Chinmay Chinara
USC-ID: 2527237452
E-mail: chinara@usc.edu

EE-569: HOMEWORK #4

Issue date: 03-26-2017 Due date: 04-23-2017

Table of Contents

Chinmay Chinara
USC-ID: 2527237452
E-mail: chinara@usc.edu

Problem-1: CNN Training and Its Application to CIFAR-10 Dataset

a. CNN Architecture and Training

I. Abstract and Motivation

Artificial Neural Network (ANN) is a breakthrough of this era in the field of Artificial Intelligence whose behavior is analogous to the neurons present in the human brain. It is a computational model that is greatly used by scientists and engineers across. Convolutional Neural Networks (CNN) also known as ConvNet, is a type of feed-forward ANN in which the connectivity pattern between its neurons is inspired by the animal visual cortex.

Since CNN can be computationally very heavy, so in practice all machine learning activities are performed on high end system having high end GPUs and multi core. For this assignment, since the training and testing of data were to be done on CPUs, so the dataset given to us was that of CIFAR-10 (which has a set of 60,000 RGB images of dimension 32x32 divided into 10 classes with 6000 images per class). The CNN architecture that needs to be designed was that of LeNet-5 which was introduced by LeCun et al. for hand written and machine printed character recognition. Since LeNet-5 is not meant for image recognition and classification, it is expected that the accuracy that will be obtained at the end using this architecture may not be a good one. So, an assumption for tradeoff between time complexity and quality has been considered for this task.

The LeNet-5 architecture given to us is as under:

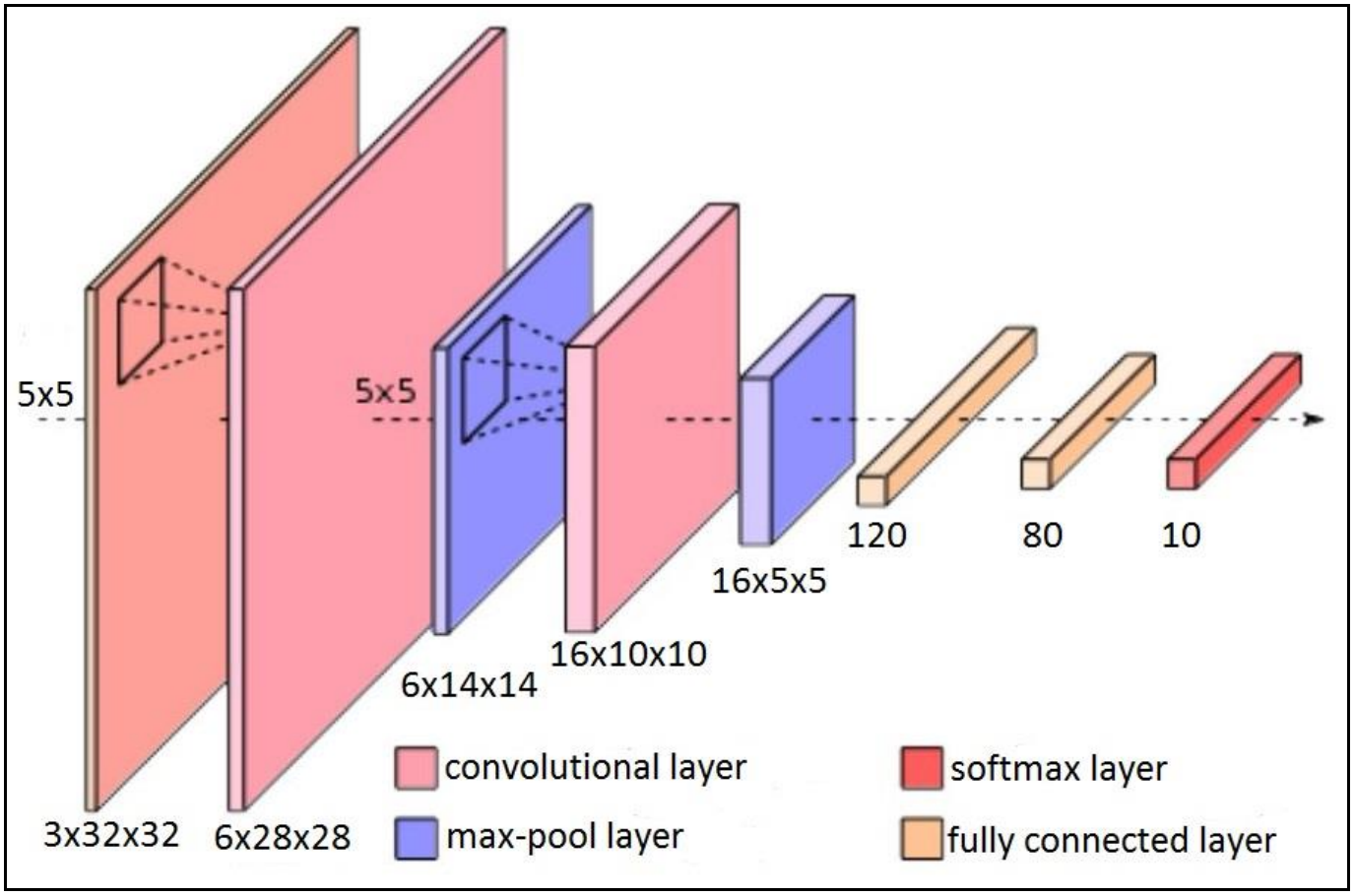

Figure-1.1: A CNN architecture derived from LeNet-5

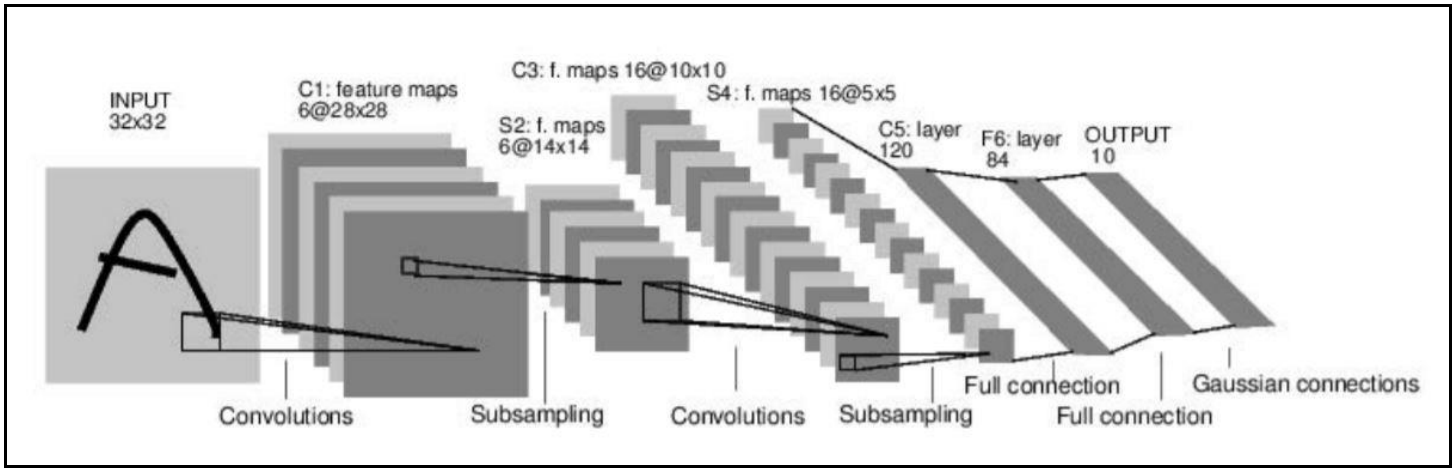

Figure-1.2: The Lenet-5 architecture as an exemplary CNN
(Source: https://arxiv.org/pdf/1701.08481.pdf)

II. Discussion

Q1. Describe CNN components in your own words:

1. **The fully connected layer**
2. **The convolutional layer**
3. **The max pooling layer**
4. **The activation function**
5. **The softmax function**

Ans: The following are the descriptions of each layer of the LeNet-5 CNN given to us in order of traversal is as under:

1. **The Convolutional Layer:**

This layer is the core building block of a CNN network which does most of the computationally heavy tasks. This layer has a set of learnable filters which is small spatially (along width and height), but covers the entire depth of the input image. This input image is forward passed by sliding/convolving each filter across the height and width of the entire image and compute the dot product of the filter elements and image elements at every position. In due course of sliding, a 2D activation map is generated representing the response of that filter in every spatial position. Depending on the number if convolution layers specified, each have their own 2D activation maps which are then stacked along the depth and an output image volume is generated.

Some important parameters that define the function of the convolutional layer are as under:

- Filter size
- Number of filters
- Stride (how much should each convolution window for the filter shift for doing the next convolution)
- Zero padding (to extend boundary so that pixels along the edges are also considered for computation)

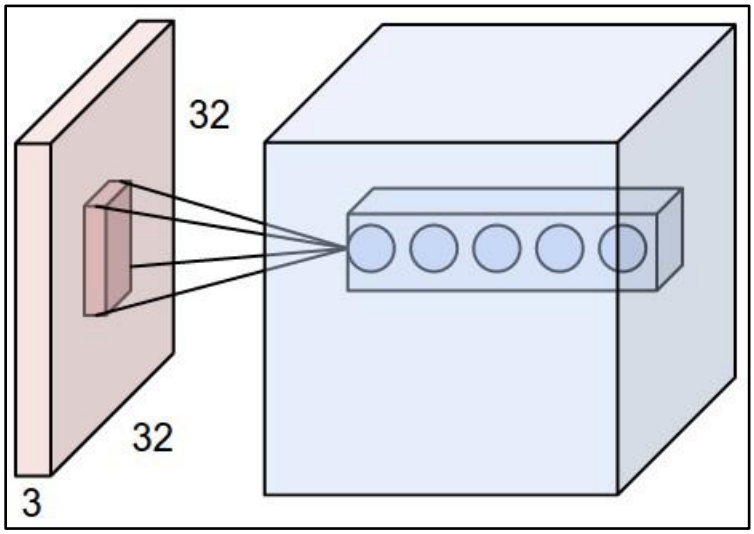

Figure-1.3: An example input volume in red (e.g. a 32x32x3 CIFAR-10 image), and an example volume of neurons in the first Convolutional layer. Each neuron in the convolutional layer is connected only to a local region in the input volume spatially, but to the full depth (i.e. all color channels)
(Source: http://cs231n.github.io/convolutional-networks/ and Discussion Slide)

2. The Max-Pooling Layer (or simply Pooling Layer):

This layer is commonly inserted between successive Convolution layers. Its major function is to progressively reduce the spatial size of the convolution layer output image to reduce the number of parameters and computation in the network so that overfitting gets controlled. The pooling layer operates independently on every depth slice of the input and resizes it spatially using the MAX operation and has zero learnable parameters. The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 down samples every depth slice in the input by 2 along both width and height, discarding 75% of the activations. Every MAX operation would in this case be taking a max over 4 numbers (little 2x2 region in some depth slice). The depth dimension remains unchanged.

Some important parameters that define the function of the convolutional layer are as under:

- Window size
- Stride

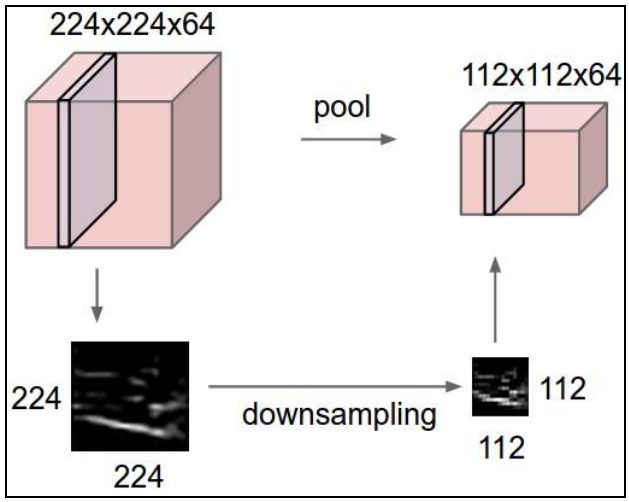

Figure-1.4: In this example, the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64].
(Source: http://cs231n.github.io/convolutional-networks/ and Discussion Slide)

3. The Fully Connected Layer:

This layer has full connections to all the activations in the previous layer. Their activations can hence be computed using matrix multiplication followed by a bias offset. This layer basically takes an input volume (whatever the output is of the Convolution or Pooling layer preceding it) and outputs an N dimensional vector where N is the number of classes that the program must choose from. It looks at the output of the previous layer (which should represent the activation maps of high level features) and determines which features most correlate to a class and have weights so that when the product between the weights and the previous layer is computed, we get the correct probabilities for the different classes.

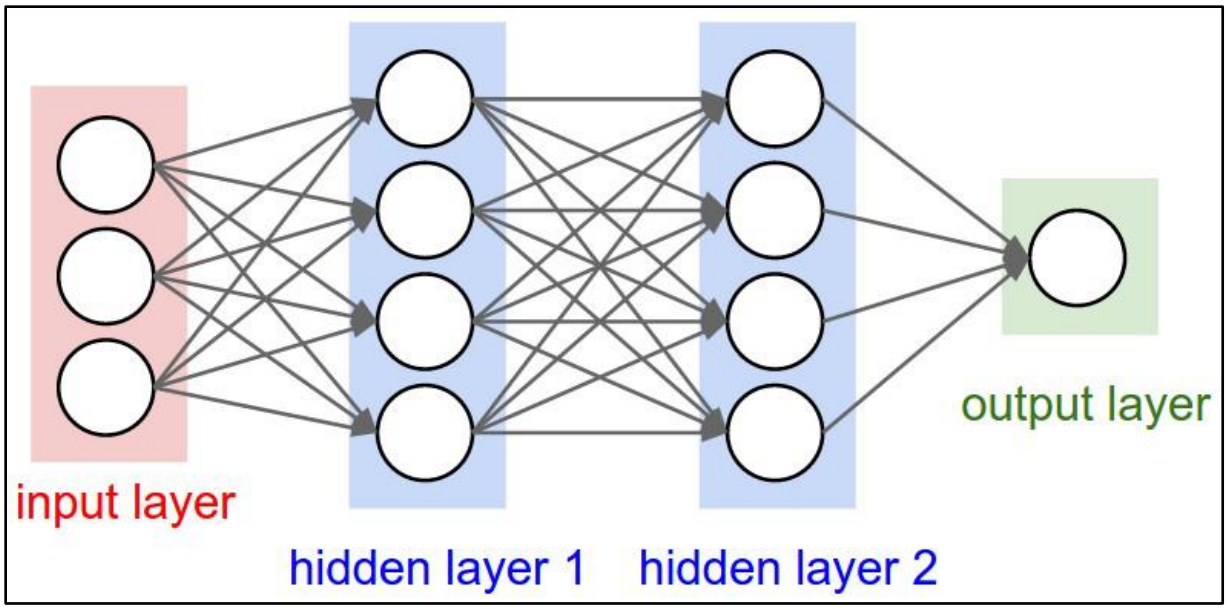

Figure-1.5: A fully connected layer
(Source: Discussion Slide)

4. The activation function:

The activation function of a node in general defines the output of that node given an input or a set of inputs. It is a non-linear function that allows networks to compute non-trivial problems using only a small number of nodes. Say we have an artificial neuron which calculates a weighted sum of its inputs, adds a bias and then gives an output. What decides whether this neuron should get fired or not is the activation function. Thus, activation functions help introducing non-linearity into the network. One key thing to note is summing up layers with linear activation function would just give another linear function due to which non-linear activation functions are always considered. These contain zero learnable parameters.

There are a lot of activation functions that have been used for performing machine learning tasks. To list a few popular ones:

i. Sigmoid function:

This takes a real valued number and squishes it into the range 0 to 1. However, it has fallen out of favor because it saturates and kills gradients. Also, sigmoid outputs are not centered to zero leading to zig-zag dynamics in gradient update which leads to slower convergence.

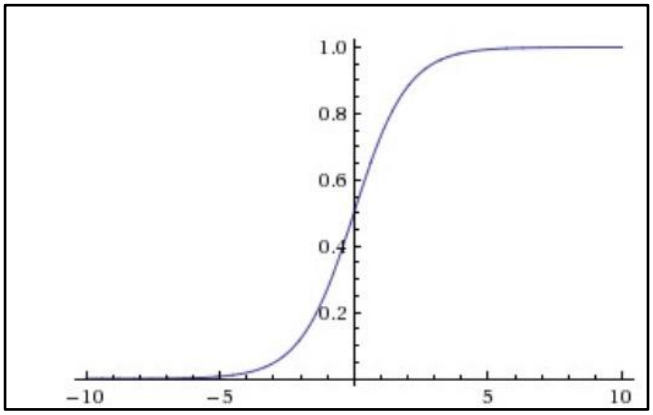

Figure-1.6: The sigmoid function

ii. Tanh function:

This takes a real valued number and squishes it into the range between -1 to 1. This too saturates and kill gradients. The outputs are zero centered and it converges much faster than the sigmoid.

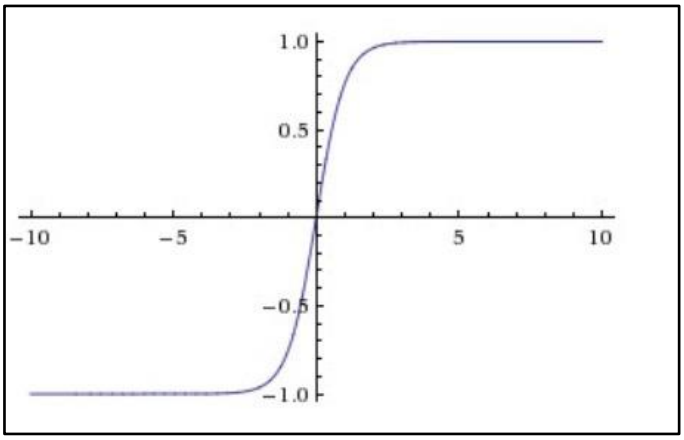

Figure-1.7: The Tanh function

iii. ReLU function (Rectified Linear Unit):

It computes the function f(x) = max(0, x) i.e. the activation is threshold at zero. It doesn't saturate in the positive region. It converges much faster than Sigmoid or Tanh functions. The output values are not zero centered. However, ReLU too has its cons. ReLUs are fragile during training and can die out if learning rate is improperly set. A dead ReLU outputs the same value for any input. Dead ReLUs take no role in discriminating between inputs. It is impossible to recover once a ReLU ends up at this stage.

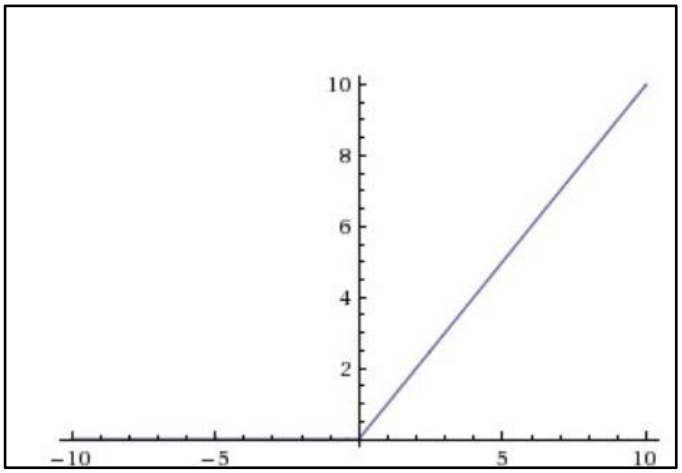

Figure-1.8: The ReLU function

iv. Leaky ReLU function (Leaky Rectified Linear Unit):

The above problem of "Dying ReLUs" can be fixed using Leaky ReLUs. Instead of the function being zero when x<0, a Leaky ReLU will instead have a small negative slope (of 0.01 or so). A mathematical approach to the statement is the function computes f(x) = 1(x<0)(Ax) + 1(x>=0)(x) where A is a small constant[1]. Leaky ReLU does not saturate. It converges much faster than Sigmoid and Tanh. The output is not zero centered. This will not die like ReLU due to improper learning rate.

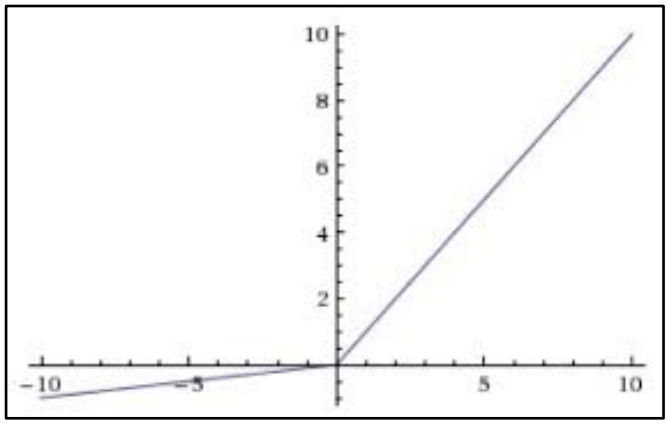

Figure-1.9: The Leaky ReLU function

5. The softmax function:

Mathematically, the softmax function is a generalization of the logistic function squashes a K-dimensional vector of arbitrary real values into a K dimensional vector of real values in the range (0,1) that add to 1. In ANN, the softmax function is used in the final layer of a neural network based classifier. Such networks are commonly trained under a cross-entropy regime, giving a non-linear variant of multinomial logistic regression. Thus, softmax in just a generalization of logistic regression.

Q2. What are the functions of these components?

Ans: The individual functions are already discussed in the question above. If we see an overall perspective, each of these components help in solving a typical machine learning problem. Image classification and text classification are some of the areas where these components are widely used.

Q3. What is the major difference between a CNN and the traditional multi-layer perceptron (MLP)?

Ans: To start off with, CNNs are MLPs with a special structure. CNNs have repetitive blocks of neurons that are applied across space be it for audio, video or images. For images, these blocks of neurons can be interpreted as 2D convolutional kernels, repeatedly applied over each patch of the image. At training time, the weights for these repeated blocks are 'shared', i.e. the weight gradients learned over various image patches are averaged.

The reason for choosing this special structure, is to exploit spatial or temporal invariance in recognition. For instance, a "dog" or a "car" may appear anywhere in the image. If we were to learn independent weights at each spatial or temporal location, it would take orders of magnitude more training data to train such an MLP. In over-simplified terms, for an MLP which didn't repeat weights across space, the group of neurons receiving inputs from the lower left corner of the image must learn to represent "dog" independently from the group of neurons connected to upper left corner, and correspondingly we will need enough images of dogs such that the network had seen several examples of dogs at each possible image location separately.

On top of this fundamental constraint, many special techniques and layers have been invented specially in the CNN context. So, a latest deep CNN might look very different from a bare bones MLP, but the above states the major difference in principle.

Q4. Why CNNs work much better than other traditional methods in many computer vision problems? You can use the image classification problem to elaborate your points.

Ans: CNNs are widely used in the computer vision field nowadays. They offer state-of-the-art solutions to many challenging vision and image processing problems such as object detection, scene classification, room layout estimation, semantic segmentation, image super-resolution, image restoration, object tracking, etc. They are the main stream machine learning tool for big visual data analytics. A great amount of effort has been devoted to the interpretability of CNNs based on various disciplines and tools, such as approximation theory, optimization theory and visualization techniques. As suggested let's take the image classification problem to delve deep as to why CNN works much better for computer vision problems.

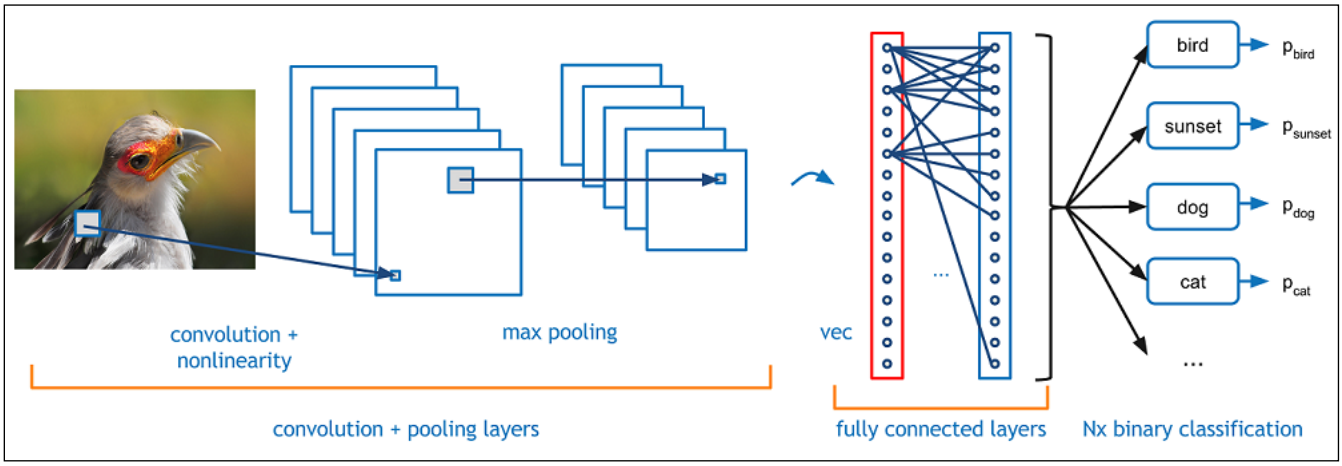

Figure-1.10: One such image classification problem using CNN

Over the years, lots of different machine learning architecture have been in place to solve the challenges of image processing and computer vision. CNN is the latest trendsetter for handling these challenges.

The basic problem space for image classification is the task of taking an input image and outputting a class (a cat, an aeroplane, a dog, etc.) or a probability of classes that best describes the image.

The table below describes the various broad classification of machine learning techniques used over the years for the challenge above and how CNN can be proved to be the best is described below:

Classification method	Advantages	Disadvantages
Artificial Neural Network	• It is a non-parametric classifier. • It is a universal functional approximator with arbitrary accuracy. • It is a data driven self-adaptive technique • Efficiently handles noisy inputs • Computation rate is high	• It is semantically poor. •The training of ANN is time consuming. • Problem of over fitting. • Difficult in choosing the type network architecture.
Decision tree	• Can handle nonparametric training data • Does not required an extensive design and training. • Provides hierarchical associations between input variables to forecast class membership and provides a set of rules n are easy to interpret. • Simple and computational efficiency is good.	•The usage of hyperplane decision boundaries parallel to the feature axes may restrict their use in which classes are clearly distinguishable. •Becomes complex calculation when various values are undecided and/or when various outcomes are correlated
Support Vector Machine	• It gains flexibility in the choice of the form of the threshold. • Contains a nonlinear transformation. • It provides a good generalization capability. • The problem of over fitting is eliminated. • Reduction in computational complexity. • Simple to manage decision rule complexity and error frequency	•Result transparency is low. •Training is time consuming. • Structure of algorithm is difficult to understand •Determination of optimal parameters is not easy when there is nonlinearly separable training data.
Fuzzy Measure	• Efficiently handles uncertainty. • Properties are described by identifying various stochastic relationships.	• Without prior knowledge output is not good • Precise solutions depend upon direction of decision
Convolutional Neural Network	• Has best-in-class performance on problems that significantly outperforms other solutions in multiple domains. This includes speech, language, vision, etc. This isn't by a little bit, but by a significant amount.	• Requires a large amount of data — if you only have thousands of examples, deep learning is unlikely to outperform other approaches. • Is extremely computationally expensive to train. The most complex models take weeks to train

	• Reduces the need for feature engineering, one of the most time-consuming parts of machine learning practice. • It is an architecture that can be adapted to new problems relatively easily • CNN has a very layered structure where each layer considers e very nitty gritty of an image to be classified with great details and accuracy which is not done in any of the above mentioned methods. • Classification accuracy is the highest in this method.	using hundreds of machines equipped with expensive GPUs. • Do not have much in the way of strong theoretical foundation. This leads to the next disadvantage. • Determining the topology/flavor/training method/hyperparameters for deep learning is a black art with no theory to guide you. • What is learned is not easy to comprehend. Other classifiers (e.g. decision trees, logistic regression etc) make it much easier to understand what's going on.

Q5. Explain the loss function and the classical back-propagation (BP) optimization procedure to train such a convolutional neural network.

Ans:

1. **Loss function:**

As per Wikipedia, *"A loss function or cost function is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event. An optimization problem seeks to minimize a loss function. An objective function is either a loss function or its negative (sometimes called a reward function, a profit function, a utility function, a fitness function, etc.), in which case it is to be maximized."*

When we define a function from the pixel values to class scores, which was parameterized by a set of weights W we don't have control over the data (it is fixed and given), but we do have control over these weights and we want to set them so that the predicted class scores are consistent with the ground truth labels in the training data. For example, say we have an image of a cat and its scores for the classes "cat", "dog" and "ship", if the set of weights are not a good set of values then the score when an image is fed will greatly deviate from the class it should represent. To measure our unhappiness with outcomes such as this one we use something called a loss function (or sometimes also referred to as the cost function or the objective) to continuously keep seeing the loss or mismatch in the classification result in due course of training and hence reduce it to a negligible amount iteratively and intelligently so that training and classification accuracy increases over time. Intuitively, the loss will be high if we're doing a poor job of classifying the training data, and it will be low if we're doing well.

2. **Back Propagation:**

As per Wikipedia, *"The backward propagation of errors or backpropagation, is a common method of training artificial neural networks and used in conjunction with an optimization method such as gradient descent. The algorithm repeats a two-phase cycle, propagation and weight update. When an input vector is presented to the network, it is propagated forward through the network, layer by layer, until it reaches the output layer. The output of the network is then compared to the desired output, using a loss function, and an error value is calculated for each of the neurons in the output layer. The error values are then propagated backwards, starting from the output, until each neuron has an associated error value which roughly represents its contribution to the original output."*

Back-propagation uses these error values to calculate the gradient of the loss function with respect to the weights in the network. In the second phase, this gradient is fed to the optimization method, which in turn uses it to update the weights, to minimize the loss function. The importance of this process is that, as the network is trained, the neurons in the intermediate layers organize themselves in such a way that the different neurons learn to recognize different characteristics of the total input space. After training, when an arbitrary input pattern is present which contains noise or is incomplete, neurons in the hidden layer of the network will respond with an active output if the new input contains a pattern that resembles a feature that the individual neurons have learned to recognize during their training.

Back-propagation requires a known, desired output for each input value to calculate the loss function gradient – it is therefore usually considered to be a supervised learning method; nonetheless, it is also used in some unsupervised networks. It is a generalization of the delta rule to multi-layered feedforward networks, made possible by using the chain rule to iteratively compute gradients for each layer. Backpropagation requires that the activation function used by the artificial neurons (or "nodes") be differentiable.

The motivation behind back-propagation was to find a way to train a multi-layered neural network such that it can learn the appropriate internal representations to allow it to learn any arbitrary mapping of input to output. The goal of backpropagation is to compute the partial derivative, or gradient, $\boldsymbol{\partial E/\partial w}$ of a loss function **E** with respect to any weight **w** in the network.

b. Application of the CNN to CIFAR-10 Dataset

I. Abstract and Motivation

LeNet-5 CNN architecture is an excellent architecture that was developed for performing text classification and recognition. The task at hand is to train the CIFAR-10 dataset using the LeNet-5 architecture.

II. Approach and Procedures

Theoretical Approach:

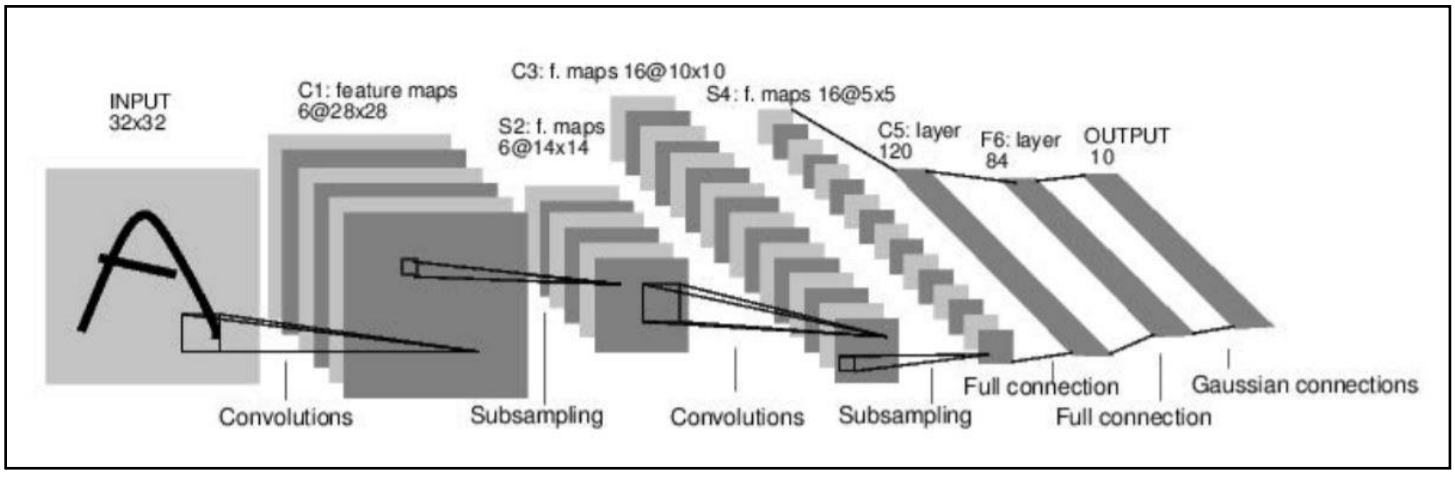

Figure-1.11: The Lenet-5 architecture as an exemplary CNN
(Source: https://arxiv.org/pdf/1701.08481.pdf)

The figure above shows the LeNet-5 architecture. It consists of two spatial convolutional and max pooling layers with two fully connected hidden layers. It contains a total of 61,482 parameters that are optimized over a training period (how this is obtained is explained below). The following are the modules that are used for training the CIFAR-10 dataset in their specific order.

- **Input Layer:** This layer provides the input image pixel values to the network. The size of this layer depends on the size of the images present in the training and testing databases. CIFAR-10 consists of RGB images that are of dimensions 32x32 both in the testing and training databases, numerically being 60,000 in the training and 10,000 in the testing databases. Hence, the input layer consists of 32x32 images as neurons spatially arranged having a depth of 3.

- **Spatial Convolution Layer 1:** In case of images, the input data coming is very high dimensional i.e. in our case being 32x32x3x60,000 = 1,84,320,000. With these many input data, the amount of weights to be calculated would be around 4,821,600 which needs to be learnt in the very first layer. This may cause the neural network to give a very poor output if the number of training examples are very few. A much better approach would be to take a local region of the input image data and connect it to each neuron in the output. This reduces the receptive field of each neuron which is compensated for the next layers of this architecture. For our assignment, this very concept of receptive field was selected to be of dimensions 5x5 thus making each output neuron see 5x5 patch from the input. The LeNet-5 architecture given to us has the receptive field set to 5x5 with no zero padding and stride 1. The input image is convolved over 6 filter banks and thus an image of size 32x32 gets transformed to 28x28 with 6 layers, each layer representing convolved output with each respective filter. That number of weights that each neuron in this layer will have is (5x5x3(weights) + 1(bias)*6 = 456 that corresponds to a 5x5x3 region in the input volume.

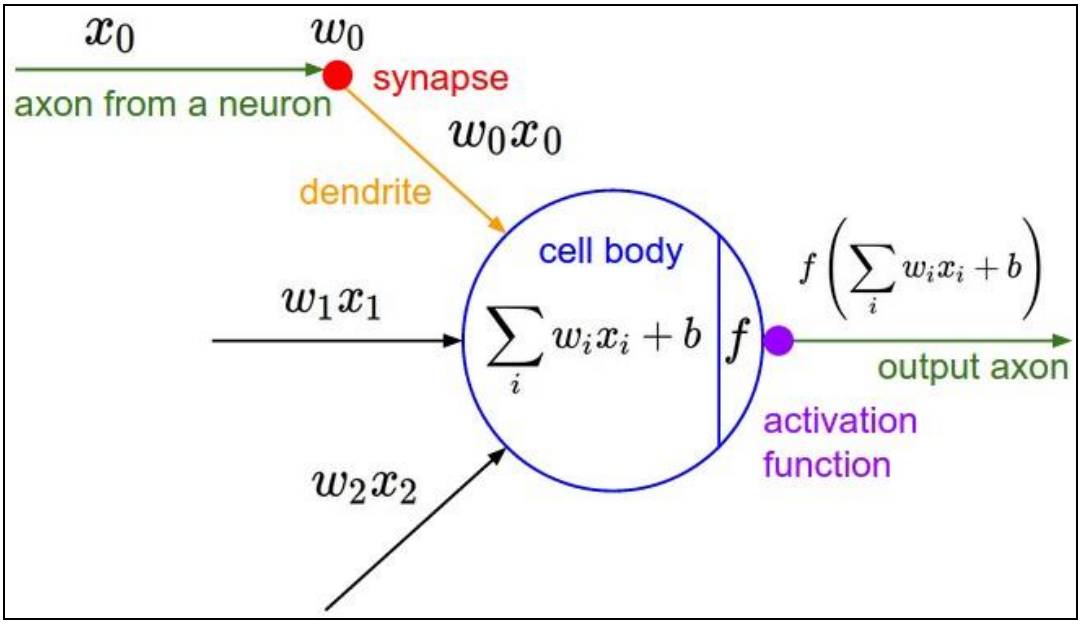

Figure-1.12: The convolution process
(Source: https://arxiv.org/pdf/1701.08481.pdf)

- **Spatial max-pooling Layer 1:** This layer takes the output of the convolution-1 layer and subsamples the output. In order to improve the receptive field w.r.t. the input layer and reduce the number of weights in the subsequent layers this step is done. According to question, a 2x2 window is considered. The dimension of the input after max-pooling with the specified hyper-parameters becomes 14x14. There are no weights in this layer.

- **ReLU Non-Linear Activation 1:** This layer will apply an elementwise activation that is threshold at zero. The input size remains unchanged and there are no weights to be learned in this layer. The ReLU function is represented as **output = max(0, input)**. Introducing non-linearity in the network is very important so this non-linear activation

function is used to cater the purpose. Nothing in the real world is linear, so if we do not train the data by non-linearity then performance degradation is observed. The reason as to why ReLU and not something like sigmoid or tanh because it has been proven by lot of experiments that it tends to converge much faster.

- **Spatial Convolution Layer 2:** Similar to spatial convolution layer 1, this takes an input which is of dimensions 14x14x6 in size and convolves with a receptive field of 5x5 as given in the question. This layer outputs 16 layered images based on the parameter of 16 filter banks to be used as mentioned in the question. The dimension of the images output here is 10x10. The number of weights this layer has is (5x5x6(weights)+1(bias)) * 16 = 2,416.

- **Spatial max-pooling Layer 2:** This layer takes the output of the convolution-2 layer and subsamples the output. According to question, a 2x2 window is considered. The dimension of the input after max-pooling with the specified hyper-parameters becomes 5x5x16 from 10x10x16. There are no weights in this layer too.

- **ReLU Non-Linear Activation 2:** This layer performs the same way as Layer-1. Until now the input image dimension has been reduced from 32x32= 1024 neurons to 20x20 = 400 neurons. The previous layers thus accounted for feature and dimensionality reduction. These features/neurons are now passed onto the fully connected layers.

- **Fully Connected Hidden Layer-1:** This layer consists of 120 neurons as mentioned in the question which is connected to 400 neurons from the previous layer. The number of weights in this layer that needs to be learnt are (20x20 + 1(bias)) * 120 = 48,120. This layer has the maximum receptive field i.e. each neuron has all the information about the input image. ReLU activation is used to introduce non-linearity.

- **Fully Connected Hidden Layer-2:** This layer consists of 80 neurons connected from 120 neurons from the previous layer. The number of weights in this layer that needs to be learnt are (120 + 1(bias)) * 80 = 9,680. ReLU activation is used to introduce non-linearity in this layer too.

- **Output Layer:** This layer consists of 10 neurons representing the output labels connected to 80 neurons in the previous layer. The number of weights in this layer that needs to be learnt are (80 + 1(bias)) * 10 = 810. This is cascaded with a log softmax layer that provides the log probabilities of the output labels. It is a method to normalize the outputs to easily determine the output label for each input. The log softmax function is applied to each neuron and is represented as $\boldsymbol{s(x) = \log\left(\frac{e^x}{\sum_{i=1}^{N} e^i}\right)}$.

  Hence the output of each neuron is x is passed to this function. All the N neurons are compared with the same formula. During the training process, these values are passed to the negative log likelihood criterion to reduce cross entropy.

  Summing up all the weights, we have (456 + 2,416 + 48,120 + 9,680 + 810) = 61,482 weights to be trained.

For training and testing the CIFAR-10 data, in addition to the steps above lot of other parameters were also involved to produce the desired output. They are discussed as under:

- **Preprocessing:** Input image data was normalized by dividing with 255 to get the pixel values in range from 0 to 1. The data was zero-centered as well by subtracting the mean of each image from each pixel in itself to prevent any high frequencies to dominate the image data.

- **Random Weight initialization:** After building the network, weights are initialized to each layer to a suitable value. The Xavier weight initialization was used for this assignment. The reason why this was used in because experimentally it has been proven to give the best results. This initializer is designed to keep the scale of the gradients roughly the same in all layers. In uniform distribution this ends up being the range: **x = sqrt(6./(in+out)); [-x, x]** and for normal distribution a standard deviation of **sqrt(3./(in+out))**.

- **Batch-size:** This is one of the most import criteria that decides the speed and accuracy of the entire training and testing process. As per experiments conducted, the minimum batch size that gives a decent output should be around 96. The size of batch contributes heavily towards convergence and performance of the neural network. This a hyper-parameter of the network that needs to be optimized using brute force. The batch sizes I used ranged from 96 to 1000. By using batches, the variance of the updates is drastically reduced. Larger batch sizes contribute towards unstable update vectors which will converge slowly.

- **Learning Rate:** Learning rate determines the time for convergence of the network. Smaller learning rates converge very slowly and larger learning rates may not converge to the global minima. For my set of experiments, I set my learning rate at 0.001 by studying various papers and experimental data available online which accounts it to be an optimal value to get good testing accuracy.

- **Optimizer:** The optimizer used in my case was not SGD (Stochastic Gradient Decent) because it converged very slowly and the difference between training and testing accuracy was very high. The Adam optimizer proved to be very useful and gave a very good output and was thus used.

- **Data shuffling:** Neural networks are a strong tool for learning. Its strengths may sometimes cause overfitting. Overfitting is the phenomenon in which the data gets modelled to the noise in the training set. This may result in poor performance in the test set. To prevent overfitting, the training set was randomly shuffled after every epoch and trained in batches.

- **Image augmentation:** The dataset of CIFAR-10 has images which are of normal orientation. To introduce a flavor of dynamic types of data, the input data is augmented in the beginning. It introduces variety in the type of data and gives a strong dataset to be trained with. By augmentation I mean introducing some scaling, rotation, translation, flipping, etc. like parameters to the image. In my case, my input data was randomly flipped left right and rotated by 0-25 degrees.

- **Epoch:** The process of forward pass, loss function, backward pass, and parameter update is generally called one epoch. The program will repeat this process for a fixed number of epochs for each set of training images or batch. Experimentally proven, an epoch of 100 was used to perform my experiments.

- **Dropout:** The underlying concept of dropout is to neglect a certain portion of neurons in layer during a batch. This reduces overfitting as it restricts the strong features to dominate over the weaker ones. I tried using dropout after every fully connected layer with the probabilities ranging from 0.5 to 0.9. It did make a significant difference in the final testing accuracy obtained.

Algorithm used to develop in TfLearn using Python:
Step-1: The CIFAR-10 dataset was downloaded.

Step-2: The input data was randomly shuffled.

Step-3: The input data was then zero centered and normalized and the following steps were performed for five different sets of data each with some change in parameters as instructed in the question.

Step-4: Augmentation was added to introduce variety in the dataset by randomly left-right flipping and rotating by 0-25 degrees in the following order.

Parameter	Set-1	Set-2	Set-3	Set-4	Set-5
Augmentation	No	No	Yes	Yes	No

Step-5: A tensor 'network' was created with all the input data specifications as introduced in the previous steps.

Step-6: The network was then passed through convolution layer 1 with the following parameters:

Parameter	Set-1	Set-2	Set-3	Set-4	Set-5
Receptive field	5x5	5x5	5x5	5x5	5x5
Padding	valid (no zero pad)	valid (no zero pad)	valid (no zero pad)	valid (no zero pad)	valid (no zero pad)
Initial weights	xavier	xavier	xavier	None	xavier
Activation function	ReLU	ReLU	ReLU	ReLU	ReLU
Filter bank size	6	6	16	16	16

Step-7: The network was then passed through max-pooling layer 1 with the following parameters:

Parameter	Set-1	Set-2	Set-3	Set-4	Set-5
Window size	2x2	2x2	2x2	2x2	2x2
Padding	valid (no zero pad)	valid (no zero pad)	valid (no zero pad)	valid (no zero pad)	valid (no zero pad)

Step-8: The network was then passed through convolutional layer-2 with the following parameters:

Parameter	Set-1	Set-2	Set-3	Set-4	Set-5
Receptive field	5x5	5x5	5x5	5x5	5x5
Padding	valid (no zero pad)	valid (no zero pad)	valid (no zero pad)	valid (no zero pad)	valid (no zero pad)
Initial weights	xavier	xavier	xavier	None	xavier
Activation function	ReLU	ReLU	ReLU	ReLU	ReLU
Filter bank size	16	16	16	16	16

Step-9: The network was then passed through max-pooling layer-2 with the following parameters:

Parameter	Set-1	Set-2	Set-3	Set-4	Set-5
Window size	2x2	2x2	2x2	2x2	2x2
Padding	valid (no zero pad)	valid (no zero pad)	valid (no zero pad)	valid (no zero pad)	valid (no zero pad)

Step-10: The network was then passed through fully connected layer-1 with the following parameters:

Parameter	Set-1	Set-2	Set-3	Set-4	Set-5
Neurons	120	120	120	120	120
Activation function	ReLU	ReLU	ReLU	ReLU	ReLU

Step-11: A drop out was set as per the table below:

Parameter	Set-1	Set-2	Set-3	Set-4	Set-5

Dropout	0.5	0.7	0.9	0.9	0.9

Step-12: The network was then passed through a fully connected layer-2 with the following parameters:

Parameter	Set-1	Set-2	Set-3	Set-4	Set-5
Neurons	80	80	80	80	80
Activation function	ReLU	ReLU	ReLU	ReLU	ReLU

Step-13: A drop out was set as per the table below:

Parameter	Set-1	Set-2	Set-3	Set-4	Set-5
Dropout	0.5	0.7	0.9	0.9	0.9

Step-14: The network was then passed through a fully connected layer which represented the output layer with the following parameters:

Parameter	Set-1	Set-2	Set-3	Set-4	Set-5
Neurons	10	10	10	10	10
Activation function	softmax	softmax	softmax	softmax	softmax

Step-15: Regression was performed on the network with the following parameters:

Parameter	Set-1	Set-2	Set-3	Set-4	Set-5
Optimizer	Adam	Adam	Adam	Adam	Adam
Loss	Categorical crossentropy	Categorical crossentropy	Categorical crossentropy	Categorical crossentropy	Categorical crossentropy
Learning rate	0.001	0.0005	0.0005	0.0005	0.06

Step-15: The network was then passed for training and testing with the following parameters:

Parameter	Set-1	Set-2	Set-3	Set-4	Set-5
Epoch	100	100	100	100	150
Shuffle	True	True	True	True	True
Batch-size	256	256	256	512	512

Step-16: The logs that were saved was then used to visualization using tensor board using the command: **tensorboard --logdir=<path where the log file was saved>** in the python terminal.

III. Experimental Results

Table-1.1: Table showing the summary of experiments conducted

Variations	mAP Training Accuracy (%)	mAP Testing Accuracy (%)	Time-taken
Set-1	66.25	63.41	1h 11m 16s
Set-2	71.10	62.86	1h 17m 33s
Set-3	69.61	69.45	1h 23m 26s
Set-4	68.09	68.38	1h 23m 39s
Set-5	79.84	63.45	1h 43m 51s

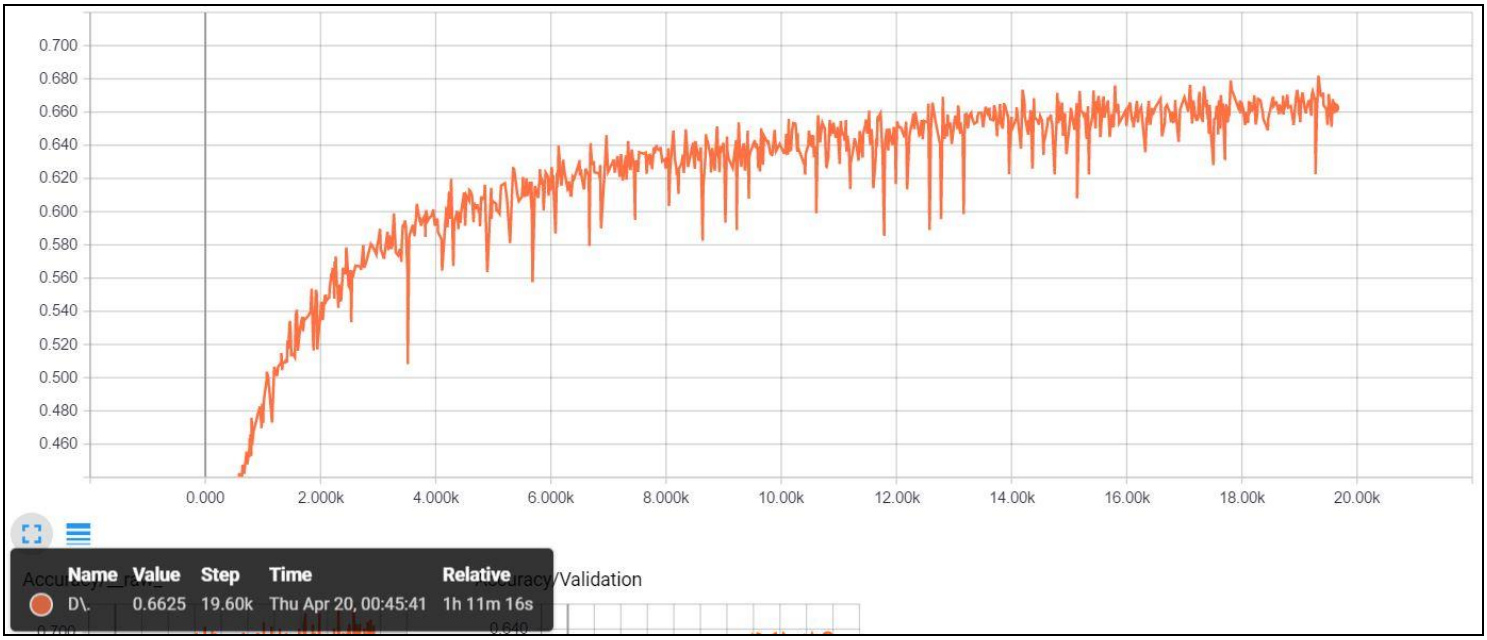

Figure-1.13: mAP curve Epoch(steps) vs training data accuracy for Set-1 configuration (66.25%)

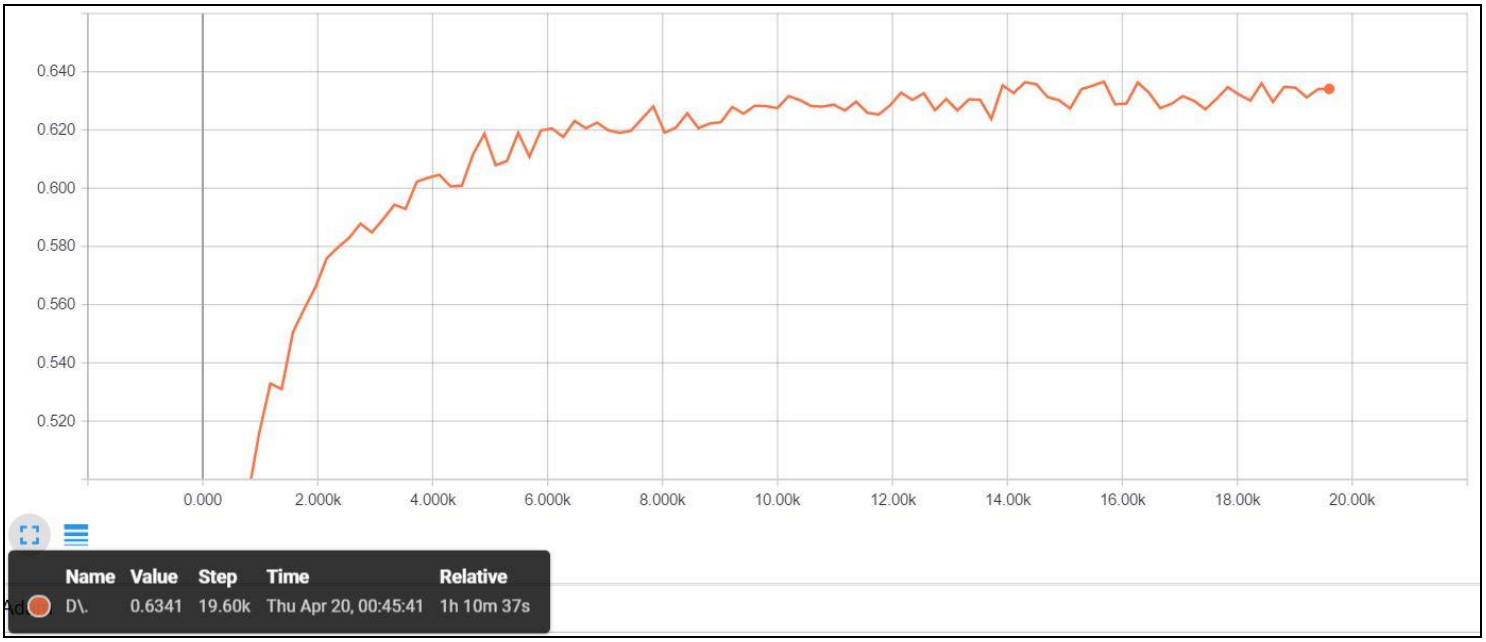

Figure-1.14: mAP curve Epoch(steps) vs testing data accuracy for Set-1 configuration (63.41%)

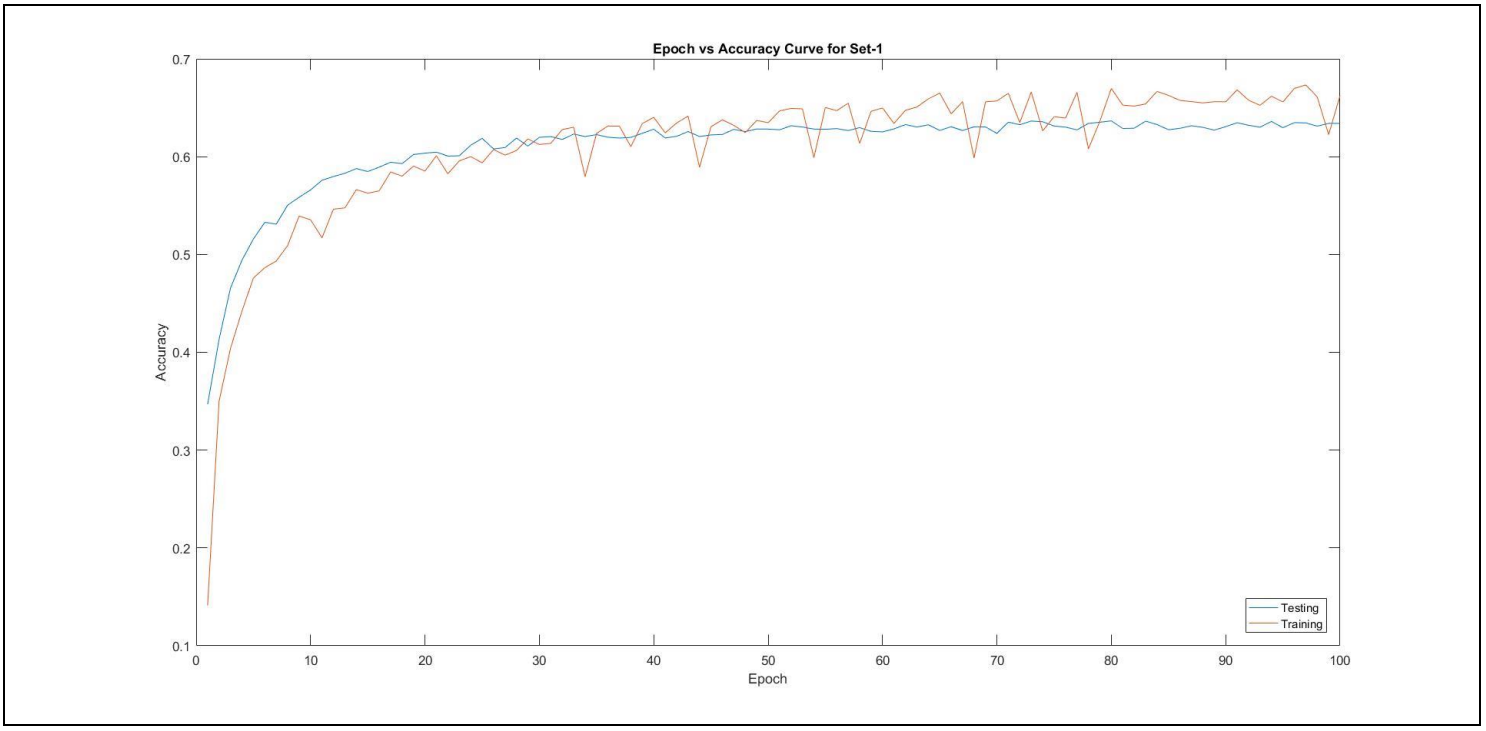

Figure-1.15: mAP curve Epoch(steps) vs training/testing data accuracy for Set-1 configuration

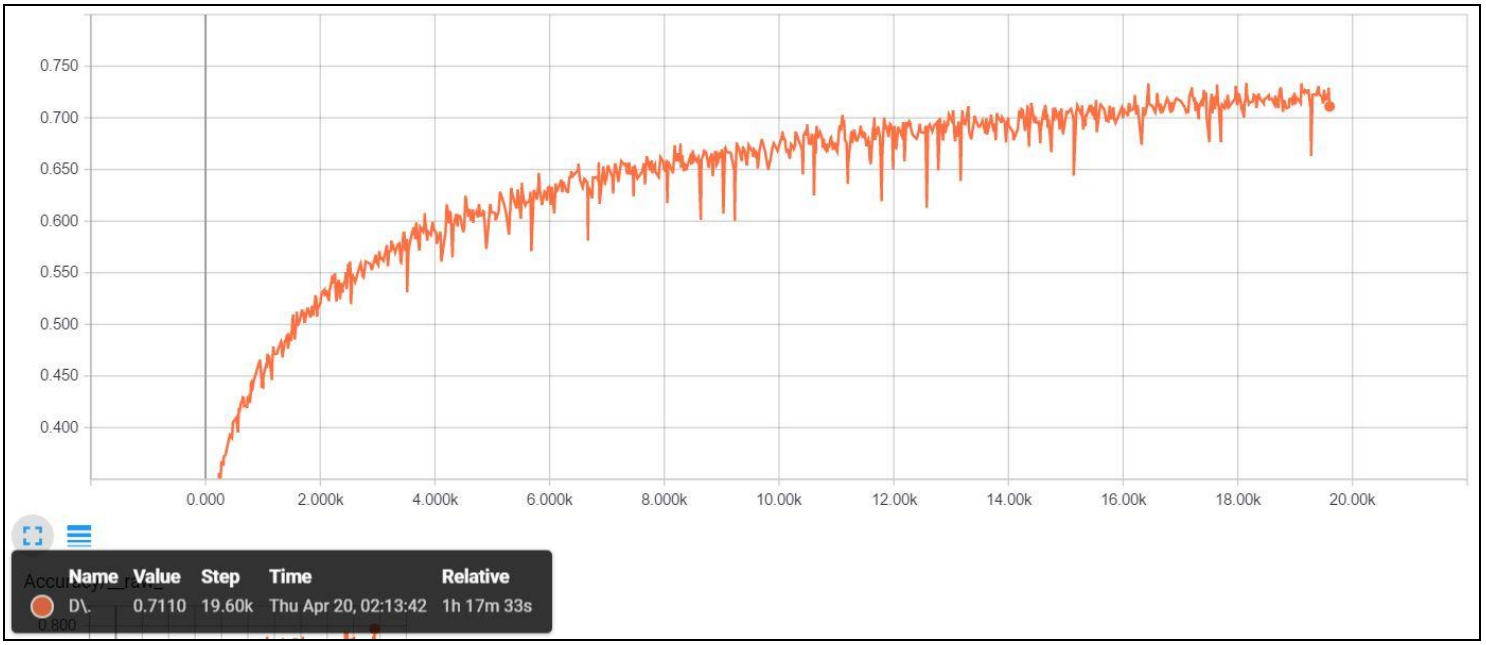

Figure-1.16: mAP curve Epoch(steps) vs training data accuracy for Set-2 configuration (71.10%)

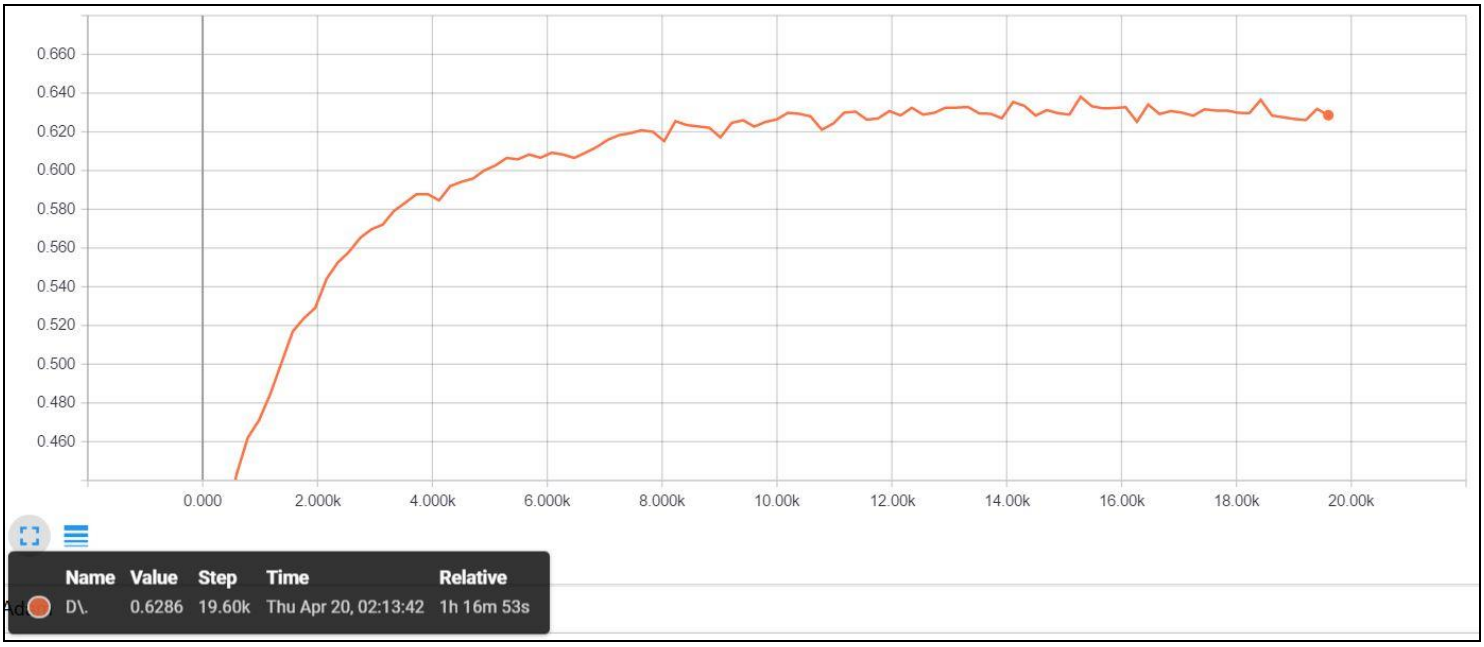

Figure-1.17: mAP curve Epoch(steps) vs testing data accuracy for Set-2 configuration (62.86%)

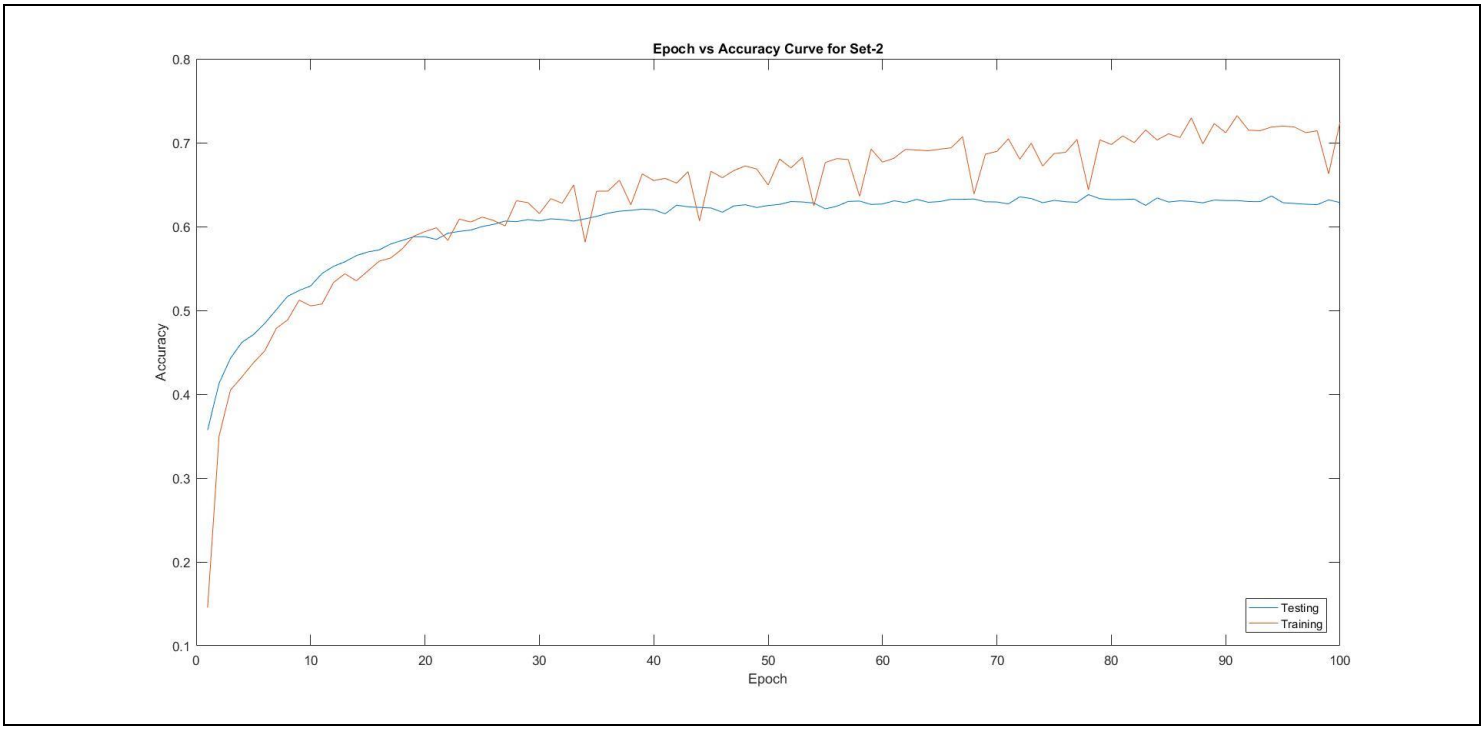

Figure-1.18: mAP curve Epoch(steps) vs training/testing data accuracy for Set-2 configuration

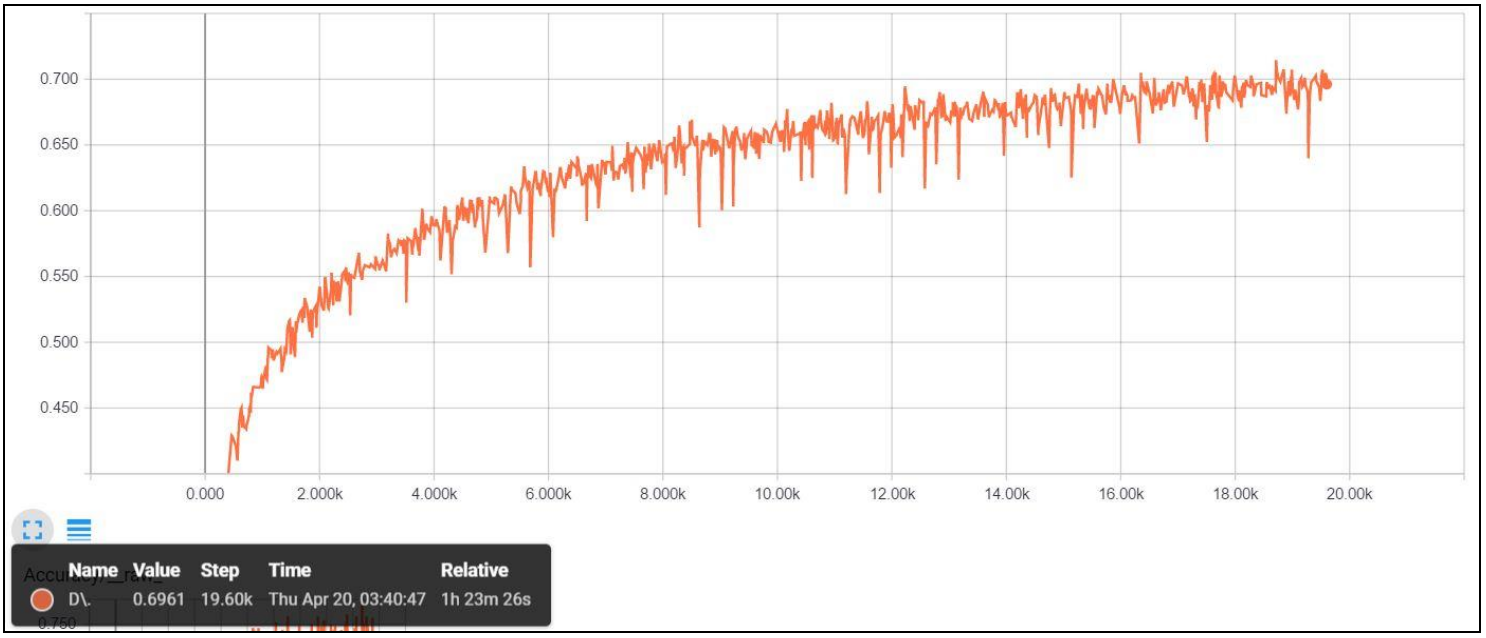

Figure-1.19: mAP curve Epoch(steps) vs training data accuracy for Set-3 configuration (69.61%)

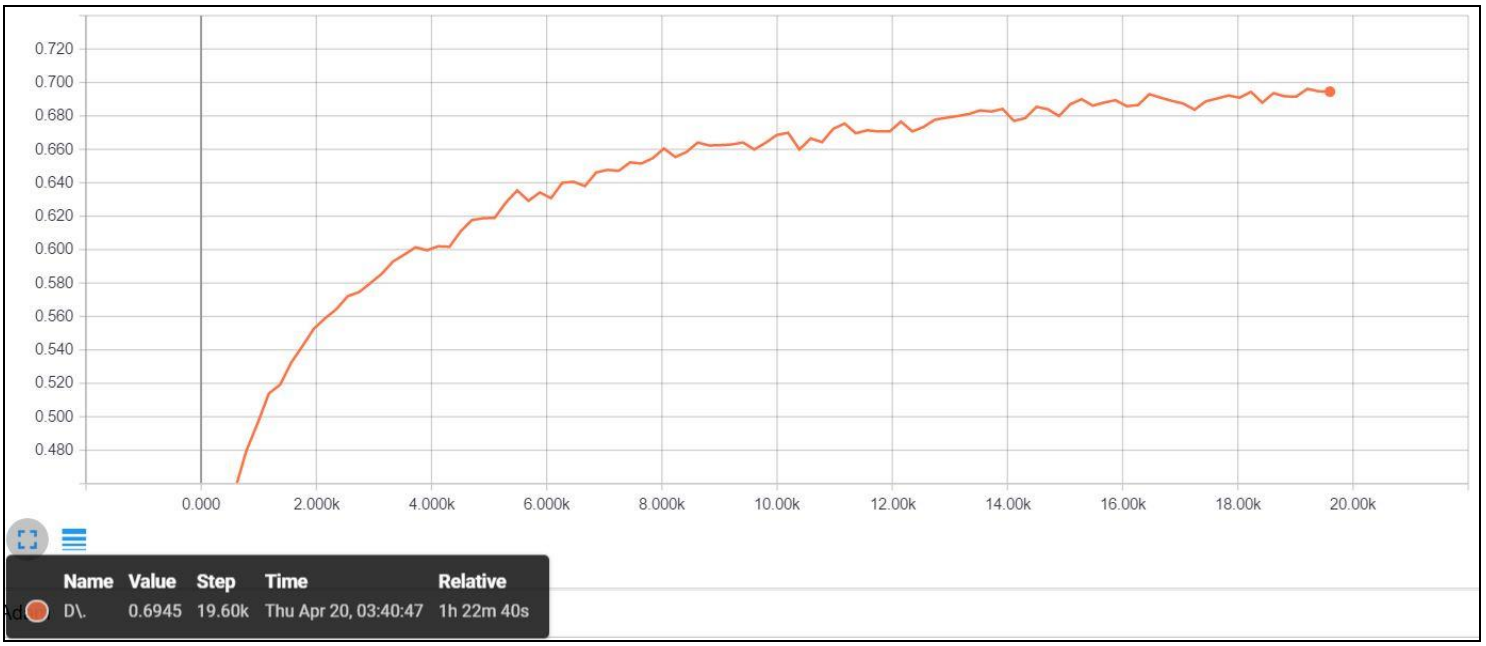

Figure-1.20: mAP curve Epoch(steps) vs testing data accuracy for Set-3 configuration (69.45%)

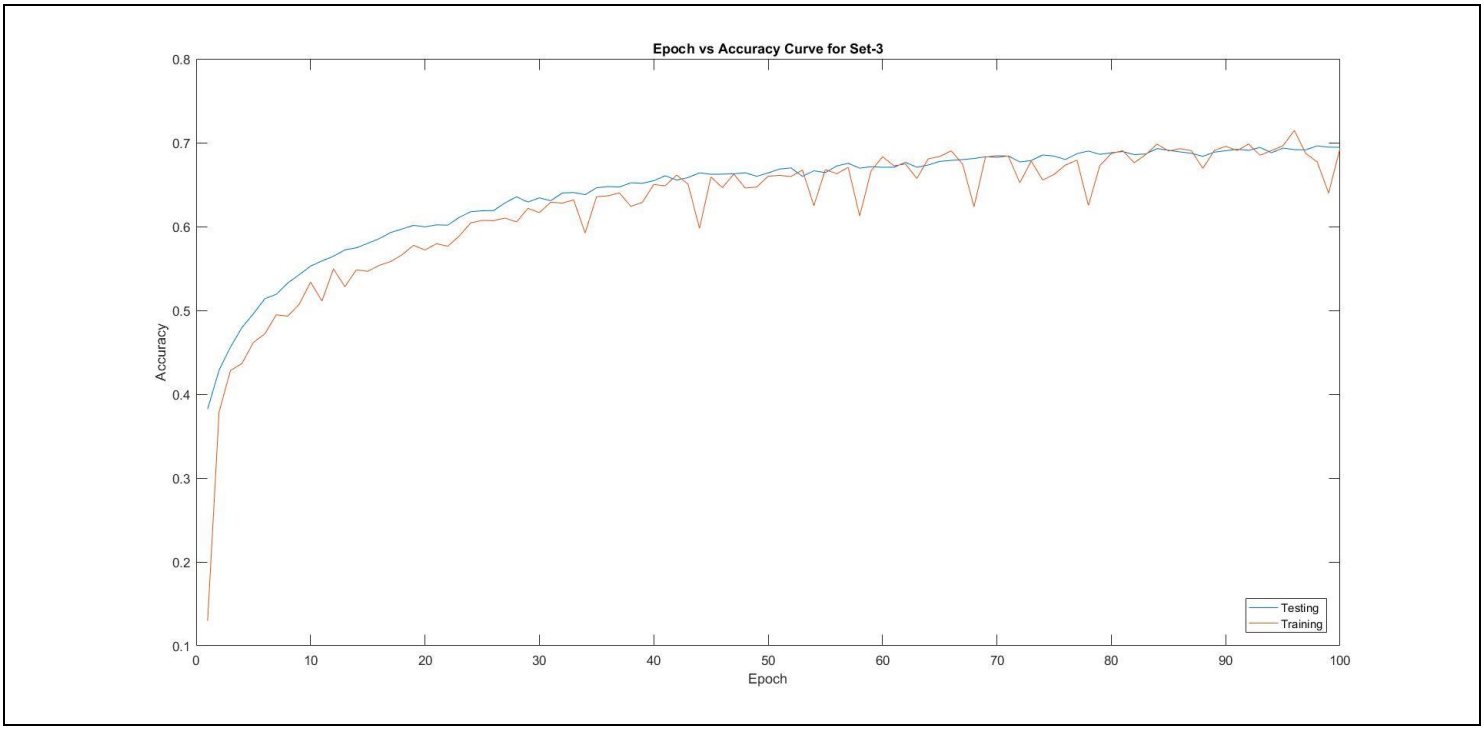

Figure-1.21: mAP curve Epoch(steps) vs training/testing data accuracy for Set-3 configuration

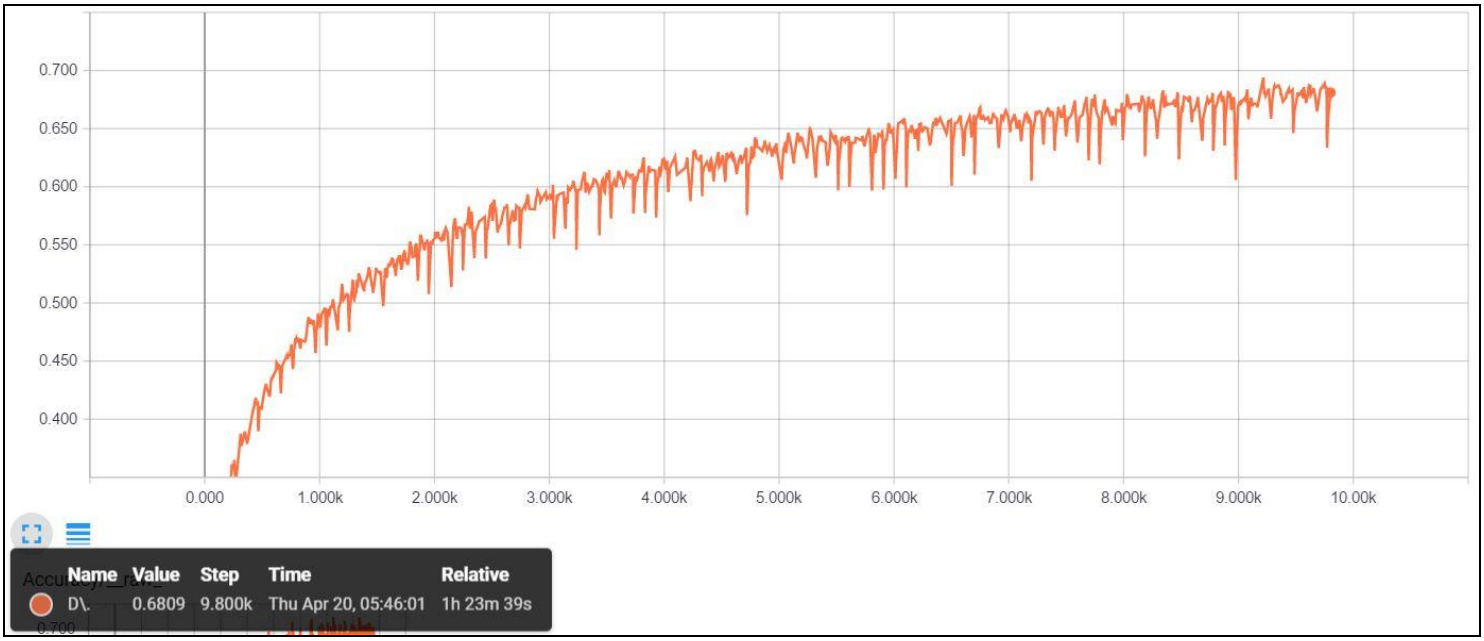

Figure-1.22: mAP curve Epoch(steps) vs training data accuracy for Set-4 configuration (68.09%)

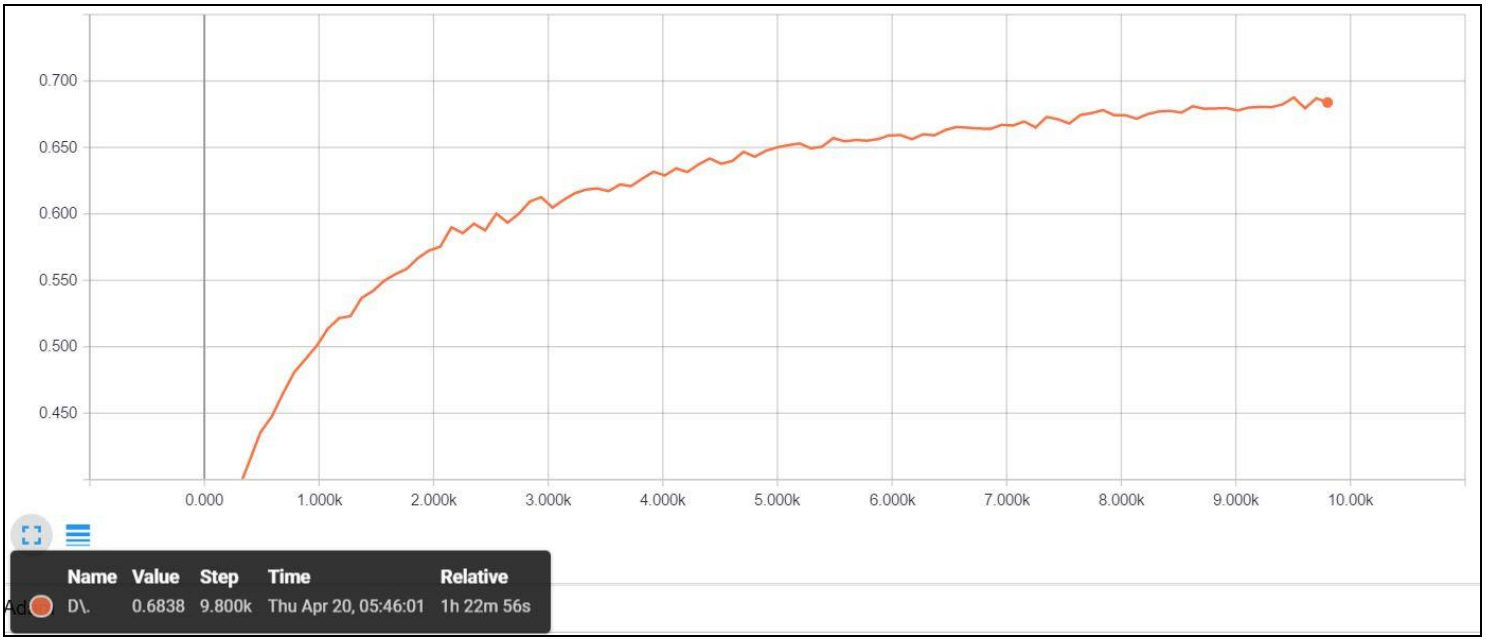

Figure-1.23: mAP curve Epoch(steps) vs testing data accuracy for Set-4 configuration (68.38%)

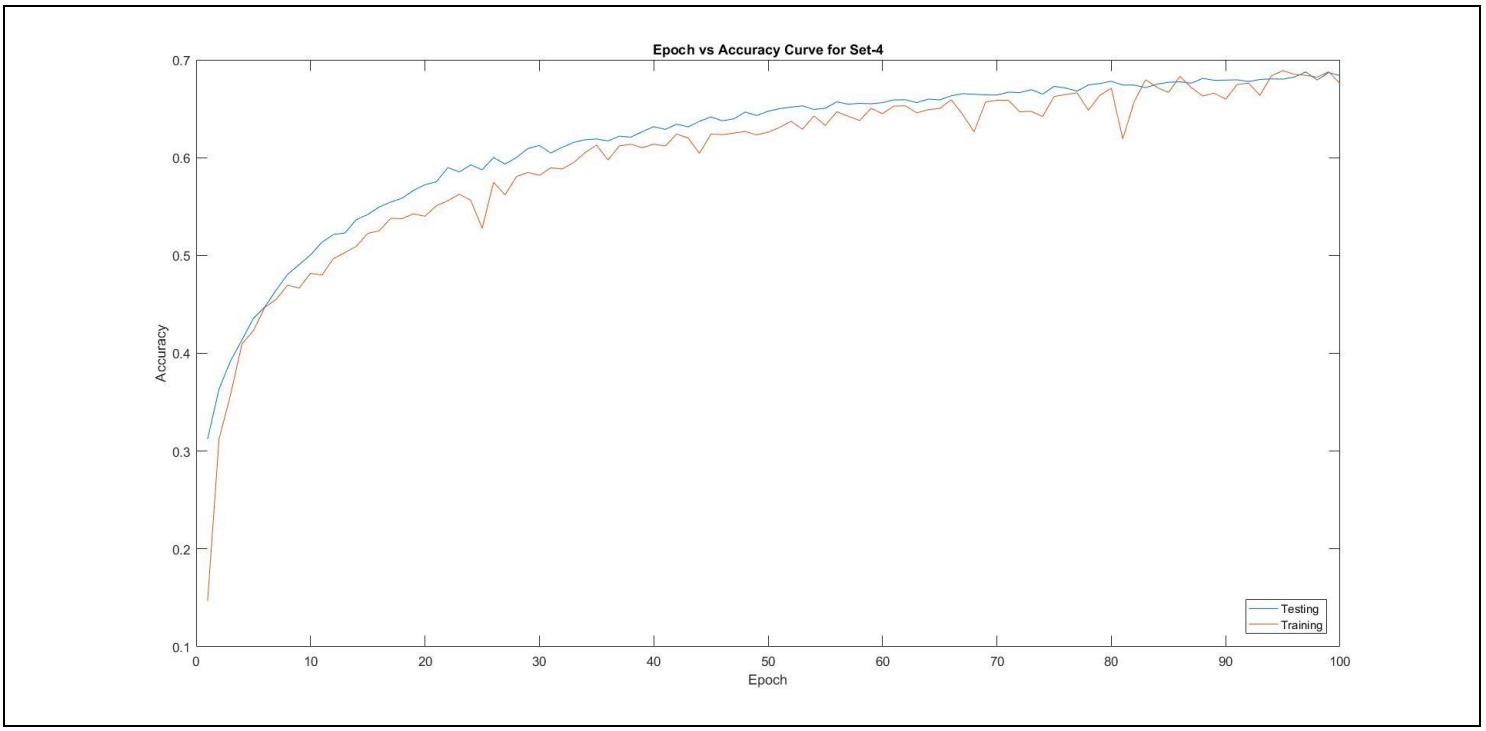

Figure-1.24: mAP curve Epoch(steps) vs training/testing data accuracy for Set-4 configuration

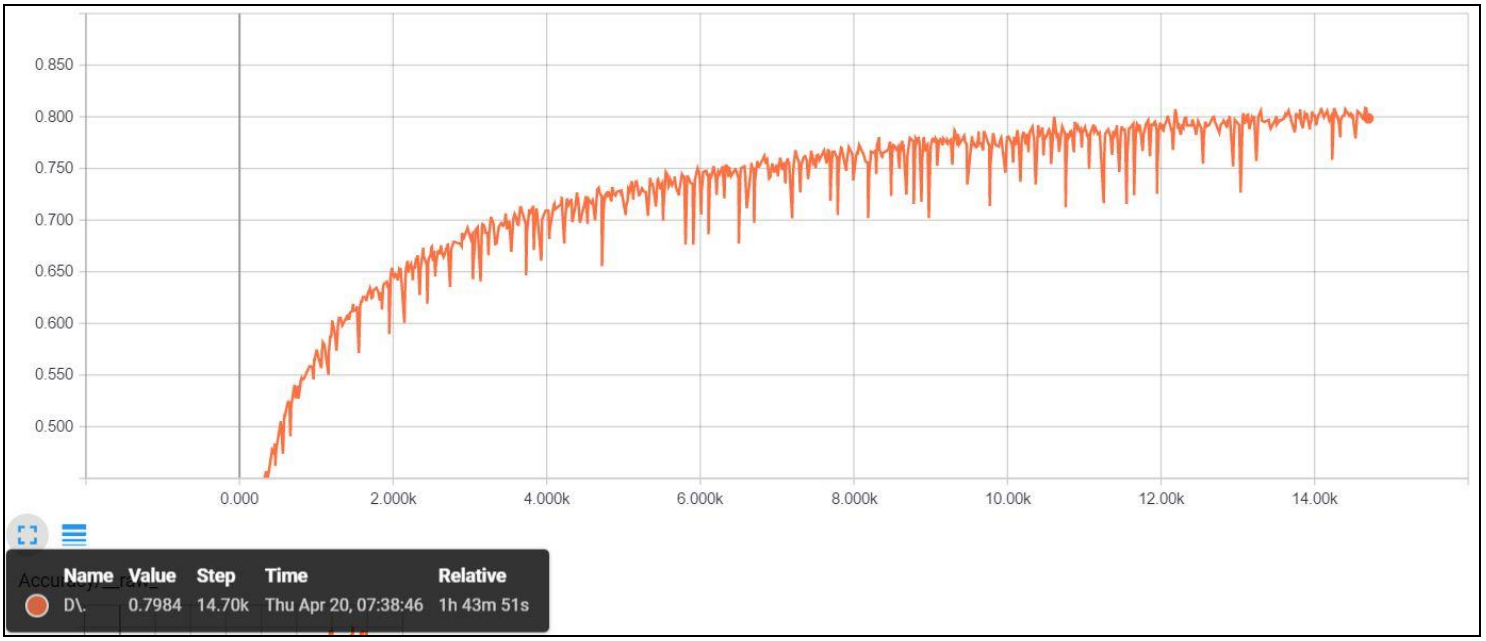

Figure-1.25: mAP curve Epoch(steps) vs training data accuracy for Set-5 configuration (79.84%)

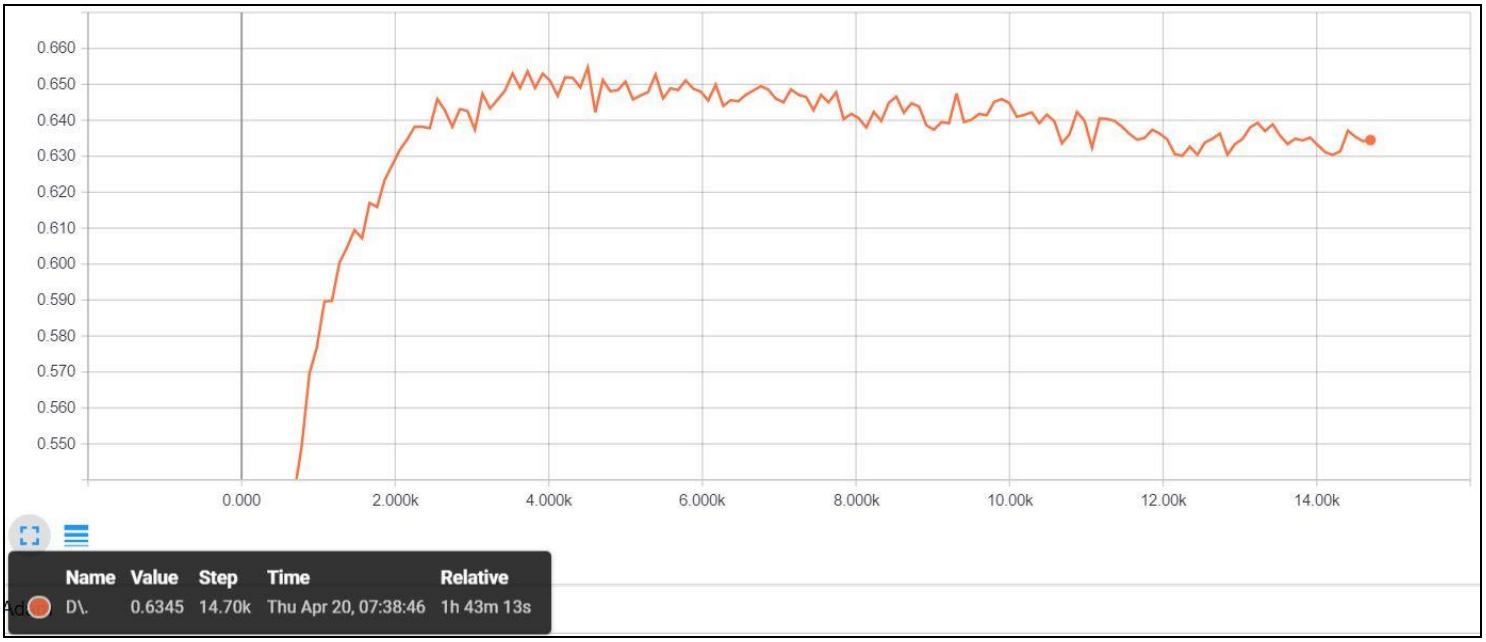

Figure-1.26: mAP curve Epoch(steps) vs testing data accuracy for Set-5 configuration (63.45%)

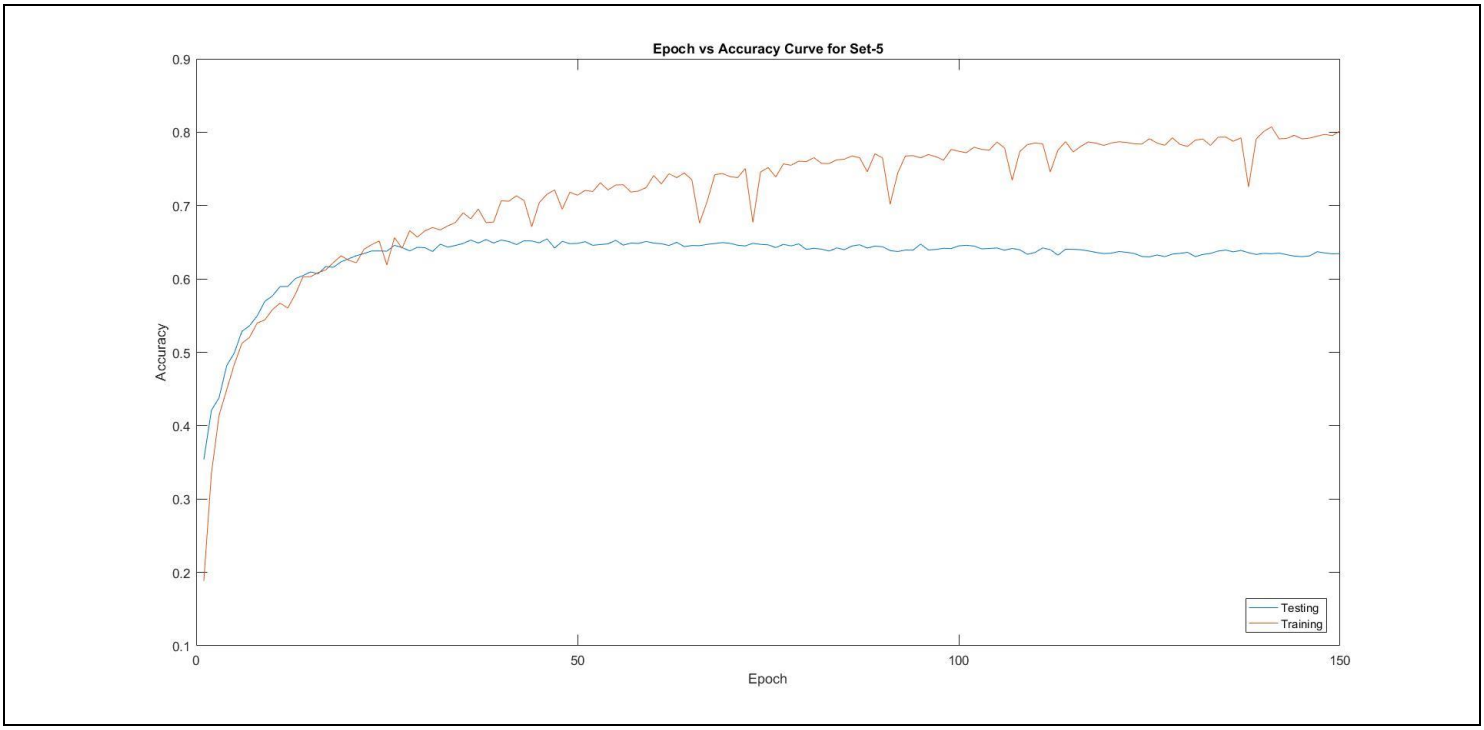

Figure-1.27: mAP curve Epoch(steps) vs training/testing data accuracy for Set-5 configuration

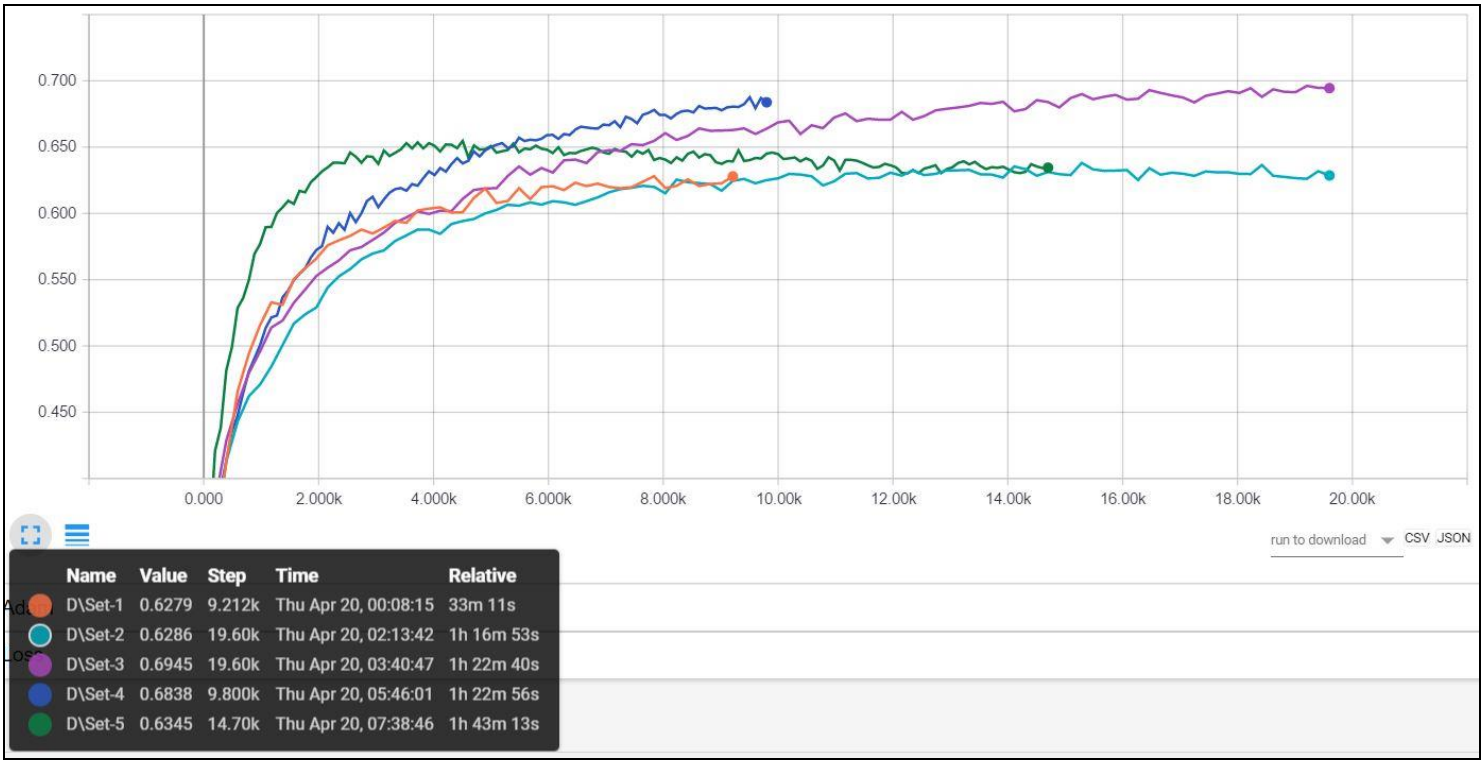

Figure-1.28: mAP curve Epoch(steps) vs testing data accuracy for Set-1 to Set-5 (comparison)

IV. Discussion

Preprocessing: Input image data was normalized by dividing with 255 to get the pixel values in range from 0 to 1. The data was zero-centered as well by subtracting the mean of each image from each pixel in itself to prevent any high frequencies to dominate the image data.

Random Weight initialization: After building the network, weights are initialized to each layer to a suitable value. The Xavier weight initialization was used for this assignment. The reason why this was used in because experimentally it has been proven to give the best results. This initializer is designed to keep the scale of the gradients roughly the same in all layers. In uniform distribution this ends up being the range: **x = sqrt(6./(in+out)); [-x, x]** and for normal distribution a standard deviation of **sqrt(3./(in+out))**.

Training and testing the CIFAR-10 dataset was achieved using brute force approach by changing the parameters. Not all parameters could be changed as it needed to be designated as a baseline CNN which would act as reference for comparison and analysis in the future sections of this assignment. The conclusions made below are based on the experimental results obtained in the section above.

1. With the Set-1 configuration where the learning rate is optimal as per scientific research studies i.e. 0.001 then with a batch size of 256 and dropout of 0.5, the convergence happens at a good pace. The accuracy increases almost linearly till a point and then oscillates at around 62% for the rest of duration. Here augmentation was not considered for which there was slight less variety in the data being trained. The testing accuracy of this setup is not the best but there is no overfitting which can be seen in the difference in the training and testing accuracy.

2. In Set-2, the learning rate was further reduced to 0.0005. It took few minutes extra as compared to Set-1 configuration, but there was no promising improvement in the testing accuracy. Here the dropout was 0.7 and still no augmentation was used. However, it can be seen from the plots that the difference between the training and testing accuracies is huge around 10%. Thus, for situations like these augmentation should have been used to reduce this difference.

3. In Set-3, just by introducing augmentation and setting the learning rate to 0.9, the accuracy of testing improved drastically. Also, the training and testing accuracy graphs were going hand in hand. An intuition that could be derived from the plots is that had the number of epochs been more, the testing accuracy could have increased significantly.

4. In Set-4, the 'xavier' weight initialization was removed and the batch size was increased to 512 as compared to Set-3 above. A similar response was observed. The accuracy was off by a percent but it did not make any significant change in the final accuracy. One more thing that was observed was for the accuracy to the level of 60s it took quite some time like around 50 epochs which in Set-3 happened in like 12 epochs i.e. the rate of convergence was slow here. A slight overfitting was seen because the testing accuracy exceeded the training accuracy by like 0.03%.

5. The Set-5 configuration proved to be the worst possible setup of all. The data augmentation was removed, learning rate increased to 0.06 and epochs were set to 150. Surprisingly, on increasing the number of epochs, the training accuracy increased but testing accuracy after a certain point began to fall gradually. This can be clearly realized in the graph of Set-5 above. Also, there was a high difference between the training and the testing accuracy i.e. in the range of around 13% and in the end the testing accuracy reaching to mere 63% which just ranked it the worst tried model in my case.

6. The spikes in the training and testing accuracy curves are due to the tradeoff between high learning rate and small batch size.

7. The training accuracy is not what matters in machine learning rather it's the testing accuracy that matters.

8. All plots were plotted on tensorboard with the smoothing set to 0 (rather than default 0.6) so that the complete accuracy curves could be studied in their true form.

9. The x-axis in all the plots indicate the steps which can be seen as Epochs as well.

c. K-means with CNNs

I. Abstract and Motivation

K-means clustering is a method of vector quantization that is popular for cluster analysis. K-means clustering aims to partition n observations into K clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells. The problem is computationally difficult, however, there are efficient heuristic algorithms that are commonly employed and converge quickly to a local optimum. K-means clustering tends to find clusters of comparable spatial extent.

The generic algorithm followed for K-means clustering is as under:
Step-1: Place K points into the space represented by the objects that are being clustered. These points represent initial group centroids.
Step-2: Assign each object to the group that has the closest centroid.
Step-3: When all objects have been assigned, recalculate the positions of the K centroids.
Step-4: Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated.

Data clustering plays a critical role in the understanding of the underlying structure of the data. The k-means algorithm, which is probably the most well-known clustering method, has been widely used in pattern recognition and supervised / unsupervised learning. Each CNN layer conducts data clustering on the surface of the high-dimensional sphere based on a rectified geodesic distance.

The way the CNN is currently initialized in by random weights using the 'xavier' weight initialization method. The task at hand is to replace this initialization by k-means based weight initialization and study its effect on the network.

II. Approach and Procedures

Theoretical Approach:

As per the professor's paper [6], it is said to apply K-means weight initialization to all the layers based on its corresponding input data samples (with zero mean and unit length normalization), and repeat this process form input to the output layer after layer. Random initialization though is generally preferred but as per [6] it has been verified that K-means initialization gives a much better result.

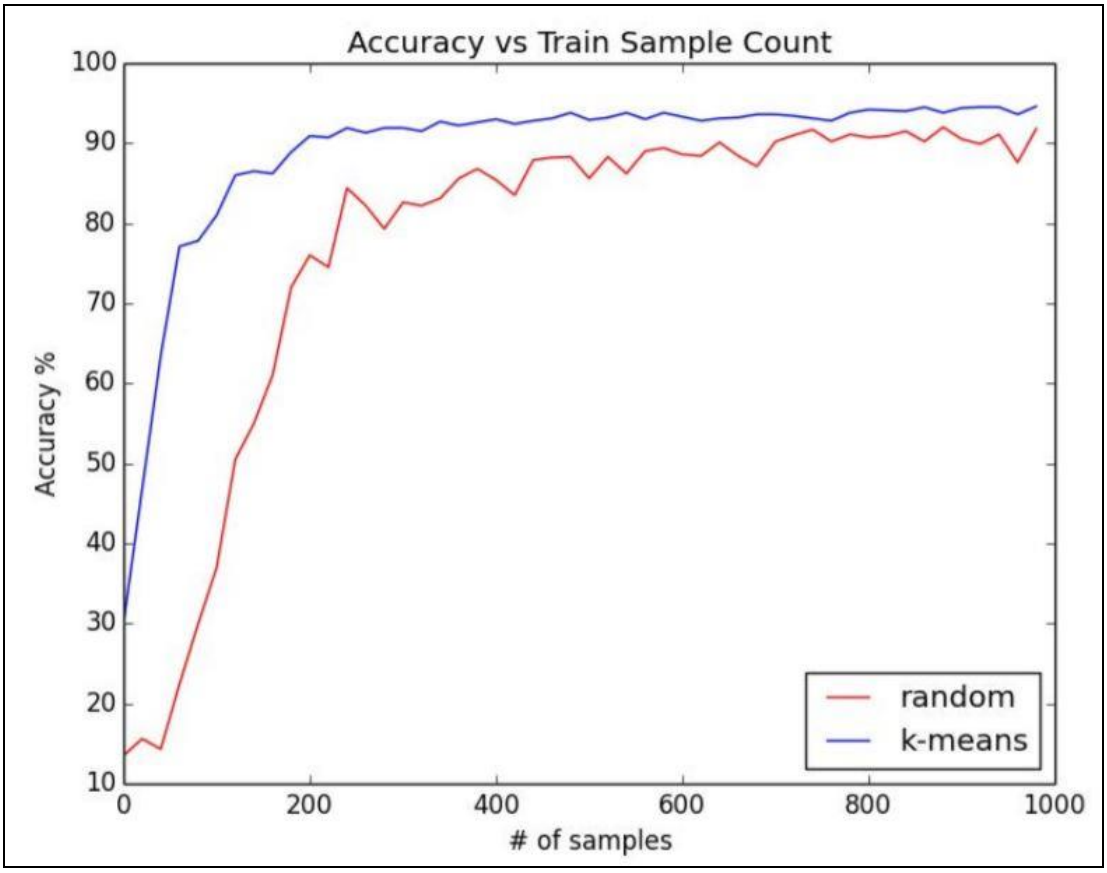

Figure-1.29: mAP curve Epoch(steps) vs testing data accuracy as per Prof's paper [6] for random and K-means

The following approach was taken for K-means weight initialization for the spatial convolution layer-1 and layer-2 for the Set-1 configuration above:

1. Extract 15 random 5x5x3 patches from each 3D input data sample over 1000 3D images.
2. Since the input data are 3D images, so for each patch, the pixel values were so arranged such that the 5x5x3 dimension of the patch was converted to a 75x1 1D vector.
3. These 1D vectors obtained were then stacked to form a matrix where the rows correspond to patches and columns correspond to pixel values.
4. The above matrix was then passed on further over which K-means was performed.
5. As per the LeNet-5 architecture in Figure-1.1, the first layer is the spatial convolution layer-1 which has 6 filter banks. So, K-means is performed with K=6.
6. After applying K-means with K=6, we will obtain 6 centroids which needs to be replaced as weights as initialization parameters to the layer.

The process from 1 to 6 needs to be repeated for the subsequent layers for weight initialization.

III. Experimental Results

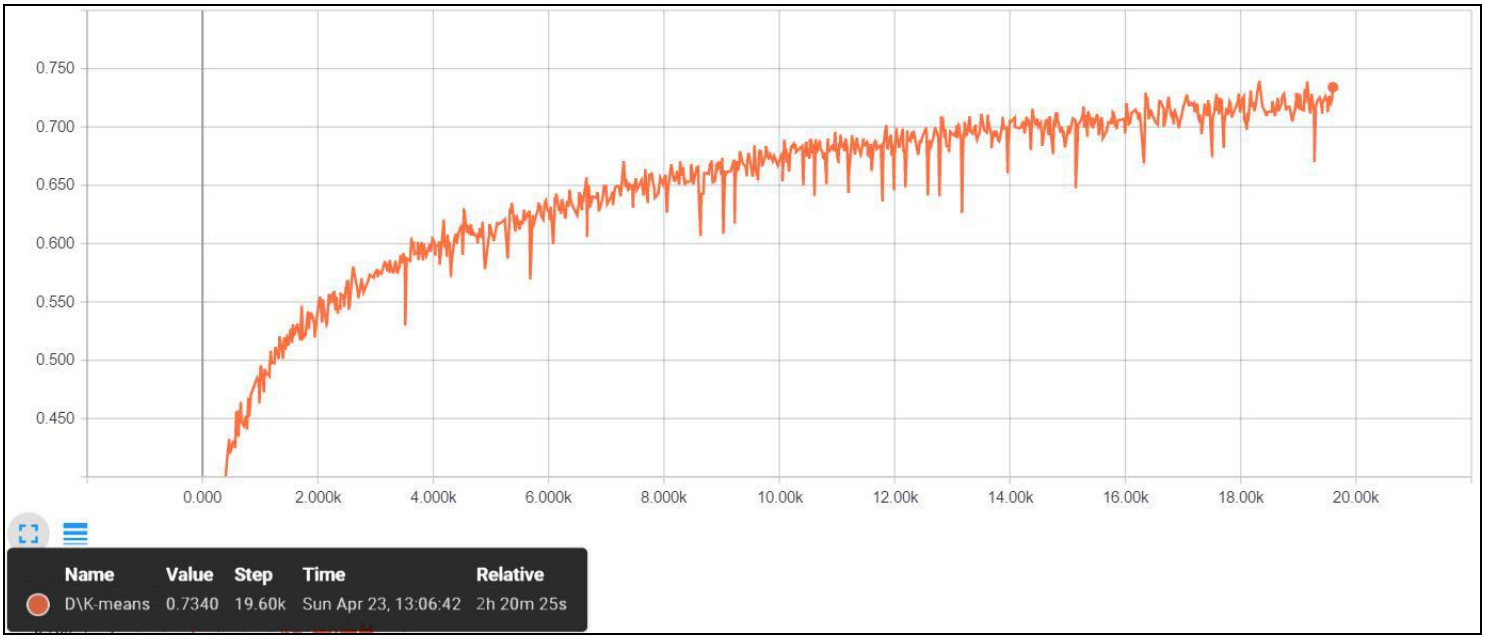

Figure-1.30: mAP curve Epoch(steps) vs testing data accuracy for K-means configuration (73.40%)

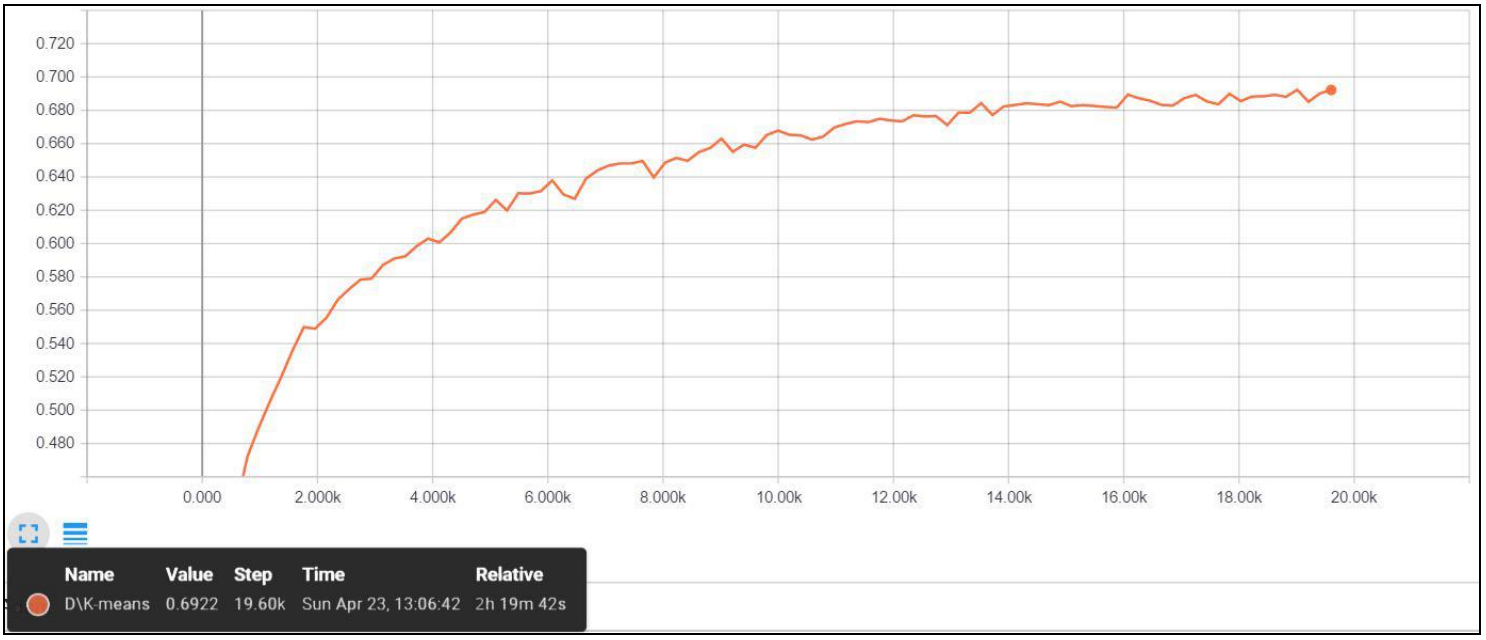

Figure-1.31: mAP curve Epoch(steps) vs testing data accuracy for K-means configuration (69.22%)

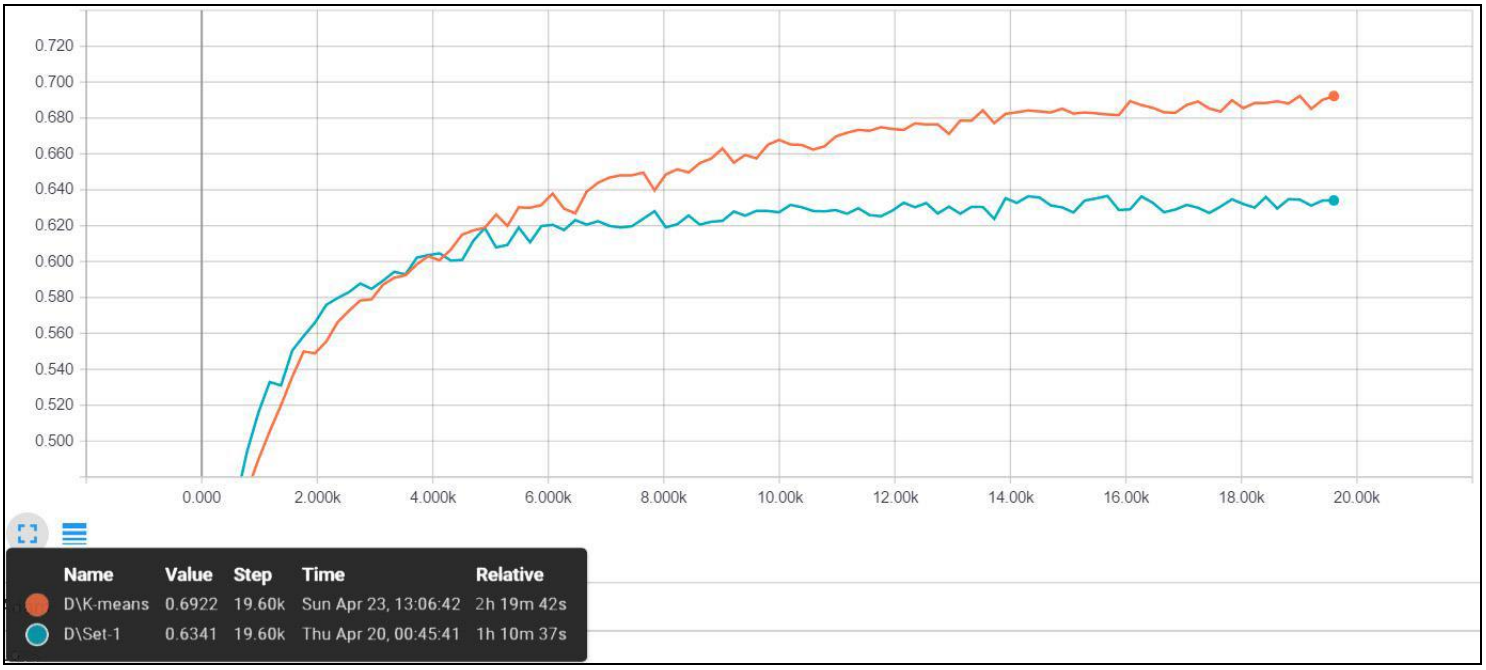

Figure-1.32: mAP curve Epoch(steps) vs testing data accuracy comparison curves (Set-1 vs K-means Set-1)
Set-1 has Epoch = 100
K-means has Epoch = 100

IV. Discussion

The following observations were made on applying the K-means weight initialization to the Set-1 configuration for spatial convolution layers 1 and 2:

1. The testing accuracy improved drastically from around 63% to 69%. Once we feed the test data to the network (an unsupervised learning methodology), it was observed that the K-means initialization provided 10 anchor vectors pointing at 10 different images while the random initialization gives different anchor vectors based on the randomization, sometimes the worst being multiple anchor vectors pointing to the same image.

2. As per professors paper [6], the accuracy expected should be an increase by 3% to 14% compared to the baseline. The one that I got is 6% which is not a bad output.

3. The process was very time consuming because apart from normal convolution as there was an iterative K-means being performed before each of the spatial convolution layer, which itself took time.

4. The spikes in the training and testing accuracy curves are due to the tradeoff between high learning rate and small batch size.

5. The training accuracy is not what matters in machine learning rather it's the testing accuracy that matters.

6. All plots were plotted on tensorboard with the smoothing set to 0 (rather than default 0.6) so that the complete accuracy curves could be studied in their true form.

7. The x-axis in all the plots indicate the steps which can be seen as Epochs as well.

8. I did not use the K-means for weight initialization in the rest of my improvement algorithm because I was able implement it just a couple of days before deadline and repeating the whole process for all my improvement algorithm would not have completed on time before assignment deadline. I had already done my improvement algorithm using 'xavier' random weight initialization.

Problem-2: Capability and Limitation of Convolutional Neural Networks

a. Improving Your Network for CIFAR-10 Dataset

I. Abstract and Motivation

The task is to improve the CIFAR-10 dataset classification by modifying the baseline CNN to improve the classification accuracy. In contradiction to the above section, here the liberty of tweaking any parameters is there, like adding layers, changing activation function, optimization, loss function, number of filters, augmentation and fine tuning some training parameters.

II. Approach and Procedures

Multiple approaches were taken for achieving an increased accuracy. Various parameters were played with by brute force and observing earlier recorded results which gave an intuition of what needs to be tweaked. With every run a visual inspection gave an intuition as to what could be changed to achieve a better result. The various tweaks made around with the architecture to get an improved result are as under:

1. Set-6 network architecture:

Taking the baseline CNN given to us as reference as per Figure-1.1, the following changes were made:

- The spatial convolution layer-1 had 24 filter banks
- The spatial convolution layer-2 had 64 filter banks
- Dropout after each fully connected layer was 0.9
- The fully connected layer-1 had 240 neurons
- The fully connected layer-2 had 160 neurons
- The number of epochs was 74
- The batch size was 1000

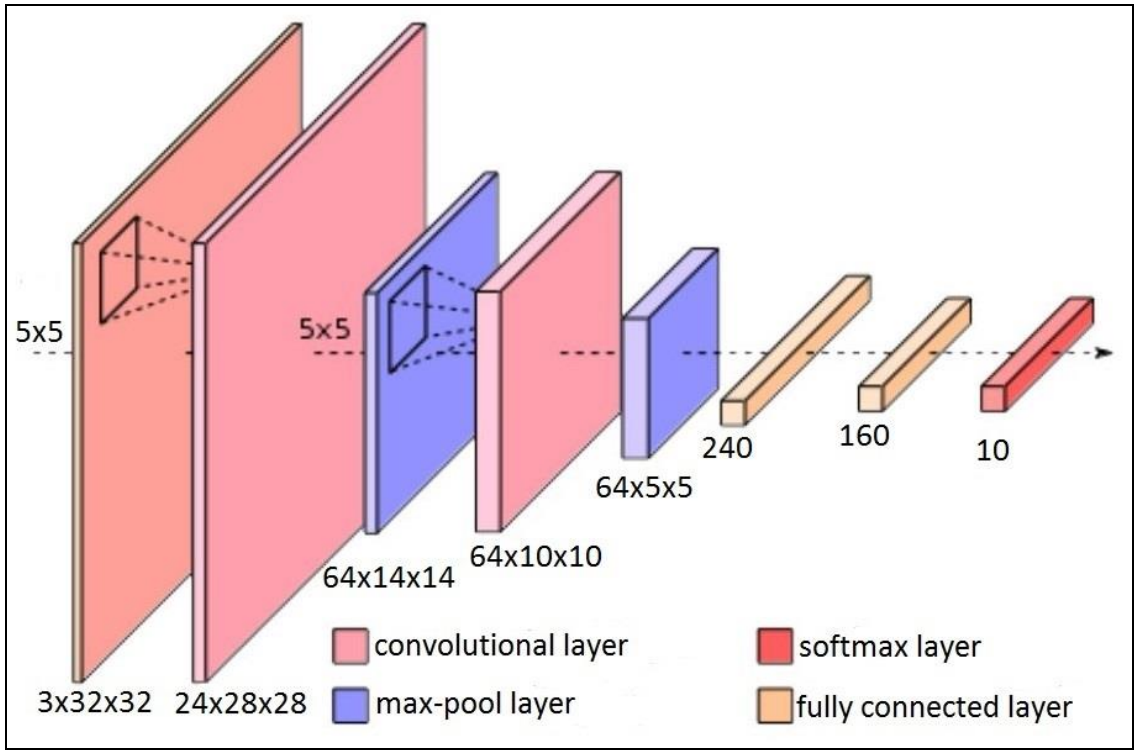

Figure-2.1: Set-6 CNN architecture

2. **Set-7 network architecture:**

Taking the baseline CNN given to us as reference as per Figure-1.1, the following changes were made:

- The spatial convolution layer-1 had 24 filter banks with activation function being Leaky ReLU
- The spatial convolution layer-2 had 64 filter banks with activation function being Leaky ReLU
- Dropout after each fully connected layer was 0.9
- The fully connected layer-1 had 240 neurons
- The fully connected layer-2 had 160 neurons
- The number of epochs was 100
- The batch size was 256

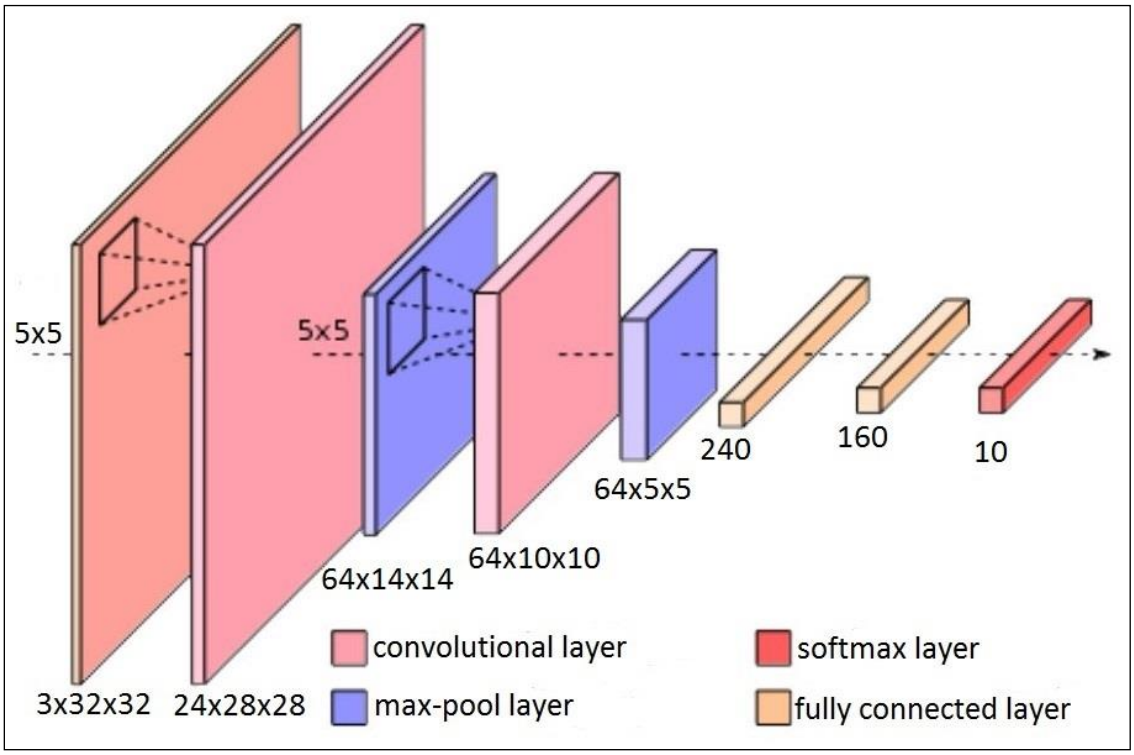

Figure-2.2: Set-7 CNN architecture

3. **Set-8 network architecture:**

Taking the baseline CNN given to us as reference as per Figure-1.1, the following changes were made:

- The spatial convolution layer-1 had 32 filter banks with activation function being Leaky ReLU
- The spatial convolution layer-2 had 64 filter banks with activation function being Leaky ReLU
- Dropout after each fully connected layer was 0.7
- The fully connected layer-1 had 360 neurons
- The fully connected layer-2 had 240 neurons
- The number of epochs was 100
- The batch size was 256

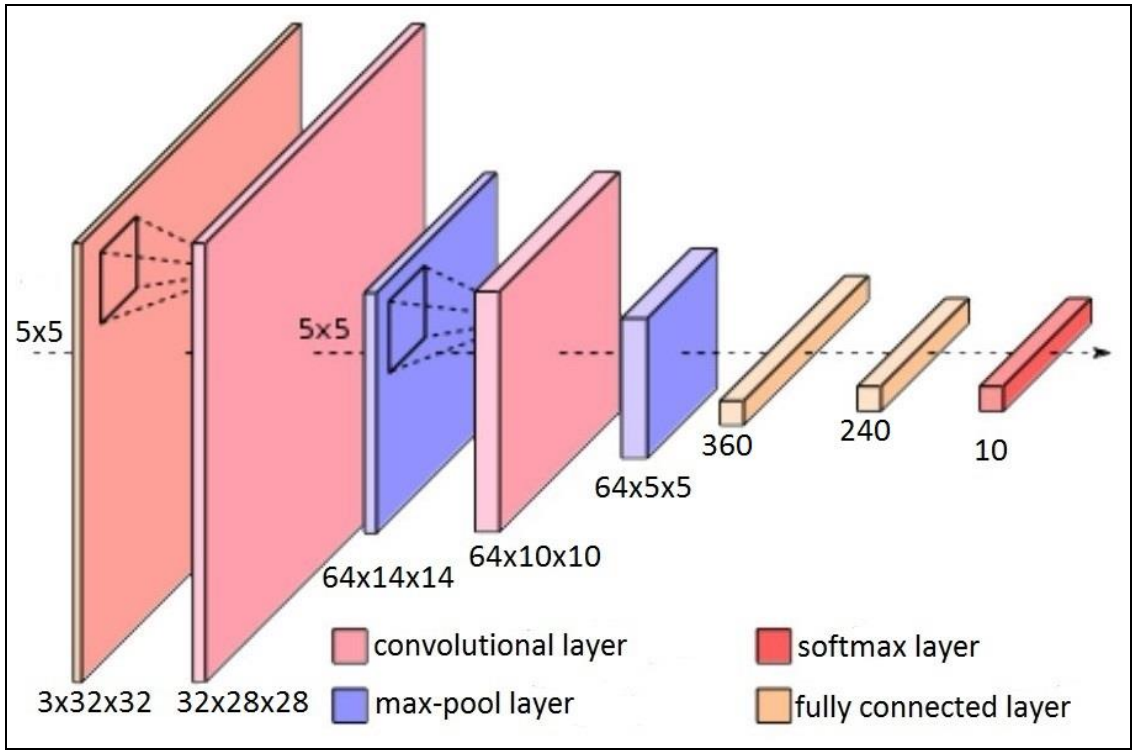

Figure-2.3: Set-8 CNN architecture

4. **Set-9 network architecture:**

The configuration that I considered is out of curiosity of trying out paper [5] in a smaller scale to see how the network behaved. It sure was computationally heavy and time consuming but it was the best result I got. The following was its configuration:

- The spatial convolution layer-1 had 96 filter banks, receptive field 5x5, activation function being Leaky ReLU, initial weights 'xavier' and no zero padding
- The spatial convolution layer-2 had 96 filter banks, receptive field 1x1, activation function being Leaky ReLU, initial weights 'xavier' and no zero padding
- The max pool layer 1 had 3x3 window size, 2 strides and no zero padding
- A network dropout of 0.5 was set
- The spatial convolution layer-3 had 192 filter banks, receptive field 5x5, activation function being Leaky ReLU, initial weights 'xavier' and no zero padding
- The spatial convolution layer-4 had 192 filter banks, receptive field 1x1, activation function being Leaky ReLU, initial weights 'xavier' and no zero padding
- The max pool layer 2 had 3x3 window size, 2 strides and no zero padding
- A network dropout of 0.5 was set
- The spatial convolution layer-5 had 192 filter banks, receptive field 3x3, activation function being Leaky ReLU, initial weights 'xavier' and no zero padding
- The spatial convolution layer-6 had 192 filter banks, receptive field 1x1, activation function being Leaky ReLU, initial weights 'xavier' and no zero padding
- The spatial convolution layer-7 had 10 filter banks, receptive field 1x1, activation function being Leaky ReLU, initial weights 'xavier' and no zero padding
- Global average pooling was applied over the network
- The output layer had 10 neurons representing the 10 classes and activation function used was softmax.

III. Experimental Results

Table-2.1: Table showing the summary of experiments conducted for improvement

Variations	mAP Training Accuracy (%)	mAP Testing Accuracy (%)	Time-taken
Set-6	83.24	77.25	2h 0m 24s
Set-7	90.64	78.16	2h 52m 45s
Set-8	88.91	80.62	5h 41m 38s
Set-9	87.31	84.58	21h 24m 24s

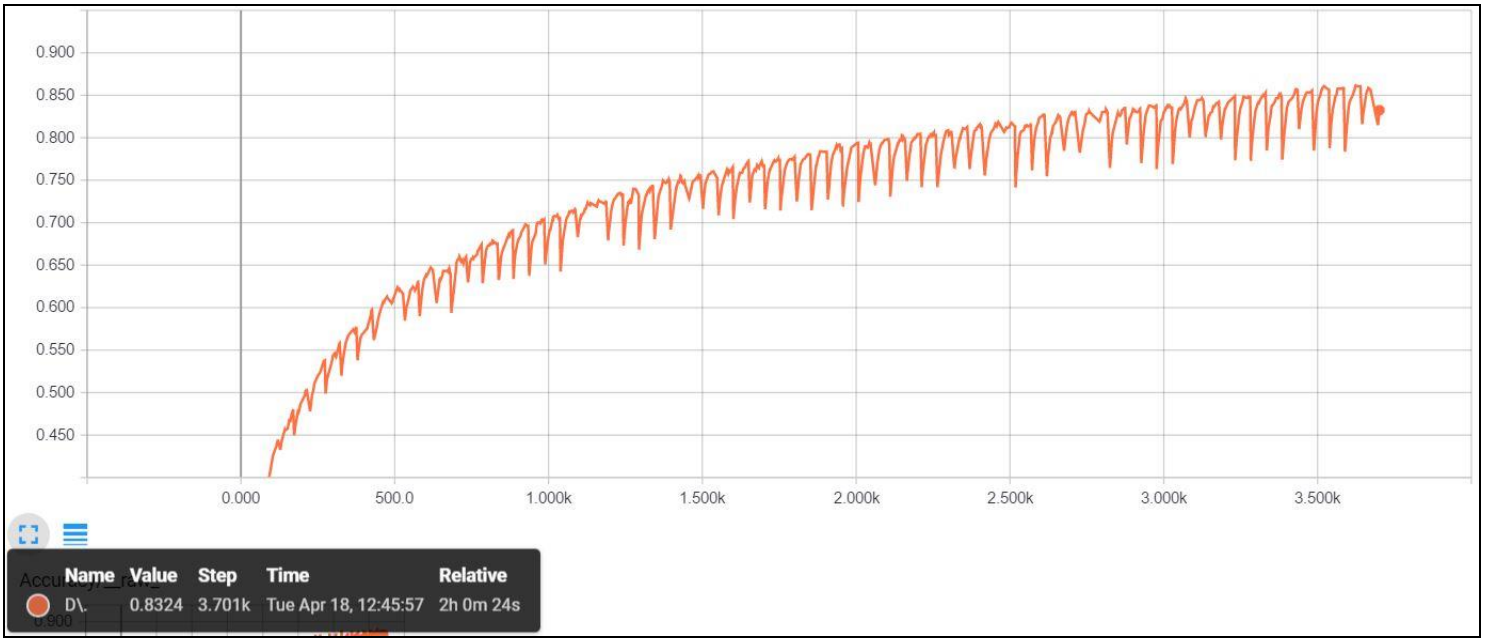

Figure-2.4: mAP curve Epoch(steps) vs training data accuracy for Set-6 configuration (83.24%)

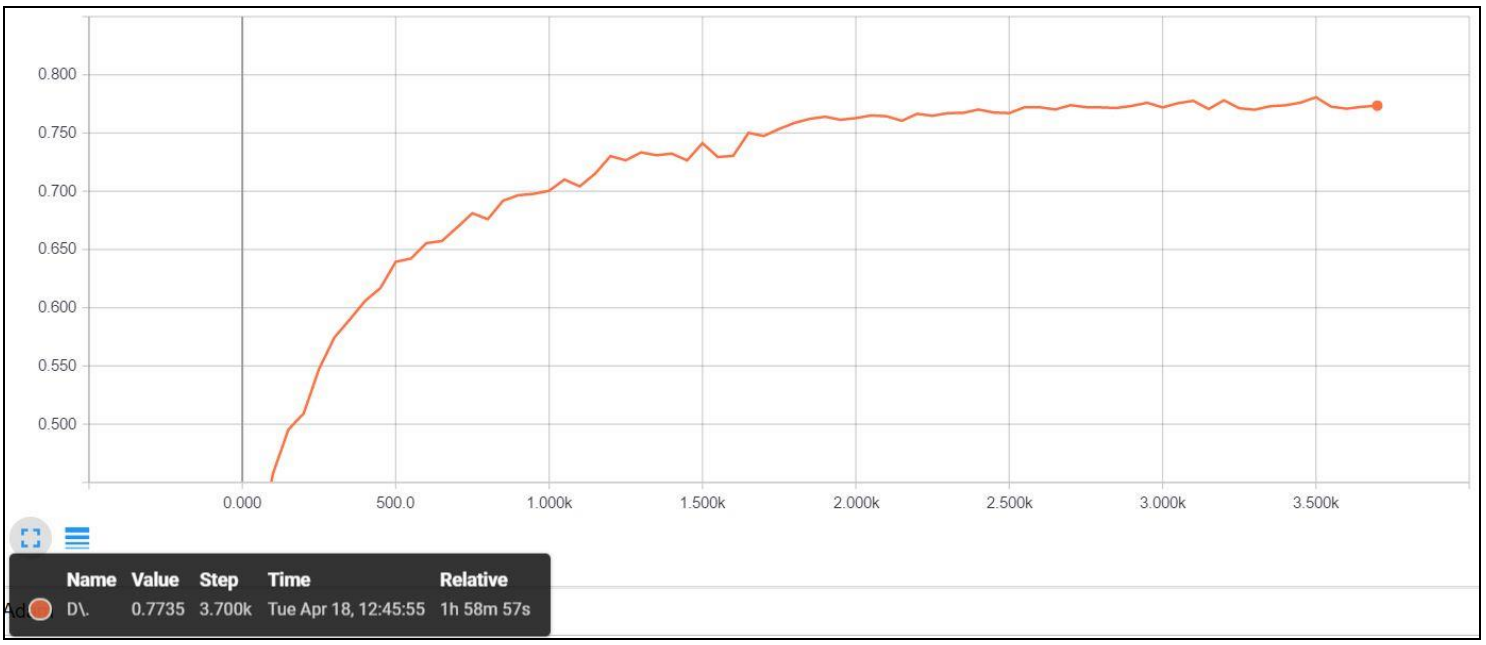

Figure-2.5: mAP curve Epoch(steps) vs testing data accuracy for Set-6 configuration (77.35%)

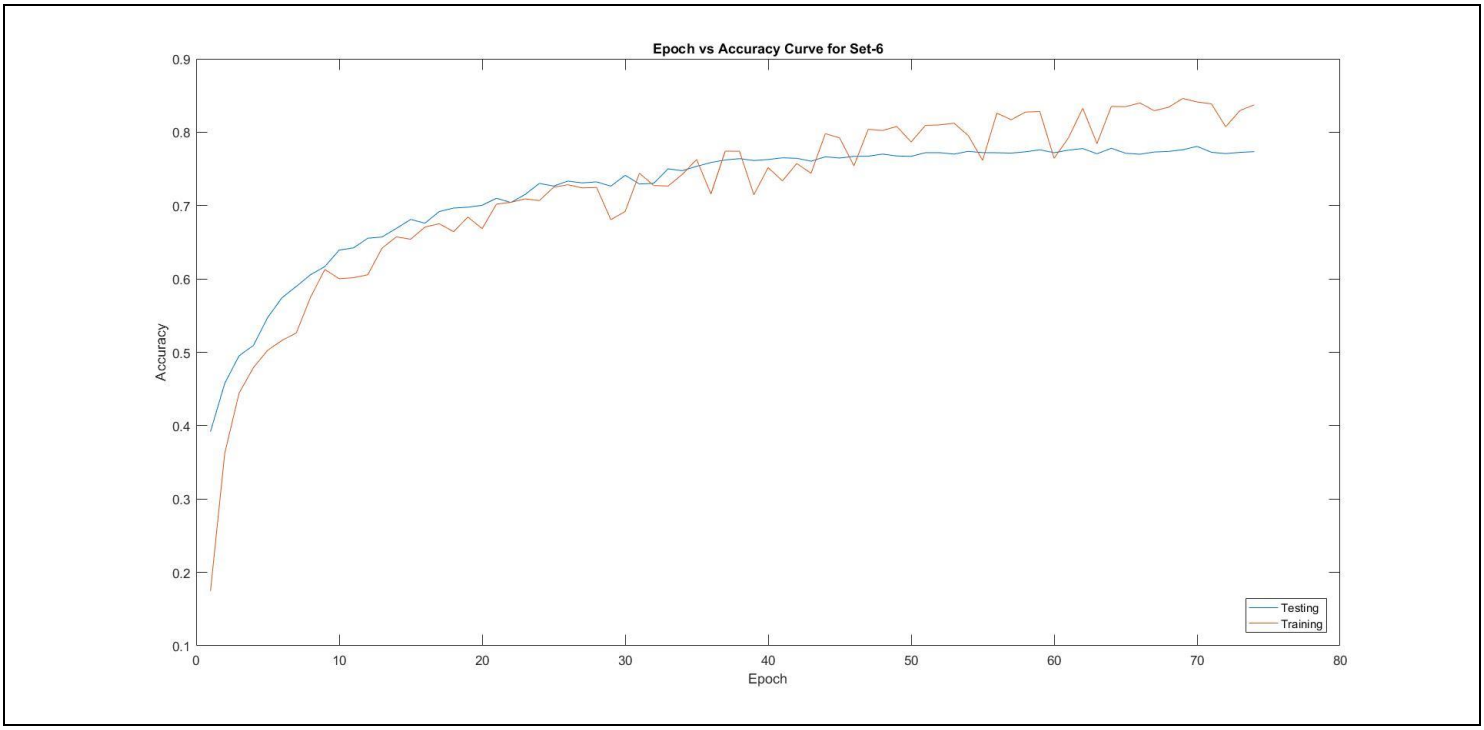

Figure-2.6: mAP curve Epoch(steps) vs training/testing data accuracy for Set-6 configuration

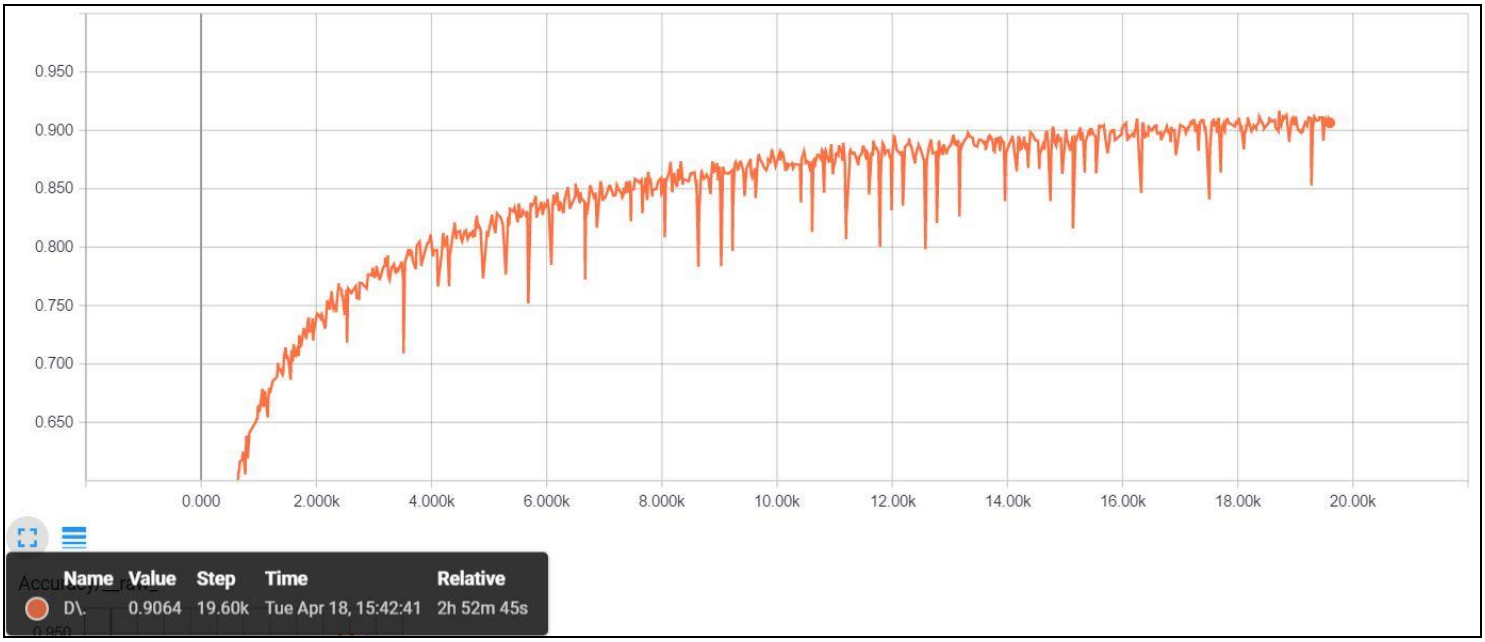

Figure-2.7: mAP curve Epoch(steps) vs training data accuracy for Set-7 configuration (90.64%)

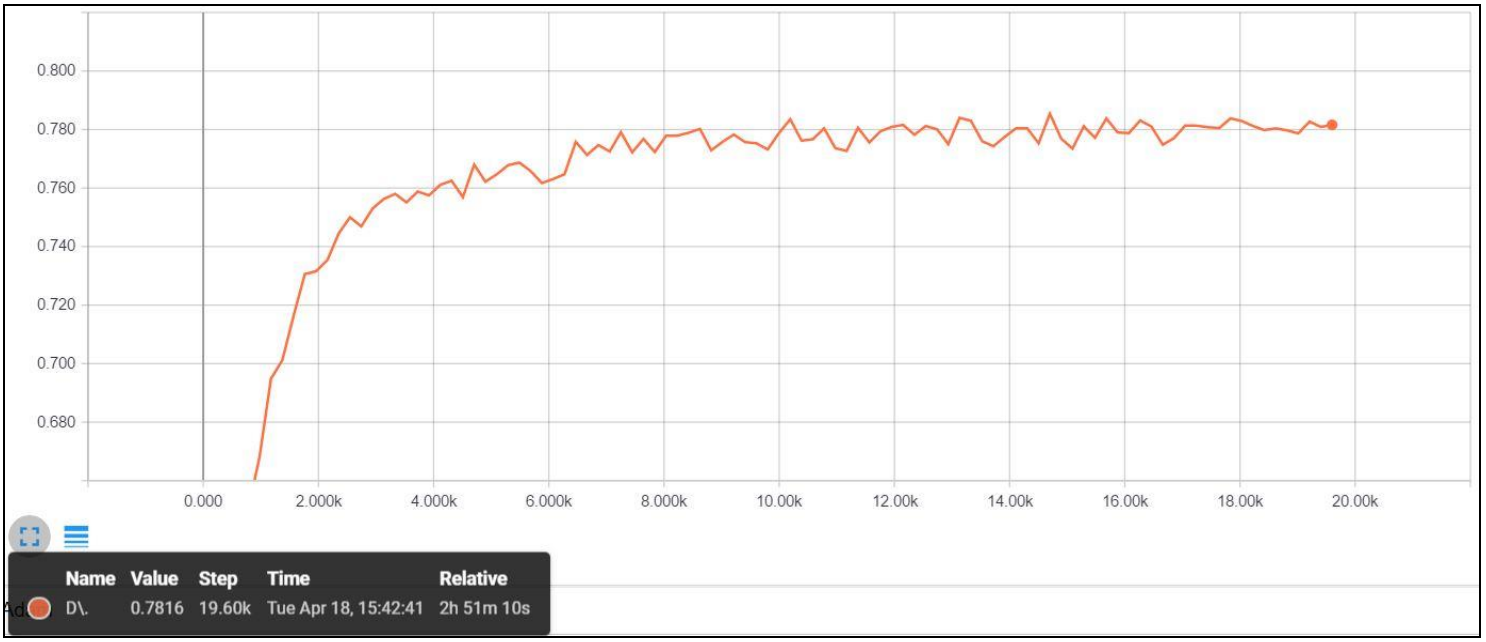

Figure-2.8: mAP curve Epoch(steps) vs testing data accuracy for Set-7 configuration (78.16%)

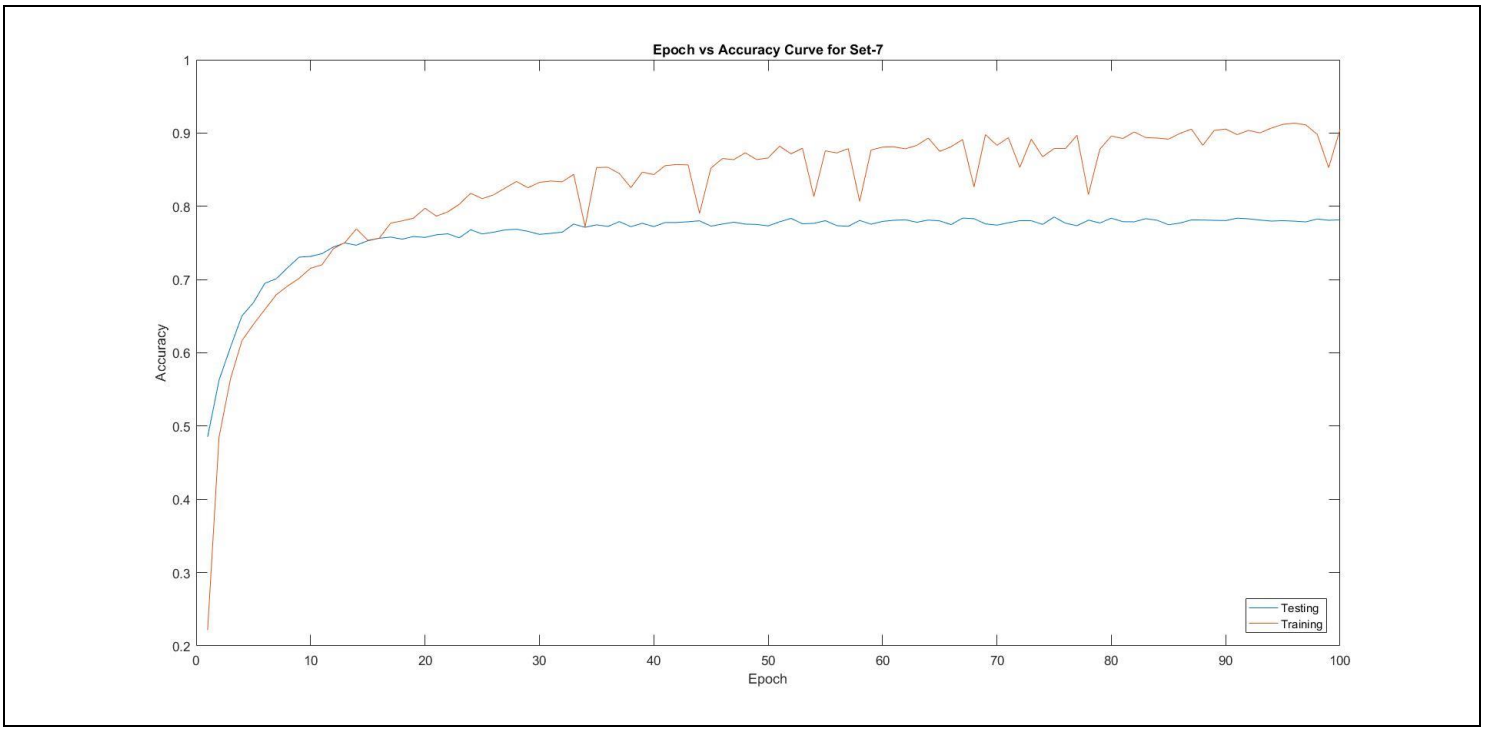

Figure-2.9: mAP curve Epoch(steps) vs training/testing data accuracy for Set-7 configuration

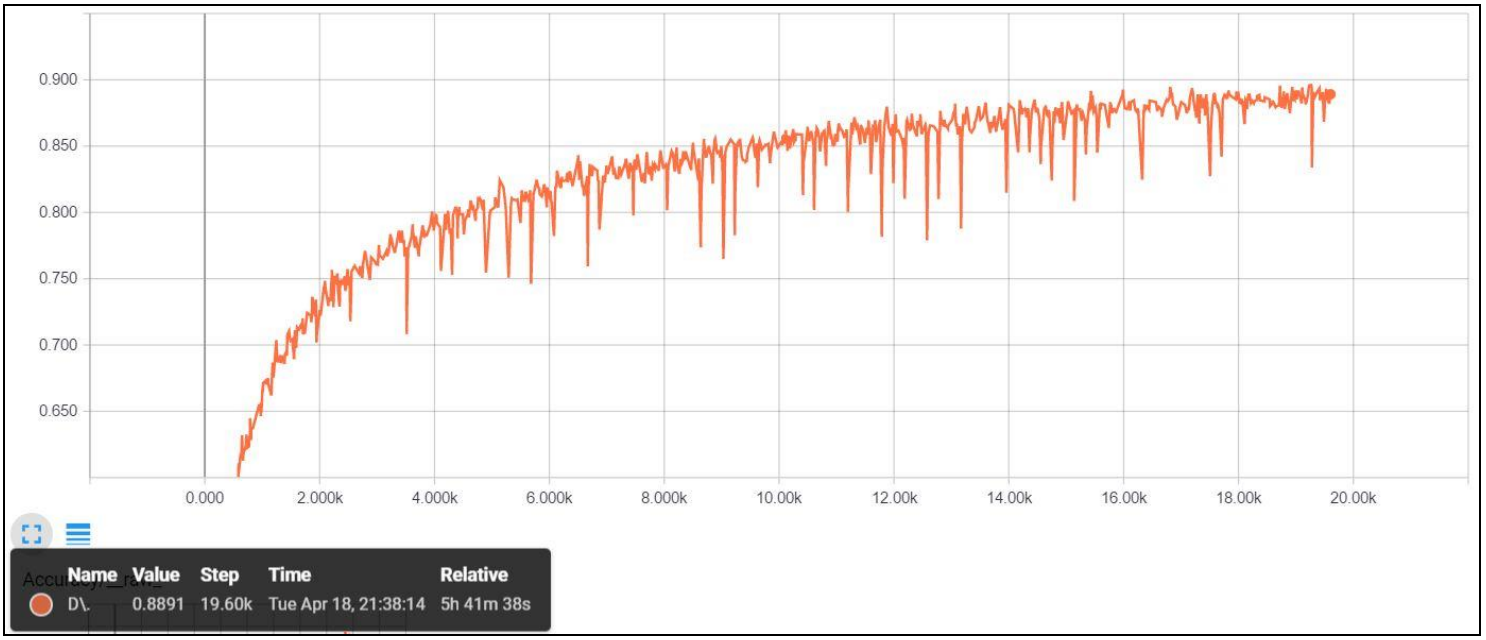

Figure-2.10: mAP curve Epoch(steps) vs training data accuracy for Set-8 configuration (88.91%)

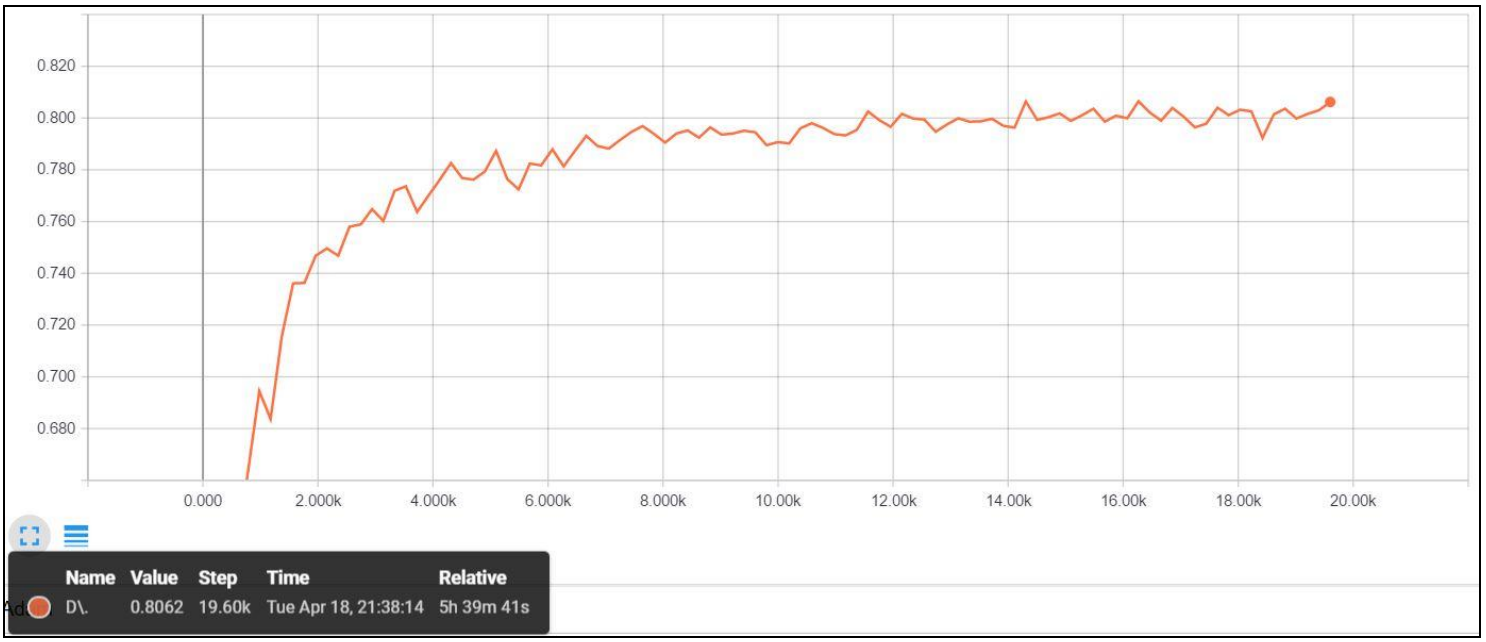

Figure-2.11: mAP curve Epoch(steps) vs testing data accuracy for Set-8 configuration (80.62%)

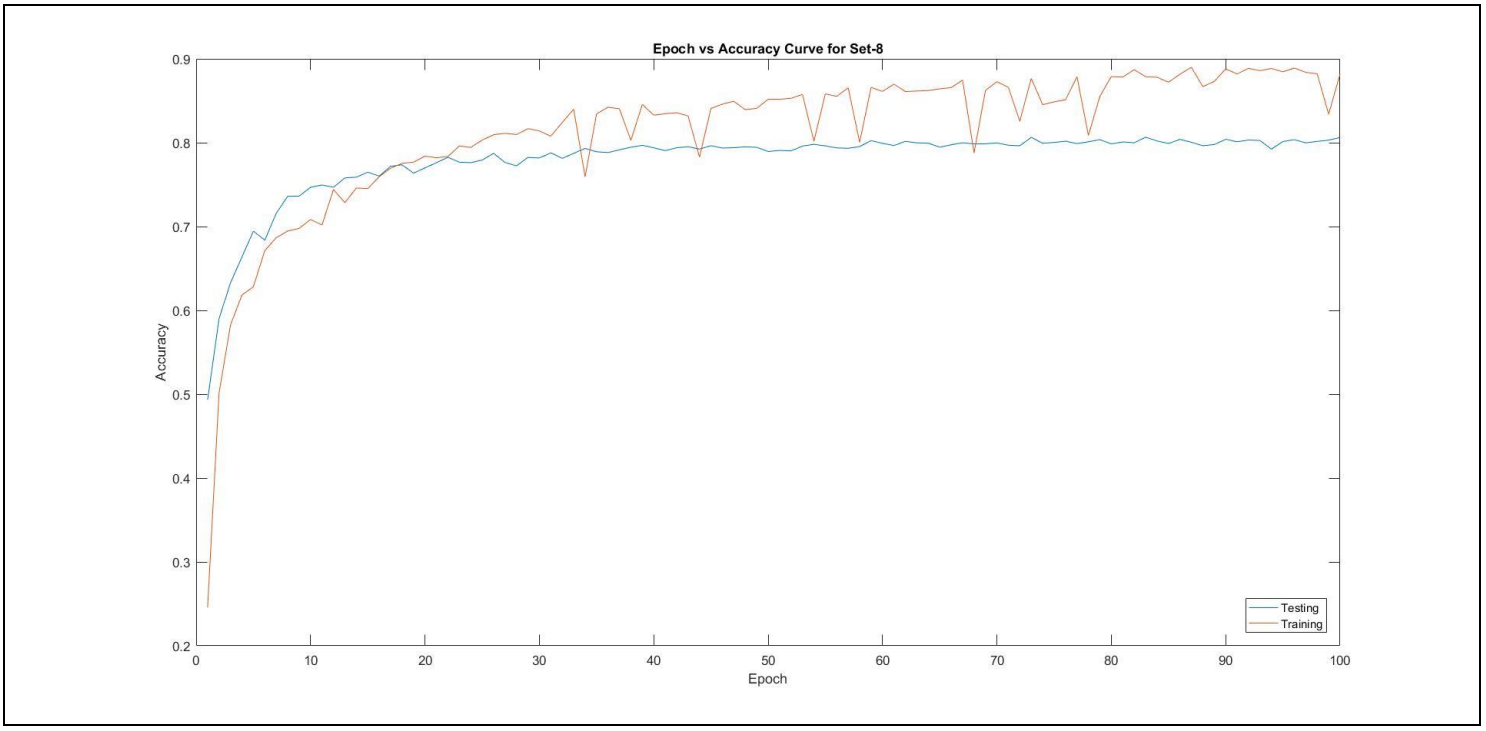

Figure-2.12: mAP curve Epoch(steps) vs training/testing data accuracy for Set-8 configuration

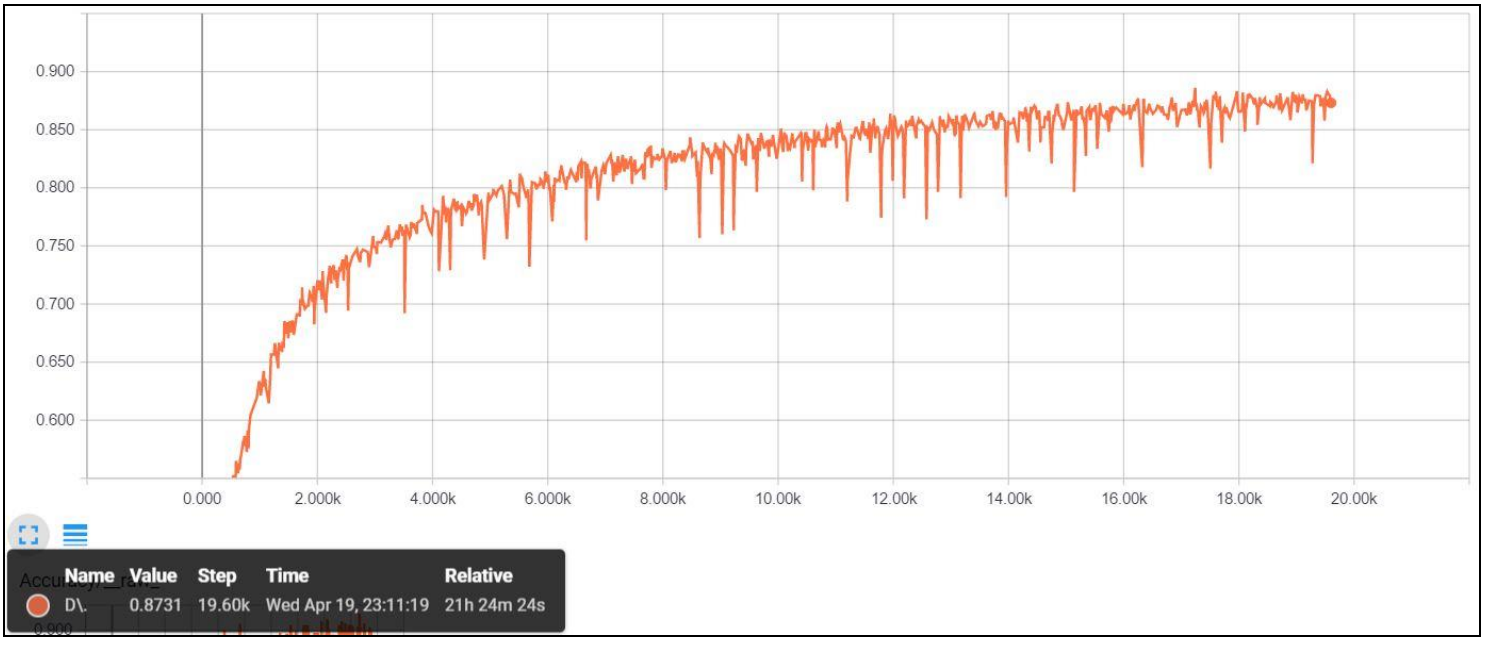

Figure-2.13: mAP curve Epoch(steps) vs training data accuracy for Set-9 configuration (87.31%)

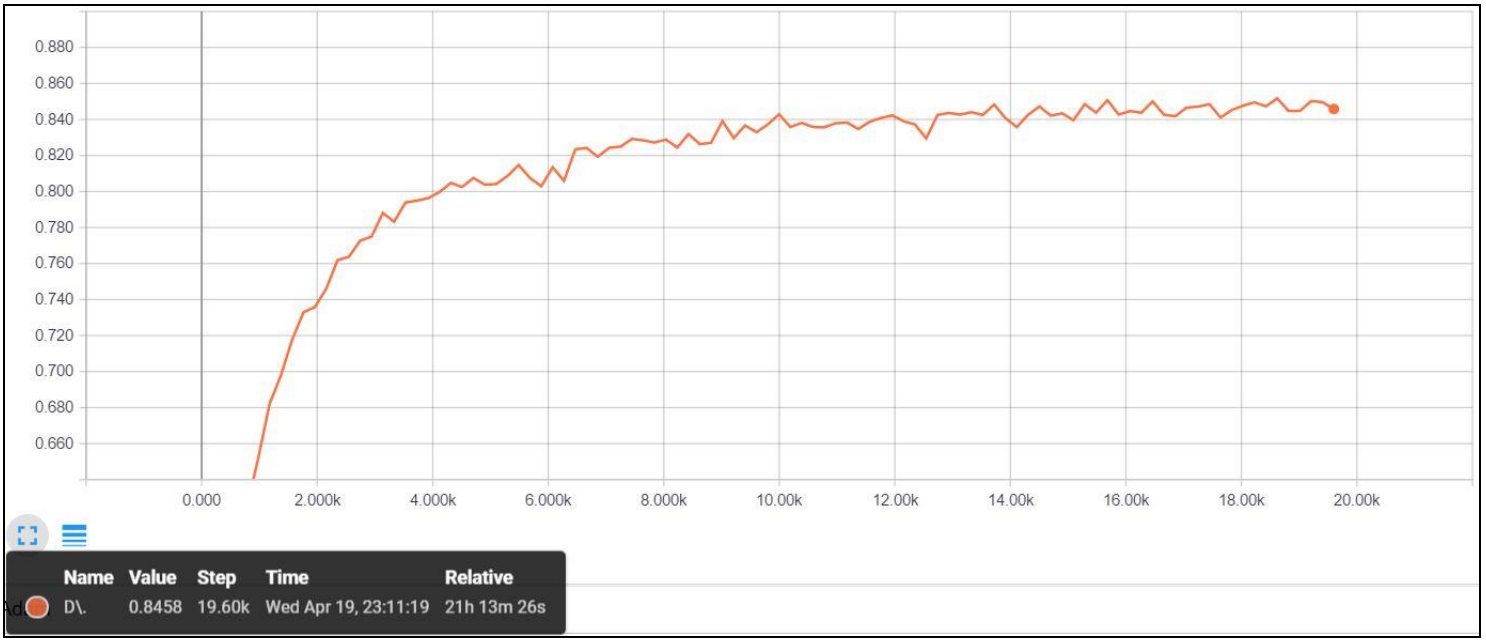

Figure-2.14: mAP curve Epoch(steps) vs testing data accuracy for Set-9 configuration (84.58%)

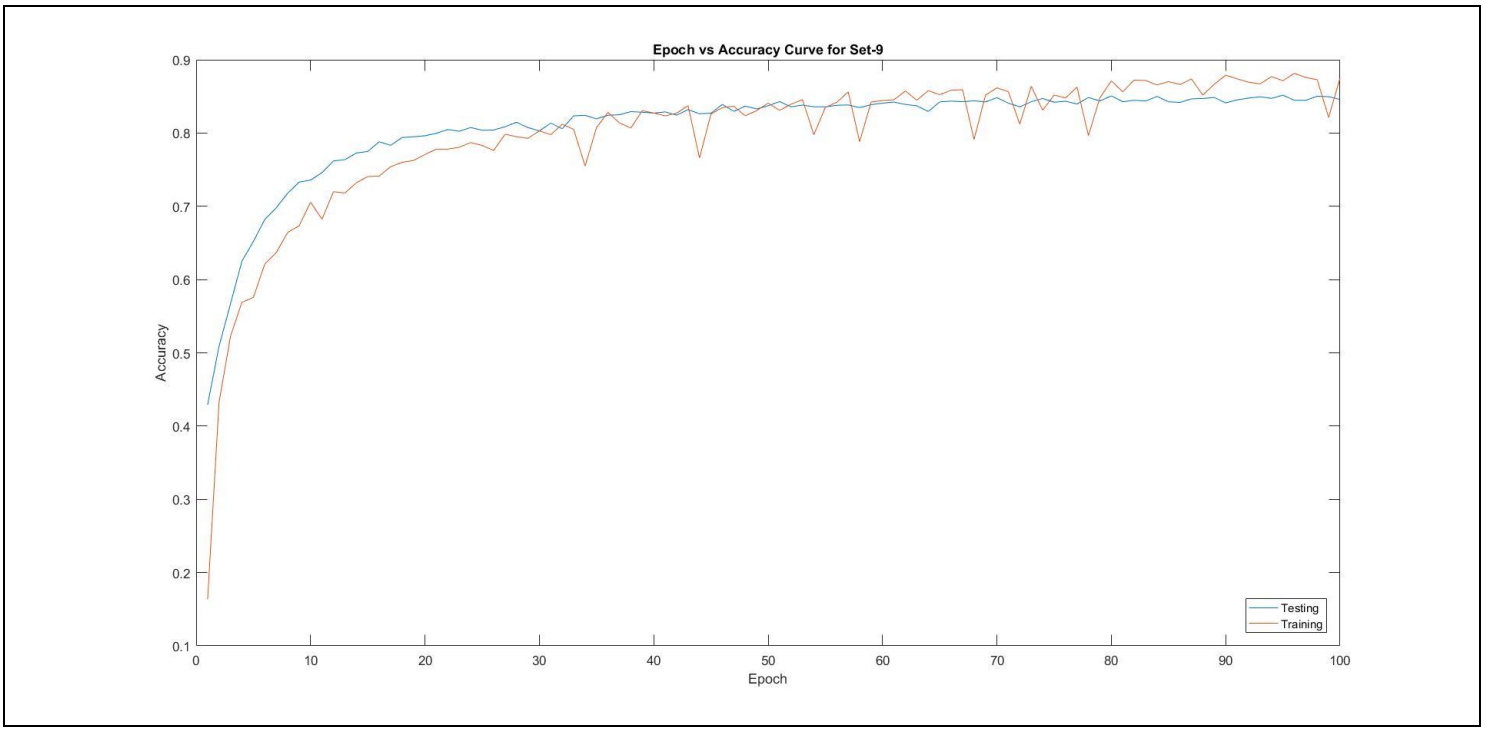

Figure-2.15: mAP curve Epoch(steps) vs training/testing data accuracy for Set-9 configuration

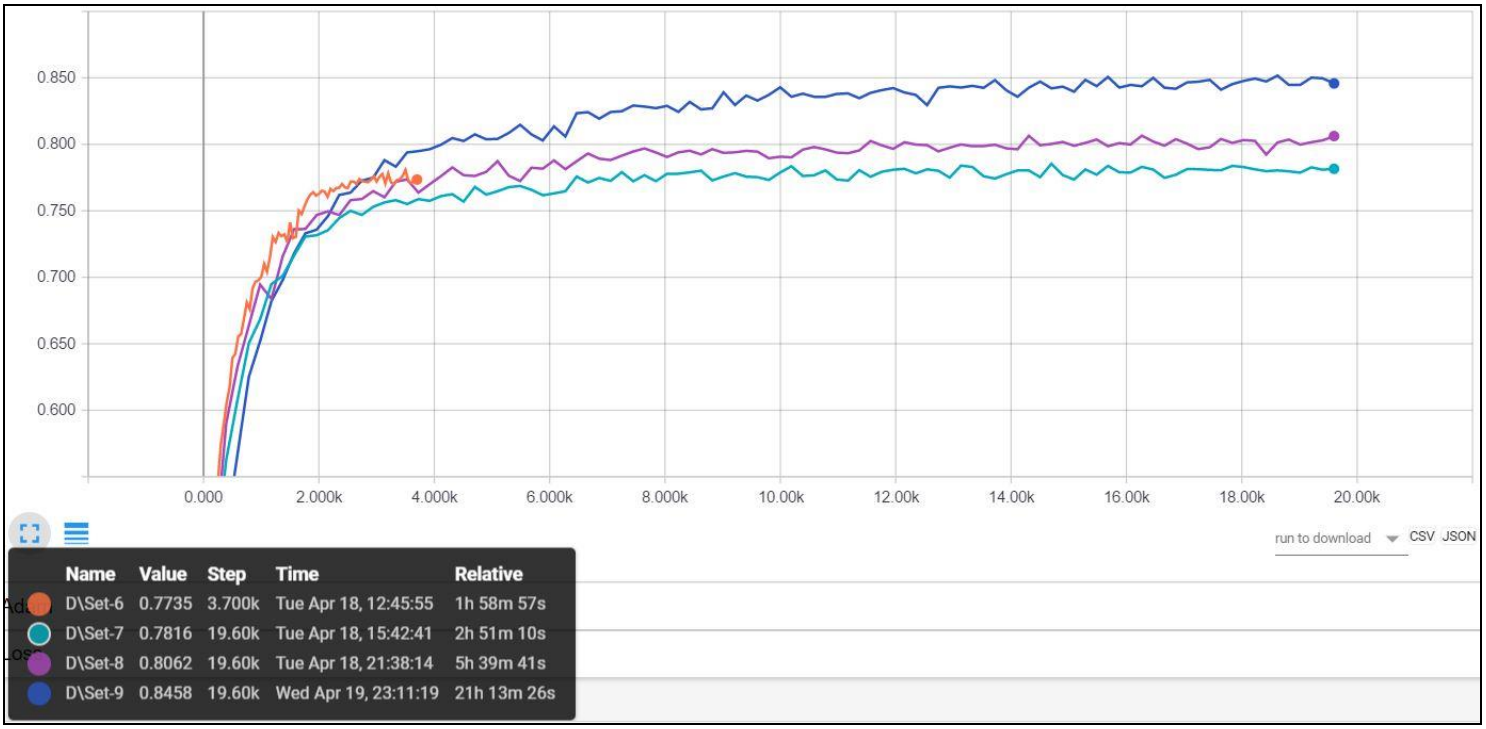

Figure-2.16: mAP curve Epoch(steps) vs testing data accuracy for Set-6 to Set-9 configuration (comparison)

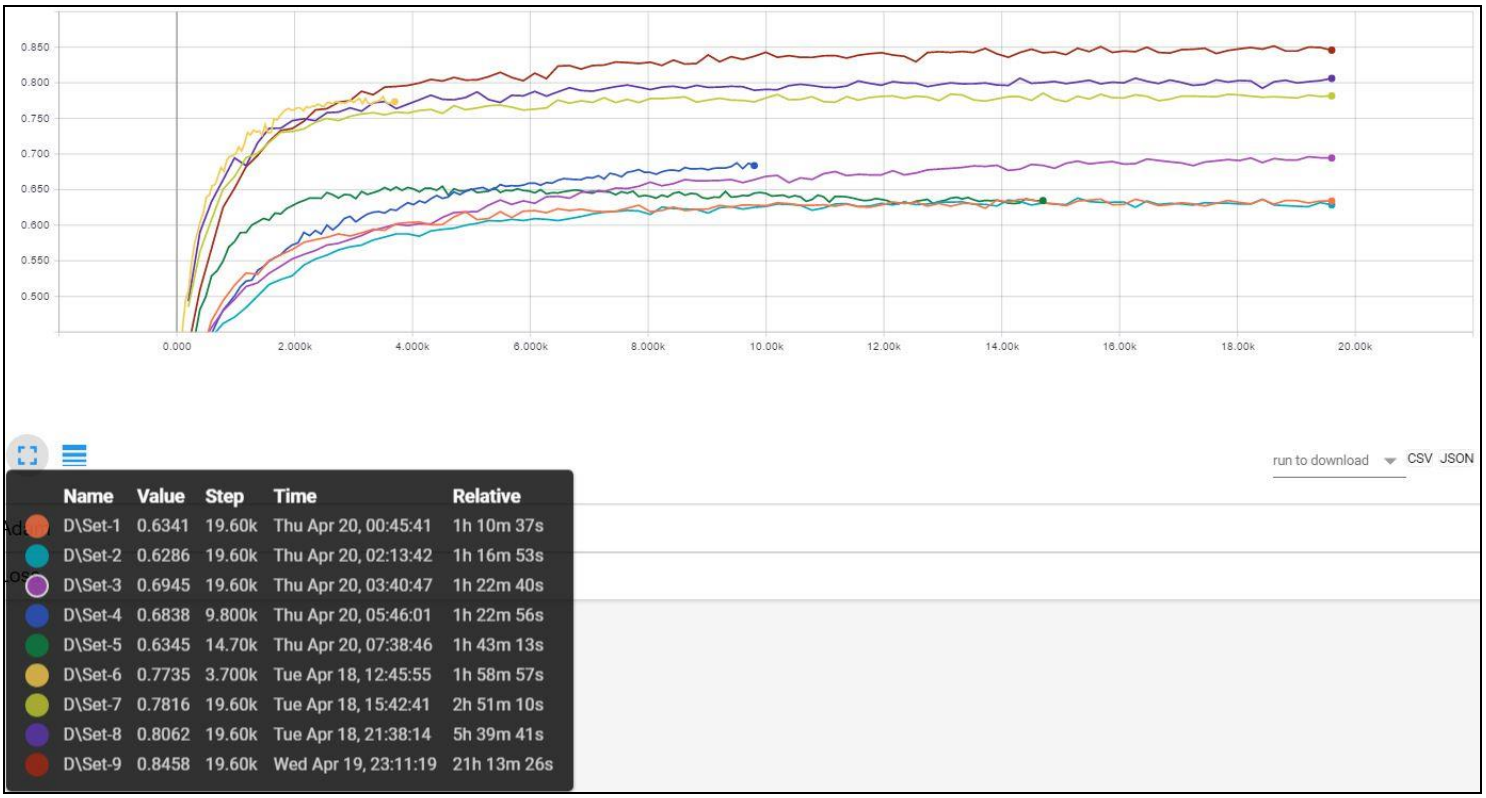

Figure-2.17: mAP curve Epoch(steps) vs testing data accuracy for Set-1 to Set-9 configuration (comparison)

IV. Discussion

In the previous section, training and testing the CIFAR-10 dataset was achieved using brute force approach by changing just the parameters without affecting any layer level changes. In this section with the freedom to play around with anything, the above 4 sets (set-6 to set-9) were selected to test for improvement. The configurations were obtained after careful studying of various papers from leaderboard [4] and some out of intuitive understanding while tweaking around with the base line CNN in section 1. The conclusions made below are based on the experimental results obtained in the experimental results section above.

1. All the sets considered above gave a very good testing curve.

2. Out of the four, I would say that Set-9 was the best improvement I achieved. It involved a lot of layers and computation time was high, but nevertheless the curve obtained in the end showed very good classification. From a meagre 65% to a whopping 84% just shows the level of improvement that was achieved. Another thing to note was the rate at which the accuracy continued to increase, gave a good estimate that had the epoch been larger, certain level of accuracy could have been improved further from the testing side. The plot of Set-9 above gives a clear visual feedback as well.

3. Set-6 to Set-8 were just the repetition of the baseline CNN but with high values of filters which helped in increasing the accuracy from 65% to 75%.

9. The spikes in the training and testing accuracy curves are due to the tradeoff between high learning rate and small batch size.

10. The training accuracy is not what matters in machine learning rather it's the testing accuracy that matters.

11. All plots were plotted on tensorboard with the smoothing set to 0 (rather than default 0.6) so that the complete accuracy curves could be studied in their true form.

12. The x-axis in all the plots indicate the steps which can be seen as Epochs as well.

b. State-of-the-Art CIFAR-10 Implementation

I. Abstract and Motivation

The task to study from [4] one paper and compare and contrast it with the baseline CNN approach taken and the improved CNN approach taken for the CIFAR-10 dataset. The Leaderboard in [4] has some good results which were achieved using systems with very high end processors and GPUs. It took days for achieving such results and tremendous amount of brute force to get to such optimality.

After carefully studying few of them I chose to compare my results with the second paper [5] in the Leaderboard [4]. The paper is titled "Striving for Simplicity: The All Convolutional Net".

II. Discussion

Q1. Describe what the authors did to achieve such a result. You do not have to implement the network.

Ans: As the name clearly suggests the authors purely use the Spatial Convolutional Layer to get to the level of accuracy they have achieved. The authors used three different architectures to achieve the result. They divided them into three models i.e. Model-A, Model-B, Model-C as shown below:

Model		
A	B	C
Input 32 × 32 RGB image		
5 × 5 conv. 96 ReLU	5 × 5 conv. 96 ReLU 1 × 1 conv. 96 ReLU	3 × 3 conv. 96 ReLU 3 × 3 conv. 96 ReLU
3 × 3 max-pooling stride 2		
5 × 5 conv. 192 ReLU	5 × 5 conv. 192 ReLU 1 × 1 conv. 192 ReLU	3 × 3 conv. 192 ReLU 3 × 3 conv. 192 ReLU
3 × 3 max-pooling stride 2		
3 × 3 conv. 192 ReLU		
1 × 1 conv. 192 ReLU		
1 × 1 conv. 10 ReLU		
global averaging over 6 × 6 spatial dimensions		
10 or 100-way softmax		

Figure-2.18: The three base networks used by the authors to classify CIFAR-10 [5]

Starting from Model-A, the depth of the number of parameters in the network gradually increases to Model-C. All base networks were considered to use a 1x1 convolution at the top to produce 10 outputs of which then an average was computed over all positions and a softmax to produce class probabilities. They also performed additional experiments with fully connected layers instead of 1x1 convolutions but found that they performed 0.5% to 1% worse than their fully convolutional counterparts. In Model-B, only one 1x1 convolution is performed after each normal convolution layer. The Model-C just replaces all 5x5 convolutions by simple 3x3 convolutions for serving two purposes i.e. it unifies the architecture to consist only of layers operating on 3x3 spatial neighborhoods of the previous layer feature map (with occasional subsampling) and the second being if max-pooling was replaced by a convolutional layer then 3x3 is the minimum filter size to allow overlapping convolution with stride 2.

Additionally, they also derived three more variants of Method-C to show the importance of pooling in the image classification process as shown in the figure below:

Model		
Strided-CNN-C	ConvPool-CNN-C	All-CNN-C
Input 32 × 32 RGB image		
3 × 3 conv. 96 ReLU 3 × 3 conv. 96 ReLU with stride $r = 2$	3 × 3 conv. 96 ReLU 3 × 3 conv. 96 ReLU 3 × 3 conv. 96 ReLU	3 × 3 conv. 96 ReLU 3 × 3 conv. 96 ReLU
	3 × 3 max-pooling stride 2	3 × 3 conv. 96 ReLU with stride $r = 2$
3 × 3 conv. 192 ReLU 3 × 3 conv. 192 ReLU with stride $r = 2$	3 × 3 conv. 192 ReLU 3 × 3 conv. 192 ReLU 3 × 3 conv. 192 ReLU	3 × 3 conv. 192 ReLU 3 × 3 conv. 192 ReLU
	3 × 3 max-pooling stride 2	3 × 3 conv. 192 ReLU with stride $r = 2$
⋮		

Figure-2.19: The three variants of Model C [5]

- A model in which max-pooling is removed and the stride of the convolution layers preceding the max-pool layers is increased by 1 (to ensure that the next layer covers the same spatial region of the input image as before). This is column "Strided-CNN-C" in the table.
- A model in which max-pooling is replaced by a convolution layer. This is column "All-CNN-C" in the table.
- A model in which a dense convolution is placed before each max-pooling layer (the additional convolutions have the same kernel size as the respective pooling layer). This is model "ConvPool-CNN-C" in the table. Experiments with this model are necessary to ensure that the effect we measure is not solely due to increasing model size when going from a "normal" CNN to its "All-CNN" counterpart.

These tweaks gave very good results which are discussed as a part of the next question below but results are shown in the figures below as stated by the paper.

CIFAR-10 classification error

Model	Error (%)	# parameters
without data augmentation		
Model A	12.47%	≈ 0.9 M
Strided-CNN-A	13.46%	≈ 0.9 M
ConvPool-CNN-A	**10.21%**	≈ 1.28 M
ALL-CNN-A	10.30%	≈ 1.28 M
Model B	10.20%	≈ 1 M
Strided-CNN-B	10.98%	≈ 1 M
ConvPool-CNN-B	9.33%	≈ 1.35 M
ALL-CNN-B	**9.10%**	≈ 1.35 M
Model C	9.74%	≈ 1.3 M
Strided-CNN-C	10.19%	≈ 1.3 M
ConvPool-CNN-C	9.31%	≈ 1.4 M
ALL-CNN-C	**9.08%**	≈ 1.4 M

Figure-2.20: Results for Model-A, Model-B, Model-C compared with other papers
[% of Accuracy is (100-Error)%]

CIFAR-10 classification error

Method	Error (%)	# params
without data augmentation		
Maxout [1]	11.68%	> 6 M
Network in Network [2]	10.41%	≈ 1 M
Deeply Supervised [3]	9.69%	≈ 1 M
ALL-CNN (Ours)	**9.08%**	≈ 1.3 M
with data augmentation		
Maxout [1]	9.38%	> 6 M
DropConnect [2]	9.32%	-
dasNet [4]	9.22%	> 6 M
Network in Network [2]	8.81%	≈ 1 M
Deeply Supervised [3]	7.97%	≈ 1 M
ALL-CNN (Ours)	**7.25%**	≈ 1.3 M

Figure-2.21: Results for variants of Model-C compared with other papers
[% of Accuracy is (100-Error)%]

Q2. Compare the solution with the baseline CNN and your improved CNN and discuss the pros and cons of three methods.

Ans:

Comparison between baseline CNN, improved CNN and CNN implemented in paper [5]:

Baseline CNN	Improved CNN (Set-9)	CNN in paper [5]
Testing accuracy here ranged from 60% – 69%	Testing accuracy ranged from 70% to 85%	Testing accuracy ranged from 85% to 97%
The Layering architecture involved very small numbers for filter bank size and number of neurons	The Layering architecture involved large numbers for filter bank size and number of neurons	The Layering architecture involved large numbers for filter bank size and number of neurons
The Layer complexity was comparatively simple	The Layer complexity was comparatively high. It has 7 convolutional, 2 max pool and 1 global averaging pool layer with 2 dropouts in between.	As compared to the Improved CNN architecture, the layer complexity was high. It has 7 convolutional, 2 max pool and 1 global averaging layer with dropouts between almost each layer. Further, the Model-C is

		divided into 3 more subtypes with lots of max-pooling. So, it is just very complex.
The number of epochs used was in the range 50 to 150.	The number of epochs used was in the range 50 to 150.	The number of epochs used was in the range 200 to 350.
Learning rate used was 0.001 and 0.0005	Learning rate used was 0.001	Learning rate was varied between 0.25, 0.1, 0.05, 0.01 and 0.001
Dropout between layers ranged from 0.5 to 0.9 e.g. 0.9 indicates 10% dropped and 90% passed (as per TfLearn).	Dropout between layers was set to 0.5 e.g. 0.5 indicates 50% dropped and 50% passed (as per TfLearn).	Dropout between layers ranged from 0.2 to 0.5 e.g. 0.2 indicates 20% dropped and 80% passed (as per paper)
Can be performed using normal CPU	Can be performed using normal CPU but using a GPU is suggested	A very high end GPU is highly recommended
Cases arise where testing accuracy is more than the training accuracy (which was very small like 0.03% as observed in Table1.1 for Set-4)	Training accuracy >= Testing accuracy always	Training accuracy >= Testing accuracy always
Augmentation of input data involved only left-right flipping and rotation by 0 to 25 degrees randomly.	Augmentation of input data involved only left-right flipping and rotation by 0 to 25 degrees randomly.	Augmentation involved horizontally flipping images and randomly translating 5 pixels in each dimension for the images.
The receptive field was fixed at 5x5, the stride was always 2 and activation was ReLU.	The receptive field was mix of 5x5, 3x3 and 1x1, the stride was always 2 and activation was Leaky ReLU.	The receptive field was mix of 5x5, 3x3 and 1x1, the stride was always 2 and activation was ReLU.
The computation time ranged from 1 hour to 2 hours when conducted on my GPU (specifications mentioned in the Index section).	The computation time taken was around 22 hours when conducted on my GPU (specifications mentioned in the Index section).	The computation time taken was around 10 hours based on the hardware configuration of the modern GPUs used by the authors.

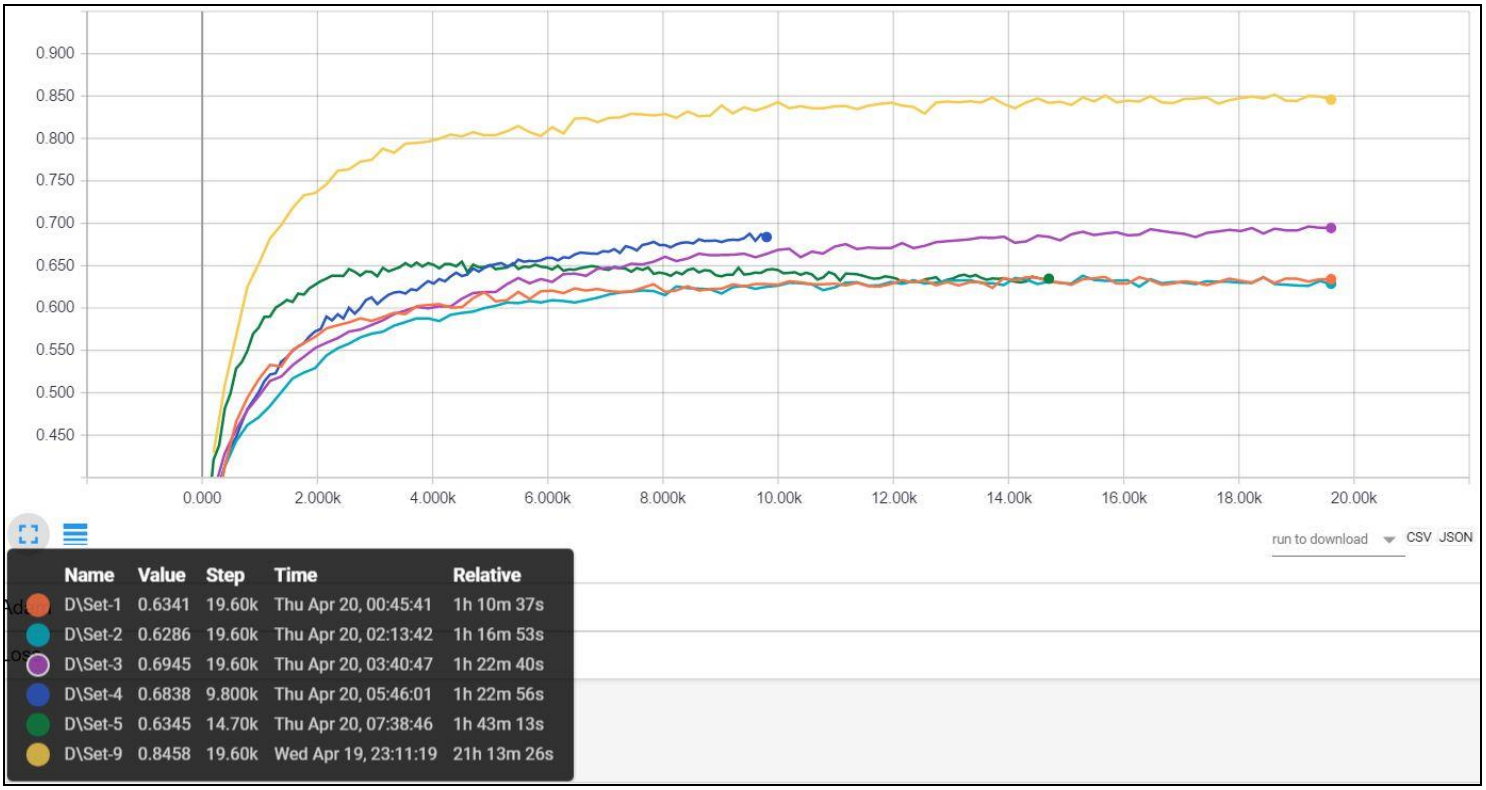

Figure-2.22: mAP curve Epoch(steps) vs testing data accuracy for baseline CNN and improved CNN comparison

Set-1 to Set-5 (baseline CNN)

Set-9 (improved CNN)

The x-axis in all the plots indicate the steps which can be seen as Epochs as well.

References

[1] http://cs231n.github.io/convolutional-networks/

[2] Digital Image Processing, Second Edition, Rafael Gonzalez, Richard E Woods, Pearson Education

[3] Digital Image Processing, Fourth Edition, William K. Pratt, Wiley-Interscience Publication

[4] The CIFAR-10 Leaderboard: http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html#43494641522d3130

[5] https://arxiv.org/pdf/1412.6806.pdf

[6] https://arxiv.org/pdf/1701.08481.pdf

Index

1. System configuration in which all the training and testing was done

Operating System	Windows 10 Professional 64-bit
Processor (CPU)	Intel® Core™ i7-6500U CPU @ 2.50GHz
Processor (GPU)	NVIDEA GDDR5 2GB