TRAFFIC HOARDINGS RECOGNITION

Abstract

Our project is to present an effective classification method for traffic signs on the basis of a Convolutional Neural Network with various dimension filters. Every model of convolutional neural network has the same architecture but different dimension of filters for convolutional layer. The studied dimensions of convolution layer filters are 3x3, 5x5, 7x7, 9x9, 11x11, 13x13, 15x15 and 19x19. The dimensions of the filters were varied from 1-15 pixels in accordance with which the input image is convolved with certain processing depth of image borders and dimensions. The complete dataset is divided into training, validation, and testing. The variation of classification accuracy and average processing time depends solely on dimension of convolutional layer filters. The highest validation accuracies for filters 5x5, 7x7 and 9x9 and their respective accuracies 0.962, 0.940 and 0.939 were recorded and average processing time for filters 7x7, 9x9, 5x5 are 0.061, 0.066, 0.0712(s) were recorded. The middle dimension filters have proved to be a reliable filter marking a high validation accuracy and rapid classification rate making them compatible for real time applications.

Main Objective

1) To predict the traffic hoardings using Convolution Neural Network
2) CNN Filters' dimensional efficacy impact on traffic hoarding classification.
3) Determination of testing accuracy and average classification time of each filter.
4) Implementation on basis of a convolution neural network with various dimension filters.

Major Stumbling Blocks

1) The experiments were carried out on 64-bit lap with AMD processor with 4 cores and 4GB of RAM.
2) Large data set (around 6.80GB) was processed and it was used to train the model for around 11 hours initially, that resulted in an execution time out error frequently due to time limit of GPU in google colab.
3) Optimized the algorithm to run efficiently in 2 hour 15 minutes.

Introduction

Contravention of road traffic occurs majorly with the speed limit. Hence categorization of traffic speed limit hoardings is essential on grounds of safety. In built automobile systems that have the capability and credibility to compare the current speed and the allowable speed on action of recognizing the sign board have been invented by global marketeers. This also has many stumbling blocks depending on weather and night hours. They complicate the classification, accuracy and processing. Research and indagation in this field have been developing and is growing at an increasing rate. The sign classification under non ideal scenarios like vibration, angle of elevation and the illuminance of light might be misconstrued. These flaws have been reasonably eliminated by convolutional neural networks. The main lead of CNN is that it is invariant with shape, rotation and intensity of the input images. This research considers the variation of dimensions of CNN filters and their pertinent effect on average processing time and classification accuracies.

Application of CNN in Traffic Hoarding Organisation

Convolutional Neural Network is grouped under Deep Learning, classification using CNN is a model of design recognition in computer vision. The methodology of working involves an input image being processed with the help of trainable filters. The kernel of this process is the bit wise augmentation of pixels and image area. The output is so obtained. The eccentricity of the convolution layer filters is that they detect similar features in different paths in the image. The data set being divided into three subsets namely training, testing and validation simultaneously conserve the image proportion for each class.

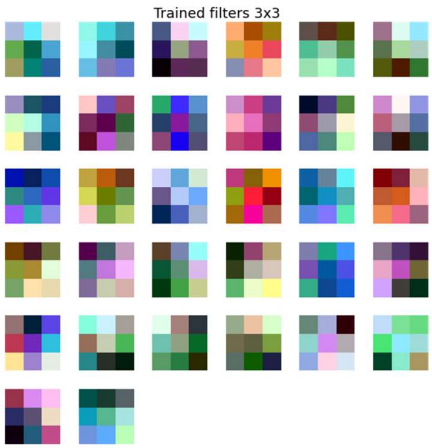
There are four main operations which are the basic blocks of Convolutional Neural Network.

1) Convolution.
2) Activation function.
3) Pooling or Sub Sampling.
4) Classification.

1) Convolution

Convolutional layer comes from the convolutional operator. The aim of this layer is to extract the features from the image. Convolution conserves the spatial relationship between pixels by learning image features using small squares of input data. Various filters are used by each convolutional layer to detect the features accurately and to extract precise amount of Edge detection, Sharpening, Blur etc. After sliding the filter through the image, we get a matrix known as the feature map.

Example of a convolutional layer filter with dimension 3x3 looks like below shown image learning to extract features from the training image dataset, develops in to feature map as shown below.



In practice, a CNN learns the values of these filters on its own during the training process. Parameters such as number of filters, filter size and network architecture are specified before launching the training process. More number of filters will always result in better network which extract good number of features to classify the images.

The size of the feature map is controlled by three parameters which are:

1) Depth: Depth corresponds to the number of filters used for the convolution operation.

2) Stride: Stride is the number of pixels by which we slide our filter matrix over the input matrix. Having a larger stride will produce smaller feature maps.

3) Zero-padding: To apply the filter to bordering elements of input image, it is convenient to pad the input matrix with zeros around the border.

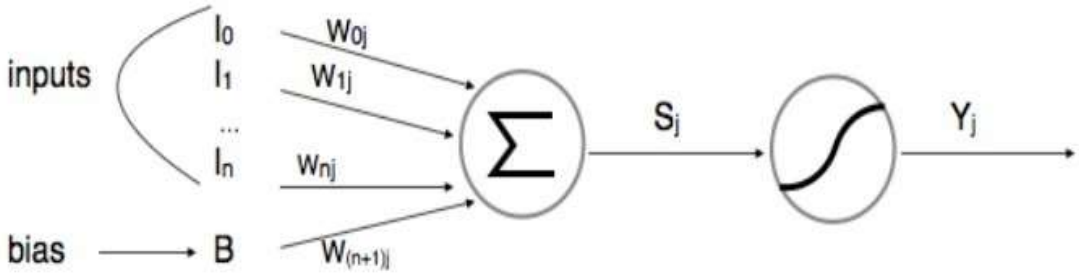
Mathematical representation of CNN operation.

$$(f \cdot g)[m, n] = \Sigma_{k,l} f[m-k, n-l] \cdot g[k, l]$$

f - input image matrix
g - filter for convolution
m - height of feature map
n - width of feature map
k - height of the filter
l - width of filter

2) Activation Function

Once the convolution has been completed, an activation function is applied to all values in the filtered image to extract nonlinear features.



$$s_k^{(l)} = f(\sigma_k^{(l)})$$

with the activation function and $\sigma_k^{(l)}$ the value of neuron k of layer l. The choice of the activation function may depend on the problem. The ReLU function replaces all negative pixel values in the feature map by zero. The purpose of ReLU is to introduce non-linearity in the CNN, since the convolution operator is linear operation and most of the CNN input data would be non-linear. Result of convolution and ReLU operation is called rectified feature map.

3) Pooling

The pooling step is also called subsampling or down sampling. It aims to reduce the dimensionality of each rectified feature map and retains the most important information.

The two most used methods to apply in this operation are the average or max pooling.

$$I_k^{(l)} = pool(s_k^{(l)})$$

The advantages of the pooling function are:

- It makes the input representations (feature dimension) smaller and more manageable.
- It reduces the number of parameters and computations in the network, therefore, controlling overfitting.
- It makes the network invariant to small transformations, distortions, and translations in the input image. In fact, a small distortion in input will not change the output of pooling since the maximum/average value in a local neighbourhood is taken.
- It helps getting an almost scale invariant representation of input image. This is very powerful since we can detect objects in an image no matter where they are located.

4) Fully Connected layer

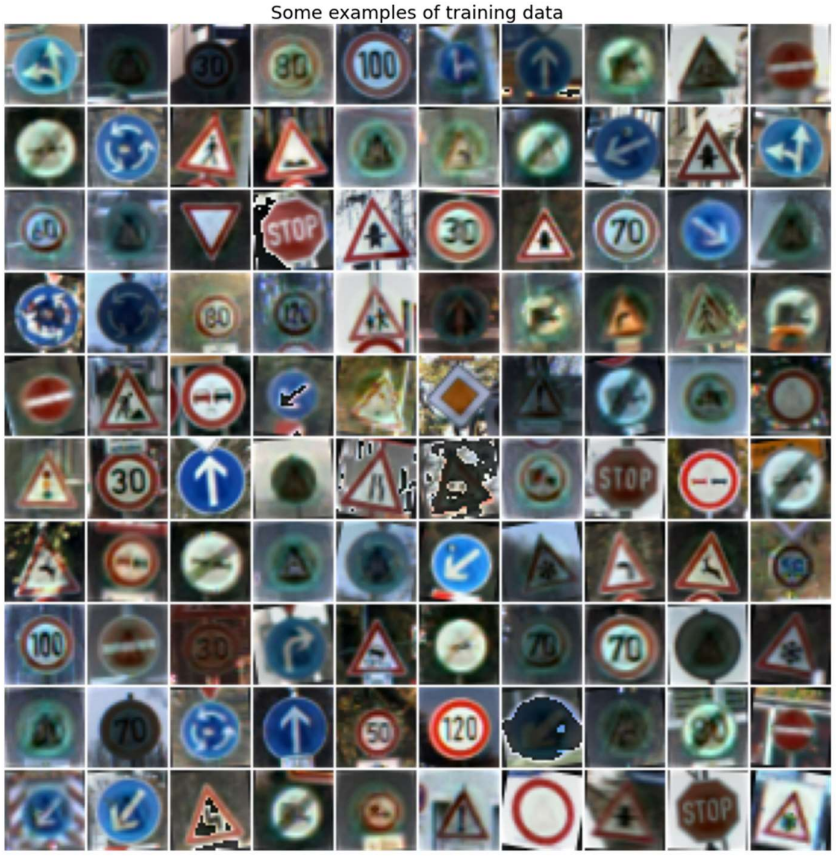
The output from the convolutional and pooling layers of a CNN is the image features vector. The purpose of the fully connected layer is to use these features vector for classifying the input images into several classes based on a labelled training dataset.

The fully connected layer is composed of two parts. The first part consists of layers so-called fully connected layers where all its neurons relate to all the neurons of the previous and next layers. In fact, CNNs seek to optimize some objective function, specifically the loss function. The well-used loss function is the SoftMax function. It normalizes the results and produces a probability distribution between the different classes (each class will have a value in the range [0, 1]). Adding a fully connected layer allows learning non-linear combinations of extracted features which might be even better for the classification task.

From the training data set the image normalization was performed by dividing with 255 and subtracting the mean image. A three channelled RGB image was prepared. number of images:

Training subset-86,989
Testing subset-12,630
Validation subset-4,410

Some images of Training dataset:



The input image obtained from the three channelled RGB is fed to the convolution layer. The three channels of every input image
makes it inevitable for the layer filters to contain three channels as well. The convolution results in 32 feature maps. These feature maps are construed in sync with the filters. Relu activation function is applied to obtained feature maps thereby eradicating negative values and restoring zeroes instead of them. This is succeeded by a layer of dimension reduction and followed by a hidden layer of neurons set by keras tuner.

Table 1: **Parameters of developed CNN**

Parameters	Description
Weight update policy	adam
Activation function	relu
Weights initialization	HE normal
Loss function	Negative log-likelihood
Cost function	Average of loss functions

The training process in CNN diminishes the cost function by backpropagation or gradient descent method. The cost function theoretically states the mean of loss functions of overall present training batch. Mathematically the cost function is:

$$J(w,b)=\frac{1}{m}\sum_{i=1}^{m} l(r^{(i)},y^{(i)}),$$

$$j(w,b)=-\frac{1}{m}\sum_{i=1}^{m}[y^{(i)}\cdot \ln r^{(i)}+(1+y^{(i)})\cdot \ln(1-r^{(i)})],$$

w-weight of output fully connected layer,
b-bias of output fully connected layer,
m-number of iterations.

Mathematical expression for negative log-likelihood function:

$$L(r, y)= -[y\cdot \ln r + (1-y)\cdot \ln(1-r)]$$

r-obtained probability with CNN.
y-True probability.

Zero frame parameter is originated around the input image before it being transmitted into the convolution layer. Mathematical equation for zero frame parameter is:

$$pad = \frac{d-1}{2},$$

d-dimension of filter

The zero-frame parameter is eminent because of the dependency of CNN on validation accuracy and processing time. This parameter increases with increase in the dimension of layer filter and the depth of image borders will also increase. This reduces the scope of missing any important information on the periphery of image.

Hyperparameters

Hyperparameters are the variables which determines the network structure (Example: Number of Hidden Units) and the variables which determine how the network is trained (Example: Learning Rate).

Hyperparameters are set before training (before optimizing the weights and bias).

Hyperparameter Tuning

Tuning hyperparameters for deep neural network is difficult as it is slow to train a deep neural network and there are numerous parameters to configure.

1) Learning Rate

Learning rate controls how much to update the weight in the optimization algorithm. We can use fixed learning rate, gradually

decreasing learning rate, momentum-based methods, or adaptive learning rates, depending on our choice of optimizer such as SGD, Adam, Adagrad, AdaDelta or RMSProp.

2) Number of Epochs

Number of epochs is the number of times the entire training set pass through the neural network. We should increase the number of epochs until we see a small gap between the test error and the training error.

3) Batch Size

Mini batch is usually preferable in the learning process of convnet. A range of 16 to 128 is a good choice to test with. We should note that convnet is sensitive to batch size.

4) Activation Function

Activation function introduces non-linearity to the model. Usually, rectifier works well with convnet. Other alternatives are sigmoid, tanh and other activation functions depending on the task.

5) Number of hidden layers and units

It is usually good to add more layers until the test error no longer improves. The trade-off is that it is computationally expensive to train the network. Having a small number of units may lead to underfitting while having more units are usually not harmful with appropriate regularization.

6) Weight Initialization

We should initialize the weights with small random numbers to prevent dead neurons, but not too small to avoid zero gradient. Uniform distribution usually works well.

7) Dropout for Regularization

Dropout is a preferable regularization technique to avoid overfitting in deep neural networks. The method simply drops out units in neural network according to the desired probability. A default value of 0.5 is a good choice to test with.

8) Grid search or Randomized search

Manually tuning hyperparameter is painful and impractical. There are two generic approaches to sampling search candidates. Grid search exhaustively search all parameter combinations for given values. Random search sample a given number of candidates from a parameter space with a specified distribution.

To implement grid search more efficiently, it is better to start with coarse ranges of hyperparameter values in the initial stage. It is also helpful to perform coarse grid search for smaller number of epochs or smaller training set. The next stage would then perform a narrow search with more epochs or entire training set.

Although grid search is useful in many machine learning algorithms, it is not efficient in tuning hyperparameter for deep neural network as the number of parameters increases, computation grows exponentially. Random search has been found more efficient compared to grid search in hyperparameter tuning for deep neural network. It is also helpful to combine with some manual tuning on hyperparameters based on prior experience.

Proposed Approach-Building CNN Model with Keras Tuner

The Keras Tuner is a library that helps you pick the optimal set of hyperparameters for your TensorFlow program. The process of selecting the right set of hyperparameters for your machine learning (ML) application is called hyperparameter tuning or hyper tuning.

Defining the Model

When you build a model for hyper tuning, you also define the hyperparameter search space in addition to the model architecture. The model you set up for hyper tuning is called a hyper model.

You can define a hyper model through two approaches:

1) By using a model builder function.
2) By subclassing the Hyper Model class of the Keras Tuner.

Here, we implemented our custom-made function (MyHyperModel)

Hyperparameter Tuning

We define a model building function to perform hyperparameter tuning for a four-layer dense neural network. It takes an argument 'hp' from which we can sample hyperparameters such as number of neurons, filter dimensions, learning rate in the fixed range, here we defined our function 'MyHyperModel' (Hypermodel sub-class).

In 'MyhyperModel', we have passed the filter size as the parameter to build function (self.classes) to implement build (self,hp) Model which is used to tune the learning rate and number of neurons in dense layer. The model type that we will be using is Sequential. Sequential is the easiest way to build a model in Keras. It allows you to build a model layer by layer. We use the 'add()' function to add layers to our model. Our first 2 layers are Conv2D layers. These are convolution layers that will deal with our input images, which are seen as 2-dimensional matrices. 64 in the first layer and 32 in the second layer are the number of nodes in each layer.

Kernel size (filter size) is the size of the filter matrix for our convolution. So, a kernel size of 3 means we will have a 3x3 filter matrix. We trained and tested our model for 3x3, 5x5, 7x7, 9x9, 11x11,

13x13, 15x15 and 19x19 filter dimensions to get the impact of various dimension filters on classification of traffic signs.

Activation is the activation function for the layer. The activation function we will be using for our first 2 layers is the ReLU, or Rectified Linear Activation.

In between the Conv2D layers and the dense layer, there is a 'Flatten' layer. Flatten serves as a connection between the convolution and dense layers. 'Dense' is the layer type we will use in for our output layer. Dense is a standard layer type that is used in many cases for neural networks. We will have 43 nodes in our output layer, one for each possible outcome (0–43).

The activation is 'softmax'. Softmax makes the output sum up to 1 so the output can be interpreted as probabilities. The model will then make its prediction based on which option has the highest probability.

We need to compile our model. Compiling the model takes three parameters: optimizer, loss and metrics. The optimizer controls the learning rate. We will be using 'adam' as our optimizer. The adam optimizer adjusts the learning rate throughout training. The learning rate determines how fast the optimal weights for the model are calculated. A smaller learning rate may lead to more accurate weights (up to a certain point), but the time it takes to compute the weights will be longer.

We will use 'categorical_crossentropy' for our loss function. This is the most common choice for classification. A lower score(loss) indicates that the model is performing better. To make things even easier to interpret, we will use the 'accuracy' metric to see the accuracy score on the validation set when we train the model.

Tuner Search

Next, let's instantiate a tuner. You should specify the model-building function, the name of the objective to optimize (whether to minimize or maximize is automatically inferred for built-in metrics), the total number of trials (max_trails) to test. In our model we have used max_trails =5. We use the overwrite argument to control whether to overwrite the previous results in the same directory or resume the previous search instead. Here we set overwrite=True to start a new search and ignore any previous results. We have used the RandomSearch tuner for our model.

Then, start the search for the best hyperparameter configuration. The call to search has the same signature as model.fit(). When search is over, you can retrieve the best model(s), Here's what happens in search: models are built iteratively by calling the model-building function, which populates the hyperparameter space (search space) tracked by the hp object. The tuner progressively explores the space, recording metrics for each configuration.

Results of the Experiment

The discrete filters like 3x3, 5x5, 7x7, 9x9, 11x11, 13x13, 15x15 and 19x19 were used to perform the training of convolutional neural network culminating in training of eight models with Keras hyperparameter tuner model and divergent filter dimensions.
The whole training process incorporated finding five models to choose the best model out of the five models (max_trails = 5) by Keras hyperparameter tuner and each model has undergone eight epochs. After every consecutive epoch-accuracies for both training and validated data set were calculated. The accuracies were updated regularly on encountering epochs with greater values, finally approaching to a state of better parameters at the end of training process.

Around twelve thousand images did not take part in the training process. The testing process for this model is forth. Twelve thousand six hundred images were passed as input for testing and result was entered in a vector. The so obtained values were compared to the true classes. The vector obtained forms an amalgamation of twelve thousand class members of traffic signs. The result is converted into accuracy between 0 and 1. The same process can be applied to any of the eight stated models with their own filtered dimensions. The described process is applied to any and all 8 models with their own filter dimension.

The best result in accordance with validation accuracy = 0.96213 and Testing Accuracy = 0.94212 is: As it is clear from below Table 2 the highest classification accuracy in validation and testing was shown by the model with convolutional layer filters of dimension equal to
5 × 5 pixels

The best result in accordance with processing time is: From Table 3 the highest classification rate = 0.06178 sec, was shown by the model with convolutional layer filters of dimension equal to 7 x 7 pixels.

Let's predict one image from the testing dataset that is obtained with the best model with convolutional layer filters of dimension equal to 5x5 pixels: we are going to test the below image from test dataset, which looks too dull and blurry, depicting a board with "50" on it.

The result from the model is: Model has predicted correctly.

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
(1, 32, 32, 3)
[2]
```

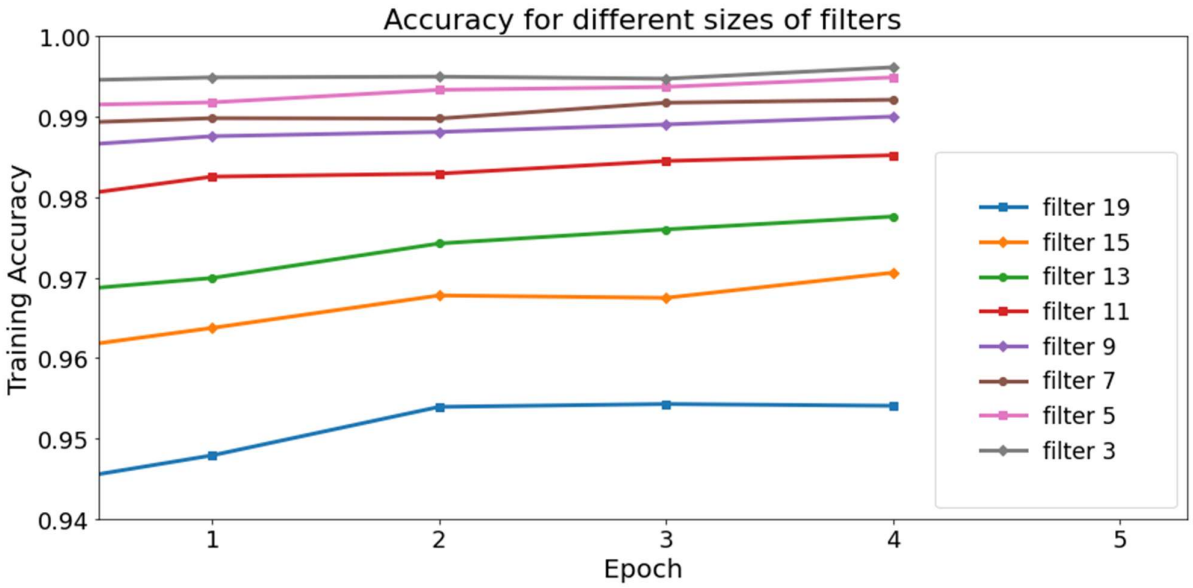
```
(43,)
ClassId: 2
Label: Speed limit (50km/h)
```

Analysis of Experimental Results

Let us visualize the plot for training and validation dataset over the training accuracy and the validation accuracy versus the number of epochs for all the eight various dimensional filter models.

Plot 1: Training Accuracy Vs Epoch

Training accuracy of models with different dimension of convolutional layer filters



Plot 2: Validation Accuracy Vs Epoch

Validation accuracy of models with different dimension of convolutional layer filters

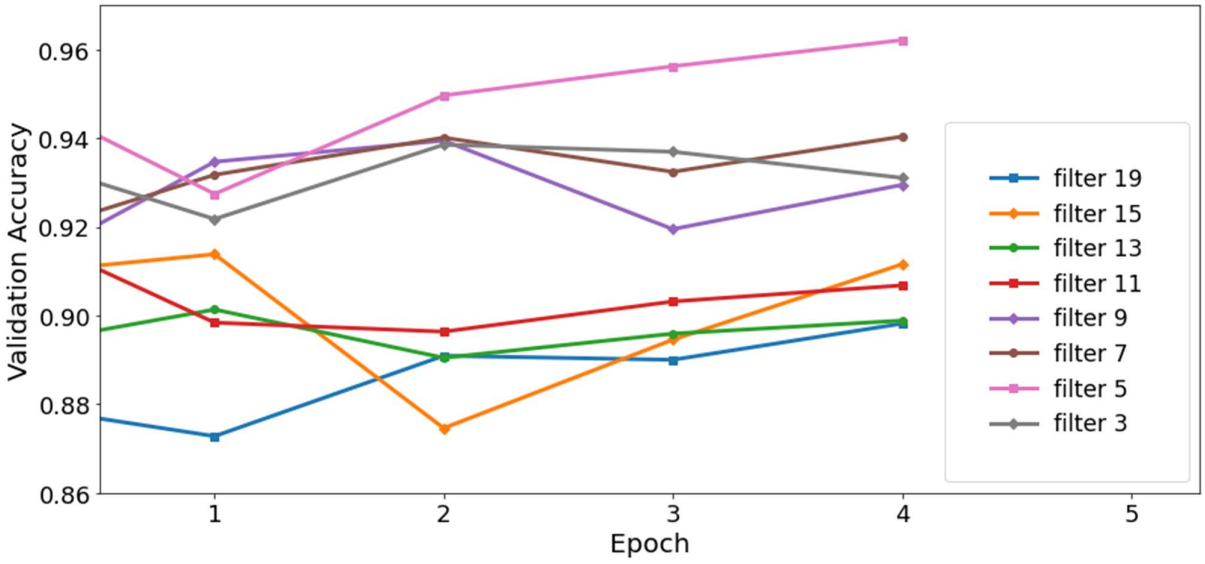


Table2: **Summarized results for accuracy of every model**

Model	Training Accuracy	Validation Accuracy	Testing Accuarcy
3x3	0.99616	0.93855	0.92486
5x5	0.99491	0.96213	0.94212
7x7	0.99213	0.94036	0.92629
9x9	0.99003	0.93946	0.91425
11x11	0.98523	0.92222	0.90713
13x13	0.97761	0.90136	0.89414
15x15	0.97063	0.91383	0.89335
19x19	0.9543	0.89819	0.8806

In addition to the accuracy, the image classification rate for each of the 8 models with their own filter dimensions is also estimated. Experimental results are presented in Table 3.

Table 3: **The rate of image classification of every model**

Model	Classification Time(s)
3x3	0.11309
5x5	0.07126
7x7	0.06178
9x9	0.06628
11x11	0.0972
13x13	0.07695
15x15	0.07587
19x19	0.0792

The initial states and the states at which the training process is completed can be completely illustrated by their CNN filters. The filters used just before the training are randomly initialised and do not possess a sigh of pattern recognition and once the training is complete the specification of the filters which get incorporated is the strategy to recognize characteristics in the form of dots, lines, curves, waves etc. Such filters are looked forward to be used to give out a desirable maximum output at the correct location on a feature map when input image is given.

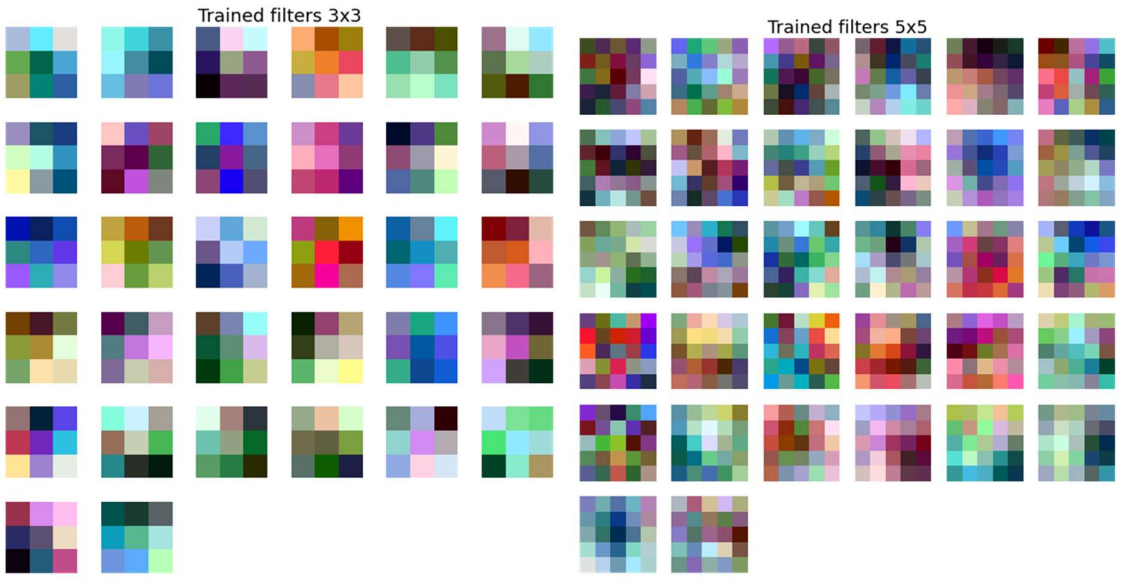
Visualization of the Trained Filters

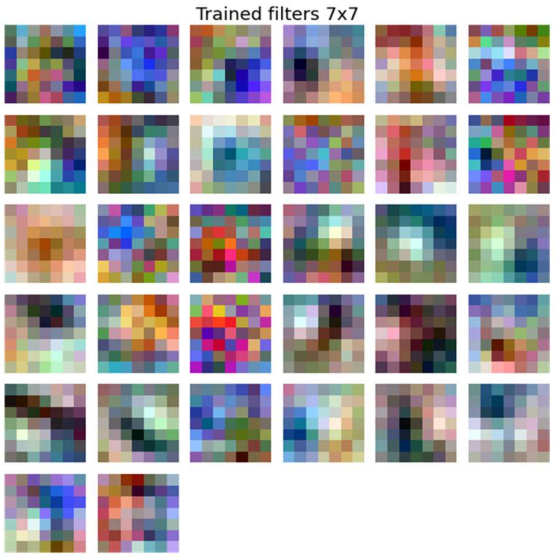
As mentioned in the results of the experiments, the initialized filters are a chaotic set of pixels of different colours. After the training has been done over the dataset, filters have specific features in the form of lines, curves, waves, dots etc. That is we can visualize a particular pattern in the filters which are formed due to the extraction of features from the given training dataset while the training process is under execution.

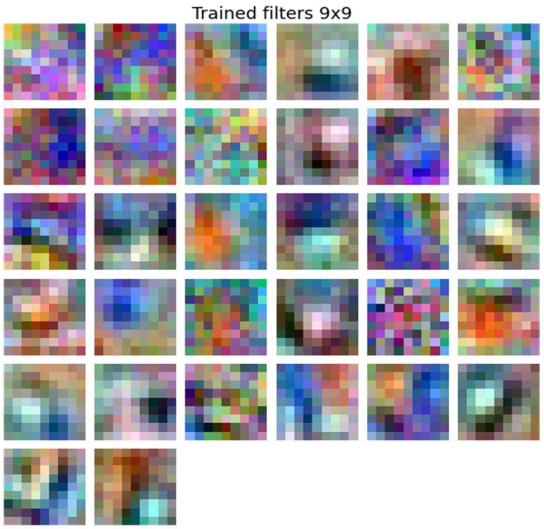
These specific filters are being looked for in the input image and, in case of finding them, the maximum response is given, which is written in the appropriate place of the feature map. Convolutional layer filters can be visualized to see the changes from the initial state when they are initialized randomly and the final state when the training process is completed. Below figures show trained filters for various dimensions of the 3x3, 5x5, 7x7, 9x9, 11x11, 13x13, 15x15 and 19x19 models.

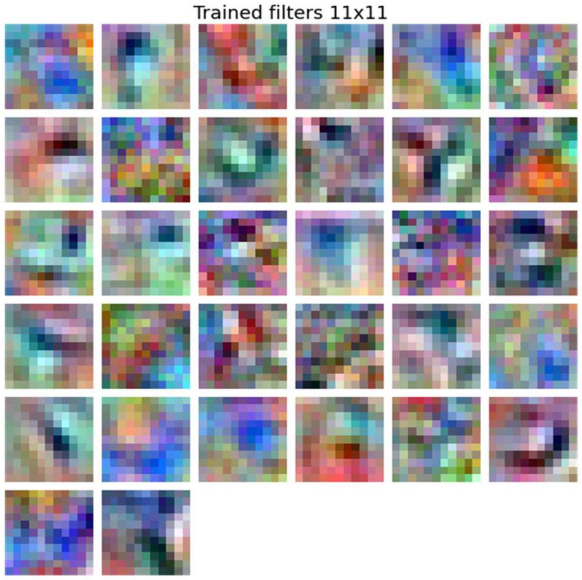
<u>Trained Filters for various dimensions of NxN models</u>

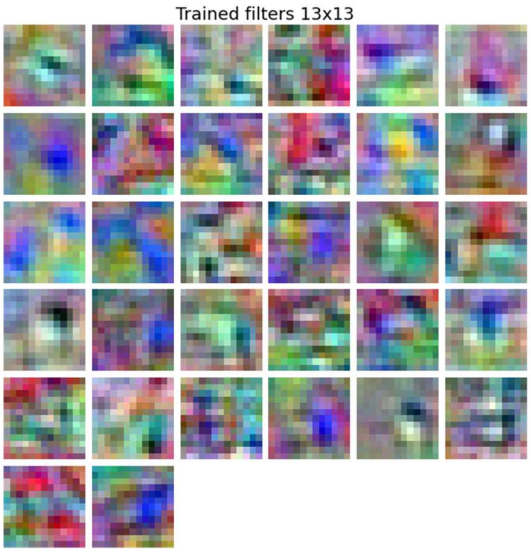
Here N = {3, 5, 7, 9, 11, 13, 15, 19}

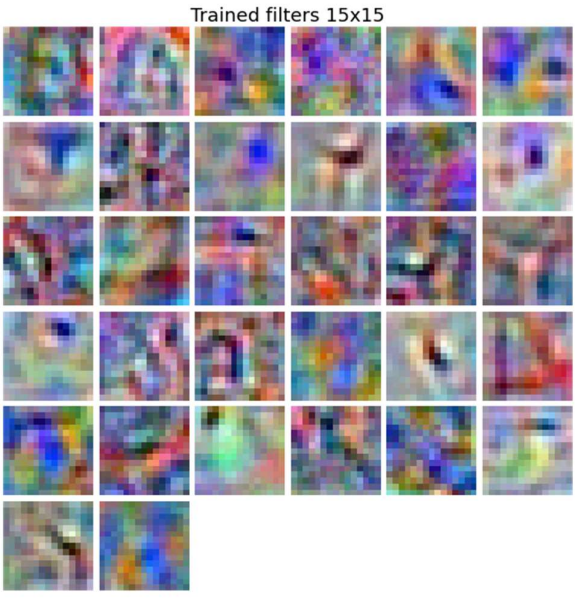


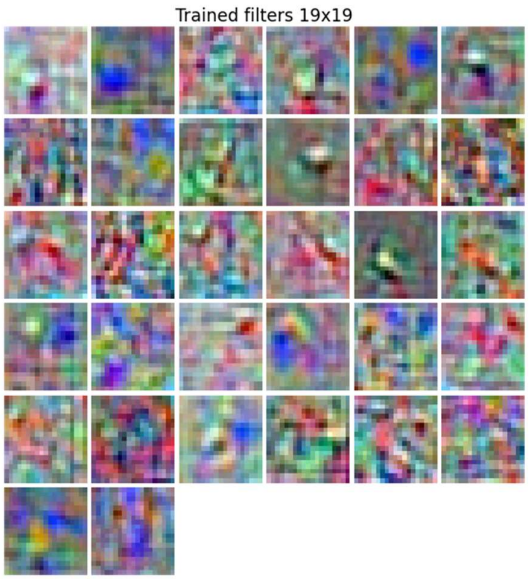

Trained filters 7x7


Trained filters 9x9


Trained filters 11x11


Trained filters 13x13


Trained filters 15x15


Trained filters 19x19

Conclusion

The understandings and analyzation of the methodologies in this project educates the execution of traffic hoardings algorithm constructed on CNN. The main criteria presumed here is the involvement CNN filter dimensions and the consequent effects of it on the rate and classification accuracy of traffic hoardings. The best accuracies were obtained from 5x5, 7x7 filters with their respective values 0.942, 0.926. The best and rapid classification time were obtained from 7x7, 9x9 filters with respective values 0.061, 0.066(s).

The 5x5 filter gave the best output in terms of classification accuracy. The 7x7 filter gave the best output in terms of classification time.

We can further develop the model by adding more dense layers and increasing the number of epochs for improving the accuracy for all the models with convolutional layer filters with different dimensions. We can also improve the accuracy by increasing the number of trials (max_trials) from the keras hyperparameter tuner which chooses the best model out all the models obtained from the max_trials set by us.

We can even extend our project for knowing the effect of various dimension convolution layer filters, by taking other dimensions of filters like 23x23, 25x25, 27x27, 29x29 , 31x31 etc, so that we can get an emphasis on accuracy and performance of classification rate of higher dimensional filters.

References

1. https://towardsdatascience.com/understanding-different-loss-functions-for-neural-networks-dd1ed0274718.

2. https://towardsdatascience.com/building-a-convolutional-neural-network-cnn-in-keras-329fbbadc5f5.

3. https://keras.io/guides/keras_tuner/getting_started/.

4. https://www.tensorflow.org/tutorials/keras/keras_tuner.

5. https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/

6. https://medium.com/deeplearningmadeeasy/negative-log-likelihood-6bd79b55d8b6.

7. https://towardsdatascience.com/a-walkthrough-of-convolutional-neural-network-7f474f91d7bd.

8. https://medium.com/swlh/classification-of-traffic-signs-using-deep-learning-d6c79e95db5f.

****