HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

MACHINE LEARNING

MINI PROJECT: TRAFFIC SIGN CLASSIFICATION

Instructors: Prof. Than Quang Khoat
Students: Nguyen Truong Truong An - 20204866
Nguyen Thanh Dat - 20204903
Nguyen Duy Khanh - 20204914
Nguyen Duc Quyet - 20200522
Nguyen Phuc Thanh - 20200594

Ha Noi, July - 2022

1

Abstract

This report describes how we implement feature engineering method, traditional Machine Learning and modern Deep Learning methods on the German Traffic Sign Recognition Benchmark(**GTSRB**) dataset to classify the traffic signs and evaluate each approach's result. Also, we will provide experimental results demonstrating the performance of the models and evaluation metrics.

1 Introduction

In recent years, Artificial Intelligence have shown to be effective in many fields like natural language processing, image processing, speech processing. In the automotive industry, many companies are developing and deploying AI applications in their vehicles, for example, self-driving car. In this project, we are working on Traffic Sign Classification which is a very useful application. The task of recognizing and following traffic signs is very important because it can reduce the number of accidents on roads and help the drivers drive more securely. In the scope of this subject, we propose a feature extractor named HOG (histogram of oriented gradient) which create feature descriptor from images and then apply machine learning method to build models which are trained and tested using the GTSRB dataset.

2 Background

This section provides a brief explanation of the theoretical background necessary to understand our project report.

2.1 Feature Extraction

Feature Extraction aims to reduce the number of features in a dataset by creating new features from the existing ones (and then discarding the original features). These new reduced set of features should then be able to summarize most of the information contained in the original set of features.

So instead of using vector of flatten pixels, which contains many redundant features (like background pixels), we will apply a feature extraction method to reduce the number of features needed for processing without losing important or relevant information.

In this problem, we use **Histogram of oriented gradient** method.

2.1.1 What is it?

Histogram of Oriented Gradients, also known as HOG, is a feature descriptor. It is used in computer vision and image processing.The technique counts occurrences of gradient orientation in the localized portion of an image.The HOG descriptor focuses on the structure or the shape of an object. It is better than any edge descriptor as it uses magnitude as well as angle of the gradient to compute the features. For the regions of the image it generates histograms using the magnitude and orientations of the gradient.

2.1.2 Calculating the Gradients

To calculate the gradient, a discrete derivative mask is used in horizontal and vertical direction, which is Sobel mask in our case.

Horizontal derivative

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \mathbf{I}$$

Vertical derivative

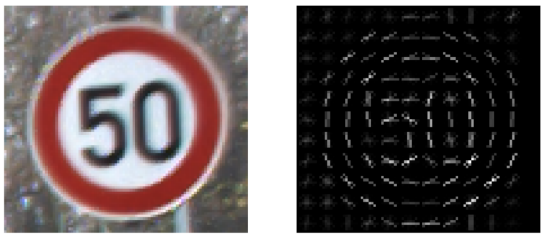
$$G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} * \mathbf{I}$$

The * sign is like a filter like convolutional filter The gradient magnitude and gradient direction can be created from G_x and G_y as below:

Gradient magnitude $G = \sqrt{G_x^2 + G_y^2}$

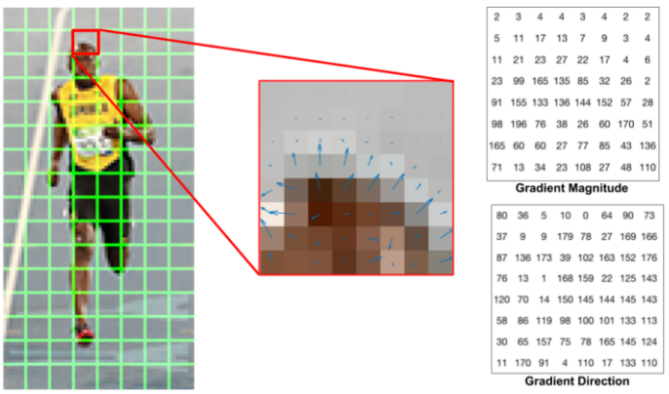
Gradient direction $G = \frac{G_y}{G_x}$

By calculating the gradient magnitude and direction, we have a feature extraction photo as below:



2.1.3 HOG process

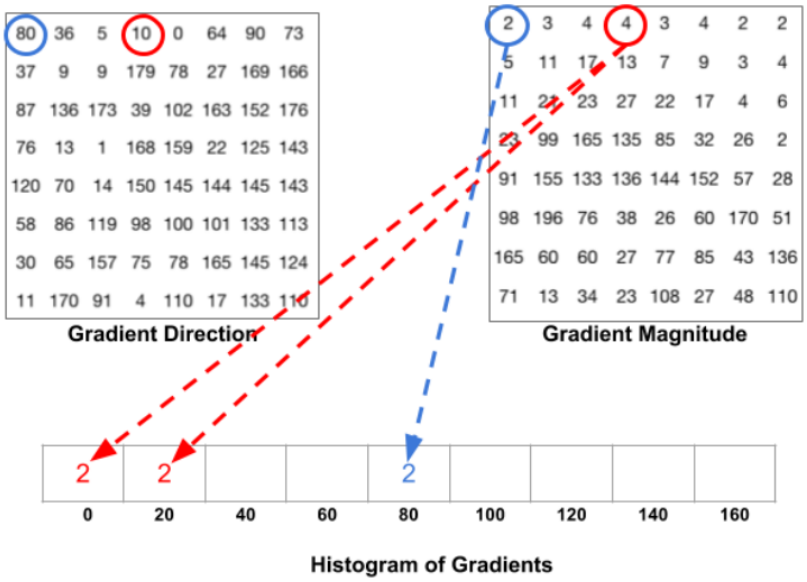
We divide photo photo into cell with size mxm pixels and calculate gradient by sliding the Sobel filter in each cell. So in each group, we get 2 matrices



In each cell, after getting two matrices, we do as follow

- Step 1: Mapping magnitude gradient into corresponding bins of gradient direction.

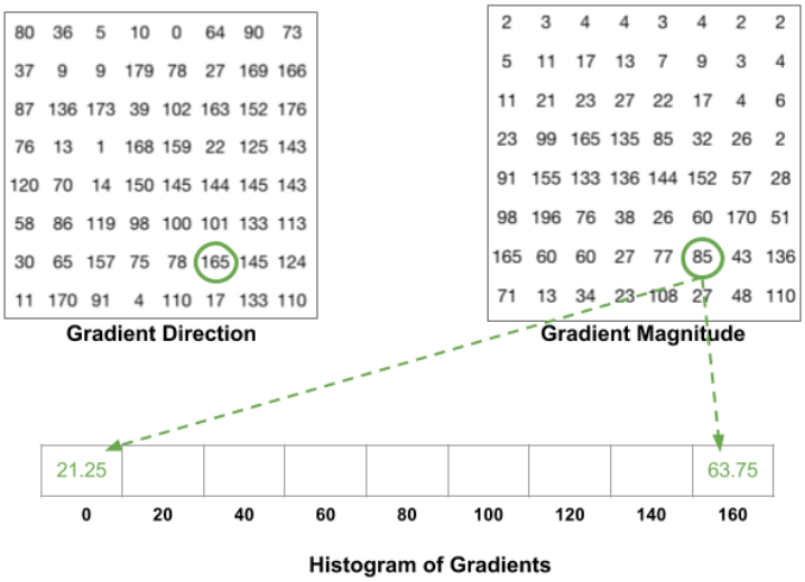
- Step 2: Organize the gradient direction in increasing order and group them into n bins. The gradient direction's magnitude will range from 0 to 180 so its bin will have 180/n length.



Each gradient direction will in pair with gradient magnitude that have same coordinate. If the gradient direction value is between the bins endpoint $[x_0, x_1]$, its corresponding gradient magnitude will be split to each one by interpolation formula.

$$x_{l-1} = \frac{(x_1 - x)}{x_1 - x_0} * y$$

$$x_l = \frac{(x - x_0)}{x_1 - x_0} * y$$



Calculate the sum all gradient magnitude in the same bin and we would get Histogram of Gradient vector.

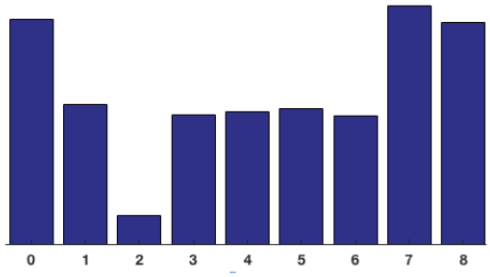


Figure 1: Train loss value during training

We can see that the histogram vector will be heavily depend on the intensity of pixels in the photo, so we will need to normalization. Group the cells into overlapping blocks of 2 × 2 cells each, so that each block has size 2m×2m pixels. Two horizontally or vertically consecutive blocks overlap by two cells, that is, the block stride is m pixels. As a consequence, each internal cell is covered by four blocks. Concatenate the four cell histograms in each block into a single block feature b and normalize the block feature by its Euclidean norm:

$$b \leftarrow \frac{b}{\sqrt{\|b\|_2^2 + \epsilon^2}}$$

After normalization, we would get the final feature vector by concatenating all the histogram of gradient vector.

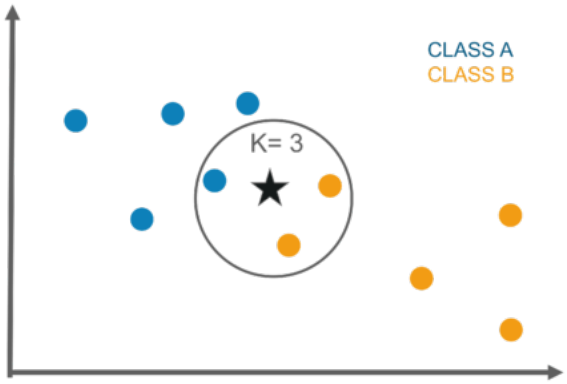


Figure 2: k = 3

2.2 Machine Learning

Machine learning (ML) is a type of artificial intelligence (AI) that allows software applications to become more accurate at predicting outcomes without being explicitly programmed to do so. Machine learning algorithms use historical data as input to predict new output values. There are two basic approaches: Supervised learning and Unsupervised learning.

Supervised learning uses a training set to teach models to yield the desired output. This training dataset includes inputs and correct outputs, which allow the model to learn over time. The algorithm measures its accuracy through the loss function, adjusting until the error has been sufficiently minimized.

Mathematical Expression: Learn a function $y = f(x)$ from a given training set $x_1, x_2, \ldots, x_N$; $y_1, y_2, \ldots, y_N$ so that $y_i \cong f(x_i)$ for every i.

Supervised learning can be separated into two types of problems—classification and regression.

2.3 K-nearest neighbors

The k-nearest neighbors algorithm, also known as KNN or k-NN, is a non-parametric, supervised learning classifier, which uses proximity to make classifications or predictions about the grouping of an individual data point. While it can be used for either regression or classification problems, it is typically used as a classification algorithm, working off the assumption that similar points can be found near one another. A new data point is classified using the K-NN algorithm based on similarity after all the existing data has been stored. This means that utilizing the K-NN method, new data can be quickly and accurately classified into a suitable category. Since K-NN is a non-parametric technique, it makes no assumptions about the underlying data. It is also known as a lazy learner algorithm since it saves the training dataset rather than learning from it immediately. Instead, it uses the dataset to perform an action when classifying data.

When tested with a new example, it looks through the training data and finds the k training examples that are closest to the new example. It then assigns the most common class label (among those k-training examples) to the test example.

In figure 1, we can see that k = 3 mean the 3 nearest neighbor points are voting for the new test data's class.

2.3.1 Data representation

Each observation is represented by a vector in an n-dimensional space, e.g., $x_i = (x_{i1}, x_{i2}, \ldots, x_{in})^T$
Each represents an *attribute/feature/variate*.
There is a set C of predefined labels.

2.3.2 Learning phase

- Simply save all the training data $\mathbb{D}$, with their labels.
- Prediction: for a new instance z.
- For each instance x in $\mathbb{D}$, compute the distance/similarity between x and z.
- Determine a set $\mathbb{NB}(z)$ of the nearest neighbors of z.
- Using majority of the labels in $\mathbb{NB}(z)$ to predict the label for z , with $|NB(z)\| = k$.
- Predict the label for z by

$$y_z = \frac{1}{k} \sum_{x \in \mathbb{NB}(z)} y_x$$

2.3.3 Distance/similarity measure

For the algorithm to work best on a particular dataset we need to choose the most appropriate distance metric accordingly. There are a lot of different distance metrics available, but we are only going to talk about a few widely used ones.

Minkowski Distance: is a distance/ similarity measurement between two points in the normed vector space (N dimensional real space) and is a generalization of the Euclidean distance and the Manhattan distance.

$$d(\vec{X}, \vec{Y}) = ((\vec{X} - \vec{Y})^p)^{\frac{1}{p}} = (\sum i = 1^n |x_i - y_i|^p)^{\frac{1}{p}}$$

Manhattan Distance: As $p = 1$, Minkowski distance is specific to Manhattan distance.

$$d(\vec{X}, \vec{Y}) = (|\vec{X} - \vec{Y}|) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{\frac{1}{p}}$$

Euclidean Distance: As p = 2, Minkowski distance is specific to Euclidean distance.

$$d(\vec{X}, \vec{Y}) = \sqrt{((\vec{X} - \vec{Y})^2)} = \sqrt{(\sum i = 1^n |x_i - y_i|^2)}$$

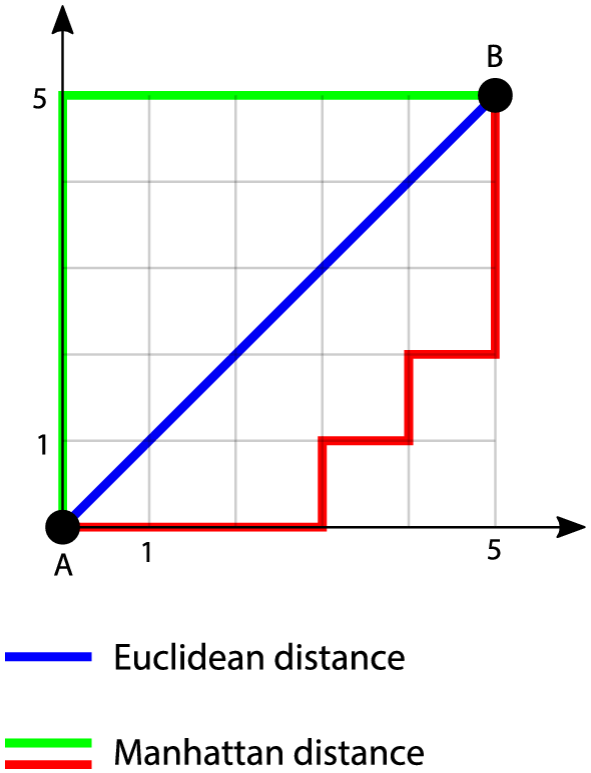


Figure 3: Manhattan Distance and Euclidean Distance

2.3.4 Weighting neighbors

Weighting the attributes is sometimes important for KNN. No weight implies that the attributes play an equal role. This is unrealistic in some applications, where an attribute might be more important than the others in prediction. Some weights (wi) on the attributes might be more suitable

2.3.5 Advantages and Limitations of KNN

- Advantages:
  - Low cost for the training phase.
  - Very flexible in choosing the distance/similarity measure : We can use many other measures.
  - KNN is able to reduce some bad effects from noises when k ¿ 1.
  - In theory, KNN can reach the best performance among all regression methods, under some conditions (this might not be true for other methods)
- Limitations:
  - Have to find a suitable distance/similarity measure for your problem.
  - Prediction requires intensive computation.

2.4 Random Forest

2.4.1 Decision Tree

A decision tree can be interpreted as a set of rules of the form: IF-THEN.

Learn a decision tree: algorithm ID3 - at each node N, select a test attribute A which can help us best do classification for the data in N.

- Advantages
  - ID3 searches for a tree that fits well with the training data.
  - ID3 just searches for only one tree
  - ID3 never backtracks
- Issue
  - The learnt trees may overfit the training data.
  - Missing or real values.
  - Is there any better measure than information gain?

2.4.2 Random Forest

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-fitting.

Main idea: prediction is based on combination of many decision trees, by taking the average of all individual predictions.

Three basic ingredients :

- Randomization and no pruning:
  - For each tree and at each node, we select randomly a subset of attributes.
  - Find the best split, and then grow appropriate subtrees.
  - Every tree will be grown to its largest size without pruning.
- Combination: each prediction later is made by taking the average of all predictions of individual trees.
- Bagging: the training set for each tree is generated by sampling (with replacement) from the original data.

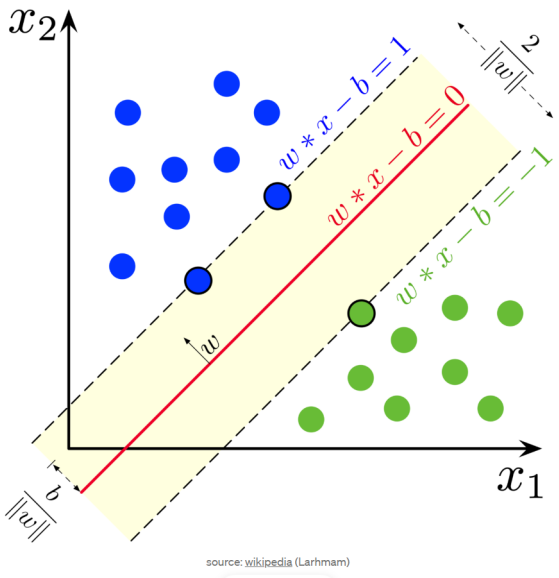
2.5 Support Vector Machine(SVM)

2.5.1 Binary SVM

The linearly separable case

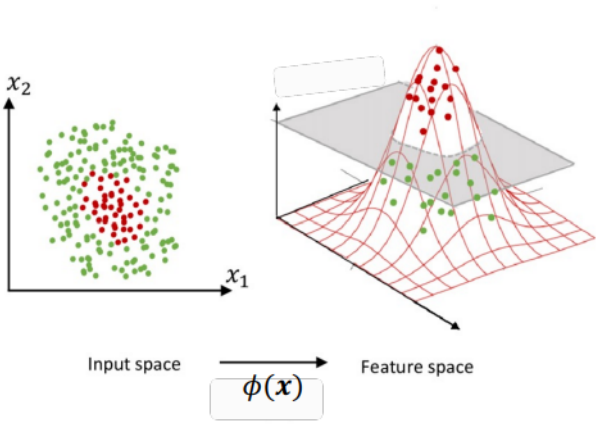
The objective is to find a hyperplane in an n-dimensional space that separates the data points to their potential classes. The hyperplane should be positioned with the maximum distance to the data points. The data points with the minimum distance to the hyperplane are called Support

Vectors.Due to their close position, their influence on the exact position of the hyperplane is bigger than of other data points. In the graphic below the Support Vectors are the 3 points (2 blue, 1 green) laying on the lines.



The non-linearly separable case

Idea of Non-linear SVM: transform the input into another space, which often has higher dimensions, so that the projection of data is linearly. Then we use linear SVM in the new space separable



Kernel Functions

The Kernel function map the input x to a new representation, using a non-linear mapping. These are some kernel functions:

1. Linear Function

$$k(x_i, x_j) = x_i * x_j$$

2. Polynomial Function

$$k(x_i, x_j) = (1 + x_i * x_j)^d$$

3. Radial Basis Function (RBF)

$$k(x_i, x_j) = exp(-\gamma \|x_i - x_j\|^2)$$

4. Sigmoid Function

$$k(x_i, x_j) = \tanh(\alpha x^T y + c)$$

2.5.2 Multiclass Classification using Support Vector Machine

In its most simple type SVM are applied on binary classification, dividing data points either in 1 or 0. For multiclass classification, the same principle is utilized. The multiclass problem is broken down to multiple binary classification cases, which is also called one-vs-one. In scikit-learn one-vs-one is not default and needs to be selected explicitly (as can be seen further down in the code). One-vs-rest is set as default. It basically divides the data points in class x and rest. Consecutively a certain class is distinguished from all other classes.

The number of classifiers necessary for one-vs-one multiclass classification is $\frac{n*(N-1)}{2}$.This technique breaks down our multiclass classification problem into subproblems which are binary classification problems. So, after this strategy, we get binary classifiers per each pair of classes. For final prediction for any input use the concept of majority voting along with the distance from the margin as its confidence criterion.

In the one-vs-one approach, each classifier separates points of two different classes and comprising all one-vs-one classifiers leads to a multiclass classifier. To predict the output for new input, just predict with each of the build SVMs and then find which one puts the prediction the farthest into the positive region (behaves as a confidence criterion for a particular SVM).

2.6 Deep Learning

Deep learning is a type of machine learning and artificial intelligence (AI) that imitates the way humans gain certain types of knowledge. Deep learning is an important element of data science, which includes statistics and predictive modeling. It is extremely beneficial to data scientists who are tasked with collecting, analyzing and interpreting large amounts of data; deep learning makes this process faster and easier.

2.6.1 Neural Networks

Neural networks, also known as artificial neural networks (ANNs) or simulated neural networks (SNNs), are a subset of machine learning and are at the heart of deep learning algorithms. Their name and structure are inspired by the human brain, mimicking the way that biological neurons signal to one another.

Artificial Neural Networks
A human brain has billions of neurons. Neurons are interconnected nerve cells in the human brain that are involved in processing and transmitting chemical and electrical signals. Dendrites are branches that receive information from other neurons.

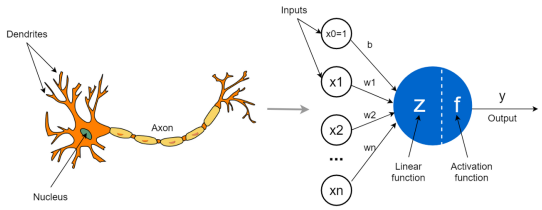


Figure 4: Biological neuron and Artificial neural network

An artificial neuron is a mathematical function based on a model of biological neurons, where each neuron takes inputs, weighs them separately, sums them up and passes this sum through a nonlinear function to produce output.

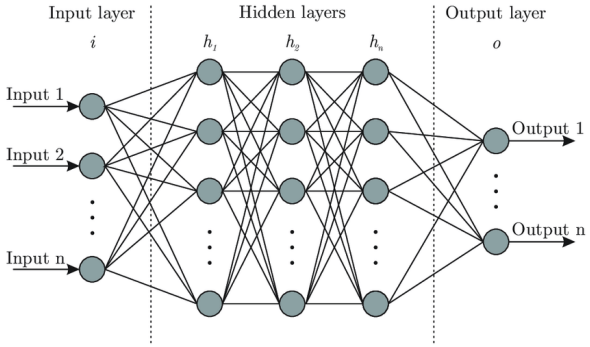


Figure 5: Artificial Neural Network architecture (Source: ResearchGate)

Artificial neural networks (ANNs) are comprised of a node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or artificial neuron, connects to another and has an associated weight and threshold. If the output of any individual node is above the specified threshold value, that node is activated, sending data to the next layer of the network. Otherwise, no data is passed along to the next layer of the network.

Forward Propagation
Neural Network is composed of the Forward propagation and Back-propagation.

In forward propagation, the input data are fed to the network in the forward direction. Each hidden layer gets the data, perform calculation and pass the result to the next layer. And Last, the output layer calculates the output of the model.

Backpropagation
Backpropagation refers to the method of calculating the gradient of neural network parameters. In short, the method traverses the network in reverse order, from the output to the input layer, according to the chain rule from calculus. The algorithm stores any intermediate variables (partial derivatives) required while calculating the gradient with respect to some parameters.

2.6.2 Convolutional Neural Network

Convolutional neural networks are distinguished from other neural networks by their superior performance with image, speech, or audio signal inputs. They have three main types of layers, which are:

- *Convolutional layer*
- *Pooling layer*
- *Fully-connected (FC) layer*

With each layer, the CNN increases in its complexity, identifying greater portions of the image. Earlier layers focus on simple features, such as colors and edges. As the image data progresses through the layers of the CNN, it starts to recognize larger elements or shapes of the object until it finally identifies the intended object.

Convolutional Layer
A convolutional layer is the main building block of a CNN. It contains a set of filters (or kernels), parameters of which are to be learned throughout the training. The size of the filters is usually smaller than the actual image. Each filter convolves with the image and creates an activation map. For convolution the filter slid across the height and width of the image and the dot product between every element of the filter and the input is calculated at every spatial position.

Convolution involving one-dimensional signals is referred to as 1D convolution or just convolution. Otherwise, if the convolution is performed between two signals spanning along two mutually perpendicular dimensions (i.e., if signals are two-dimensional in nature), then it will be referred to as 2D convolution. This concept can be extended to involve multi-dimensional signals due to which we can have multi-dimensional convolution.

Convolution 1D
Mathematical expression:

$$(f * g)(t) = \int_{\tau} f(\tau)g(t-\tau)d\tau$$

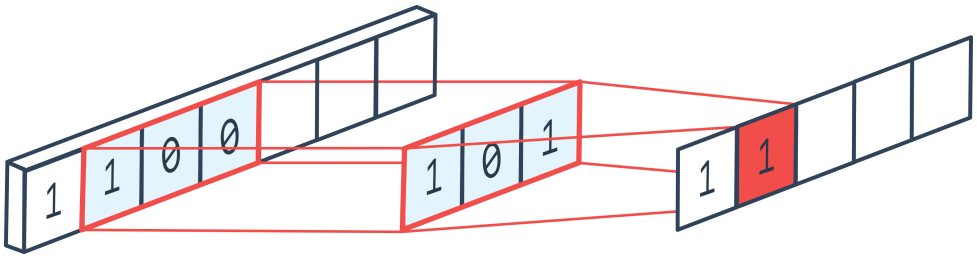


Figure 6: Example of a Conv1D layer

Convolution 2D

Mathematical expression:

$$(f * g)(u, v) = \int_{u,v} f(u, v)g(x - u, y - v)dudv = (g * f)(u, v)$$

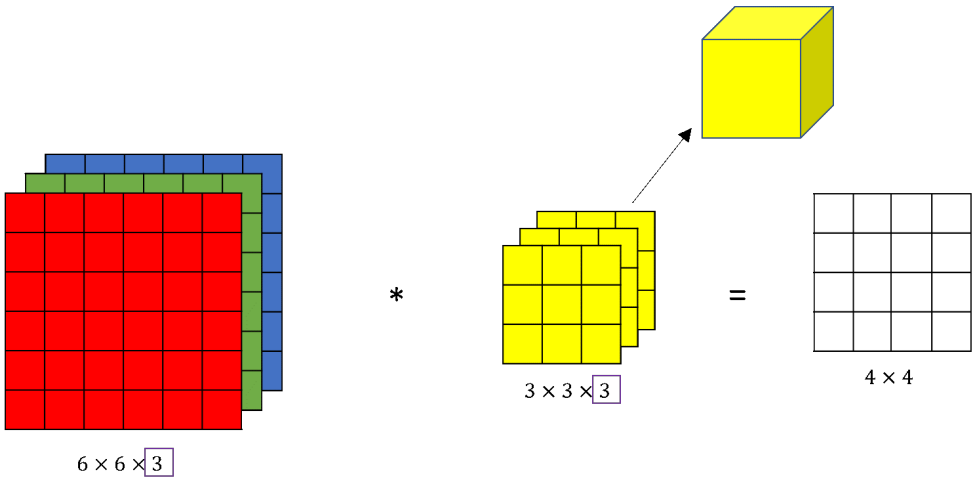


Figure 7: Math behind 2D convolution

Pooling Layer

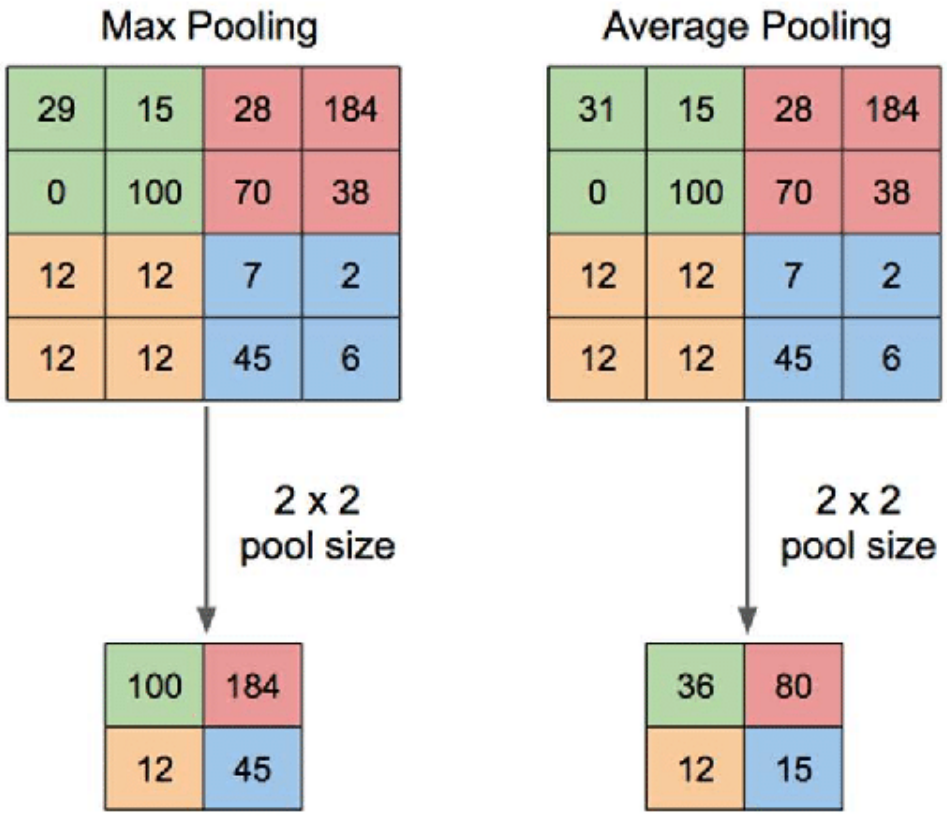


Figure 8: Max pooling and Average pooling

Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter. Thus, the output after max-pooling layer would be a feature map containing the most prominent features of the previous feature map.

Average pooling involves calculating the average for each patch of the feature map. This means that each 2×2 square of the feature map is down sampled to the average value in the square.

Batch Normalization

$$z^N = \left(\frac{z - m_z}{s_z} \right)$$

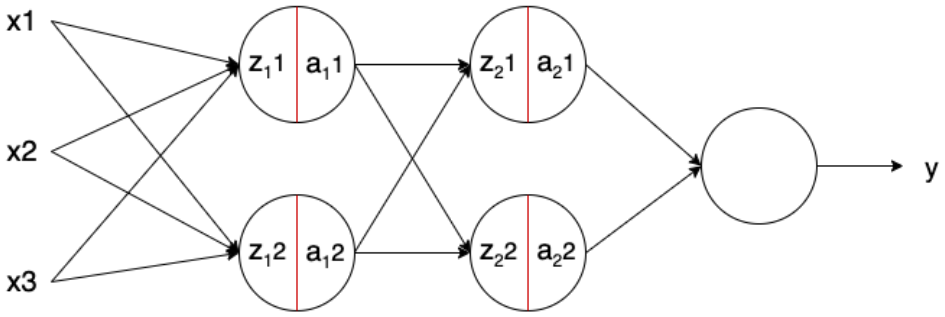
Batch Norm is a normalization technique done between the layers of a Neural Network instead of in the raw data. It is done along mini-batches instead of the full data set. It serves to speed up training and use higher learning rates, making learning easier.

Following the technique explained in the previous section, we can define the normalization formula of Batch Norm as:

$$z^N = \left(\frac{z - m_z}{s_z} \right)$$

being m_z the mean of the neurons' output and s_z the standard deviation of the neurons' output.

In the following image, we can see a regular feed-forward Neural Network: $x_i$are the inputs, z the output of the neurons, a the output of the activation functions, and y the output of the network:



$$z = g(w, x) + b$$
$$z = f(z)$$

being $g()$ the linear transformation of the neuron, w the weights of the neuron, b the bias of the neurons, and $f()$ the activation function. The model learns the parameters w and b. Adding Batch Norm, it looks as:

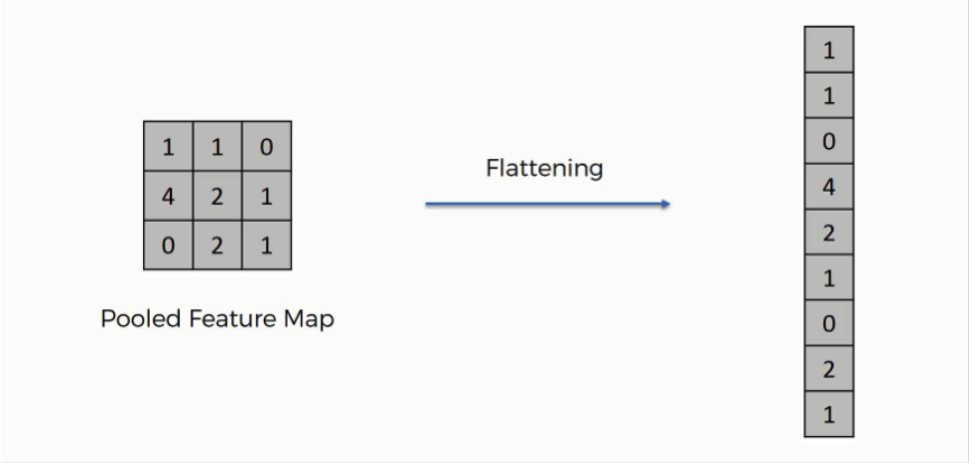
$$z = g(w, x) + b$$
$$z^N = \left(\frac{z - m_z}{s_z}\right) \cdot \gamma$$
$$\alpha = f(z^N)$$

being z^N the output of Batch Norm, m_z the mean of the neurons' output, s_z the standard deviation of the output of the neurons, and γ and β learning parameters of Batch Norm. Note that the bias of the neurons (b) is removed. This is because as we subtract the mean m_z, any constant over the values of z – such as b – can be ignored as it will be subtracted by itself.

The parameters β and γ shift the mean and standard deviation, respectively. Thus, the outputs of Batch Norm over a layer results in a distribution with a mean β and a standard deviation of γ. These values are learned over epochs and the other learning parameters, such as the weights of the neurons, aiming to decrease the loss of the model.

Flatten

After finishing the pooling layer, we're supposed to have a pooled feature map by now. As the name of this step implies, we are literally going to flatten our pooled feature map into a column like in the image below.



Dense

The dense layer is a neural network layer that is connected deeply, which means each neuron in the dense layer receives input from all neurons of its previous layer. The dense layer is found to be the most commonly used layer in the models.

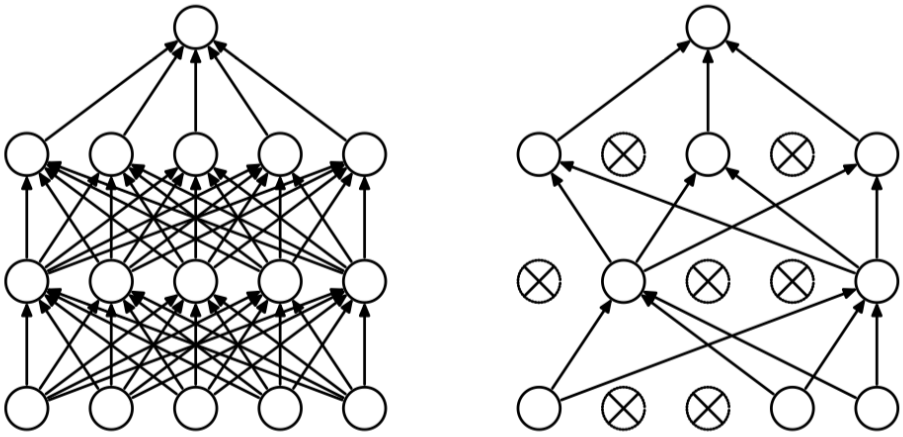
In the background, the dense layer performs a matrix-vector multiplication. The values used in the matrix are actually parameters that can be trained and updated with the help of backpropagation.

The output generated by the dense layer is an 'm' dimensional vector. Thus, dense layer is basically used for changing the dimensions of the vector. Dense layers also applies operations like rotation, scaling, translation on the vector.

Dropout

Another typical characteristic of CNNs is a Dropout layer. The Dropout layer is a mask that nullifies the contribution of some neurons towards the next layer and leaves unmodified all others. We can apply a Dropout layer to the input vector, in which case it nullifies some of its features; but we can also apply it to a hidden layer, in which case it nullifies some hidden neurons.

Dropout layers are important in training CNNs because they prevent overfitting on the training data. If they aren't present, the first batch of training samples influences the learning in a disproportionately high manner. This, in turn, would prevent the learning of features that appear only in later samples or batches:



(a) Standard Neural Net (b) After applying dropout.

2.7 Over-sampling

2.7.1 Naive random over-sampling

Random oversampling is the simplest oversampling technique to balance the imbalanced nature of the dataset. It balances the data by replicating the minority class samples. This does not cause any loss of information, but the dataset is prone to overfitting as the same information is copied.

2.7.2 SMOTE

In the case of random oversampling, it was prone to overfitting as the minority class samples are replicated, here SMOTE comes into the picture. SMOTE stands for Synthetic Minority Oversampling Technique. It creates new synthetic samples to balance the dataset.

SMOTE works by utilizing a k-nearest neighbor algorithm to create synthetic data. Steps samples are created using Smote:

- Identify the feature vector and its nearest neighbor.
- Compute the distance between the two sample points.
- Multiply the distance with a random number between 0 and 1.
- Identify a new point on the line segment at the computed distance.
- Repeat the process for identified feature vectors.

3 Method

To complete this project, we build a HOG feature descriptor with Machine Mearning algorithms and a Convolution Neural Networks model to classify traffic signs in the GTSRD dataset. After training and testing, we do some experiments by tuning the parameters and try other approach to achieve better results. This Method section will discuss:

- Exploring the dataset.
- Pre-processing the images with HOG descriptor.
- Building machine learning models.
- Building a CNN model.
- Tuning the parameters.
- Presenting the experiment results.

3.1 Some Libraries used in this project

We build and deploy our model on Jyputer Notebooks and Google Colab which are really famous and efficient environment for Data Science. They have a lot of useful libraries that can help us significanly in building and training models. Here are some libraries that we import in our code:

- numpy
- pandas
- matplotlib
- os
- cv2
- tensorflow
- sklearn

3.2 Data Overview

3.2.1 The Dataset

The German Traffic Sign Recognition Benchmark (GTSRB) contains 43 classes of traffic signs, split into 39,209 training images and 12,630 test images. The images have varying light conditions and rich backgrounds. There are 43 classes of traffic signs. As we can see from the graph below, the data is pretty imbalanced.

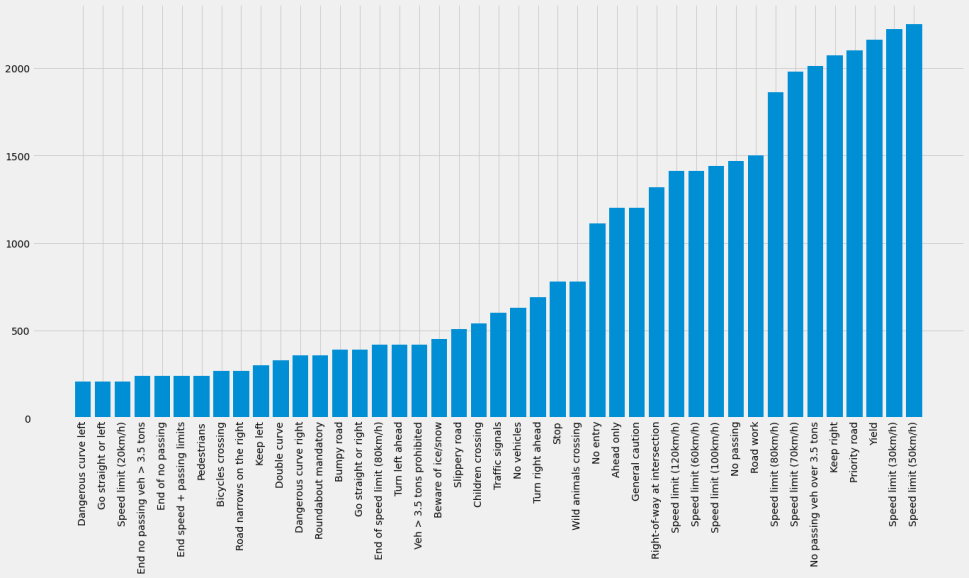


Figure 9: The number of images for each classes

3.3 Data Processing

First, all images is resized to size 40x40

3.3.1 Machine learning algorithm

Since all of our machine learning algorithm require vector input, we will apply SMOTE and HOG to get better performance and generalization

3.3.2 For Convolution Neuron Network

If we want to apply SMOTE to an image data, we need to flatten the image tensor first. As a result, the spatial information will not be preserved (which is very important for CNN) and the image will be badly augmented (We used to apply SMOTE here and get bad images- hard even for human to recognize the class of the image). Therefore, we decide not to apply SMOTE here.

So we use another augment method:

Zooming: This method randomly zooms the image either by zooming in or it adds some pixels around the image to enlarge the image

Figure 10: Some example of the images

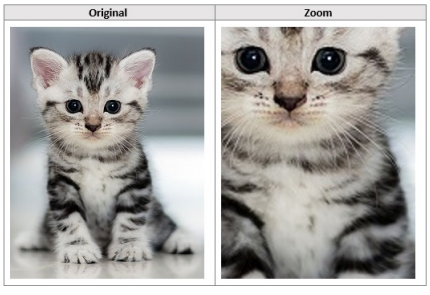


Figure 11

Shearing: The image will be distorted along an axis, mostly to create or rectify the perception angles. It's usually used to augment images so that computers can see how humans see things from different angles

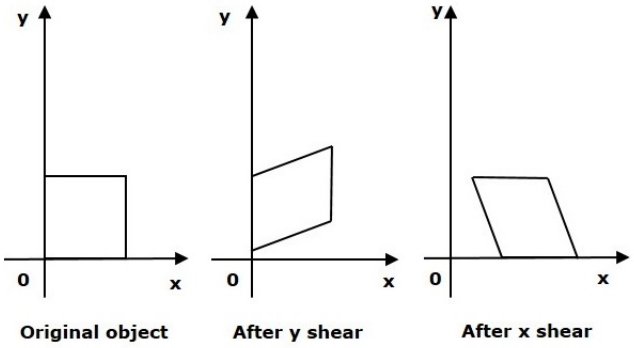


Figure 12

Width shift: shift the image to the left or right(horizontal shifts)
Height shift:It works same as width shift but shift vertically(up or down)

Figure 13

3.4 Building the Machine learning Model

3.4.1 KNN

Try KNN with different parameters:

- Numbers of neighbors(when metric = 'manhattan'): With 5 neighbors, accuracy of model is only 0.39. The accuracy is strongly increase to 0.86 with 10 neighbors. With 15 neighbors, the accuracy doesn't change too much.

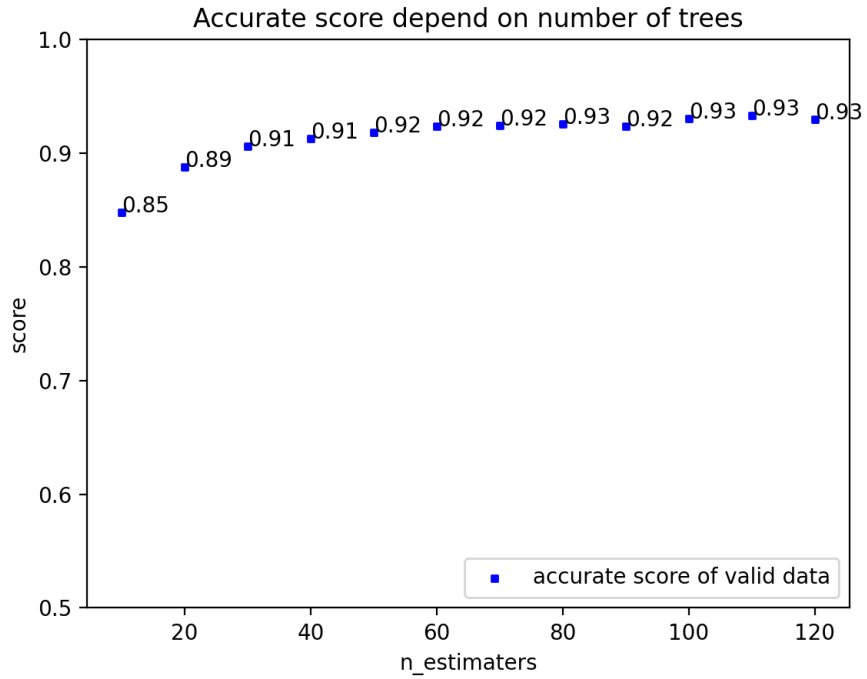
- Distance metric: Manhaatan has the highest accuracy with 0.86 (with 10 neighbors), some others like 'minkowski', 'euclidean', 'hamming' have the accuracy approximately 0.82.

- Some parameters like algorithm used to compute the nearest neighbors, weights function used in prediction does not change accuracy too much.

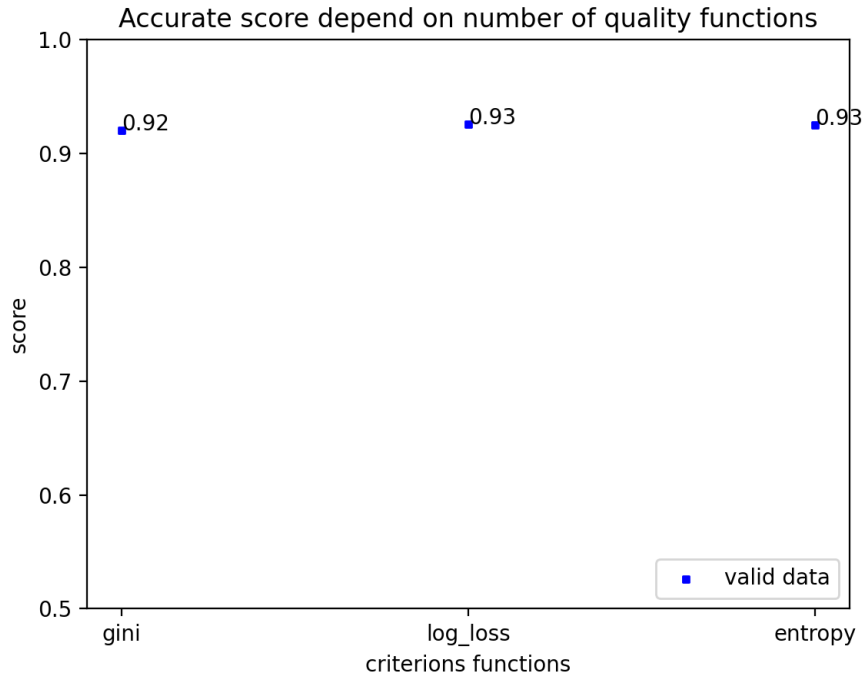
3.4.2 Random Forest Model

RandomForestClassification model from sklearn.ensemble is used with parameter:

- Parameter "n_estimators": determine number of trees grown in the forest. The value 60-80 is enough instead of 100 by default. n_estimators=80 is set.

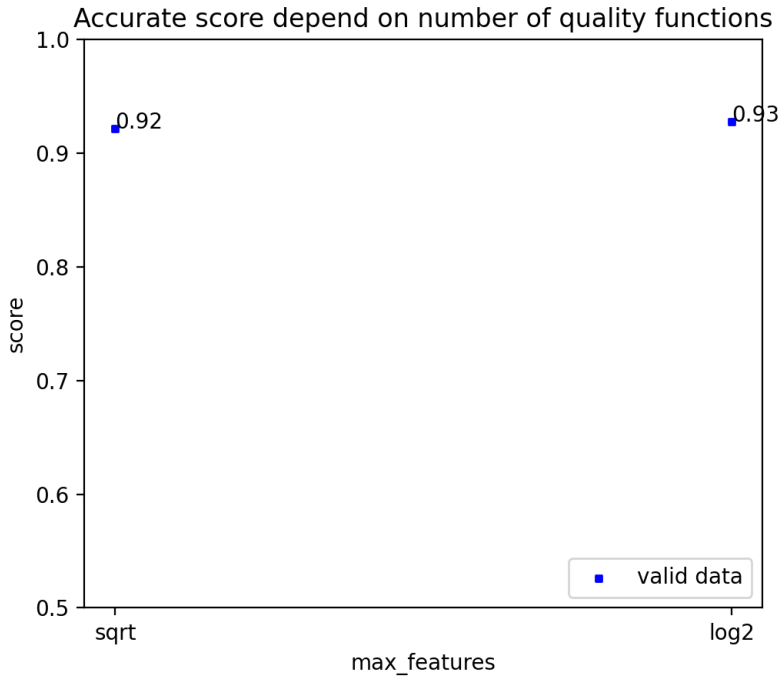


- Parameter "criterions": determine the function for measuring the quality of a split.



[H]

- Parameter "max_features": determine the number of features to consider when looking for the best split. The value "log2" is set instead of "sqrt" by default.



3.4.3 SVM

Parameter "kernel function": Try SVM with different kernel like "Poly" and "rbf" Parameter "C- regularization parameter": using GridSearch to test with C=1,7,12 and find the one with best result.

3.5 Building the CNN model

3.5.1 Model architecture

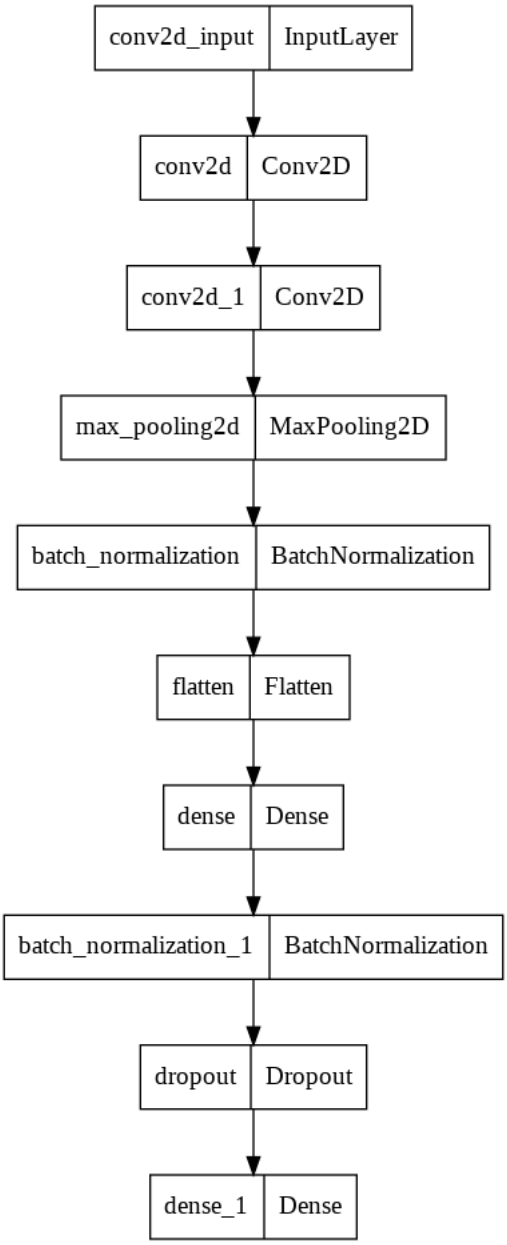


Figure 14: The CNN Model Architecture

Our CNN architecture is made up of 9 layers. The layer composition consists of 2 convolutional layers, 1 max pooling layer, 2 batch normalization layers, and 2 fully connected layers.

After building, the model has compiled with adam optimizer along with accuracy metric and a cross entropy loss.

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 38, 38, 16)        448

 conv2d_9 (Conv2D)           (None, 36, 36, 32)        4640

 max_pooling2d_4 (MaxPooling  (None, 18, 18, 32)       0
 2D)

 batch_normalization_2 (Batc  (None, 18, 18, 32)       128
 hNormalization)

 flatten_3 (Flatten)         (None, 10368)             0

 dense_6 (Dense)             (None, 512)               5308928

 batch_normalization_3 (Batc  (None, 512)              2048
 hNormalization)

 dropout_3 (Dropout)         (None, 512)               0

 dense_7 (Dense)             (None, 43)                22059

=================================================================
Total params: 5,338,251
Trainable params: 5,337,163
Non-trainable params: 1,088
```

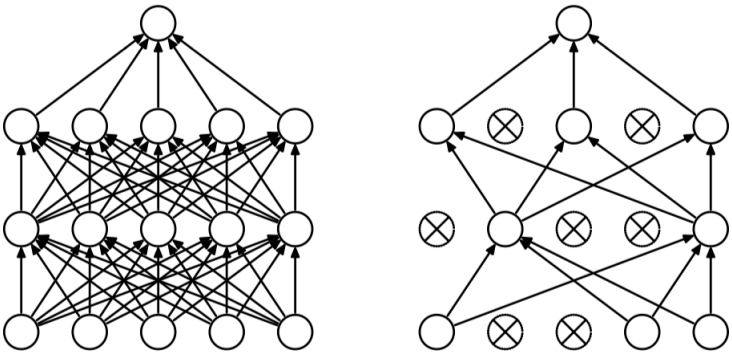
Figure 15: The CNN Model Summary

3.5.2 Learning Regularization

Drop out

Deep neural networks are arguably the most powerful machine learning models available to us today. Due to a large number of parameters, they can learn extremely complex functions. But this also makes them very prone to overfitting the training data.

Compared to other regularization methods such as weight decay, or early stopping, dropout also makes the network more robust. This is because when applying dropout, you are removing different neurons on every pass through the network. Thus, you are actually training multiple networks with different compositions of neurons and averaging their results.



(a) Standard Neural Net (b) After applying dropout.

Figure 16

3.5.3 Parameter Tuning

After training and tuning hyperparameters, we see that batchsize = 32 with optimizer "Adam" is most stable to converge.

3.6 Results and Analysis

We use 4 metrics to evaluate our method: accuracy, precision , recall and f1_score.

By training on the training dataset aned parameter tuning on valid set to get the best result on valid set. We evaluate and compare the performance of model by testing on the test dataset

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1\text{-}score = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Figure 17

The graphs show accuracy and loss of CNN

There are significant difference in performance between methods

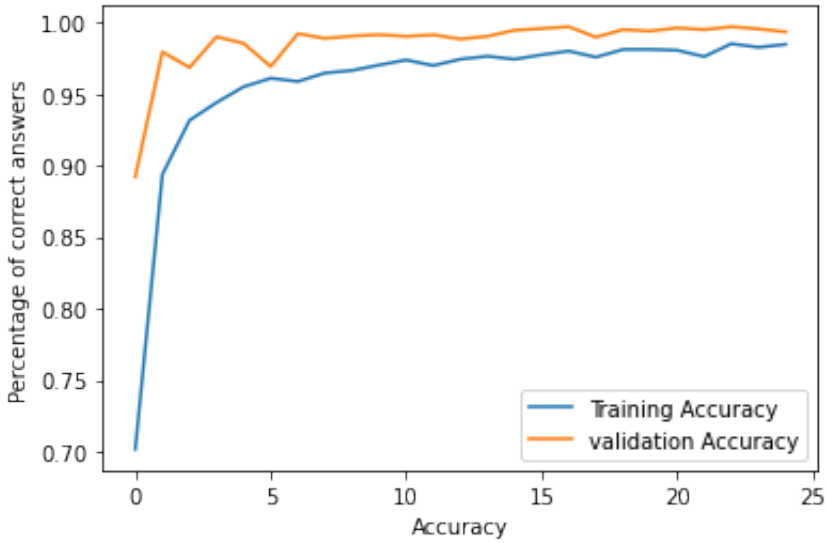


Figure 18

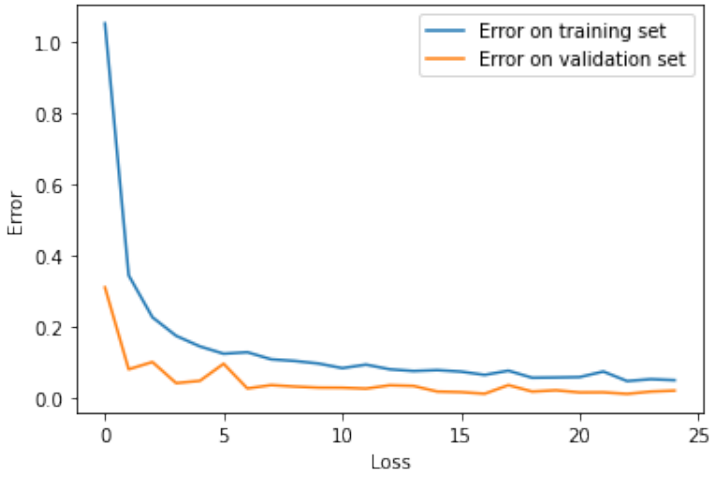


Figure 19

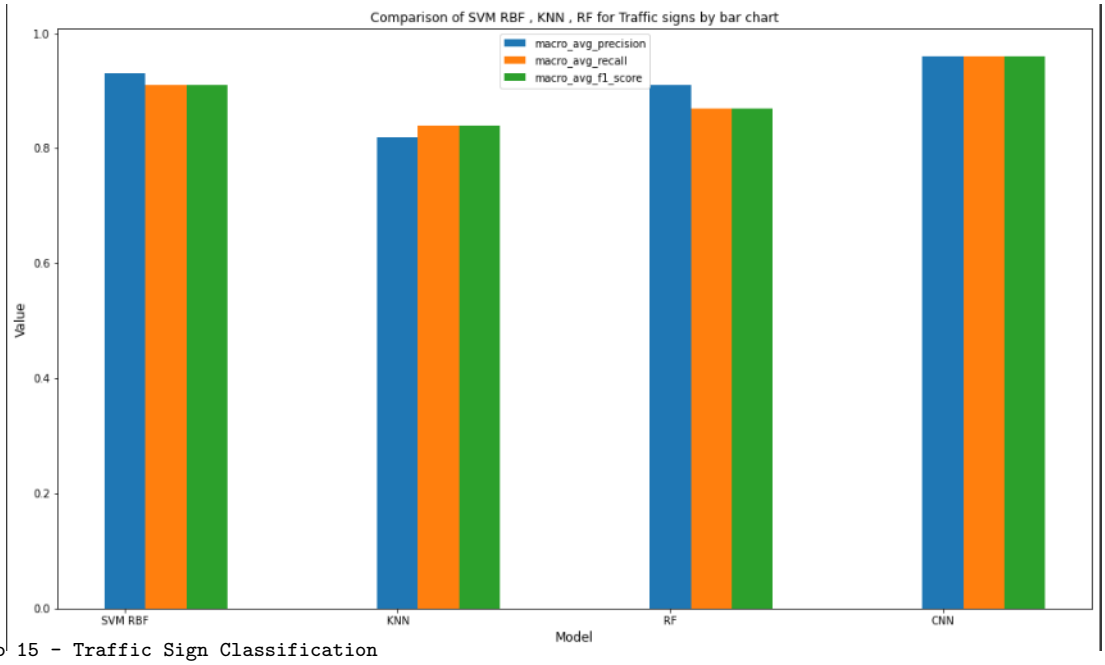


Figure 20

	precision	recall	f1-score	support
0	0.94	0.83	0.88	60
1	0.91	0.76	0.83	720
2	0.83	0.90	0.86	750
3	0.86	0.81	0.84	450
4	0.89	0.91	0.90	660
5	0.68	0.77	0.72	630
6	0.80	0.82	0.81	150
7	0.86	0.80	0.82	450
8	0.73	0.80	0.76	450
9	0.94	0.99	0.96	480
10	0.96	0.97	0.96	660
11	0.75	0.87	0.81	420
12	0.98	0.98	0.98	690
13	0.99	1.00	1.00	720
14	0.99	0.97	0.98	270
15	0.91	0.99	0.95	210
16	0.99	0.93	0.96	150
17	1.00	1.00	1.00	360
18	0.97	0.85	0.91	390
19	0.97	0.98	0.98	60
20	0.89	0.90	0.90	90
21	0.92	0.72	0.81	90
22	0.99	0.83	0.90	120
23	0.86	0.97	0.91	150
24	0.92	0.91	0.92	90
25	0.85	0.94	0.89	480
26	0.83	0.74	0.78	180
27	0.97	0.58	0.73	60
28	0.88	0.80	0.84	150
29	0.85	0.92	0.88	90
30	0.78	0.61	0.68	150
31	0.92	0.97	0.95	270
32	0.97	0.57	0.72	60
33	0.93	1.00	0.96	210
34	0.97	0.99	0.98	120
35	0.94	0.96	0.95	390
36	0.98	0.97	0.97	120
37	1.00	0.95	0.97	60
38	0.97	0.99	0.98	690
39	1.00	1.00	1.00	90
40	0.93	0.87	0.90	90
41	0.96	0.73	0.83	60
42	0.95	0.69	0.80	90
accuracy			0.90	12630
macro avg	0.91	0.87	0.89	12630
weighted avg	0.90	0.90	0.90	12630

Figure 21: Random Forest

	precision	recall	f1-score	support
0	0.71	0.73	0.72	60
1	0.95	0.63	0.76	720
2	0.92	0.68	0.78	750
3	0.71	0.83	0.76	450
4	0.94	0.87	0.90	660
5	0.54	0.74	0.63	630
6	0.91	0.79	0.85	150
7	0.69	0.76	0.72	450
8	0.60	0.80	0.68	450
9	0.93	0.98	0.95	480
10	0.99	0.93	0.96	660
11	0.86	0.45	0.59	420
12	1.00	1.00	1.00	690
13	1.00	1.00	1.00	720
14	1.00	1.00	1.00	270
15	0.94	1.00	0.97	210
16	0.80	0.97	0.88	150
17	1.00	0.99	1.00	360
18	0.92	0.81	0.86	390
19	0.63	0.97	0.76	60
20	0.69	0.92	0.79	90
21	0.70	0.77	0.73	90
22	0.81	0.67	0.73	120
23	0.78	0.65	0.71	150
24	0.69	0.84	0.76	90
25	0.88	0.62	0.73	480
26	0.41	0.60	0.49	180
27	0.42	0.95	0.58	60
28	0.48	0.65	0.55	150
29	0.46	0.86	0.60	90
30	0.40	0.47	0.43	150
31	0.89	0.97	0.93	270
32	0.97	1.00	0.98	60
33	1.00	1.00	1.00	210
34	0.97	1.00	0.98	120
35	0.99	0.95	0.97	390
36	0.99	0.98	0.99	120
37	0.98	0.95	0.97	60
38	1.00	0.99	1.00	690
39	0.99	1.00	0.99	90
40	0.93	0.94	0.94	90
41	1.00	0.45	0.62	60
42	0.81	0.92	0.86	90
accuracy			0.84	12630
macro avg	0.82	0.84	0.82	12630
weighted avg	0.86	0.84	0.84	12630

Figure 22: KNN

	precision	recall	f1-score	support
0	0.98	0.85	0.91	60
1	0.86	0.90	0.88	720
2	0.86	0.93	0.89	750
3	0.92	0.84	0.88	450
4	0.96	0.94	0.95	660
5	0.77	0.84	0.80	630
6	0.92	0.76	0.83	150
7	0.92	0.89	0.90	450
8	0.91	0.85	0.88	450
9	0.96	0.99	0.98	480
10	0.99	0.98	0.99	660
11	0.81	0.86	0.84	420
12	1.00	1.00	1.00	690
13	1.00	1.00	1.00	720
14	0.99	0.99	0.99	270
15	1.00	1.00	1.00	210
16	1.00	0.97	0.98	150
17	1.00	1.00	1.00	360
18	0.97	0.90	0.93	390
19	0.95	0.97	0.96	60
20	0.97	0.99	0.98	90
21	0.91	0.76	0.82	90
22	1.00	0.74	0.85	120
23	0.99	0.93	0.96	150
24	0.82	0.94	0.88	90
25	0.85	0.90	0.88	480
26	0.86	0.73	0.79	180
27	0.85	0.85	0.85	60
28	0.85	0.87	0.86	150
29	0.76	0.93	0.84	90
30	0.66	0.63	0.65	150
31	0.93	0.99	0.96	270
32	1.00	1.00	1.00	60
33	1.00	1.00	1.00	210
34	1.00	0.99	1.00	120
35	0.98	0.97	0.98	390
36	0.99	0.99	0.99	120
37	1.00	0.95	0.97	60
38	0.99	1.00	1.00	690
39	0.99	1.00	0.99	90
40	0.94	0.93	0.94	90
41	0.97	0.65	0.78	60
42	0.93	0.83	0.88	90
accuracy			0.93	12630
macro avg	0.93	0.91	0.92	12630
weighted avg	0.93	0.93	0.93	12630

Figure 23: SVM with rbf kernel

	precision	recall	f1-score	support
0	1.00	0.85	0.92	60
1	0.95	0.99	0.97	720
2	1.00	0.96	0.98	750
3	0.98	0.91	0.95	450
4	0.99	0.98	0.99	660
5	0.90	0.99	0.94	630
6	1.00	0.86	0.92	150
7	0.99	0.99	0.99	450
8	0.98	0.96	0.97	450
9	0.99	1.00	0.99	480
10	1.00	1.00	1.00	660
11	0.92	1.00	0.96	420
12	1.00	0.98	0.99	690
13	1.00	1.00	1.00	720
14	0.99	1.00	0.99	270
15	0.98	1.00	0.99	210
16	0.98	0.99	0.99	150
17	0.99	1.00	0.99	360
18	0.99	0.91	0.95	390
19	0.80	0.98	0.88	60
20	0.81	1.00	0.90	90
21	0.95	1.00	0.97	90
22	0.99	0.78	0.87	120
23	0.88	0.93	0.91	150
24	1.00	0.92	0.96	90
25	0.97	0.97	0.97	480
26	0.97	0.97	0.97	180
27	0.87	0.98	0.92	60
28	0.98	0.97	0.97	150
29	0.88	0.99	0.93	90
30	0.96	0.68	0.80	150
31	0.99	0.97	0.98	270
32	0.98	1.00	0.99	60
33	0.99	1.00	0.99	210
34	0.99	1.00	1.00	120
35	0.99	0.98	0.99	390
36	0.96	0.93	0.94	120
37	1.00	0.98	0.99	60
38	0.98	1.00	0.99	690
39	0.92	0.96	0.94	90
40	0.94	1.00	0.97	90
41	1.00	0.98	0.99	60
42	0.83	1.00	0.90	90
accuracy			0.97	12630
macro avg	0.96	0.96	0.96	12630
weighted avg	0.97	0.97	0.97	12630

Figure 24: CNN

We can see that the deep learning method CNN outperformed other tradition method. Furthermore, condidering in machine learning method, SVM is slightly better than others. After training and testing, we see that the predicting process of tradition method cost much more time than CNN.

4 Conclusion

So far, we have introduced 4 different algorithms to predict the problem: K-Nearest Neighbors, Random Forest, Support Vector Machine and Convolutional Neural Network. Despite some difficulties in understanding algorithms, all 4 models perform well predictions on both test and valid dataset. The most effective algorithm is CNN as shown on Results and Analysis.

The well performances also depend on the prepared dataset, so we need to study more to gain more effective models when dealing with real data