0.35	99.76	97.37	97	98.96
0.5	100.00	98.55	99	99.34
0.6	100.00	99.23	99	99.23
0.75	100.00	99.27	99	99.30
0.8	99.68	97.44	97	99.12
0.9	99.82	98.23	98	98.44

Table 13. Pool 2 dropout size

Changing dropout values for pooling layers did not seem to affect model performance. And the model seems to perform the worst for very high dropout values. Taking both dropout values 0.25.

7.11) Kernel and Activity Regularization

Mini-VGG Net employees batch regularization through all the layers in the network, which on its own has a regularizing effect. Therefore, adding additional Ridge, LASSO, or Elastic regression parameters may not be useful. However, just to be sure a range of values for both kernel regularization and activity regularization were tested using L1 and L2 parameters.

The table below shows the results for L1 kernel regularization.

Kernel L1 Reg.	Train Set Accuracy	Validation Set Accuracy	Test Set Accuracy
0	100.00	99.14	99
0.1	66.51	58.28	53
0.01	89.35	82.52	-
0.001	96.47	96.15	92
0.0001	98.42	90.66	87
0.00001	99.45	93.40	91
0.000001	99.97	98.12	98

Table 14. L1 kernel regularization

The table below shows the results for L2 kernel regularization.

Kernel L2 Reg.	Train Set Accuracy	Validation Set Accuracy	Test Set Accuracy
0	100.00	98.89	99
0.1	89.91	75.22	67
0.01	95.81	86.28	81
0.001	98.47	94.56	91
0.0001	99.46	94.90	98
0.00001	100.00	99.00	99
0.000001	100.00	99.05	98

Table 15. L2 kernel regularization

The table below shows the results for L1 activity regularization.

Activity L1 Reg.	Train Set Accuracy	Validation Set Accuracy	Test Set Accuracy
0	100.00	99.21	99
0.1	2.59	0.68	00
0.01	2.31	2.06	02
0.001	2.86	2.04	02
0.0001	2.57	00.68	00
0.00001	10.46	3.20	04
0.000001	97.88	23.90	23

Table 16. L1 activity regularization

The table below shows the results for L2 activity regularization.

Activity L2 Reg.	Train Set Accuracy	Validation Set Accuracy	Test Set Accuracy
0	99.99	98.05	98
0.1	2.65	0.68	00
0.01	2.89	0.68	01
0.001	2.79	0.68	00
0.0001	90.14	74.54	67
0.00001	99.64	95.24	96
0.000001	99.73	97.28	97

Table 17. L2 activity regularization

8. Snapshot Ensemble

Snapshot ensemble was tested on the final model. In one version the ensemble was tested with Adam optimizer, which is the default optimizer for our model with its default (0.001) learning rate as the start point. In the second, it was tested with SGD optimizer with default (0.01) learning rate as the initial point.

The image on the left below shows the results of snapshot ensemble using Adam optimizer, and the one on the right is using the SGD optimizer.

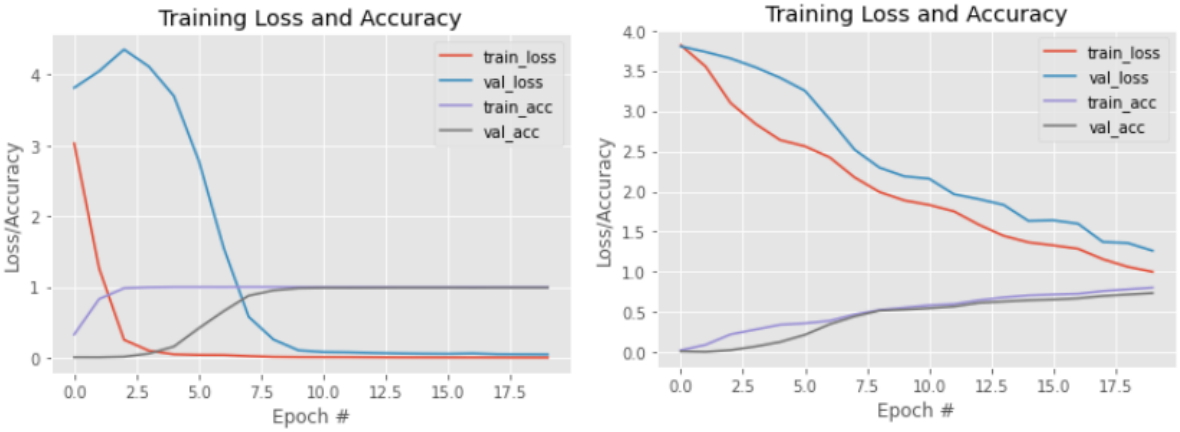

Table 18. Snapshot Ensemble (Adam with initial lr= 0.001 on the left, SGD with initial lr=0.01 on the right)

From the SGD plot above, it looks like the initial learning rate was too small. Therefore another snapshot ensemble test was run with a larger SGD initial learning rate of 0.1 and 1. The image below illustrates its output.

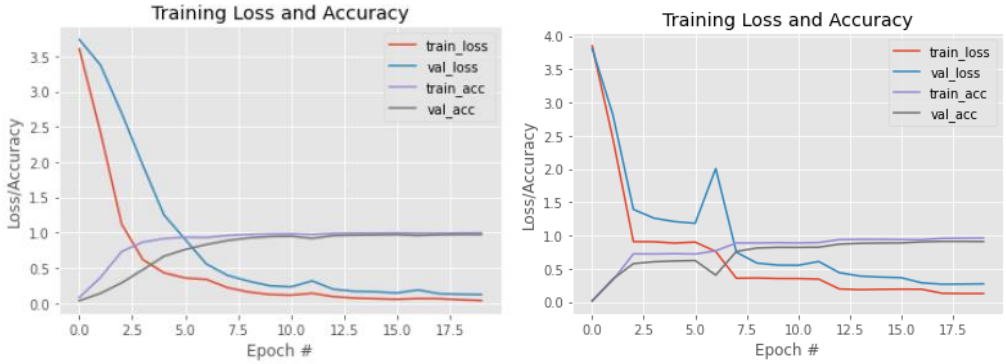

Table 19. Snapshot ensemble (SGD with initial lr=0.1(left), lr=1(right))

As is evident from the two ensembles above; even with a large initial learning rate at the beginning of each learning rate cycle, neither the cost function nor the accuracies change much unless the learning rate is exceptionally high. This possibly indicates that the model potentially has a very wide or flat local/global minima that it tends to stay in.

9. Final Model Training and Results

The final model was trained on Mini-VGG Net (illustrated in Table 3) with the hyperparameters that was found via all the testing done in the section 7 above.

Hyperparameter	Value	
Optimizer	Adam(lr=0.001)	
Weight Initialization	Random Uniform	
Activation	Initial Layers: ReLU Final Layer: SoftMax	
Loss Function	Categorical cross-entropy	
Kernel Size	Conv layer 1	(3,3)
	Conv layer 2	(5,5)
	Conv layer 3	(3,3)
	Conv layer 4	(3,3)
Convolution Layer Size	Conv layer 1	64
	Conv layer 2	128
	Conv layer 3	256
	Conv layer 4	512
Pool Size and Stride Length (Size), (Stride)	Pool layer 1	(3,3), (1,1)
	Pool Layer 2	(3,3), (1,1)
Dense Layer Sizes	First dense layer	128
	Output layer	43
L1, L2 regularization	Kernel	None
	Activation	None

Table 20. Final model hyperparameters

The final model was trained with above hyperparameters and the figures below illustrate the Train and Validation Set Accuracy (left), and Loss (right).

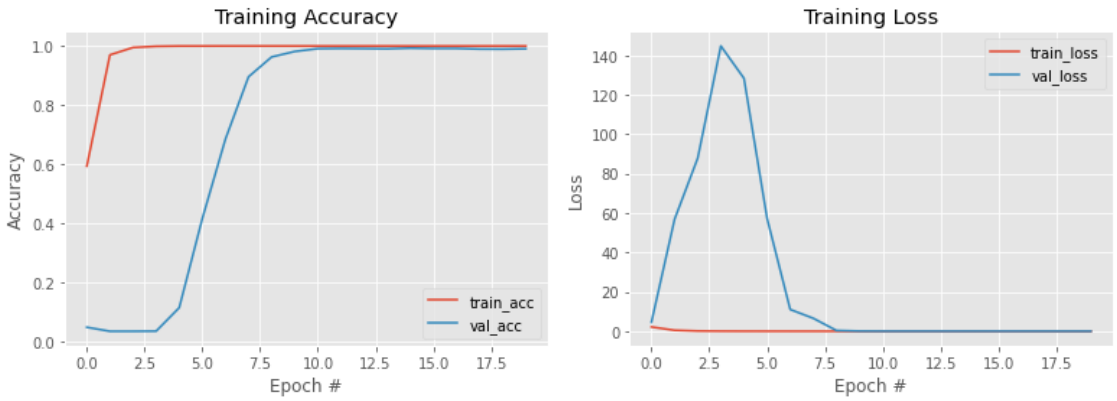

Figure 12. Final train and validation set accuracy (left), loss (right)

Final Train, Validation, and Test Set accuracies are shown in the table below.

	Train Set Accuracy	Validation Set Accuracy	Test Set Accuracy
Mini-VGG Net	100.00	99.16	99

Figure 13. Final Model Accuracy

A classification report for all the three image sets was generated and can be found under Appendix: in section 11.

10. Conclusion

The Mini-VGG Net based CNN could learn to generalize traffic very well and the objectives of this project were achieved. For many real-world applications however, real time predictions are required. Such as for use in autonomous vehicles. This means that for such specific applications fasters CNN's that can perform object detection and classification almost simultaneously are required.

Most real-time CNN's such as faster R-CNN, YOLO and its variants are more suitable for such applications. But these networks usually fall behind in prediction performance with YOLOv2 boasting only around 70-80% accuracy. CNN's such as the one trained in this project can be used to provide a feedback to real-time CNN algorithms like YOLO to improve its weights in real-time.

11. Appendix:

11.1) Classification report for Train set

Train accuracy

	precision	recall	f1-score	support
Speed limit (20km/h)	1.00	1.00	1.00	2010
Speed limit (30km/h)	1.00	0.99	1.00	2010
Speed limit (50km/h)	1.00	1.00	1.00	2010
Speed limit (60km/h)	1.00	0.99	1.00	2010
Speed limit (70km/h)	1.00	1.00	1.00	2010
Speed limit (80km/h)	0.99	1.00	0.99	2010
End of speed limit (80km/h)	1.00	1.00	1.00	2010
Speed limit (100km/h)	1.00	1.00	1.00	2010
Speed limit (120km/h)	1.00	1.00	1.00	2010
No passing	1.00	1.00	1.00	2010
No passing for vehicles over 3.5 metric tons	1.00	1.00	1.00	2010
Right-of-way at the next intersection	1.00	1.00	1.00	2010
Priority road	1.00	1.00	1.00	2010
Yield	1.00	1.00	1.00	2010
Stop	1.00	1.00	1.00	2010
No vehicles	1.00	1.00	1.00	2010
Vehicles over 3.5 metric tons prohibited	1.00	1.00	1.00	2010
No entry	1.00	1.00	1.00	2010
General caution	1.00	1.00	1.00	2010
Dangerous curve to the left	1.00	1.00	1.00	2010
Dangerous curve to the right	1.00	1.00	1.00	2010
Double curve	1.00	1.00	1.00	2010
Bumpy road	1.00	1.00	1.00	2010
Slippery road	1.00	1.00	1.00	2010
Road narrows on the right	1.00	1.00	1.00	2010
Road work	1.00	1.00	1.00	2010
Traffic signals	1.00	1.00	1.00	2010
Pedestrians	1.00	1.00	1.00	2010
Children crossing	1.00	1.00	1.00	2010
Bicycles crossing	1.00	1.00	1.00	2010
Beware of ice/snow	1.00	1.00	1.00	2010
Wild animals crossing	1.00	1.00	1.00	2010
End of all speed and passing limits	1.00	1.00	1.00	2010
Turn right ahead	1.00	1.00	1.00	2010
Turn left ahead	1.00	1.00	1.00	2010
Ahead only	1.00	1.00	1.00	2010
Go straight or right	1.00	1.00	1.00	2010
Go straight or left	1.00	1.00	1.00	2010
Keep right	1.00	1.00	1.00	2010
Keep left	1.00	1.00	1.00	2010
Roundabout mandatory	1.00	1.00	1.00	2010
End of no passing	1.00	1.00	1.00	2010
End of no passing by vehicles over 3.5 metric tons	1.00	1.00	1.00	2010
accuracy			1.00	86430
macro avg	1.00	1.00	1.00	86430
weighted avg	1.00	1.00	1.00	86430

11.2) Classification report for Validation set

Validation accuracy

	precision	recall	f1-score	support
Speed limit (20km/h)	1.00	1.00	1.00	30
Speed limit (30km/h)	1.00	0.99	1.00	240
Speed limit (50km/h)	1.00	1.00	1.00	240
Speed limit (60km/h)	1.00	0.96	0.98	150
Speed limit (70km/h)	0.99	1.00	0.99	210
Speed limit (80km/h)	0.97	0.99	0.98	210
End of speed limit (80km/h)	1.00	1.00	1.00	60
Speed limit (100km/h)	0.92	1.00	0.96	150
Speed limit (120km/h)	0.99	0.99	0.99	150
No passing	0.99	1.00	1.00	150
No passing for vehicles over 3.5 metric tons	1.00	1.00	1.00	210
Right-of-way at the next intersection	1.00	1.00	1.00	150
Priority road	1.00	1.00	1.00	210
Yield	1.00	1.00	1.00	240
Stop	1.00	1.00	1.00	90
No vehicles	1.00	1.00	1.00	90
Vehicles over 3.5 metric tons prohibited	1.00	1.00	1.00	60
No entry	1.00	1.00	1.00	120
General caution	0.98	0.99	0.99	120
Dangerous curve to the left	1.00	1.00	1.00	30
Dangerous curve to the right	1.00	0.93	0.97	60
Double curve	1.00	0.87	0.93	60
Bumpy road	1.00	1.00	1.00	60
Slippery road	0.91	0.98	0.94	60
Road narrows on the right	1.00	0.97	0.98	30
Road work	0.99	1.00	1.00	150
Traffic signals	1.00	1.00	1.00	60
Pedestrians	1.00	1.00	1.00	30
Children crossing	1.00	1.00	1.00	60
Bicycles crossing	1.00	1.00	1.00	30
Beware of ice/snow	0.97	1.00	0.98	60
Wild animals crossing	0.98	1.00	0.99	90
End of all speed and passing limits	1.00	1.00	1.00	30
Turn right ahead	1.00	1.00	1.00	90
Turn left ahead	1.00	1.00	1.00	60
Ahead only	1.00	1.00	1.00	120
Go straight or right	1.00	1.00	1.00	60
Go straight or left	1.00	1.00	1.00	30
Keep right	1.00	1.00	1.00	210
Keep left	1.00	1.00	1.00	30
Roundabout mandatory	0.98	0.78	0.87	60
End of no passing	1.00	0.97	0.98	30
End of no passing by vehicles over 3.5 metric tons	0.97	1.00	0.98	30
accuracy			0.99	4410
macro avg	0.99	0.99	0.99	4410
weighted avg	0.99	0.99	0.99	4410

11.3) Classification report for Test set

Test accuracy

	precision	recall	f1-score	support
Speed limit (20km/h)	1.00	1.00	1.00	60
Speed limit (30km/h)	1.00	0.99	1.00	720
Speed limit (50km/h)	1.00	1.00	1.00	750
Speed limit (60km/h)	1.00	0.94	0.97	450
Speed limit (70km/h)	0.99	0.99	0.99	660
Speed limit (80km/h)	0.95	1.00	0.97	630
End of speed limit (80km/h)	0.99	0.95	0.97	150
Speed limit (100km/h)	1.00	1.00	1.00	450
Speed limit (120km/h)	0.98	1.00	0.99	450
No passing	1.00	1.00	1.00	480
No passing for vehicles over 3.5 metric tons	1.00	1.00	1.00	660
Right-of-way at the next intersection	1.00	1.00	1.00	420
Priority road	1.00	0.99	0.99	690
Yield	1.00	1.00	1.00	720
Stop	1.00	1.00	1.00	270
No vehicles	0.97	1.00	0.98	210
Vehicles over 3.5 metric tons prohibited	1.00	1.00	1.00	150
No entry	1.00	1.00	1.00	360
General caution	0.99	0.97	0.98	390
Dangerous curve to the left	0.98	1.00	0.99	60
Dangerous curve to the right	0.91	1.00	0.95	90
Double curve	1.00	0.99	0.99	90
Bumpy road	0.94	0.98	0.96	120
Slippery road	0.97	1.00	0.99	150
Road narrows on the right	0.98	0.99	0.98	90
Road work	1.00	0.99	0.99	480
Traffic signals	0.96	0.96	0.96	180
Pedestrians	0.95	1.00	0.98	60
Children crossing	0.99	1.00	0.99	150
Bicycles crossing	0.98	1.00	0.99	90
Beware of ice/snow	1.00	0.91	0.95	150
Wild animals crossing	1.00	0.99	0.99	270
End of all speed and passing limits	0.97	1.00	0.98	60
Turn right ahead	1.00	1.00	1.00	210
Turn left ahead	1.00	0.99	1.00	120
Ahead only	1.00	1.00	1.00	390
Go straight or right	0.99	1.00	1.00	120
Go straight or left	1.00	1.00	1.00	60
Keep right	1.00	0.99	1.00	690
Keep left	1.00	1.00	1.00	90
Roundabout mandatory	0.95	1.00	0.97	90
End of no passing	1.00	1.00	1.00	60
End of no passing by vehicles over 3.5 metric tons	1.00	1.00	1.00	90
accuracy			0.99	12630
macro avg	0.99	0.99	0.99	12630
weighted avg	0.99	0.99	0.99	12630