

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 INSTITUTE 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



CERTIFICATE

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:

Traffic sign recognition is an important problem for autonomous cars and driver assistance systems. With recent developments in the field of machine learning, high performance can be achieved.

Traffic sign detection is a high relevance computer vision problem and is the basis for a lot of applications in industry such as Automotive etc. Traffic signs can provide a wide range of variations between classes in terms of color, shape, and the presence of pictograms or text.

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

This project work aims to investigate different 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.

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

Table of Contents

LIST OF FIGURES.....	8
LIST OF TABLES.....	9
1. INTRODUCTION	1
1.1 AIM.....	2
1.2. PROBLEM STATEMENTS.....	2
1.3 ALGORITHMIC PROCESS	2
1.4 LIMITATION.....	2
1.5 PERFORMANCE EVALUATION METRICS	2
1.6 TEAM BACKGROUND	4
2. DATASET	5
2.1. GTSRB DATASET OVERVIEW.....	5
2.2 DATASET SUMMARY & EXPLORATION.....	6
3. DATA PREPROCESSING	9
3.1 PREPROCESSING TECHNIQUES	10
3.2 DATA AUGMENTATION	10
3.2.1 Slight Rotation of Images	11
3.2.2 Image Translation	12
3.3 SHUFFLING	12
3.4 BILATERAL FILTERING.....	13
3.4.1 Grayscaleing.....	13
3.4.2 Local Histogram Equalization	14
3.3 NORMALIZATION	15
4. CLASSIFIER: SKLEARN'S SUPPORT VECTOR MACHINE	17
4.1. HYPER-PARAMETERS OF SVM	19
4.1.1. Choosing Right values of Gamma for Gaussian (rbf) Kernel.....	20
4.1.2 Choosing Right values of Regularization for Gaussian (rbf) Kernel	21
4.2. SVM LEARNING CURVE	22
4.3 TESTING SVM USING TEST DATASET.....	23
4.4 CONFUSION METRIC.....	23
4.5 TESTING SVM MODEL ON NEW IMAGES.....	23
4.6 SUMMARY	23
5. CLASSIFIER: SKLEARN'S MLPCLASSIFIER NEURAL NET.....	24
5.1 HYPER-PARAMETERS OF MLP	24
5.2 MLP LODD CURVE	24
5.3 TESTING MLP USING TEST DATASET	24
5.4 CONFUSION METRIC.....	24
5.5 TESTING MLP MODEL ON NEW IMAGES.....	24
5.6 SUMMARY	24
6. DEEP LEARNING MODELS	25
6.1. LENET-5	25
6.1.1. Model Architecture.....	25
6.1.2. Model Performance Curve.....	25
6.1.3. Testing LeNet-5 using Test dataset	25
6.1.4. Confusion Metric	25
6.1.5. Testing LeNet-5 Model on New Images.....	25
6.1.6. Summary	25

6.2. VGGNET.....	26
6.2.1. MODEL ARCHITECTURE	26
6.2.2. <i>Model Performance Curve</i>	26
6.2.2. <i>Testing VGGNet using Test dataset</i>	26
6.2.3. <i>Confusion Metric</i>	26
6.2.4. <i>Testing VGGNet Model on New Images</i>	26
6.2.3. <i>Summary</i>	26
SUMMARY	27
CONCLUSIONS AND RECOMMENDATIONS	28
DIRECTIONS OF FUTURE WORK	29
BIBLIOGRAPHY	30
REFERENCES.....	31

List of Figures

FIGURE 1.1: ILLUSTRATION OF TRAFFIC SIGN CLASSIFICATION SYSTEM.....	1
FIGURE 1.2: PARTITIONING THE TRAFFIC SIGN CLASSIFICATION PROBLEM USING REGION INFORMATION.	2
FIGURE 2.1: RANDOM SAMPLES FROM EACH OF THE 43 CLASSES IN THE GTSRB DATASET.	6
FIGURE 2.2: HISTOGRAM OF THE COUNT OF IMAGES IN EACH DATA SET	7
FIGURE 2.3: DATA DISTRIBUTION IN TRAINING AND VALIDATION DATASET	7
FIGURE 2.4: RATIO BETWEEN THE VALIDATION AND TRAINING REPRESENTATION, PER CLASS	8
FIGURE 3.1: SAMPLE BASELINE DENSE NETWORK ARCHITECTURE	9
FIGURE 3.1.1: SAMPLE BASELINE MODEL ACCURACY WITHOUT ANY PREPROCESSING OR DATA AUGMENTATION.....	9
FIGURE 3.1.2: TRAINING DATA AFTER AUGMENTATION	10
FIGURE 3.2: IMAGES AFTER 10-DEGREE ROTATION	11
FIGURE 3.3: IMAGES AFTER TRANSLATION	12
FIGURE 3.4: IMAGES AFTER GRAYSCALING.....	13
FIGURE 3.5: IMAGES AFTER LOCAL HISTOGRAM EQUALIZATION.....	14
FIGURE 3.6: IMAGES AFTER NORMALIZATION.....	15
FIGURE 3.7: SAMPLE BASELINE MODEL ACCURACY WITH DATA AUGMENTATION	16
FIGURE 4.0: SVM HYPER-PARAMETERS – KERNELS	18
FIGURE 4.1: SVM HYPER-PARAMETERS – REGULARIZATION (C)	18
FIGURE 4.2: SVM HYPER-PARAMETERS – GAMMA	19
FIGURE 4.3: SVM VALIDATION CURVE OF GAMMA FOR GAUSSIAN (RBF) KERNEL.....	20
FIGURE 4.4: SVM VALIDATION CURVE OF REGULARIZATION FOR GAUSSIAN (RBF) KERNEL	21
FIGURE 4.5: LEARNING CURVES (SVM, RBF KERNEL, $\gamma=0.0046415888$, $C=100$)	22

List of Tables

TABLE 1: GTSRB - LABELS FROM WITH CLASS IMAGES AND DESCRIPTIONS.	5
TABLE 2.1: REPRESENTATION STATS FOR VALIDATION AND TRAINING DATASETS PER-CLASS.....	8

1. Introduction

Traffic sign recognition is a key problem in the context of autonomous cars and driver assistance systems. This thesis focuses on the classification step, assuming the detection step is implemented separately.

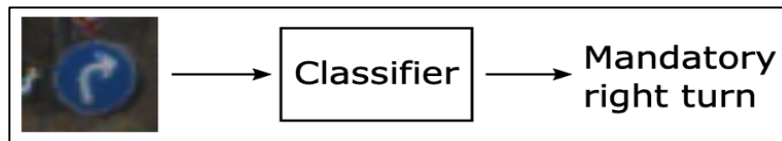


Figure 1.1: Illustration of traffic sign classification system.

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. This means that algorithms are not hand-crafted, but rather generated using data. In particular, deep convolutional neural networks (CNN) have been shown to work well for image recognition tasks.

For classification tasks, training data consisting of images annotated with the correct class and a loss function that quantifies how “bad” the network performs, is needed. The annotations are typically added by humans and constitute a large expense when implementing this type of system, since large datasets are typically required to train the system. 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.

International traffic sign classification results in large differences within classes, solving the entire classification problem directly is expected to require large networks. Large networks might not be feasible to use for real-time applications from computational cost perspective. Usually different methods of using prior knowledge to partition the problem into smaller sub-problems, which hope- fully can be solved using smaller networks with lower computational complexity.

For instance, samples from different countries/regions can be handled separately (see Figure 1.2).

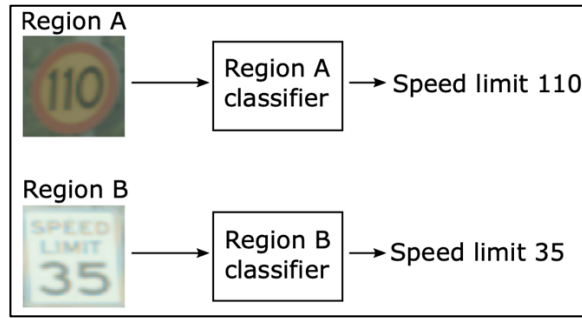


Figure 1.2: Partitioning the traffic sign classification problem using region information.

1.1 Aim

The main focus of this thesis is to study the relation between classification error, by training and evaluating models of different complexity.

1.2. Problem Statements

Classification of German Traffic Sign Recognition Benchmark dataset. Challenges Wide variability in visual appearance. Illumination, weather condition, partial occlusions Goal High accuracy in recognizing signs in real world.

1.3 Algorithmic Process

Similar to any machine learning model building process we had also executed the same steps defined below:

- 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

This thesis only considers the classification part of a full traffic sign recognition system, so detections are assumed to be handled by a separate system. Therefore, the input to the system is image patches approximately cropped to the traffic signs or a false positive, that is, something that looks like a traffic sign, but is not.

Some traffic signs have additional sub-signs, typically with text. The sub-signs can contain extra information, such as which time of day the main sign applies. These additional sub-signs are not included in the classification problem studied in this thesis.

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:

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

And *error rate* is simply

$$E = 1 - A \quad (2.3)$$

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

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

The average *classwise accuracy* as

$$A_c = \frac{1}{n} \sum_{i=1}^n A_i \quad (2.5)$$

The *average classwise error rate* as

$$E_c = 1 - A_c \quad (2.6)$$

The evaluation and results of trained models is calculated by common classification metrics which are defined as follows:

- *Precision (positive predictive value)* - $TP / (TP+FP)$
- *Recall (true positive rate)* = $TP/(TP+FN)$
- *F1-Score* = $2*TP / (2*TP+FP+FN)$

Where TP is the number of positive cases which are labelled correctly, TN is the number of negative cases which are labelled correctly, FP is the number of positive cases which are labelled falsely, and FN is the number of negative cases which are labelled falsely. Also, since the distribution of the number of samples among database are highly unbalanced, F1-score result which is the harmonic mean of precision and recall, is reported and in order to have a graphical view of the trade-off between sensitivity and specificity metrics, ROC curves and their associated AUC values are used.

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. Source Code

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

2. Dataset

In this thesis one public dataset are used to perform the experiments. The dataset used is:

- 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. The pickled data is a dictionary with 4 key/value pairs:

- **'features'** is a 4D array containing raw pixel data of the traffic sign images, (num examples, width, height, channels).
- **'labels'** is a 1D array containing the label/class id of the traffic sign. The file signnames.csv contains id -> name mappings for each id
- **'sizes'** is a list containing tuples, (width, height) representing the original width and height the image.
- **'coords'** is a list containing tuples, (x1, y1, x2, y2) representing coordinates of a bounding box around the sign in the image.

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 with class images and descriptions.

2.2 Dataset Summary & Exploration

A set of random samples from each of the 43 classes in the training data can be seen in Figure 2.1 (a), with 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 in training data can be seen in Figure 2.1 (b).

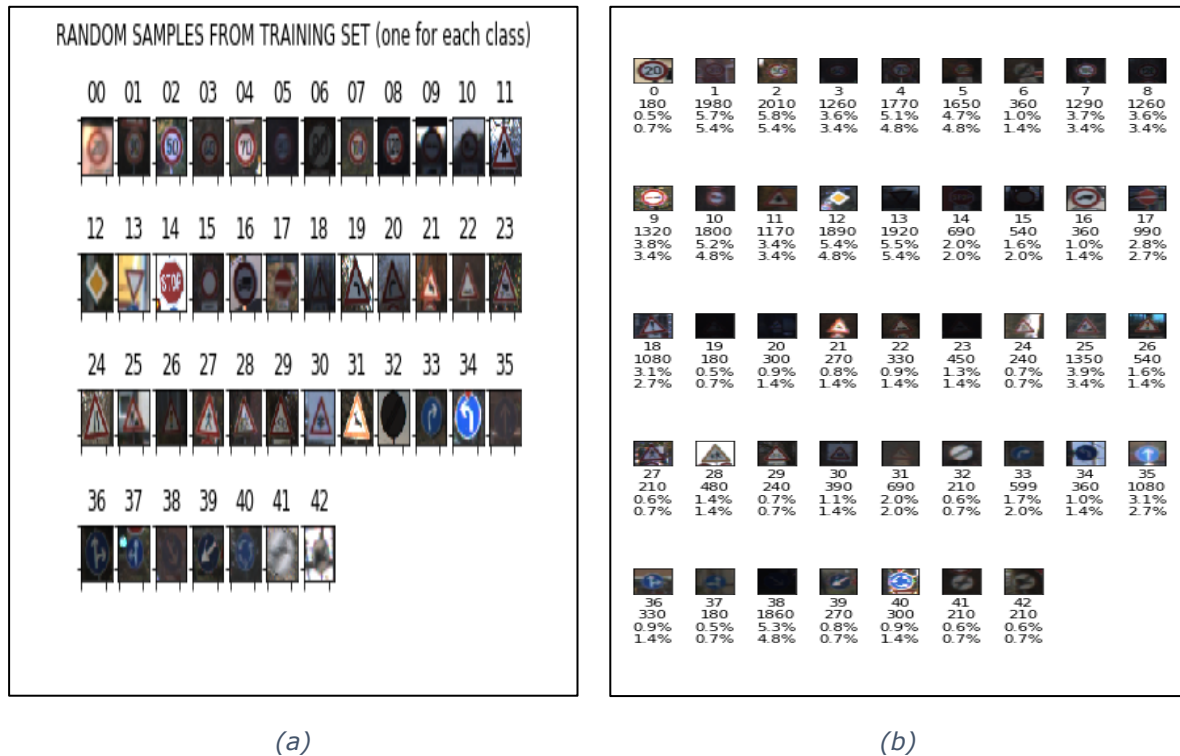
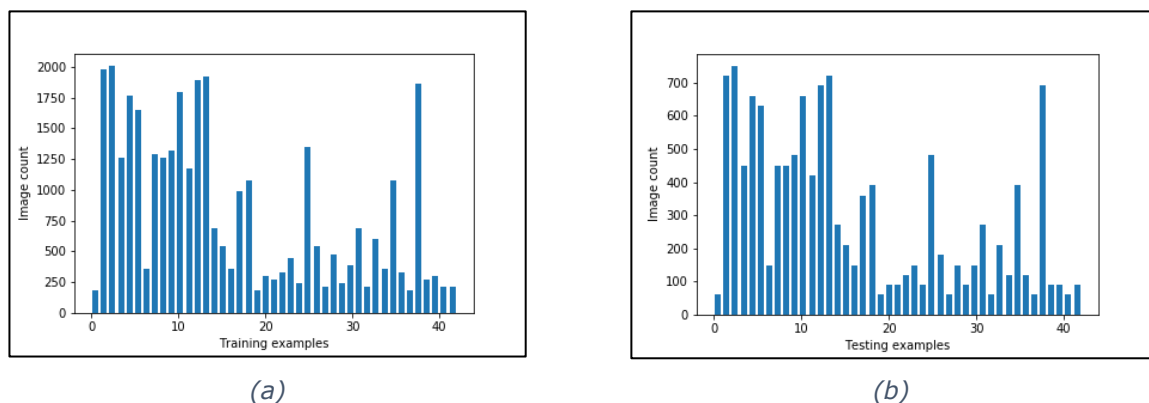
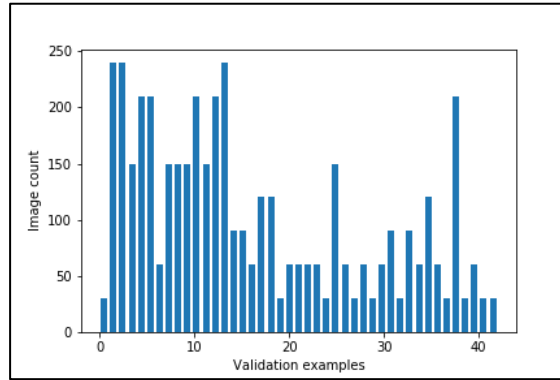


Figure 2.1: Random samples from each of the 43 classes in the GTSRB dataset.

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





(c)

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

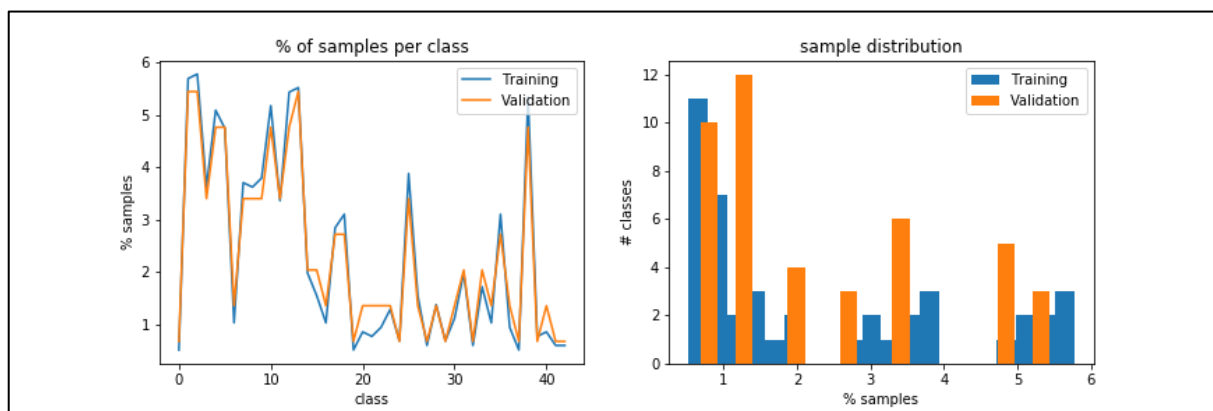


Figure 2.3: Data distribution in Training and Validation dataset

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%

As we can observe in the Table 2.1 and Figure 2.4, a ratio close to 1 indicates that there about the same fraction of validation samples as training samples, in the specific class.

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

Ratio Stats.:

Median=1.01

Average =1.11

Validation (%)	Training (%)	Ratio (%)	
Class			
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

Validation (%)	Training (%)	Ratio (%)	
Class			
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

Validation (%)	Training (%)	Ratio (%)	
Class			
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

Ratio between the validation and training representation, per class

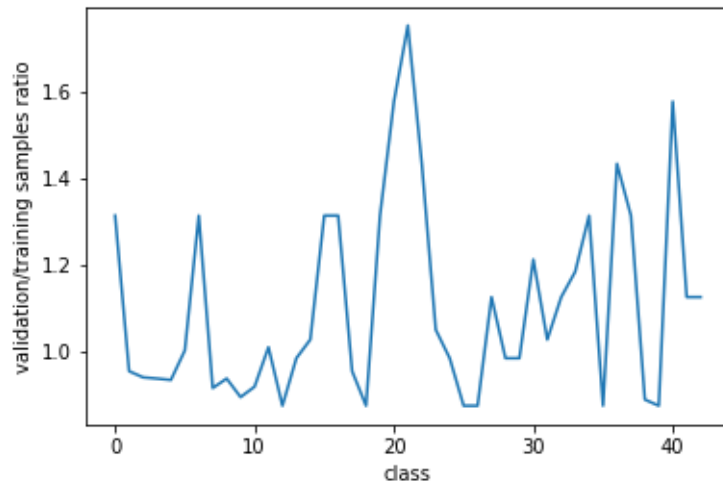


Figure 2.4: Ratio between the validation and training representation, per class

Couple of inferences from the data exploration that we had tackled during the preprocessing stage

1. Class bias issue as some classes seem to be underrepresented
2. Image contrast seems to be low for lot of images

3. Data Preprocessing

It's always a good practice to understand where your model stands without doing any data preprocessing as that would help us to establish a score for a model, which we could improve upon each iteration. The evaluation metric we have used for our model is "accuracy" score. For this purpose, we have used a simple "dense" or "fully" connected neural network architecture for baseline scores and other testing.

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

The input shape is 32*32*3 (as images have 3 color channels). The number of parameters on the first layer would be 393344 ((32*32*3*128) + 128)). We can calculate the number of parameters for the other layers in the same fashion.

The Activation function is "relu". During hyperparameters optimization we can check with Tanh, Sigmoid and other activation function if they are better suited for the task. For now, we stick on to "relu".

There are 4 hidden layers of 128 neurons with relu activation and after each hidden layer except the last one a dropout (50%) function is included.

The output layer has the softmax activation since we are dealing with 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
 - Grayscaleing
 - Local Histogram Equalization
- Normalization

3.2 Data Augmentation

Data Augmentation is used to increase the training set data. Augmenting the data is basically creating more images from the available images but with slight alteration of the images. We generally need data proportional to the parameters we feed the neural networks.

Fixing Class Bias with Data augmentation: We have increased the training set images with data augmentation, to address the class bias issue.

Hence during augmentation, all the classes were fed with 4000 images. In the original dataset Class 2 had the maximum number of training images with 2010 records. The number 4000 (Max class records * ~2) is an arbitrary number we took to make all classes have same number of records.

Check class bias after training data augmentation.

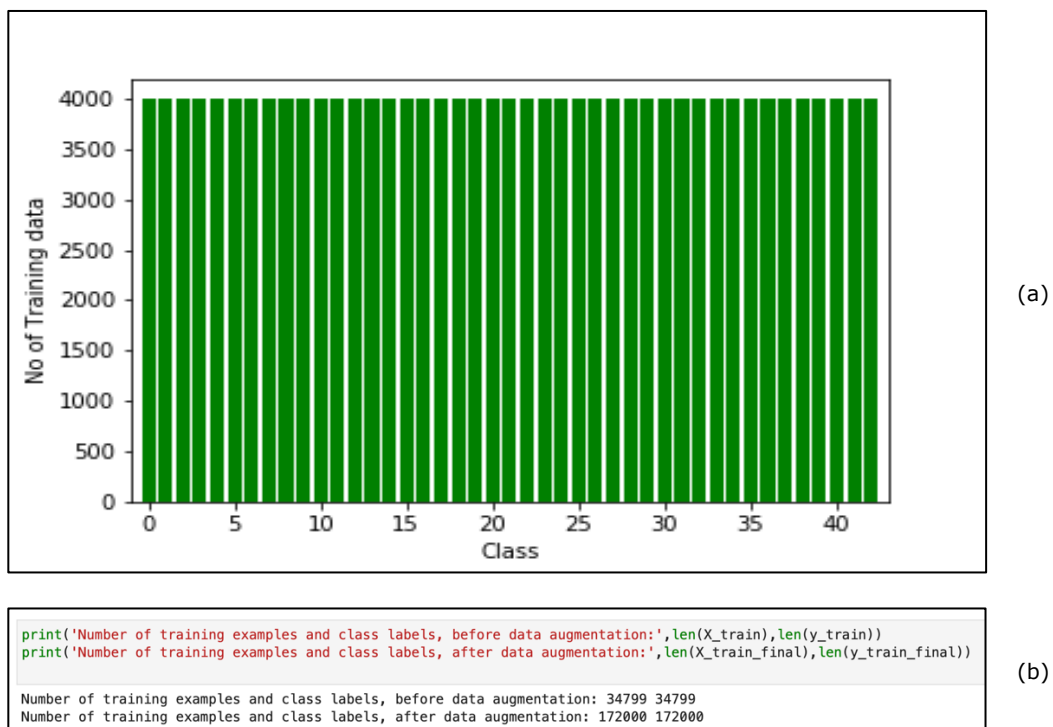


Figure 3.1.2: Training data after augmentation

We have used OpenCV open source library which is excellent for image preprocessing. Some of the techniques used in the process are Rotation, Translation, Bi lateral filtering, Grayscale and Local Histogram Equalization.

3.2.1 Slight Rotation of Images

We used 10 degrees rotation of images. It would not make much sense to rotate images more than that as that might lead to wrong representations of the traffic signs. Let's view a few images after slight rotation, as can be seen in Figure 3.2 (not that noticeable in a few images also).

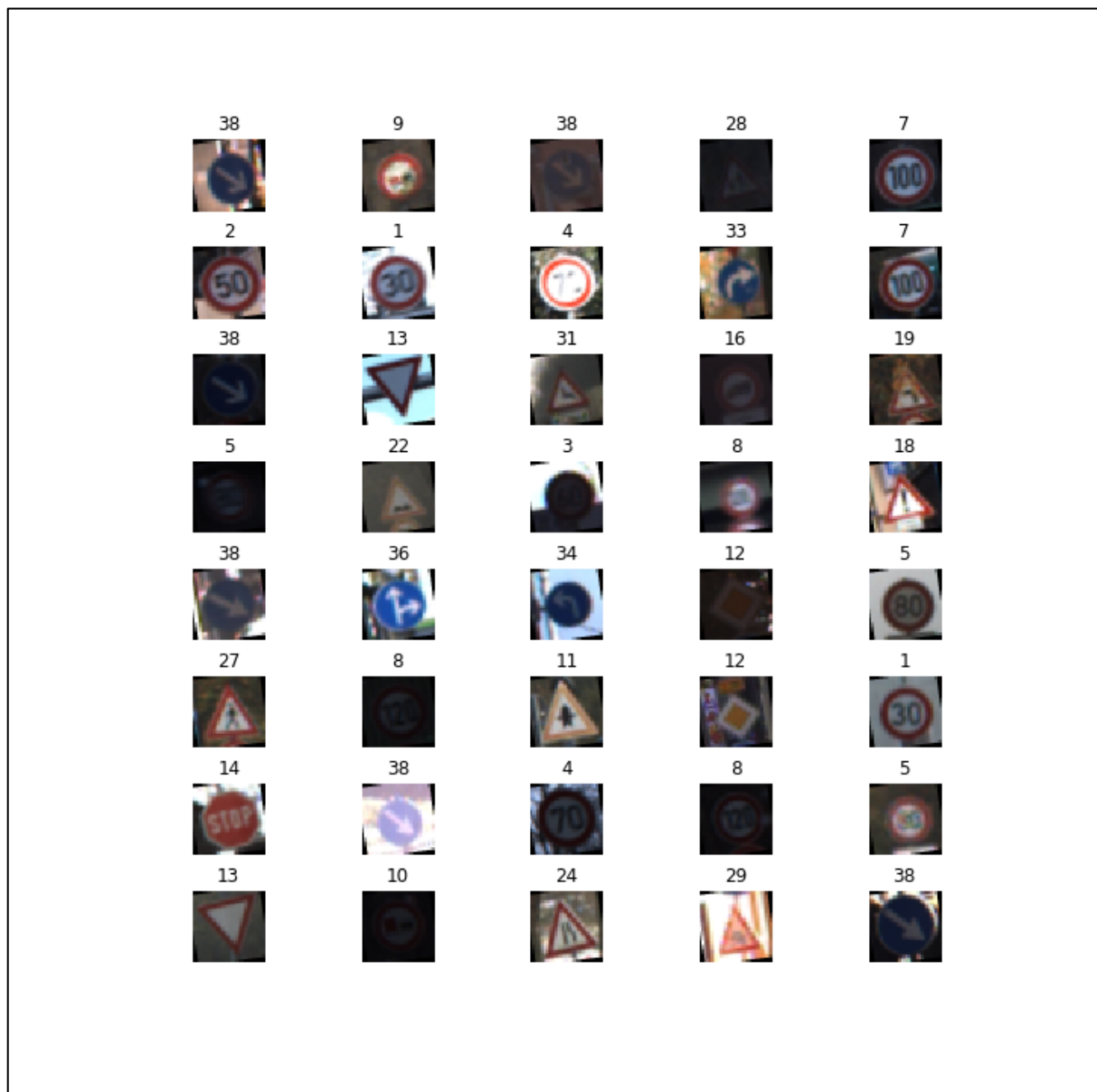


Figure 3.2: Images after 10-degree rotation

3.2.2 Image Translation

This is a technique by which you shift the location of the image. In layman terms, if the image's location is (x_1, y_1) position, after translation it is moved to (x_2, y_2) position. As we can see from the below images, the location is slightly moved downwards.

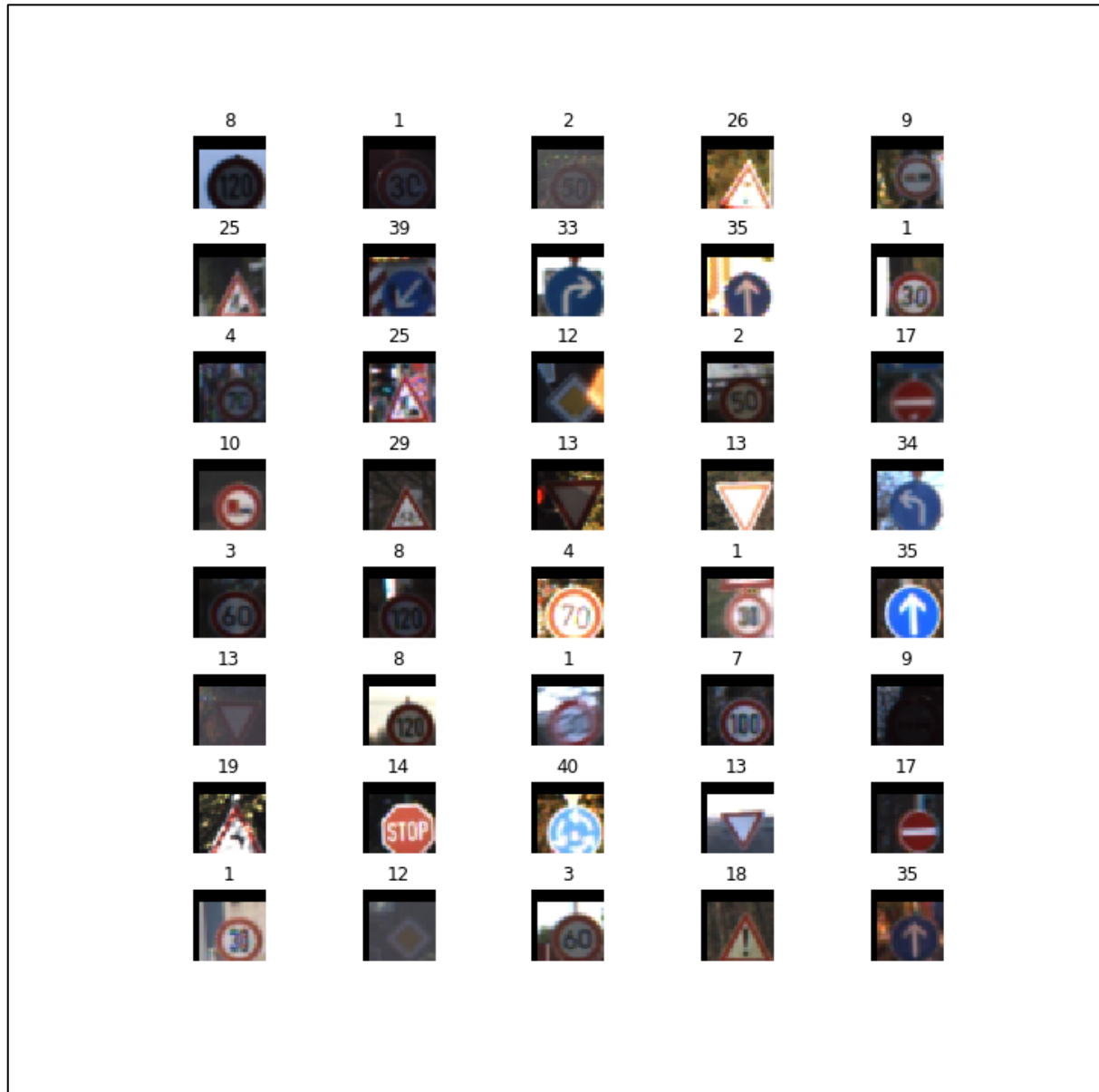


Figure 3.3: Images after translation

3.3 Shuffling

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

3.4 Bilateral Filtering

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

3.4.1 Grayscale

Gray scaling of images is done to reduce the information provided to the pixels and also reduces complexity. We have used OpenCV to convert the training images into grey scale.



Figure 3.4: Images after Grayscale

3.4.2 Local Histogram Equalization

This is done 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

3.3 Normalization

Normalization is a process that changes the range of pixel intensity values. Usually the image data should be normalized so that the data has mean zero and equal variance.



Figure 3.6: Images after Normalization

Model Score after Data augmentation and after fixing class Bias:

The same sample baseline dense neural network architecture as one used above was able to better it's 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 thesis, we have used augmented training data to train all classifier.

4. Classifier: Sklearn's Support Vector Machine

The ability of a machine learning model to classify or label an image into its respective class with the help of learned features from hundreds of images is called as Image Classification.

This is typically a supervised learning problem where we humans must provide training data (set of images along with its labels) to the machine learning model so that it learns how to discriminate each image (by learning the pattern behind each image) with respect to its label.

In this thesis, we will look into one such image classification problem namely Road/Traffic Sign Recognition, which is a hard problem because there are many such signs. As we know machine learning is all about learning from past data, we need huge dataset of road/traffic sign to perform real-time signs recognition. We have performed image classification task using computer vision and machine learning algorithms with the help of Python.

As part of thesis, 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, and VGGNet, Chapter - 6.

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

A Support Vector Machine is a supervised machine learning algorithm which can be used for both classification and regression problems. It follows a technique called the kernel trick to transform the data and based on these transformations, it finds an optimal boundary between the possible outputs. In simple words, it does some extremely complex data transformations to figure out how to separate the data based on the labels or outputs defined.

In order to improve the SVM model accuracy, there are several parameters need to be tuned. Three major parameters including:

1. Kernels: The main function of the kernel is to take low dimensional input space and transform it into a higher-dimensional space. It is mostly useful in non-linear separation problem.

We have Gaussian (rbf), and sigmoid (sigmoid) kernels to see which one works better for our problem.

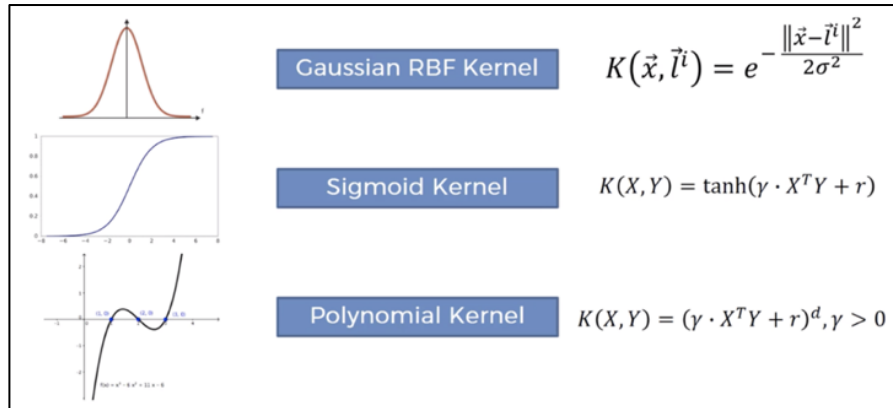


Figure 4.0: SVM Hyper-Parameters – Kernels

2. C (Regularization): C is the penalty parameter, which represents misclassification or error term. The misclassification or error term tells the SVM optimization how much error is bearable. This is how you can 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.

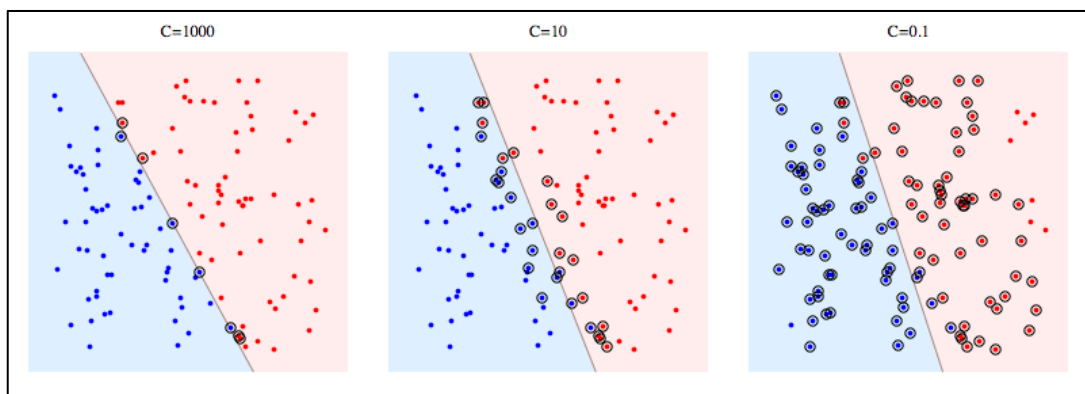


Figure 4.1: SVM Hyper-Parameters – Regularization (C)

3. Gamma

A lower value of Gamma will loosely fit the training dataset, whereas a higher value of gamma will exactly fit the training dataset, which causes over-fitting. In other words, you can say a low value of gamma considers only nearby points in calculating the separation line, while a value of gamma considers all the data points in the calculation of the separation line.

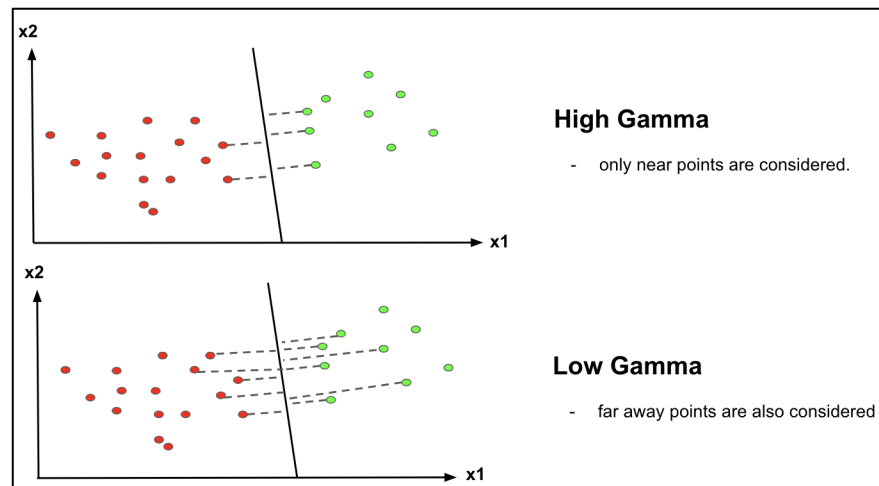


Figure 4.2: SVM Hyper-Parameters – Gamma

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

We have explored the power of SVMs for classification.

4.1. Hyper-parameters of SVM

Hyper-parameters are parameters that are not directly learnt within estimators. 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', '10.0'] for finding validation accuracy.

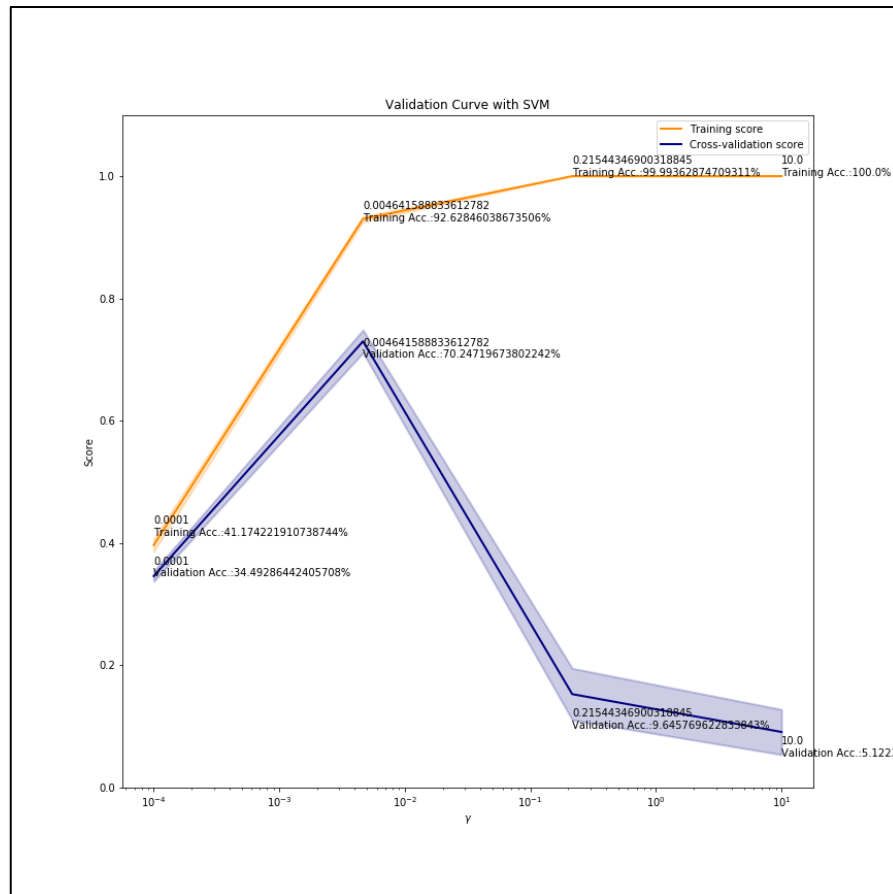


Figure 4.3: SVM Validation Curve of Gamma for Gaussian (rbf) Kernel

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

1. For very low values of gamma (γ), both the training score and the validation score are low. This is called **underfitting**.
2. Medium values of gamma (γ) will result in high values for both scores, i.e. the classifier is performing fairly well.
3. If gamma (γ) 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 (γ) as 0.0046415888**, because this 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.

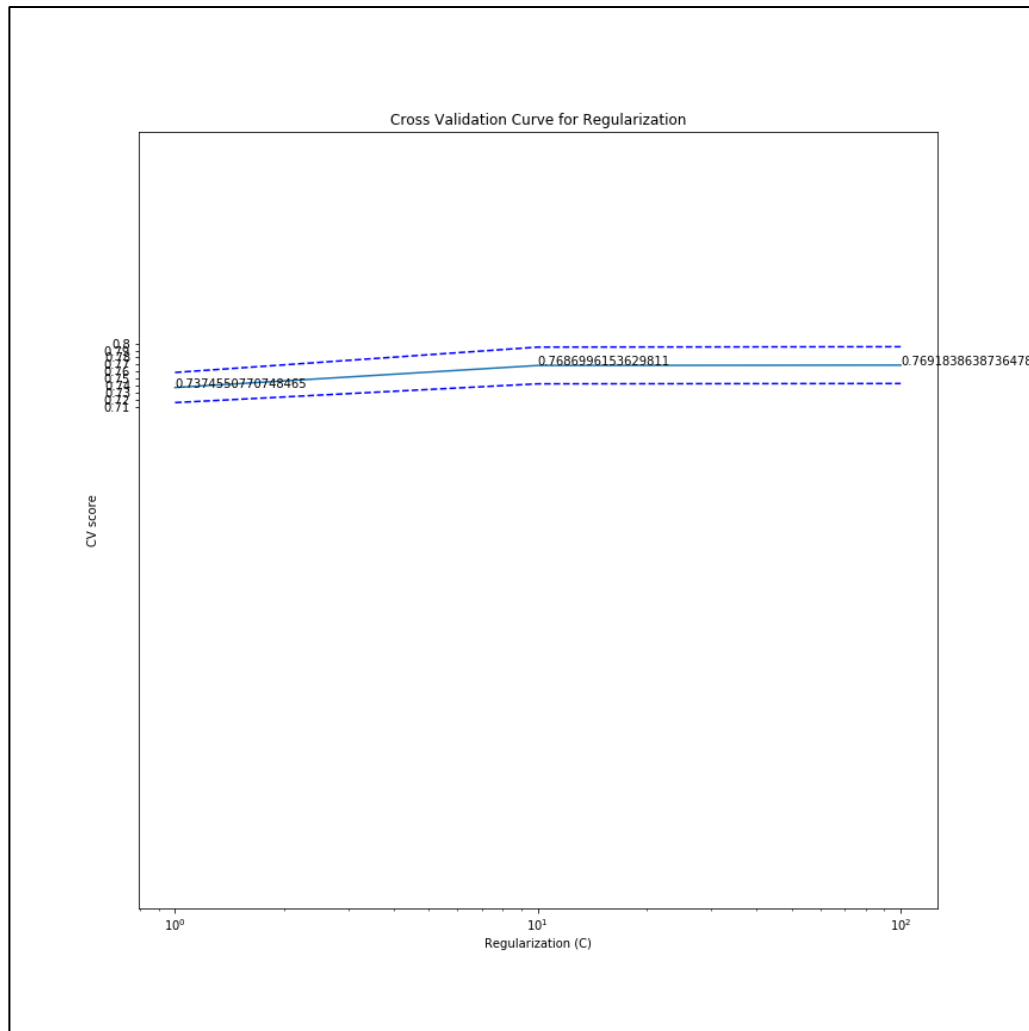


Figure 4.4: SVM Validation Curve of Regularization for Gaussian (rbf) Kernel

4.2. SVM Learning Curve

In order to determine cross-validated training and test scores for different training set sizes, we have used learning curve from scikit-learn library.

A cross-validation generator splits the whole dataset 100 times in training and test data. Subsets of the training set with varying sizes will be used to train the estimator and a score for each training subset size and the test set will be computed. Afterwards, the scores will be averaged over all k runs for each training subset size.

Figure 4.5: Learning Curves (SVM, RBF kernel, $\gamma=0.0046415888$, $C=100$)

We can see clearly that the training score is still around the maximum and the validation score could be increased with more training samples.

4.3 Testing SVM using Test dataset

4.4 Confusion Metric

4.5 Testing SVM Model on New Images.

4.6 Summary

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.

Architecture and Other Specifications

Need to specify a few more things about our model and the way it should be fit. First of all, we need to give it a fixed architecture for the net. Professor. N.G. gives us these rules of thumb:

- the number of input units will be the number of features
- for multiclass classification the number of output units will be the number of labels
- try a single hidden layer, or if more than one then each hidden layer should have the same number of units
- the more units in a hidden layer the better, try the same as the number of input features up to twice or even three or four times that

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

5.2 MLP Loss Curve

5.3 Testing MLP using Test dataset

5.4 Confusion Metric

5.5 Testing MLP Model on New Images.

5.6 Summary

6. Deep Learning Models

6.1. LeNet-5

6.1.1. Model Architecture

6.1.2. Model Performance Curve

6.1.3. Testing LeNet-5 using Test dataset

6.1.4. Confusion Metric

6.1.5. Testing LeNet-5 Model on New Images

6.1.6. Summary

6.2. VGGNet

6.2.1. Model Architecture

6.2.2. Model Performance Curve

6.2.2. Testing VGGNet using Test dataset

6.2.3. Confusion Metric

6.2.4. Testing VGGNet Model on New Images

6.2.3. Summary

Summary

Conclusions and Recommendations

.

Directions of Future Work

Bibliography

1. BOOK

Ben Stopford. Designing Event-Driven Systems Concepts and Patterns for Streaming Services with Apache Kafka. Sebastopol- CA 95472: O'Reilly Media, Inc. 2018

Eben Hewitt. Cassandra: The Definitive Guide. Sebastopol-CA: O'Reilly Media, Inc., 2011.

Munish K. Gupta. Akka Essentials. Birmingham - Mumbai: Packt Publishing Ltd,2012

Nishant Garg. Learning Apache Kafka Second Edition. Birmingham - Mumbai: Packt Publishing Ltd,2012

2. DISSERTATION AND THESES

Linus Magnusson. Event-driven architecture for data collection in smart factories. Master of Science Thesis TPRMM [Master Thesis]. STOCKHOLM: KTH Department of Production Engineering; 2017

Behrooz Malekzadeh. Event-Driven Architecture and SOA in collaboration. Master of Thesis in IT-Management [Master Thesis]. Gothenburg-Sweden: University of Gothenburg; 2010.

Maxime Klusman. Event Driven Architecture in software development projects. Master Thesis Computing Science [Master Thesis]. Nijmegen: Radboud University; 2016

References

1.

Appendices

This appendix lists textual class labels used in the original papers for the GTSRB datasets, along with class images.