Recognizing Traffic Signs Using Deep Learning

Batch-03: Project

Submitted to

Prof. Arnab Kumar Laha

by

ASHISH PANCHAL [2239973]

SANTOSH SHINDE [2239960]

TANMAY JAIN [2239954]

INDIAN INSTITUE OF MANAGEMENT - AHMEDABAD

Executive Education

May 2020

Recognizing Traffic Signs Using Deep Learning

Batch-03: Project

by

ASHISH PANCHAL [2239973]

SANTOSH SHINDE [2239960]

TANMAY JAIN [2239954]

Submitted in partial fulfillment of Executive Programme in Advanced Business Analytics (EPABA)

Under the Supervision of

Mr. Ankur Sinha, Associate Professor

Indian Institute of Management - Ahmedabad

Ahmedabad, India

Executive Programme in Advanced Business Analytics (EPABA)

Executive Education

Batch - 03

PROJECT

ABSTRACT

NAME OF THE STUDENTS : **ASHISH PANCHAL [2239973]**
epababl03.ashishp@iima.ac.in

SANTOSH SHINDE [2239960]
epababl03.santoshs@iima.ac.in

TANMAY JAIN [2239954]
epababl03.tanmayj@iima.ac.in

SUPERVISOR'S NAME : **Mr. Ankur Sinha**

PROJECT WORK TITLE : **Recognizing Traffic Signs Using Deep Learning**

ABSTRACT:

Computer Vision is an important field that recently gains popularity because of the rise of automation, be it the financial industry, Mapmaking industry, or automotive industry.

For this project work, we have focused on Computer Vision for the automotive industry's dream innovation that is autonomous cars and driver assistance systems. To have an autonomous car on the road, there are many obstacles and one of them is recognizing Traffic signs to take/adhere to local traffic rules.

Recognizing traffic signs in real-time is a high relevance computer vision problem because these signs have a wide range of variations between categories in terms of color, shape, and the presence of pictograms or text.

This project work aims to aim to research various supervised classification algorithms accuracy for the visual recognition problem of classifying traffic signs. In the experiments, Support Vector Machine (SVM), Multi-Layer Perceptron (MLP), and deep learning algorithms such as LeNet-5, VGGNet, are used and trained on German traffic sign images and then classify the unlabeled traffic signs.

For this investigation work, we have used German Traffic Signs, which is a multi-class labeled dataset.

With the current advancement in Computer Vision based on Machine Learning/Deep Learning has made it possible to recognize Traffic signs in real-time as to how humans recognize it.

Acknowledgements

We would like to express our sincere gratitude to our mentor Mr. Ankur Sinha, Associate Professor, IIM-A for his support throughout our project. His guidance and support have given us the strength to complete this journey. We also thank you for the supportive meetings and for reading many versions of the thesis, providing very useful feedback.

Finally, we would like to express thanks to all my friends who helped in different ways to accomplish this Project.

ASHISH PANCHAL	SANTISH SHINDE	TANMAY JAIN
2239973	2239960	2239954

Table of Contents

List of Figures

List of Tables

CHAPTER ONE

1.Introduction

Recognizing Traffic signs in real-time is a prevailing problem for autonomous cars and driver assistance systems. We have focused on the classification step and did not include anything on the detection.

Figure 1.1: Illustration of traffic sign classification system.

Image classification is the task of assigning input images labels from a discrete set of classes based on the appearance of the image (see Figure 1.1). As with many recognition tasks, the best performing methods are typically using deep learning. In particular, deep convolutional neural networks (CNN) have been shown to work well for image recognition tasks.

For Image classification tasks, we need training data, consisting of annotated images with the correct class (ground truth), classification algorithm's loss functions how bad/good algorithm performed to identify an image, is needed. The network is trained by updating the parameters of the network based on the gradients of the loss function, using iterative optimization algorithms, such as stochastic gradient descent.

1.1 Aim

In this project, we focused to study the relation between classification error, by training and evaluating models of different complexity.

1.2. Problem Statements

To classify Traffic Sign, which has variability in visual appearance because of the light, weather condition, or might be partial occlusions. Our goal is to understand which classification algorithm gives better/high accuracy in recognizing signs in the real world.

1.3 Methodology

We had the following steps:

1 Understand the data
2 Preprocess the data
3 Build the architecture of the model

4 Test the model
5 Iterate the same (out of scope for this project work)

1.4 Limitation

We only focused on the classification part of a full traffic sign, no detections. That is why the input to the algorithm is an image patch approximately cropped to the traffic signs or a false positive, that is, something that looks like a traffic sign, but is not.

1.5 Performance Evaluation Metrics

An important part is evaluating the trained model on a test dataset which has not been used during training in any way. The evaluations typically result in a performance metric that quantifies how well the system performs. For classification, a common performance metric is the *confusion matrix*.

The confusion matrix is defined as an $n \times n$ matrix C where n is the number of classes and element $c_{i,j}$ is the number of times the true class is class *i* and the model predicts class j. This gives a detailed description of how the system performs.

In many cases, however, a more compact metric is desired. Multiple such metrics can be computed from the confusion matrix. The *accuracy* can be computed as the sum of the diagonal elements divided by the sum of all elements:

$$A = \frac{\text{trace}(\mathbf{C})}{\text{sum}(\mathbf{C})} \tag{2.2}$$

And *error rate* is simply

$$E = 1 - A \tag{2.3}$$

The *class wise accuracy* A_i for class i can be computed as:

$$A_i = \frac{c_{i,i}}{\sum_{j=1}^{n} c_{i,j}}, \tag{2.4}$$

The average *classwise accuracy* as

$$A_c = \frac{1}{n} \sum_{i=1}^{n} A_i \tag{2.5}$$

The *average classwise error rate* as

$$E_c = 1 - A_c \tag{2.6}$$

1.6 Team Background

Team members who worked on this thesis work, are working for different organization, operates on diversified business domain.

Ashish Panchal | Engineering Manager at HERE Solution Pvt Ltd, Mumbai, is a company that designs and sells software for autonomous driving and active safety. HERE has extensive experience within the field of machine learning and computer vision.

Santosh Shinde | Manager – IT & Digital Initiative. (MIS and Analysis) | Tata Motors insurance broking and advisory services Limited. TMIBASL is a wholly owned subsidiary of Tata Motors Ltd, a leading global automobile manufacturer with a portfolio that covers a wide range of Commercial and Passenger Vehicles. TMIBASL has empaneled itself with various public and private insurance companies to offer customized solutions to customers.

Tanmay Jain | BHR-West Zone at Piramal Capital and Housing Finance Ltd, is a company which is engaged in various financial services business. It provides both wholesale and retail funding opportunities across industry sectors.

1.7. Project Source Repository

Source code found at https://github.com/apanchal/iima-epaba-batch03

CHAPTER TWO

2. Dataset

For this project work we have used The German Traffic Sign Recognition Benchmark (GTSRB)

2.1. GTSRB Dataset Overview

The GTSRB dataset contains 51839 images of German traffic signs labeled with 43 different classes. The images correspond to around 1700 different traffic sign instances (i.e. some images are of the same traffic sign instance). The sizes of the image patches vary from 15×15 to 222×193 pixels and contain a margin of 10% around the traffic signs.

This dataset comes with a predefined split into training data (contains 39209 images) and test data (contains 12630 images) into pickled format.

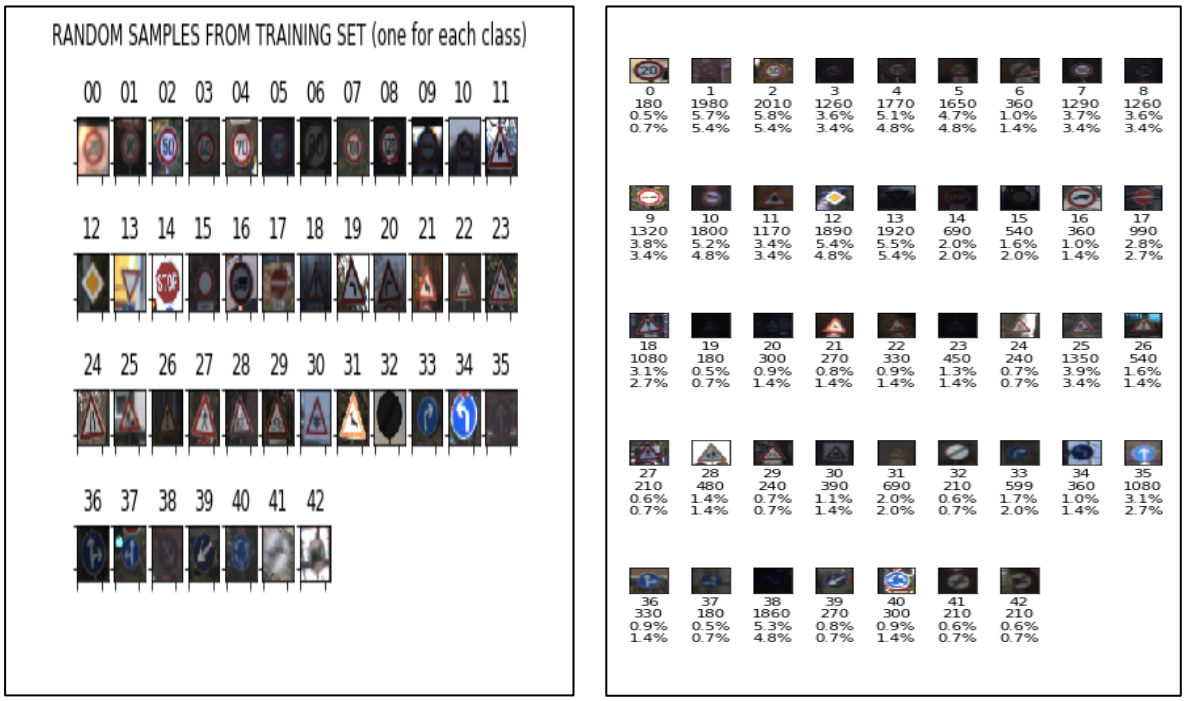
A list of all classes with labels from along with class images can be seen in Table C.1.

Class label	Class image	Class label	Class image
0		21	
1		22	
2		23	
3		24	
4		25	
5		26	
6		27	
7		28	
8		29	
9		30	
10		31	
11		32	
12		33	
13		34	
14		35	
15		36	
16		37	
17		38	
18		39	
19		40	
20		41	
21		42	

Table 1: GTSRB - Labels from class images and descriptions.

2.2 Dataset Summary & Exploration

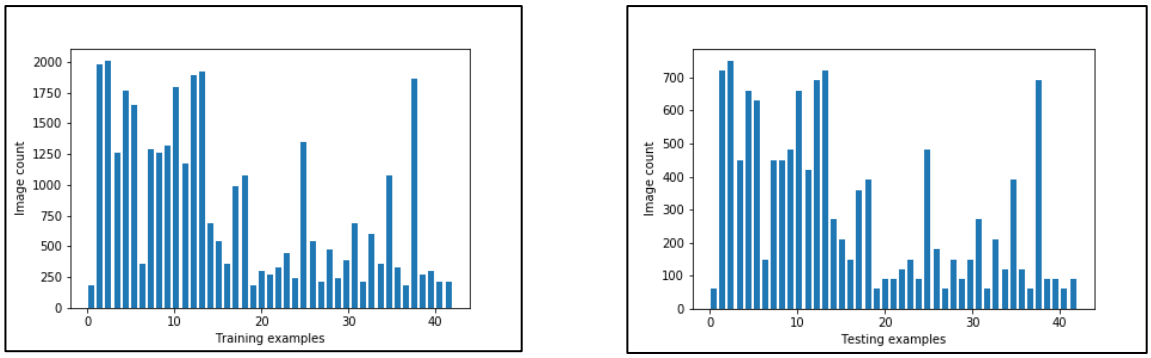
Figure 2.1 (a) depicts a set of random samples for each of the 43 classes in the training set, whereas Figure 2.2 (b) highlights more details such as the number of training samples, the percent of training samples, and the percent of validation samples for each of the 43 classes.



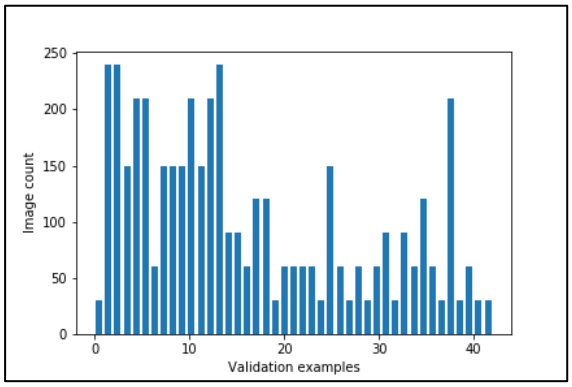
(a) *(b)*

Figure 2.1: Random samples from each of the 43 classes in the GTSRB dataset.

A histogram of the classes in the training set (a), testing set (b), and validation set (c) can be seen in Figure 2.2.



(a) *(b)*



(c)

Figure 2.2: Histogram of the count of images in each data set

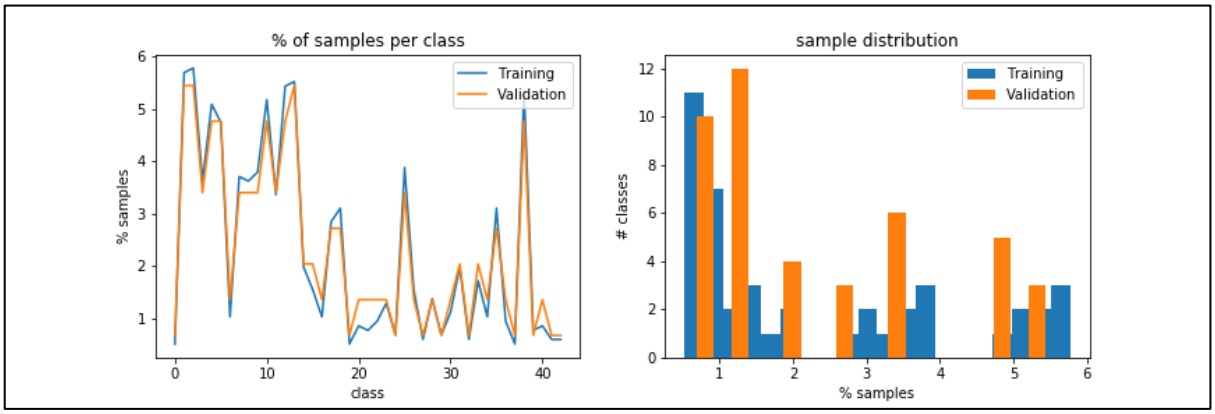


Figure 2.3: Data distribution in Training and Validation dataset

Observation from Figure 2.3:

Training Dataset

- # of Samples = 34799
- Median=1.55%
- Mean=2.33%

Validation Dataset

- # of Samples = 4410
- Median=1.36%
- Mean=2.33%

We can notice that there about the same fraction of validation samples as training samples, in the specific class (ratio closes to 1 indicate the same, refer Table 2.1 and Figure 2.4)

A high ratio means that the class has a larger representation in the validation dataset.

Ratio Stats.:

Median=1.01

Average =1.11

Class	Validation (%)	Training (%)	Ratio (%)
0	0.680272	0.517256	1.31515
1	5.44218	5.68982	0.956476
2	5.44218	5.77603	0.942201
3	3.40136	3.62079	0.939396
4	4.7619	5.08635	0.936212
5	4.7619	4.74152	1.0043
6	1.36054	1.03451	1.31515
7	3.40136	3.707	0.91755
8	3.40136	3.62079	0.939396
9	3.40136	3.79321	0.896697
10	4.7619	5.17256	0.920608
11	3.40136	3.36217	1.01166
12	4.7619	5.43119	0.87677
13	5.44218	5.5174	0.986366
14	2.04082	1.98282	1.02925

Class	Validation (%)	Training (%)	Ratio (%)
15	2.04082	1.55177	1.31515
16	1.36054	1.03451	1.31515
17	2.72109	2.84491	0.956476
18	2.72109	3.10354	0.87677
19	0.680272	0.517256	1.31515
20	1.36054	0.862094	1.57819
21	1.36054	0.775884	1.75354
22	1.36054	0.948303	1.43471
23	1.36054	1.29314	1.05212
24	0.680272	0.689675	0.986366
25	3.40136	3.87942	0.87677
26	1.36054	1.55177	0.87677
27	0.680272	0.603466	1.12728
28	1.36054	1.37935	0.986366

Class	Validation (%)	Training (%)	Ratio (%)
29	0.680272	0.689675	0.986366
30	1.36054	1.12072	1.21399
31	2.04082	1.98282	1.02925
32	0.680272	0.603466	1.12728
33	2.04082	1.72131	1.18562
34	1.36054	1.03451	1.31515
35	2.72109	3.10354	0.87677
36	1.36054	0.948303	1.43471
37	0.680272	0.517256	1.31515
38	4.7619	5.34498	0.890911
39	0.680272	0.775884	0.87677
40	1.36054	0.862094	1.57819
41	0.680272	0.603466	1.12728
42	0.680272	0.603466	1.12728

Table 2.1: Representation stats for validation and training datasets per-class

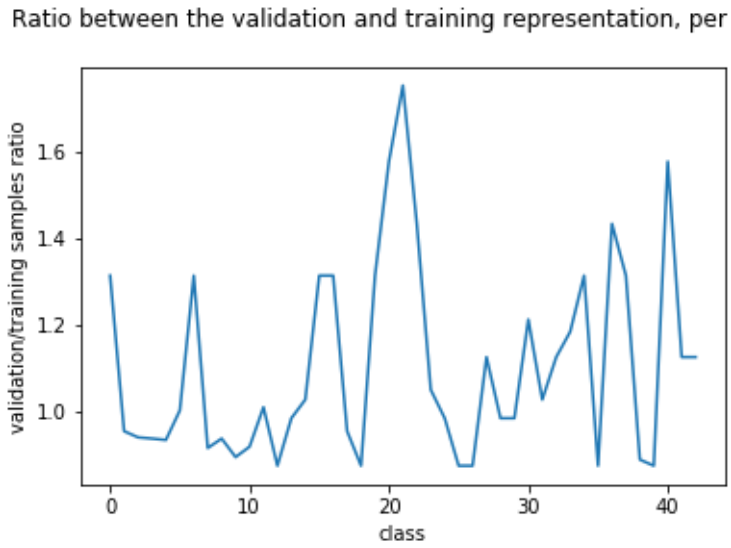


Figure 2.4: Ratio between the validation and training representation, per class

From data exploration we could make the following inferences, which we need to handle during the preprocessing stage:

1. There is a class bias issue as some classes seem to be underrepresented
2. Low image contrast for a lot of images

2.3. Source Code

https://github.com/apanchal/iima-epaba-batch03/blob/master/notebooks/0000_001_1.ipynb

CHAPTER THREE

3. Data Preprocessing

"Do we need data pre-processing?" We have taken an approach where we build simple Dense NN which helps us to answer the above question by establishing a score for a model, which could be improved upon each iteration. We have used our model "accuracy" score.

Model Testing without any preprocessing - Establishing Baseline

Neural Network Architecture

```
model = Sequential()
model.add(Dense(128, activation='relu', input_shape=(32*32*3,)))
model.add(BatchNormalization())
model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.5))
model.add(Dense(128, activation='relu'))
model.add(BatchNormalization())
model.add(Dense(n_classes, activation='softmax'))
```

Figure 3.1: Sample Baseline Dense Network Architecture

As our images are colored, our input shape is 32*32*3. Sample Dense network's first layer would have 393344 ((32*32*3*128) + 128 parameters

We have "ReLU" as an activation function.

Our sample Dense network has 4 hidden layers of 128 neurons with ReLU activation and after each hidden layer, we have the last one a dropout (50%) function is included.

We have softmax activation function, as have a multi-class classification and there are 43 classes.

The Sample baseline model was able to achieve an accuracy ***score of 86%*** without any preprocessing.

```
Pred = model.evaluate(X_test_baseline, y_test_baseline, verbose=0)
print("Dense fully connected network results on the test data - Baseline ")
print(" ")
print("%s- %.2f" % (model.metrics_names[0], Pred[0]))
print("%s- %.2f" % (model.metrics_names[1], Pred[1]))
```

```
Dense fully connected network results on the test data - Baseline

loss- 1.05
accuracy- 0.86
```

Figure 3.1.1: Sample baseline model accuracy without any preprocessing or data augmentation

3.1 Preprocessing Techniques

We have applied several preprocessing steps to the input images to achieve the best possible results. we have used the following preprocessing techniques:

- Data Augmentation
  - Slight Rotation of Images
  - Image Translation
- Shuffling
- Bilateral Filtering
- Grayscaling
- Local Histogram Equalization
- Normalization

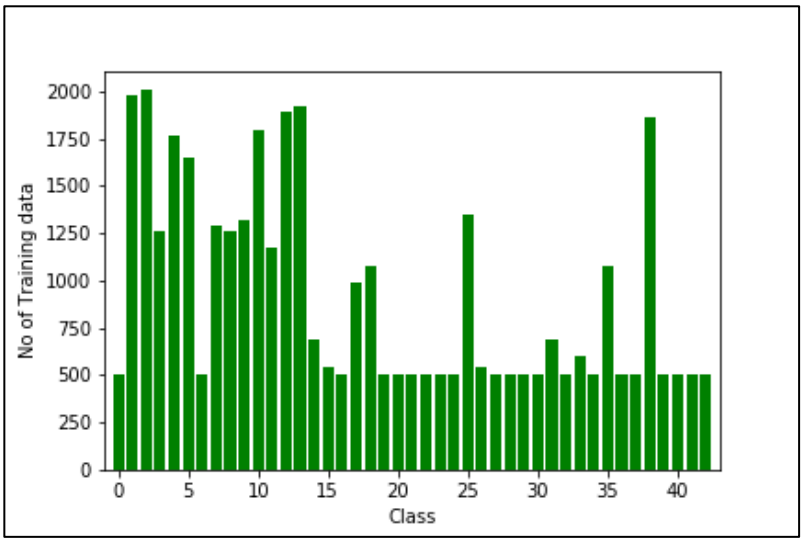
3.2 Data Augmentation

Data Augmentation is used to increase the training set data. Augmenting means generating more data(images) from the available images by adding slight alteration of the available images.

We have increased the training set images, to address the class bias issue to some extent we could have make all classes images number same, but it would be led to very high computational resources.

Hence during augmentation, all the classes were less than 500 images in the original. The number 500 is an arbitrary number we took to make all classes have the same number of records.

Check class bias after training data augmentation.



(a)

```
print('Number of training examples and class labels, before data augmentation:',len(X_train),len(y_train))
print('Number of training examples and class labels, after data augmentation:',len(X_train_final),len(y_train_final))
```

```
Number of training examples and class labels, before data augmentation: 34799 34799
Number of training examples and class labels, after data augmentation: 39239 39239
```

(b)

Figure 3.1.2: Training data after augmentation

We have used OpenCV open-source techniques such as Rotation, Translation, Bilateral filtering, Grayscaling, and Local Histogram Equalization.

3.2.1 Slight Rotation of Images

We used 10 degrees of rotation of images. We could not rotate images more because of the fact that it might lead to wrong representations of the traffic signs. Figure 3.2 depicts few images after slight rotation (not that noticeable in a few images also).

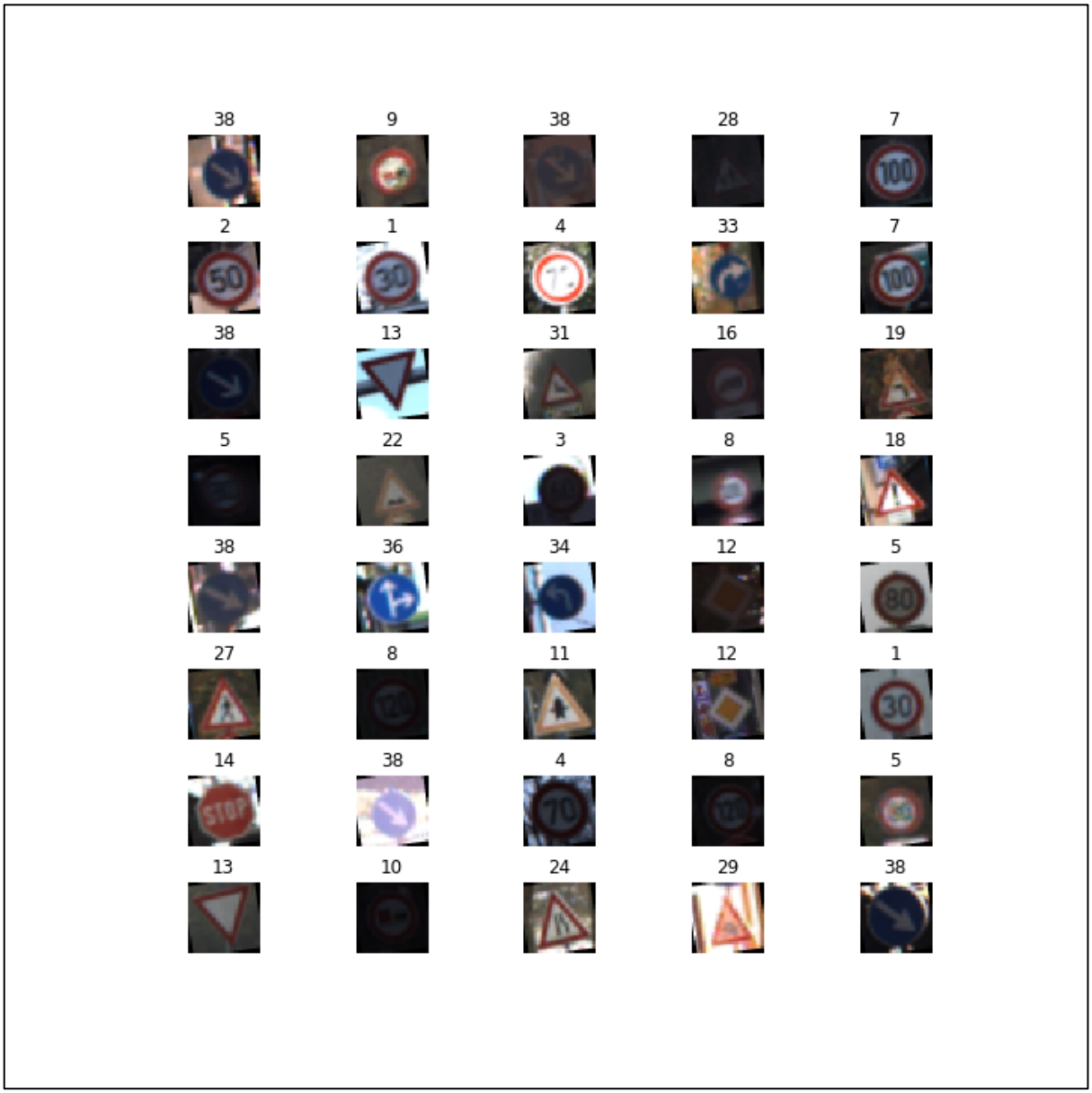


Figure 3.2: Images after 10-degree rotation

3.2.2 Image Translation

Image Translation is nothing but shifting the location of the image. In simple words, if the image's location is (x1,y1) position, after translation it is moved to (x2,y2) position. Figure 3.3 depicts that the location is slightly moved downwards.

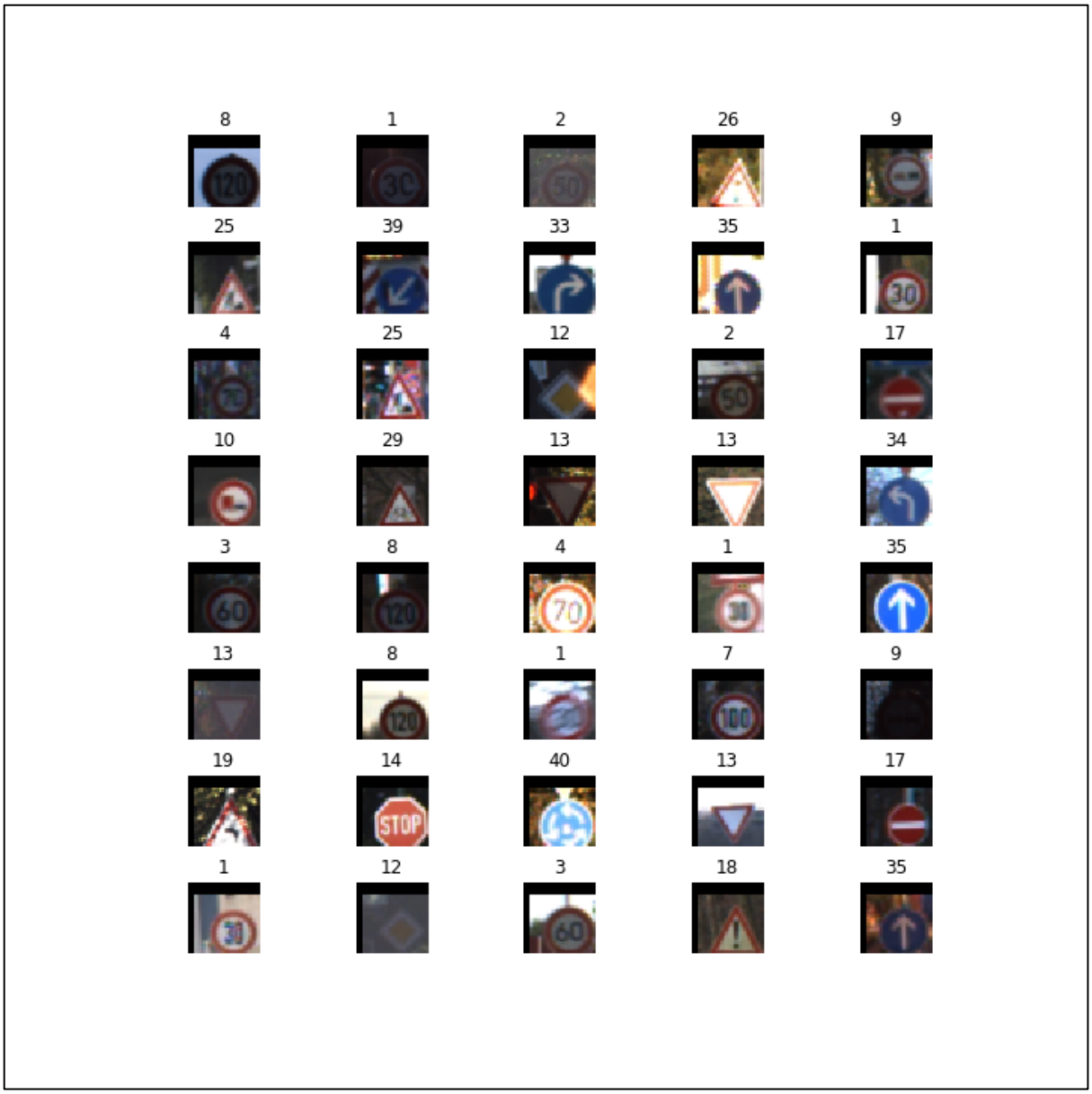


Figure 3.3: Images after translation

3.3 Shuffling

In general, we shuffle the training data to increase randomness and variety in the training dataset, in order for the model to be more stable.

3.4 Bilateral Filtering

Bilateral filtering is a noise-reducing, edge-preserving smoothening of images.

3.5 Grayscaling

Gray scaling of images helps to reduce the weight provided to pixels information and reduces complexity. We have used OpenCV to convert the training images into greyscale.

Figure 3.4: Images after Grayscaling

3.6 Local Histogram Equalization

It helps to increase the contrast of the images as we had identified during "Data exploration" that the images might need an increase in contrast.

Figure 3.5: Images after Local histogram equalization

3.7 Normalization

It helps to change the range of pixel intensity values, which helps to achieve zero mean and equal variance for image data.

Figure 3.6: Images after Normalization

The same sample baseline dense neural network architecture as one used above was able to better its accuracy score to **87%** after data preprocessing, which suggests to us that preprocessing of the images (Augmenting the data) was worth the effort.

```
Pred = model.evaluate(X_test_aug, y_test_aug, verbose=0)
print("Dense fully connected network results on the test data - After Data Augmentation ")
print(" ")
print("%s- %.2f" % (model.metrics_names[0], Pred[0]))
print("%s- %.2f" % (model.metrics_names[1], Pred[1]))
```

```
Dense fully connected network results on the test data - After Data Augmentation

loss- 1.20
accuracy- 0.87
```

Figure 3.7: Sample baseline model accuracy with data augmentation

For the project, we have used augmented training data to train all classifiers.

3.9 Source Code

https://github.com/apanchal/iima-epaba-batch03/blob/master/notebooks/0000_001_2.ipynb

CHAPTER FOUR

4. Classifier: Sklearn's Support Vector Machine

As part of project goal/study, we have picked up supervised learning algorithms from basic to deep learning to classify GTSRB signs. We have applied Support Vector Machine (SVM), Multi-Layer Perceptron (MLP) Chapter -5, LeNet-5 Chapter - 6.1, and VGGNet, Chapter – 6.2.

For SVM and MLP we have used scikit-learn library, for Lenet-5 and VGGNet we have used TensorFlow and Keras.

To optimize SVM, there are several parameters need to be tuned. Three major parameters we considered are as below:

1. Kernels: The main function of the kernel is to take low dimensional input space and transform it into a higher-dimensional space. We have Gaussian (rbf) kernel better for our problem.

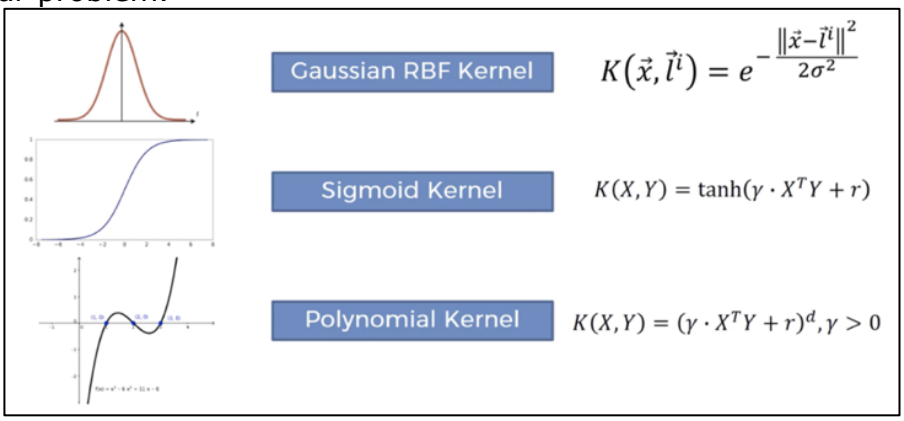


Figure 4.0: SVM Hyper-Parameters – Kernels

2. C (Regularization): C is the penalty parameter, which represents misclassification or error term. This is to instruct SVM optimization how much error is acceptable. This is to control the trade-off between decision boundary and misclassification term. A smaller value of C creates a small-margin hyperplane and a larger value of C creates a larger-margin hyperplane.

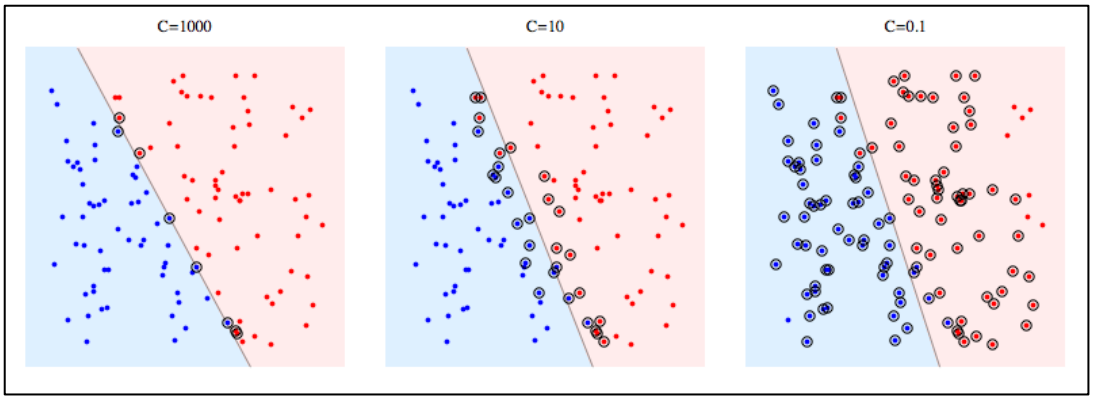


Figure 4.1: SVM Hyper-Parameters – Regularization (C)

3. Gamma

Low gamma value considers only nearby points while calculating the separation line, whereas High gamma value considers all the data points in the calculation of the separation line.

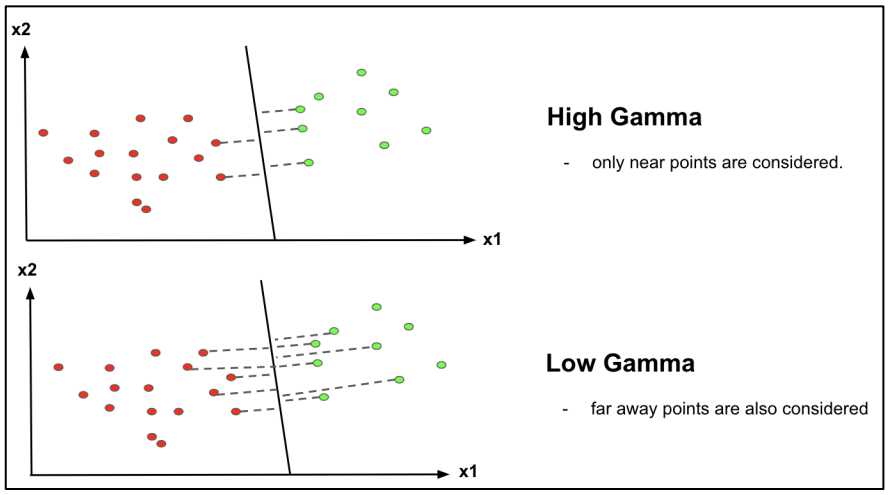


Figure 4.2: SVM Hyper-Parameters – Gamma

when gamma is higher, nearby points will have high influence; low gamma means far away points also be considered to get the decision boundary.

4.1. Hyper-parameters of SVM

In scikit-learn, they are passed as arguments to the constructor of the estimator classes. Grid search is commonly used as an approach to hyper-parameter tuning that will methodically build and evaluate a model for each combination of algorithm parameters specified in a grid.

We have use Gaussian (rbf) kernel.

4.1.1. Choosing Right values of Gamma for Gaussian (rbf) Kernel

To investigate on right value of Gamma for 'rbf' kernel for our dataset, we used Validation curve from scikit-learn library. It determines training and test scores for varying parameter values. Compute scores for an estimator with different values of a specified parameter. This is similar to grid search with one parameter.

We have used 5-fold cross validation techniques and Gamma Range ['0.0001', '0.0046415888', '0.215443469', '10.'] for finding validation accuracy.

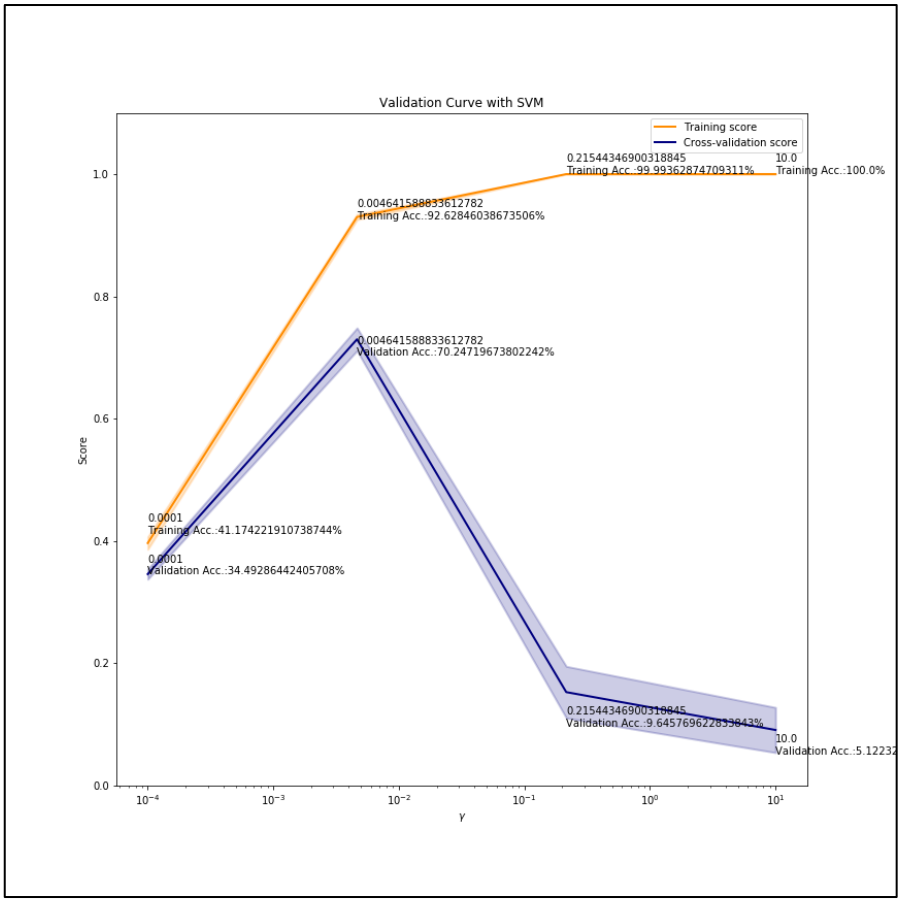


Figure 4.3: SVM Validation Curve of Gamma for Gaussian (rbf) Kernel

Above plot shows that, training scores and validation scores of an SVM for different values of the 'rbf' kernel parameter gamma (ϒ) we can conclude following:

1. For very low values of gamma (ϒ), both the training score and the validation score are low. This is called *underfitting*.
2. Medium values of gamma (ϒ) will result in high values for both scores, i.e. the classifier is performing fairly well.
3. If gamma (ϒ) is too high, the classifier will *overfit*, which means that the training score is good, but the validation score is poor.

For our GTSRB dataset, **we have gamma (ϒ) as 0.0046415888**, because this point training accuracy is 92.62 % and validation accuracy is 70.24 %.

4.1.2 Choosing Right values of Regularization for Gaussian (rbf) Kernel

To investigate on right value of Regularization for 'rbf' kernel for our dataset, we have used Cross-validation from scikit-learn library.

We have used 5-fold cross validation techniques and Regularization Range [1, 10, 100] for finding validation accuracy.

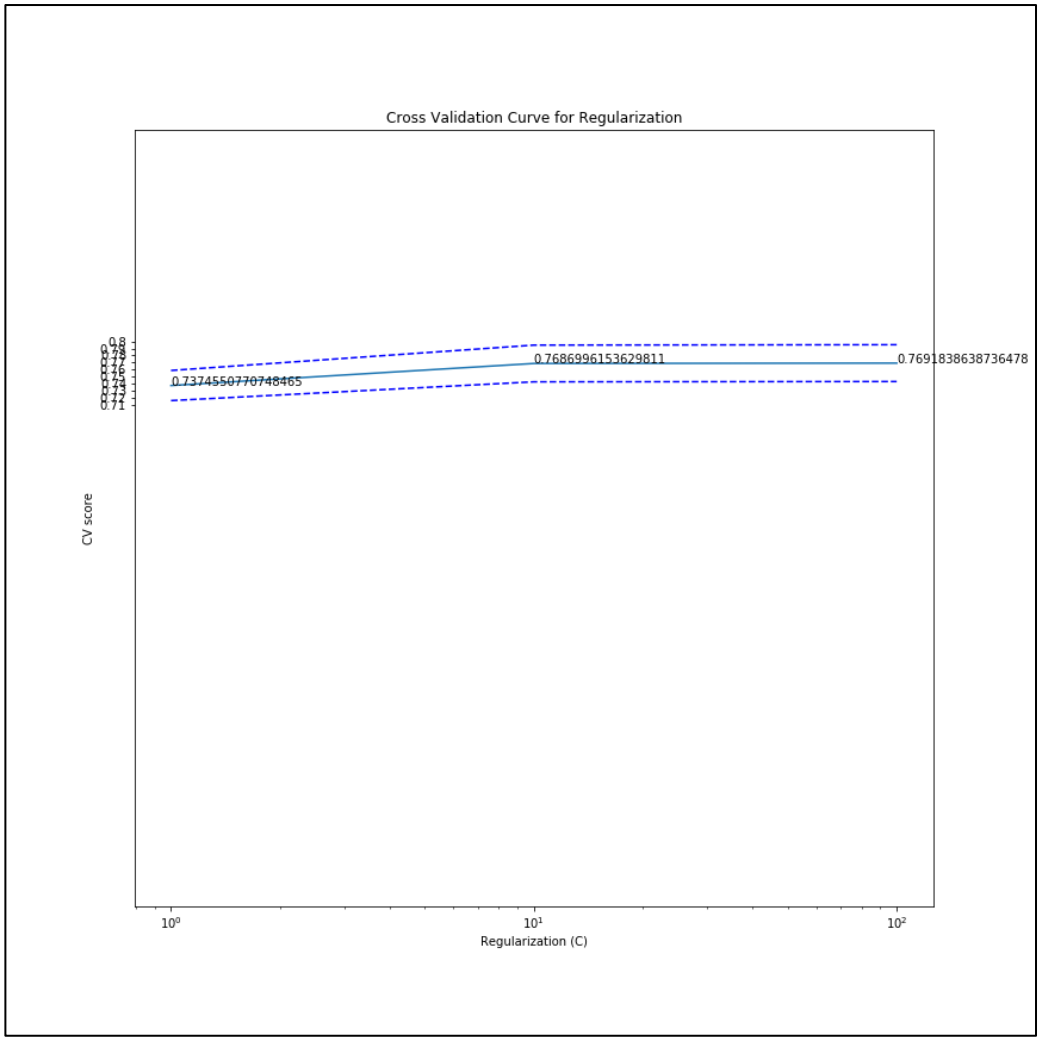


Figure 4.4: SVM Validation Curve of Regularization for Gaussian (rbf) Kernel

4.3 Testing SVM using Test dataset

```
X_test_pp = X_test_preprocessed.reshape((len(X_test_preprocessed)),-1)
y_pred = best_svm_classifier.predict(X_test_pp)
acc = accuracy_score(y_test,y_pred)
print("Test accuracy:",acc)
```

```
Test accuracy: 0.8197149643705464
```

Figure 4.6: SVM Test Accuracy

4.4 Confusion Metric

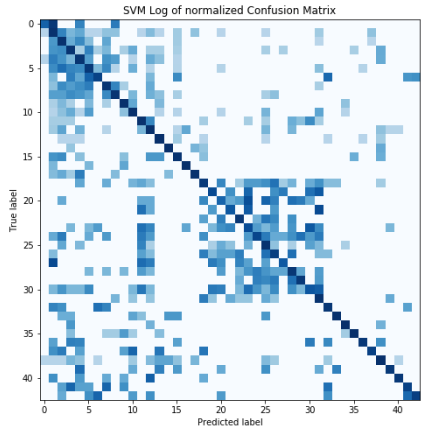


Figure: 4.7: SVM - Log of Normalized Confusion Matrix

	precision	recall	f1-score	support
0	0.77	0.17	0.27	60
1	0.76	0.89	0.82	720
2	0.75	0.90	0.82	750
3	0.75	0.83	0.79	450
4	0.77	0.80	0.78	660
5	0.77	0.69	0.73	630
6	0.72	0.51	0.60	150
7	0.77	0.78	0.77	450
8	0.88	0.82	0.85	450
9	0.96	0.96	0.96	480
10	0.93	0.93	0.93	660
11	0.80	0.89	0.84	420
12	0.87	0.93	0.90	690
13	0.95	0.98	0.97	720
14	0.94	0.97	0.96	270
15	0.86	0.85	0.85	210
16	0.93	0.97	0.95	150
17	0.97	0.91	0.94	360
18	0.85	0.67	0.75	390
19	0.52	0.38	0.44	60
20	0.19	0.16	0.17	90
21	0.55	0.24	0.34	90
22	0.81	0.76	0.78	120
23	0.36	0.36	0.36	150
24	0.57	0.43	0.49	90
25	0.80	0.87	0.84	480
26	0.59	0.59	0.59	180
27	0.61	0.23	0.34	60
28	0.63	0.75	0.69	150
29	0.76	0.70	0.73	90
30	0.33	0.26	0.29	150
31	0.65	0.87	0.74	270
32	0.53	0.75	0.62	60
33	0.96	0.89	0.93	210
34	0.96	0.96	0.96	120
35	0.99	0.96	0.98	390
36	0.98	0.72	0.83	120
37	0.89	0.55	0.68	60
38	0.96	0.92	0.94	690
39	0.93	0.69	0.79	90
40	0.97	0.73	0.84	90
41	0.65	0.52	0.57	60
42	0.91	0.56	0.69	90
accuracy			0.82	12630
macro avg	0.77	0.71	0.72	12630
weighted avg	0.82	0.82	0.81	12630

Figure 4.8: SVM - Class wise Classification report

4.5 Testing SVM Model on New Images.

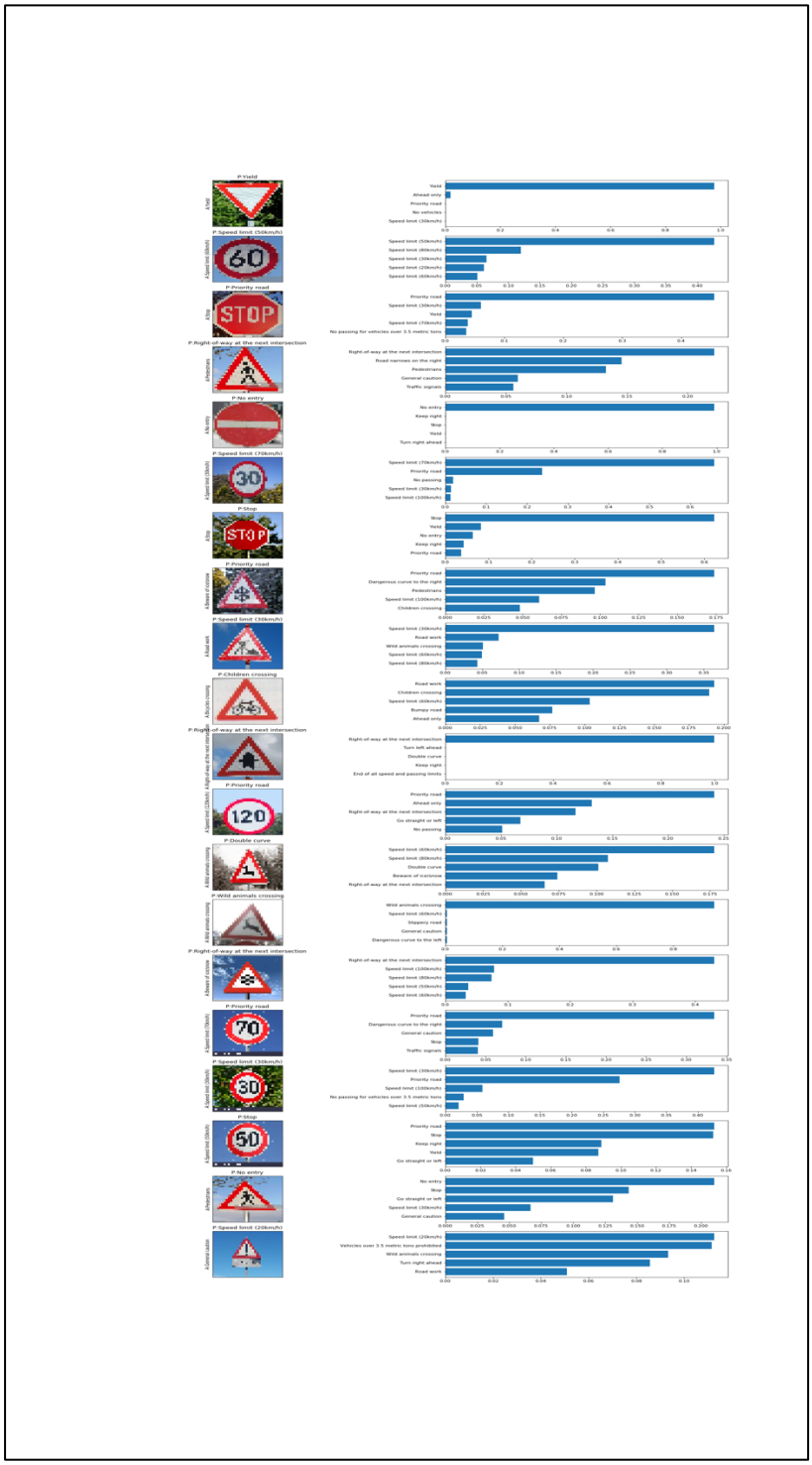


Figure 4.9: SVM - Top 5 Prediction with Predicted Probability

4.6 Source Code

https://github.com/apanchal/iima-epaba-batch03/blob/master/notebooks/0000_002_1.ipynb

CHAPTER FIVE

5. Classifier: Sklearn's MLPClassifier Neural Net

Multi-layer Perceptron (MLP) classifier which in the name itself connects to a Neural Network. Unlike other classification algorithms such as Support Vectors or Naive Bayes Classifier, MLP Classifier relies on an underlying Neural Network to perform the task of classification.

Each training point (a 32x32 image) has 1024 features, MLPClassifier is smart enough to figure out how many output units need based on the dimension of the y's you feed it. We also need to specify the "activation" function that all these neurons will use - this means the transformation a neuron will apply to its weighted input.

5.1 Hyper-parameters of MLP

It is an important step for improving algorithm performance. We have used ***Random Search Parameter*** Tuning; Random search is an approach to parameter tuning that will sample algorithm parameters from a random distribution (i.e. uniform) for a fixed number of iterations.

We have tried RandomizedSearchCV to find optimized set of hyper-parameters. we tried following combination:

- `hidden_layer_sizes':(1024,) Test Accuracy: 82.9% New Test Images Accuracy: 30%`
- `hidden_layer_sizes':(32,64,43) Test Accuracy: 80% New Test Images Accuracy: 20%`
- `hidden_layer_sizes':(100,64,43) Test Accuracy: 81% New Test Images Accuracy: 25%`
- `hidden_layer_sizes':(1024,1024,43) Test Accuracy: 83% New Test Images Accuracy: 20%`
- `hidden_layer_sizes':(1024,750,43) Test Accuracy: 83% New Test Images Accuracy: 20%`

As we can see that, #1 and #5 has almost same test accuracy but have significant different performance/accuracy on new test images. So we have selected #1.

```
Best Parameter found is: {'solver': 'sgd', 'max_iter': 1000,
'hidden_layer_sizes': (1024, 750, 43), 'alpha': 0.001}
```

Figure 5.0: MLP Hyper-Parameters - RandonizedSearchCV

5.2 MLP Loss Curve

We have used above hyper-parameters along different learning rate schedules and momentum parameters.

```
# different learning rate schedules and momentum parameters
params = [{'solver': 'sgd', 'learning_rate': 'constant', 'momentum': 0,
           'learning_rate_init': 0.001, 'hidden_layer_sizes': (1024), 'alpha':0.001},{'solver': 'sgd', 'learning_rate': 'constant', 'momentum': .9,
           'nesterovs_momentum': False, 'learning_rate_init': 0.2, 'hidden_layer_sizes': (1024), 'alpha':0.001},
          {'solver': 'sgd', 'learning_rate': 'constant', 'momentum': .9,
           'nesterovs_momentum': True, 'learning_rate_init': 0.2, 'hidden_layer_sizes': (1024), 'alpha':0.001},
          {'solver': 'sgd', 'learning_rate': 'invscaling', 'momentum': 0,
           'learning_rate_init': 0.2, 'hidden_layer_sizes': (1024), 'alpha':0.001},
          {'solver': 'sgd', 'learning_rate': 'invscaling', 'momentum': .9,
           'nesterovs_momentum': True, 'learning_rate_init': 0.2, 'hidden_layer_sizes': (1024), 'alpha':0.001},
          {'solver': 'sgd', 'learning_rate': 'invscaling', 'momentum': .9,
           'nesterovs_momentum': False, 'learning_rate_init': 0.2, 'hidden_layer_sizes': (1024), 'alpha':0.001},
          {'solver': 'adam', 'learning_rate_init': 0.01, 'hidden_layer_sizes': (1024), 'alpha':0.001}
          ]
# labels for each collection of parameters
labels = ["constant learning-rate", "constant with momentum",
          "constant with Nesterov's momentum",
          "inv-scaling learning-rate", "inv-scaling with momentum",
          "inv-scaling with Nesterov's momentum", "adam"]
# plotting arguments
plot_args = [{'c': 'red', 'linestyle': '-'},
             {'c': 'green', 'linestyle': '-'},
             {'c': 'blue', 'linestyle': '-'},
             {'c': 'red', 'linestyle': '--'},
             {'c': 'green', 'linestyle': '--'},
             {'c': 'blue', 'linestyle': '--'},
             {'c': 'black', 'linestyle': '-'}]

# plot the results
fig, axes = plt.subplots(1, 1, figsize=(15, 10))
mlps = []

# loop through each set of parameters
for label, param, args in zip(labels, params, plot_args):
    print("training: %s" % label)
    mlp = MLPClassifier(verbose=4, random_state=0, max_iter = 1000, **param)

    # some parameter combinations will not converge as can be seen on the
        # plots so they are ignored here
    with warnings.catch_warnings():
        warnings.filterwarnings("ignore", category=ConvergenceWarning,
                                module="sklearn")

    mlp.fit(X_train, y_train_final)
    mlps.append(mlp)
    print("Training set score: %f" % mlp.score(X_train, y_train_final))
    print("Training set loss: %f" % mlp.loss_)
    print('')
    axes.plot(mlp.loss_curve_, **args)

# show the plot
fig.legend(axes.get_lines(), labels=labels, ncol=3, loc="upper center")
plt.xlabel("number of steps")
plt.ylabel("loss function")
plt.grid(True)
plt.savefig('../plots/classifier/mlp/mlp_loss_curve_v2_1000.png')
plt.show()
```

Figure 5.1: MLP - Comparing Different Learning Strategies for the Network

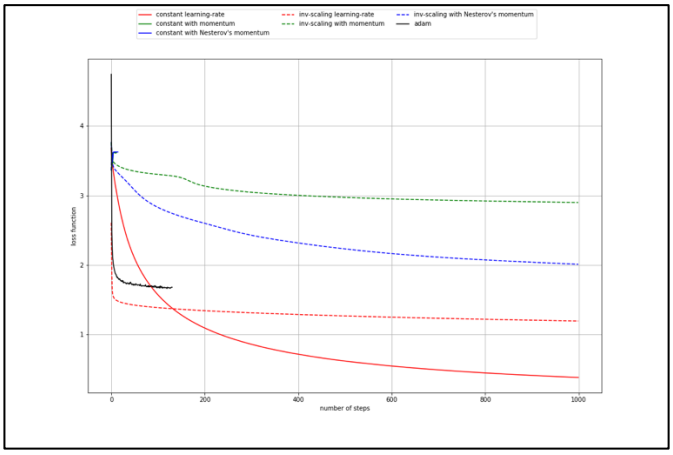


Figure 5.2: MLP – Loss Curve with Different Learning Rate

Constant Learning Rate has good learning rate.

5.3 Testing MLP using Test dataset

```
[102]: #best_mlp_classifier = random_search_clf.best_estimator_
       best_mlp_classifier = mlps[0]
       print(best_mlp_classifier)
       X_test_pp = X_test_preprocessed.reshape((len(X_test_preprocessed)),-1)
       y_pred = best_mlp_classifier.predict(X_test_pp)
       acc = accuracy_score(y_test,y_pred)
       print("Test accuracy:",acc)

       MLPClassifier(activation='relu', alpha=0.001, batch_size='auto', beta_1=0.9,
                     beta_2=0.999, early_stopping=False, epsilon=1e-08,
                     hidden_layer_sizes=1024, learning_rate='constant',
                     learning_rate_init=0.001, max_fun=15000, max_iter=1000,
                     momentum=0, n_iter_no_change=10, nesterovs_momentum=True,
                     power_t=0.5, random_state=0, shuffle=True, solver='sgd',
                     tol=0.0001, validation_fraction=0.1, verbose=4, warm_start=False)
       Test accuracy: 0.8135391923990499
```

Figure 5.3: MLP – Test Accuracy

5.4 Confusion Metric

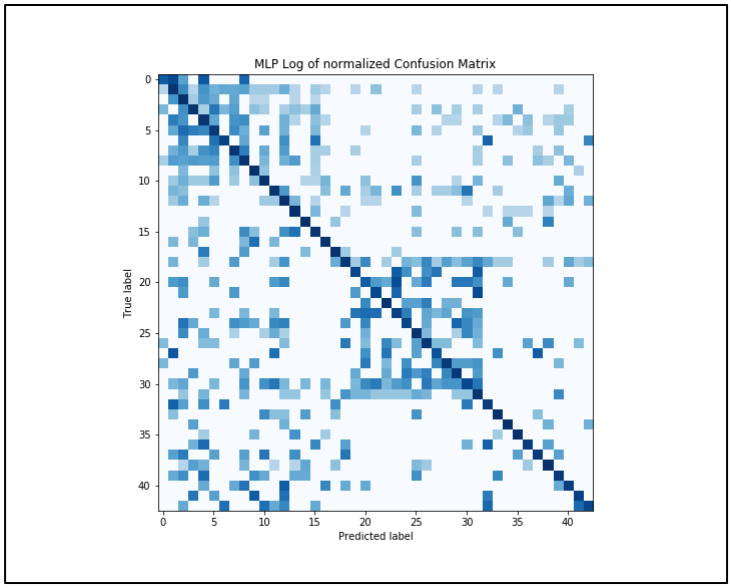


Figure 5.4: MLP - Log of Normalized Confusion Matrix

	precision	recall	f1-score	support
0	0.63	0.20	0.30	60
1	0.80	0.85	0.82	720
2	0.75	0.92	0.83	750
3	0.82	0.80	0.81	450
4	0.74	0.79	0.77	660
5	0.77	0.66	0.71	630
6	0.75	0.51	0.61	150
7	0.77	0.78	0.78	450
8	0.75	0.81	0.78	450
9	0.91	0.98	0.94	480
10	0.91	0.89	0.90	660
11	0.90	0.80	0.84	420
12	0.85	0.92	0.89	690
13	0.93	0.98	0.95	720
14	0.93	0.93	0.93	270
15	0.78	0.80	0.79	210
16	0.88	0.87	0.88	150
17	0.97	0.91	0.94	360
18	0.88	0.69	0.78	390
19	0.55	0.45	0.50	60
20	0.19	0.20	0.20	90
21	0.50	0.41	0.45	90
22	0.84	0.85	0.85	120
23	0.51	0.52	0.51	150
24	0.55	0.46	0.50	90
25	0.88	0.86	0.87	480
26	0.70	0.76	0.73	180
27	0.45	0.28	0.35	60
28	0.84	0.73	0.78	150
29	0.63	0.73	0.68	90
30	0.41	0.42	0.42	150
31	0.69	0.85	0.76	270
32	0.36	0.62	0.46	60
33	0.94	0.89	0.92	210
34	0.91	0.96	0.93	120
35	0.98	0.96	0.97	390
36	0.87	0.62	0.72	120
37	0.71	0.65	0.68	60
38	0.94	0.93	0.94	690
39	0.68	0.70	0.69	90
40	0.69	0.61	0.65	90
41	0.68	0.53	0.60	60
42	0.72	0.46	0.56	90
accuracy			0.81	12630
macro avg	0.74	0.71	0.72	12630
weighted avg	0.82	0.81	0.81	12630

Figure 5.5: MLP - Class wise Classification report

5.5 Testing MLP Model on New Images.

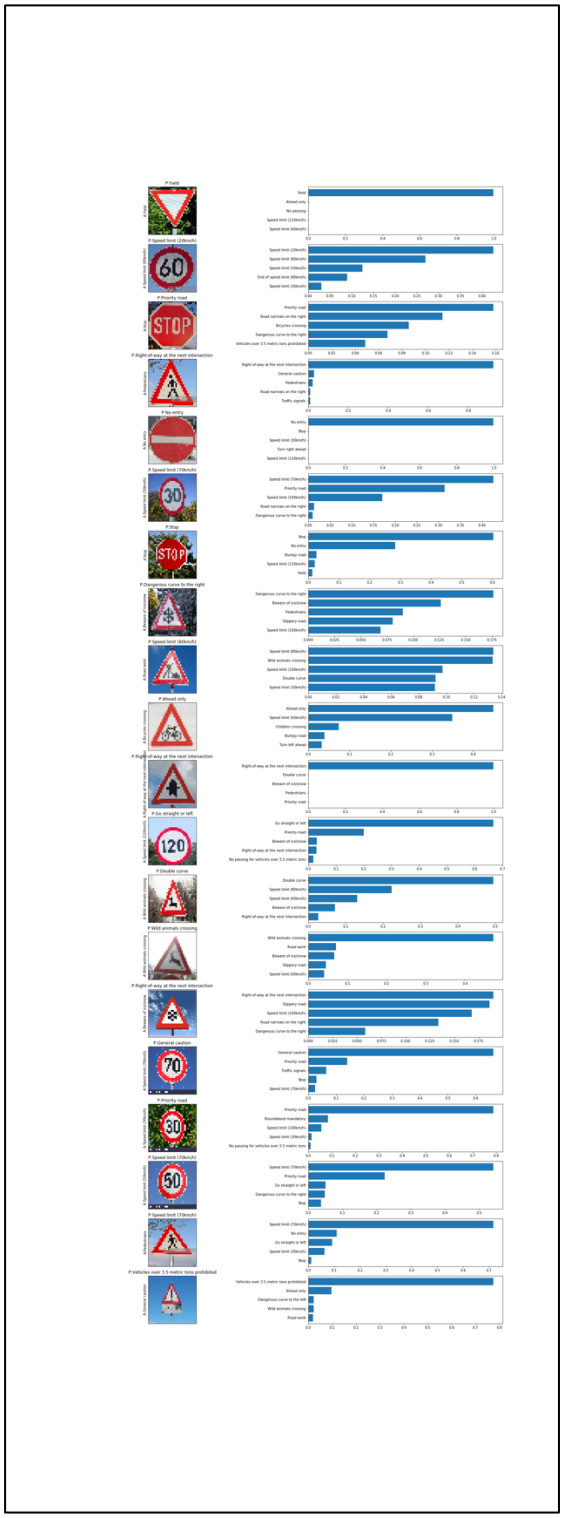


Figure 5.6: MLP - Top 5 Prediction with Predicted Probability

5.6 Source Code

https://github.com/apanchal/iima-epaba-batch03/blob/master/notebooks/0000_002_2.ipynb

CHAPTER SIX

6. Deep Learning Models

6.1. LeNet-5

The LeNet-5 architecture consists of two sets of convolutional and average pooling layers, followed by a flattening convolutional layer, then two fully connected layers and finally a softmax classifier.

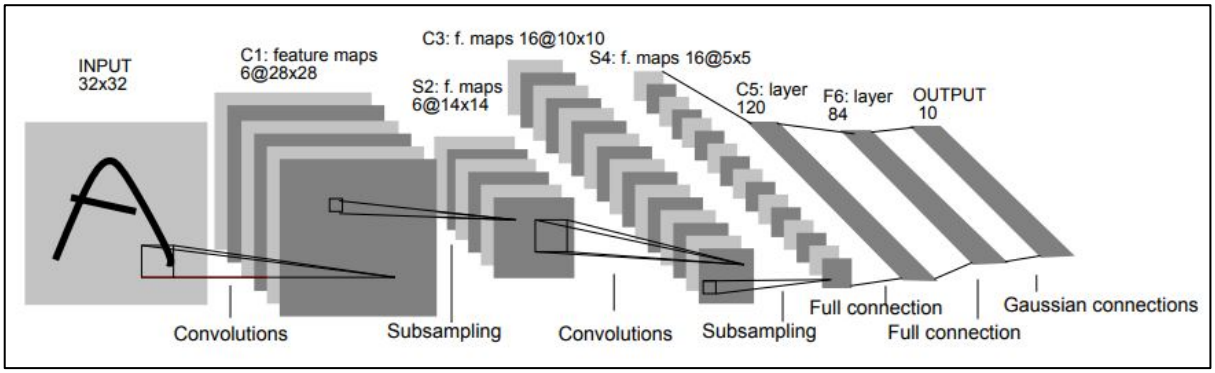


Figure 6.1: LeNet-5 Architecture - Original Image published in [LeCun et al., 1998]

6.1.1. Model Architecture

Image => Convolution => ReLU => Pooling => Convolution => ReLU => Pooling => FullyConnected => ReLU => FullyConnected

Layer 1 (Convolutional): The output: 28x28x6

Activation: ReLU

Pooling. The output: 14x14x6

Layer 2 (Convolutional): The output: 10x10x16

Activation: ReLU

Pooling. The output: 5x5x16

Flattening: Flatten the pooling layer output shape such that it's 1D instead of 3D.

Layer 3 (Fully Connected): The output :120

Activation: ReLU

Layer 4 (Fully Connected): the output: 84

Activation: ReLU

Layer 5 (Fully Connected): The Final output: 43

6.1.2. Model Performance Curve

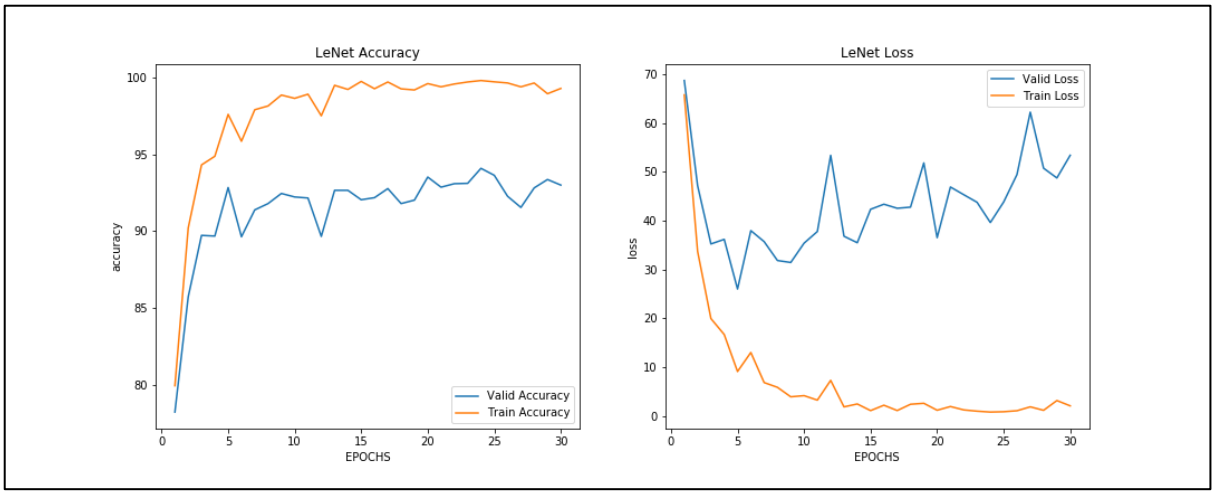


Figure 6.2: LeNet – 5 Accuracy and Loss Curves

6.1.3. Testing LeNet-5 using Test dataset

```
[33]: with tf.Session() as sess:
          LeNet_Model.saver.restore(sess, os.path.join('/Users/apanchal/EPABA/iima-epaba-batch03/Saved_Models/LeNet/', model_name))
          y_pred = LeNet_Model.y_predict(X_test_preprocessed)
          test_accuracy = sum(y_test == y_pred)/len(y_test)
          print("Test Accuracy = {:.1f}%".format(test_accuracy*100))

      INFO:tensorflow:Restoring parameters from /Users/apanchal/EPABA/iima-epaba-batch03/Saved_Models/LeNet/LeNet
      Test Accuracy = 89.5%
```

Figure 6.3: LeNet-5 test accuracy

6.1.4. Confusion Metric

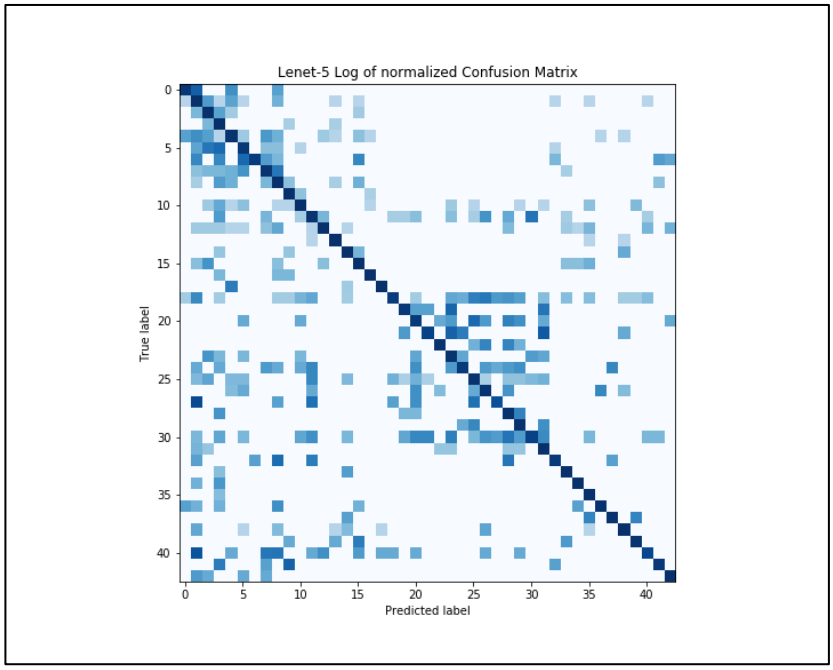


Figure 6.4: LeNet-5 Log of Normalized Confusion Matrix

Observation:

1. We can see two major clusters in above confusion matrix, first cluster is between classes 0-8 and 20 – 30, and these classes ranges turns out to be speed limits signs and some road signs with triangular shapes. We observed the same in Figure 6.4 that speed signs are misclassified among themselves by our model, same case with traffic signs with triangular shapes.

2. It possible to improve above misclassification by introducing ML pipeline in which we can first identify Groups of signs (like Speed limit/triangular shapes signs versus others) and then have second classifier to which can classify image based on more finer features(such as the actual speed limit).

6.1.5. Testing LeNet-5 Model on New Images

```
INFO:tensorflow:Restoring parameters from /Users/apanchal/EPABA/iima-epaba-batch03/Saved_Models/LeNet/LeNet
New Images Test Accuracy = 55.0%
```

Figure 6.5: LeNet-5 – New Image Test Accuracy

Figure 6.6: LeNet - Top 5 Prediction with Predicted Probability

6.1.6. Source Code

https://github.com/apanchal/iima-epaba-batch03/blob/master/notebooks/0000_004_1.ipynb

6.2. VGGNet

VGGNet was first introduced in 2014 by K. Simonyan and A. Zisserman from the University of Oxford in a paper called Very Deep Convolutional Networks for Large-Scale Image Recognition. They were investigating the convolutional network depth on its accuracy in the large-scale image recognition setting.

6.2.1. Model Architecture

We have made some modification in original VGGNet architecture, by reducing last four layers (original VGGNet architecture has 16-19 layers) and kept only 12 layers to accommodate with available computational resources.

This Convolution Net follows these steps:

Input => Convolution => ReLU => Convolution => ReLU => Pooling => Convolution => ReLU => Convolution => ReLU => Pooling => Convolution => ReLU => Convolution => ReLU => Pooling Layer => FullyConnected => ReLU => FullyConnected => ReLU => FullyConnected

Layer 1 (Convolutional): The output: 32x32x32.

Activation: ReLU (Can be any activation function)

Layer 2 (Convolutional): The output: 32x32x32.

Activation: ReLU (Can be any activation function)

Layer 3 (Pooling) The output: 16x16x32.

Layer 4 (Convolutional): The output: 16x16x64.

Activation: ReLU (Can be any activation function)

Layer 5 (Convolutional): The output: 16x16x64.

Activation: ReLU (Can be any activation function)

Layer 6 (Pooling) The output: 8x8x64.

Layer 7 (Convolutional): The output: 8x8x128.

Activation: ReLU (Can be any activation function)

Layer 8 (Convolutional): The output: 8x8x128.

Activation: ReLU (Can be any activation function)

Layer 9 (Pooling) The output: 4x4x128.

Flattening: Flatten the pooling layer output shape such that it's 1D instead of 3D.

Layer 10 (Fully Connected): The output: 128
Activation: ReLU (Can be any activation function)

Layer 11 (Fully Connected): The output: 128
Activation: ReLU (Can be any activation function)

Layer 12 (Fully Connected): The output: 43

6.2.2. Model Performance Curve

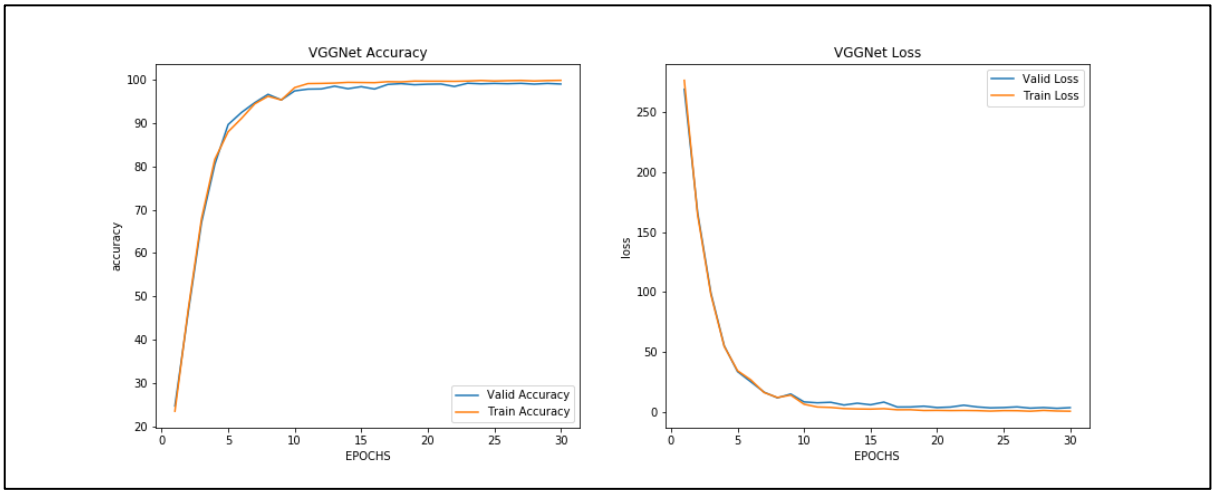


Figure 6.7: VGGNet Accuracy and Loss Curves

6.2.2. Testing VGGNet using Test dataset

Now, we'll use the testing set to measure the accuracy of the model over unknown examples.

```
with tf.Session() as sess:
    VGGNet_Model.saver.restore(sess, os.path.join(DIR, "VGGNet"))
    y_pred = VGGNet_Model.y_predict(X_test_preprocessed)
    test_accuracy = sum(y_test == y_pred)/len(y_test)
    print("Test Accuracy = {:.1f}%".format(test_accuracy*100))
```

```
INFO:tensorflow:Restoring parameters from ../Saved_Models/VGGNet/VGGNet
Test Accuracy = 97.2%
```

Figure 6.8: VGGNet – Test Accuracy

6.2.3. Confusion Metric

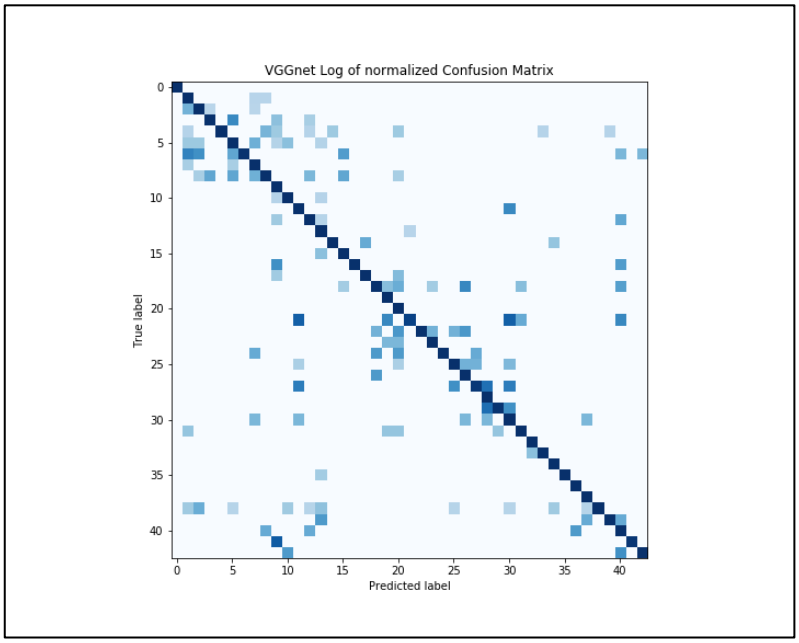


Figure 6.9: VGGNet Log of Normalized Confusion Matrix

6.2.4. Testing VGGNet Model on New Images

```
INFO:tensorflow:Restoring parameters from ../Saved_Models/VGGNet/VGGNet
New Images Test Accuracy = 75.0%
```

Figure 6.10: VGGNet Accuracy on New Images

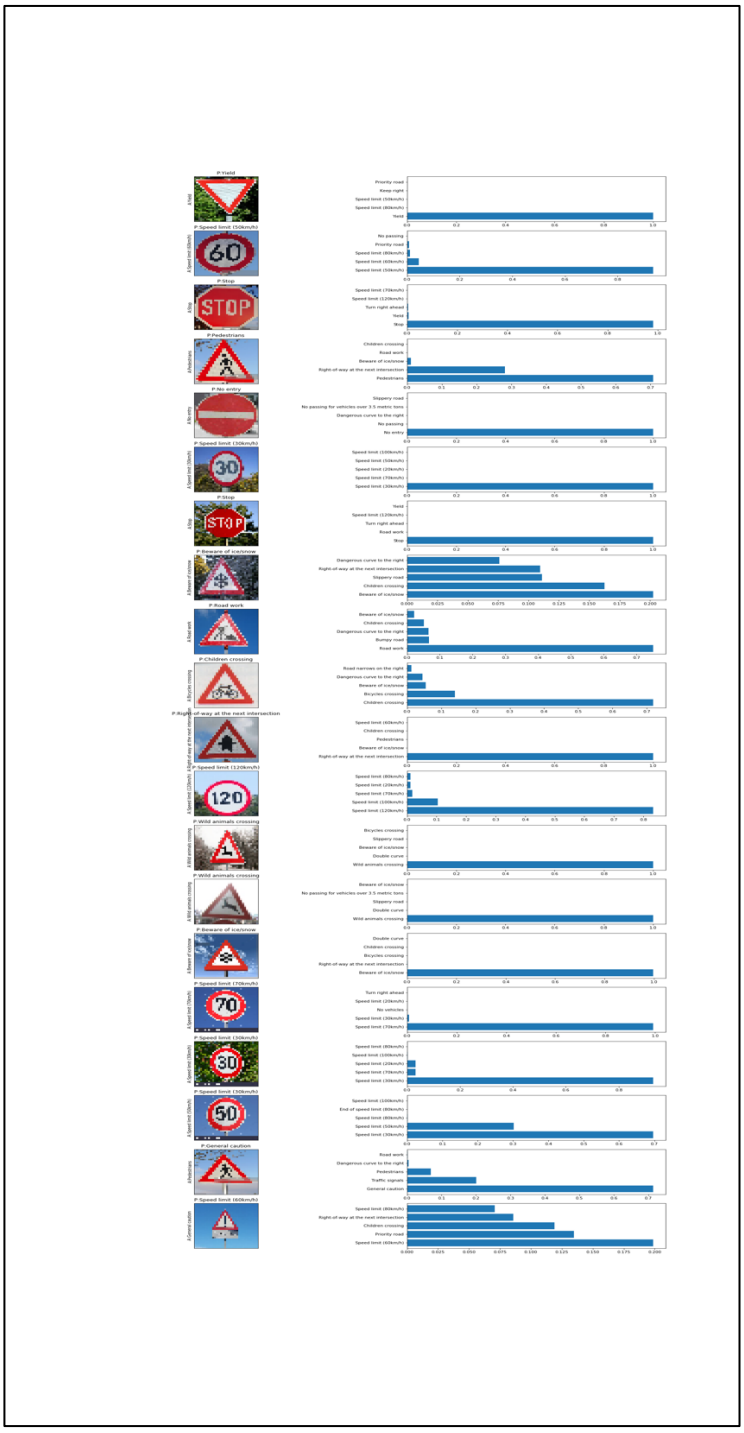


Figure 6.11: VGGNet - Top 5 Prediction with Predicted Probability

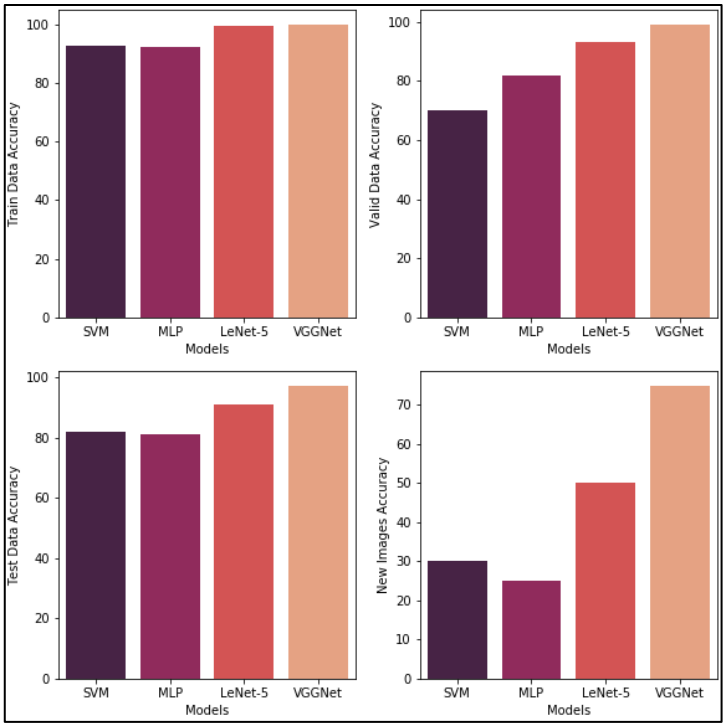
6.2.3. Source Code

https://github.com/apanchal/iima-epaba-batch03/blob/master/notebooks/0000_005_1.ipynb

Summary

In this thesis, we have studied right from traditional classification algorithms such as SVM, MLP to state-of-the-art algorithms using Deep Learning. Below table highlight accuracy of used algorithms:

	Models	Test Data Accuracy	Train Data Accuracy	Valid Data Accuracy	New Images Accuracy
0	SVM	82.0	92.62	70.24	30.0
1	MLP	81.0	92.10	81.79	25.0
2	LeNet-5	90.8	99.26	92.99	50.0
3	VGGNet	97.2	99.88	99.04	75.0



There are two major reason why traditional classification algorithms such as SVM, MLP has low scores compare to CNN:

1. Flattening the image to a 1D vector input leads to losing the spatial features of 2D images

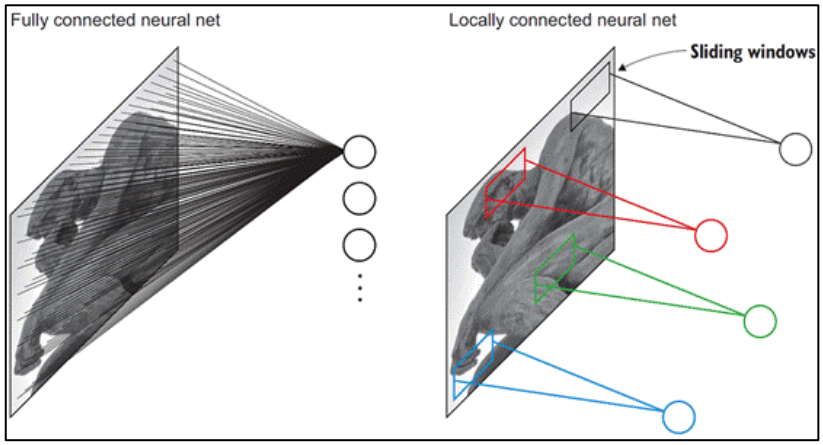
As we seen in this thesis, before feeding the image to the hidden layers/classification algorithms, we must flatten the image matrix to a 1D vector because SVM/MLPs take a flatten vector as an input. Which means throwing away all the 2D information contained in an image. Treating an input as a simple vector of number with no special structure might work well for 1D signals, like the housing price. But in 2D images, it will lead to information loss because the network doesn't relate the pixel values to each other when trying to find patterns. SVM/MLPs have no knowledge of the fact that these pixel numbers were originally spatially arranged in a grid and that they are connected to each other.

CNNs on the other hand, do not require flattening the image. We can feed the raw image matrix of pixels to our network. This will allow the CNN to understand that pixels that are close to each other are heavily related than the pixels that are far apart.

2. Fully connected (dense) layers - For MLP ONLY.

MLPs are composed of Dense layers that are fully connected to each other. Fully connected means that every node in one layer is connected to ALL nodes from the previous layer and all nodes in the next layer. In this scenario, each neuron will have parameters (weights) to train per each neuron from the previous layer. if we have an image with dimensions = 32x32), it will yield a million 1024 for each single node in the first hidden layer, it yields to high number of parameters in such a small network.

CNNs on the other hand are locally connected layers. Where their nodes are connected to only a small subset of the previous layers' nodes. Locally connected layers use far fewer parameters than a densely connected layer as we will see when we discuss CNNs.



Conclusions and Recommendations

We have concluded that the loss of information caused by flattening the 2D image matrix to a 1D vector and the computational complexity of fully connected layers with images suggest that we need an entirely new way of processing the image input where the 2D information is not entirely lost. This is where Convolutional Networks come in. CNNs accept the full image matrix as an input which helps the network significantly understand the pattern contained in the pixel values.

.

Directions of Future Work

I would definitely like to explore relation between model classification performance and computational complexity.

Something that was not studied in this thesis is the effect of using Feature Engineering to find out important features before feeding into Classification algorithms such as SVM and MLP.

Bibliography

1. BOOK AND REFERENCES

Handmann, U., Kalinke, T., Tzomakas, C., Werner, M., Seelen, W.: An image processing system for driver assistance. Image Vis. Comput. 18(5), 367–376 (2000)

Google Scholar

Swathi, M., et al.: Automatic traffic sign detection and recognition: a review. International Conference on Algorithms, Methodology, Models and Applications in Emerging Technologies (ICAMMAET), pp. 1–17 (2017)

Google Scholar

2. DISSERTATION AND THESES

Álvaro Arcos-García, Juan A. Álvarez García, and Luis M. Soria-Morillo. Deep neural network for traffic sign recognition systems: An analysis of spatial transformers and stochastic optimisation methods. Neural Networks, 99:158 – 165, 2018. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2018.01.005 .

URL http://www.sciencedirect.com/science/article/pii/S0893608018300054.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. Deep Learning. MIT Press, 2016. URL http://www.deeplearningbook.org.

Carl Ekman. raffic Sign Classification Using Computationally Efficient Convolutional Neural Networks, Corpus ID: 196179100

URL: https://liu.diva-portal.org/smash/get/diva2:1324051/FULLTEXT01.pdf

Karen Simonyan, Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition

https://arxiv.org/pdf/1409.1556.pdf

3. WEB REFERENCES:

scikit-learn - Machine Learning in Python Examples

URL: https://scikit-learn.org/0.21/modules/generated/sklearn.model_selection.validation_curve.html

(Tutorial) Support Vector Machines (SVM) in Scikit-learn - DataCamp

URL: https://www.datacamp.com/community/tutorials/svm-classification-scikit-learn-python

LeNet-5 – A Classic CNN Architecture

URL: https://engmrk.com/lenet-5-a-classic-cnn-architecture/

German-Traffic-Sign-Classification-Using-TensorFlow

URL: https://github.com/mohamedameen93/German-Traffic-Sign-Classification-Using-TensorFlow?files=1