## Table of Contents

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

## 1 (a) CNN architecture and training

Convolutional Neural Networks are one of the most commonly used computer vision algorithms that are used extensively for object detection, tracking, semantic labelling, room overlay, classification and a variety of such applications. It outperforms the low level image processing by taking the images as data features and application of smart machine vision algorithms to establish a sequence of outcomes that are state of the art. CNN uses the machine learning algorithm to establish relations using the knowledge from approximation theory, optimization theory and visualization techniques.

They work similar to the Neural networks where each neuron is made up of weights and biases that receive input, perform some dot products which later are passed on to a non - linear function for evaluation. It is made up of convolutional layers which begin to learn lower level features initially and then gradually extract higher order of features followed by a fully connected layer similar to a Multi-Layer Perceptron (MLP) which then computes the loss in the form of Softmax and helps in Back Propagation (BP).

Neural Networks try to replicate the performances of a neuron in the brain. It was proposed by McCulloch and Pitts through their model that a system of neurons could be considered a mathematical model of a sum of weights and biases to an input neuron. Yann LeCun in the early 1980s wrote a paper on handwriting recognition which was widely accepted by the AI fraternity. This architecture was then utilised my researchers on the larger data sets such as ImageNet, CIFAR-10, CIFAR- 100, etc. There have been huge improvements in terms of accuracy and reduction of losses in the entire architecture through different manipulations and changes in the base architecture proposed by LeCun in his LeNet-5 as shown below in figure 1.

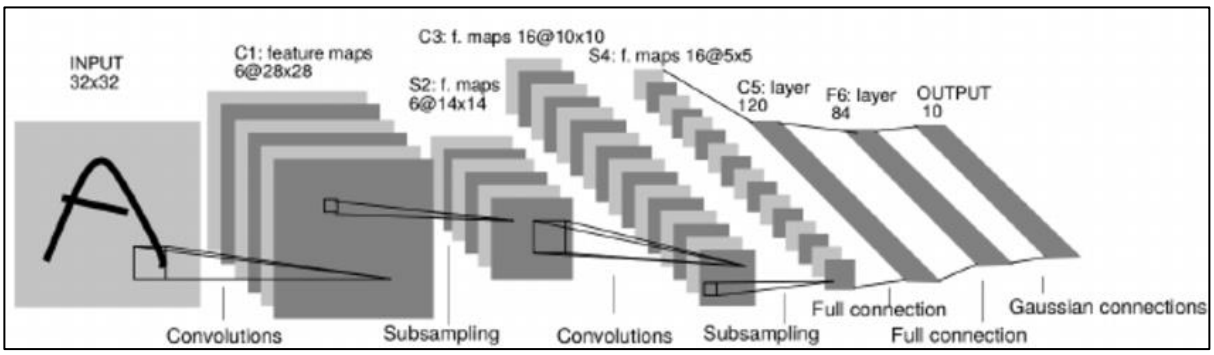



Fig 1. The LeNet-5 architecture introduced by LeCun

The LeNet-5 can be broadly said to be composed of two portions: feature extraction (FE) subnet and the decision making (DM) subnet as stated by Kuo in his paper on CNN as a Multi-layer RECOS transform. FE subnet is made up of multiple convolutional layers while

the DM subnet entails a couple of fully connected layers. It consists of two spatial convolutional layers followed by max pooling layers along with two fully connected layers. It is hence important to understand each of these layer part by part along with their functionality. The parameters mentioned in the above figure are utilised in this question to test the CIFAR-10 data set that consists of 50000 test images with 10000 images for validation of size 32x32x3. Softmax function is stated to be the cost function for the regression in this model and different non-linear activations such as Sigmoid, RELU, Leaky RELU are utilised. The stages in the architecture are explained in brief below:

1. INPUT LAYER:

The input layer takes the 50000 test images of size 32x32x3 as input patches and applies filters or weights on them in order to extract the lower level features in the image with 6 filters. This layer provides the pixels in the image as input neurons to the network and hence the size of the image plays a key role in deciding the number of input neurons that will be produced. Since the image is RGB colour image it has 3 dimensions as depth. Hence the 32x32 pixels are the input neurons which need to be classified into 10 classes.

2. SPATIAL CONVOLUTION LAYER 1

This layer is also sometimes called as the convolutional layer. It consists of a set of learnable parameters of learnable filters. In our case we utilise a 5x5x3 filter which is used to slide upon the image and convolve to compute the dot products between the entries of above filter at any input position. This produces an activation map which describes the 2 dimensional activation map of the spatial responses of the filters. We stack the activation maps together to always produce an output volume as shown in Fig 2.

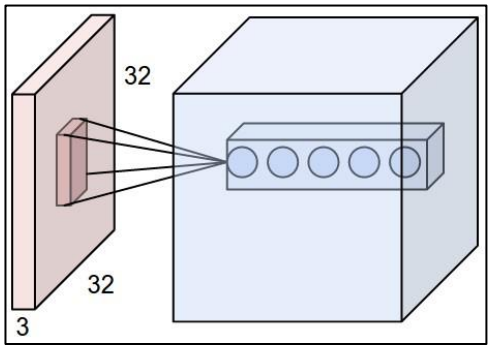



Fig 2: Stacking up of activation maps to create output volume with 5x5x3 as initial filter for 1st layer.

Convolution is important and also includes parameters such as filter size, number of filters, stride, zero padding. Generally, the output width is given by the formula stated below:

$$output\ width = \frac{(Input\ image\ width - filter\ size + (2 \times padding))}{Stride} + 1.$$

Where the stride is the parameter that equals to jumps or sliding of the filter and padding signifies whether zero padding is carried out or not. This method helps in connecting the output to a local region in the input. The receptive field selected is 5x5. Hence each neuron would see a 5x5 patch from input. Sharing parameter helps in reducing weights which helps the neurons share the same set of weights across the architecture assuming that these are useful throughout the image patches. In this case the receptive field is 5x5 and the stride =1

with no zero padding involved. The input is convolved to produce a 6(depth)x(28x28) stack of volume. The total number of learnable parameters for each neuron in this case are given by (5x5x3+1(bias)) =76 weights. The figure below shows the w as the weight associated to a single neuron in the system x. Effectively for this layer since we have parameter sharing available we would have weights=(5x5x3x6) + 6(bias)= 456 weights for each of the 28x28 neurons in each depth slice.

3. MAX POOLING LAYER 1

The max pooling layer is an essential portion after the convolution over the volume is carried out and hence it method of calculation of the L-∞ norm which basically calculates the maximum value. This layer contains 0 learnable parameters and reduces the spatial size of the representation which effectively removes the parameters and computation in the network. In LeNet-5 a stride of 2 along with a filter size of 2 are chosen in general. The figure below shows the maximum pooling concept with down-sampling by 2. Hence in the LeNet algorithm we are left with 6x14x14 responses from 6x28x28.

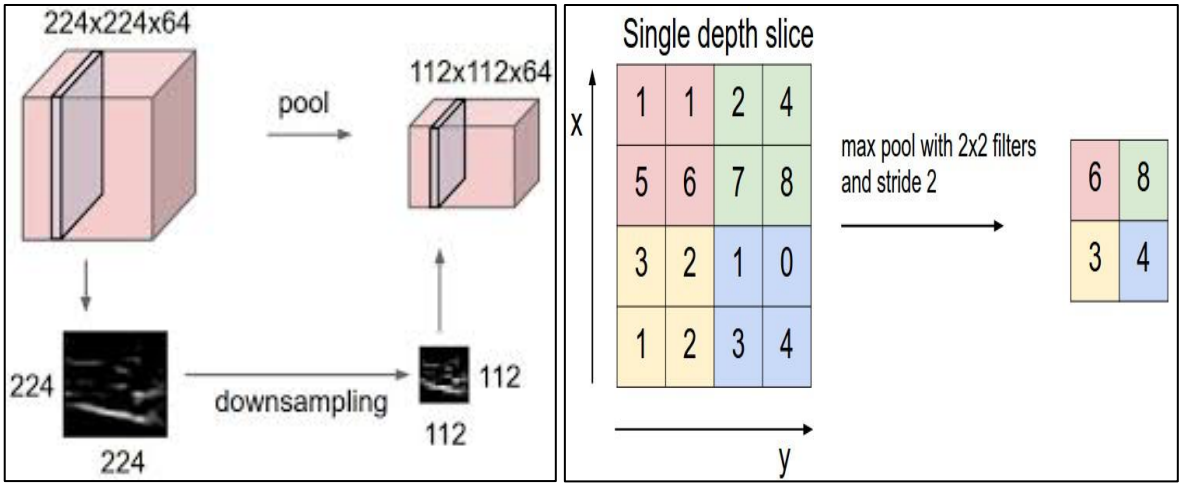



Fig 3: The max pooling operation with stride 2 and filter size 2.

4. NON-LINEAR ACTIVATION 1

These functions introduce non linearity in the system which is necessary in making decisions. The linear sum of learnable parameters along with their biases are passed on to the non-linear function such as Sigmoid, Tanh, Relu, Leaky Relu.

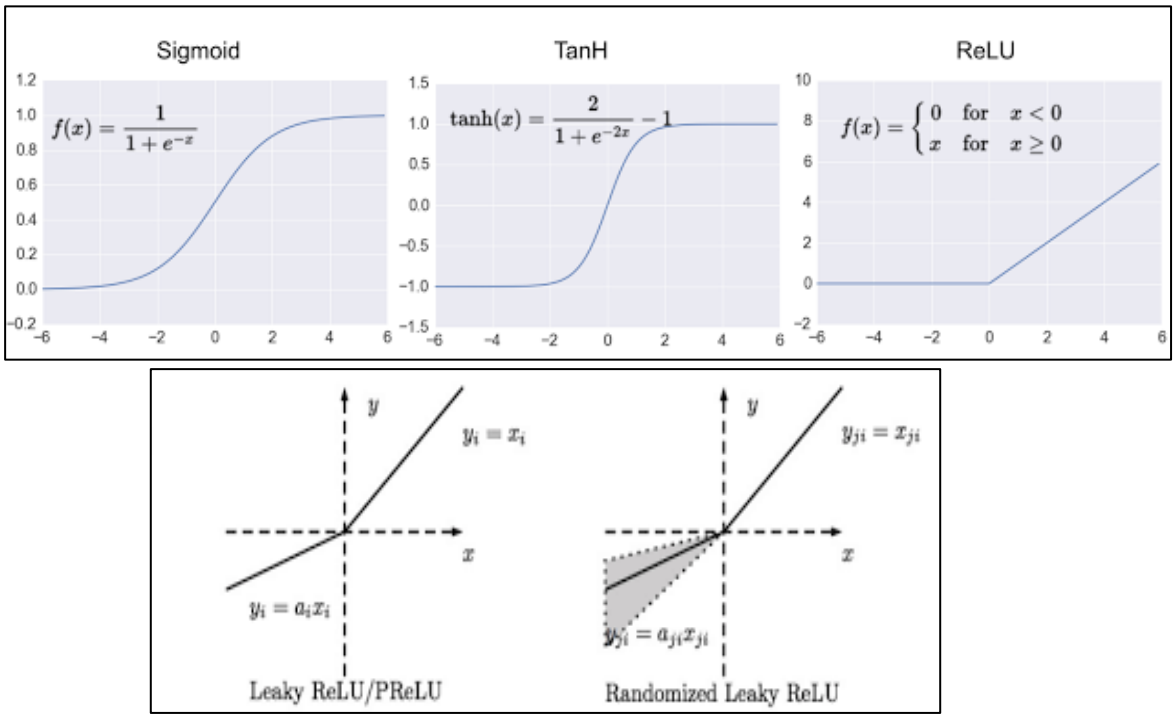



Fig 4: The various non-linear activation functions are mentioned above.

Sigmoid function is generally a standard non-linear activation function as it takes a real valued number as squashes it between 0 and 1. It has certain drawbacks such as gradient vanishing problem along with zig-zagging dynamics in the gradient update which leads to slower convergence.

Tanh on the other hand squashes the function in between 1 and -1 but also has drawbacks such as gradient vanishing problem. It is zero centered unlike sigmoid and converges faster than it. Relu on the other hand has non zero values only for positive responses and converges faster than both the above discussed functions without large vanishing problems. The outputs are not zero centered but are fragile during training and die out if improper learning rate is chosen. Dead Relu can hence not be used in the entire architecture and hence is considered a large drawback. This drawback is addressed by the Leaky Relu which has a small positive slope in the negative region which prevents it from fading away due to improper learning rates.

Effectively the human brain equivalent is shown in the figure below where w represents the weights, x is the neuron for activation along with the bias term which is fed into f which is a non-linear activation function.

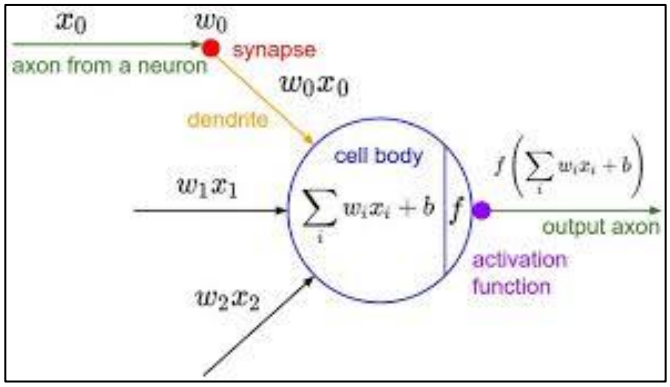



Fig 5: The brain equivalent model to activation by non-linear functions.

Since the real world data is non-linear it serves as a good approximation. We use Leaky Relu due to its advantages over the others.

5. SPATIAL CONVOLUTION LAYER 2

Similar to the concepts involved in the first convolutional layer, here the 6x14x14 act like input neurons which are again dealt with 5x5 activation maps with 16 depths. The output to this layer is of dimensions 16x10x10. The number of weights associated to this layer per neuron is 5x5x6+ 1(bias)=151. Hence the total weights associated with layer turn out to be 5x5x6x16+16(bias)=2416 weights in total. This layer is responsible for extracting higher level features such as textures whereas the 1$^{st}$ layer is able to extract only edges and contours.

6. MAX POOLING LAYER 2

Similar to the previous case, we again apply maximum pooling technique to reduce or down-sample from 16x10x10 to 16x5x5. Window size and the stride for this case are 2 similar to previous case.

7. NON-LINEAR ACTIVATION 2

Leaky Relu is applied again in this case for non-linear activation because of its advantages specified previously. Since this layer does not add any learnable parameters, the size remains equal to the previous stage size. The input 32x32x3 neurons are hence finally reduced to 5x5x16= 400 neurons which is large reduction in the image data.

8. FULLY CONNECTED LAYER 1

The full connected layer is similar to a MLP which carries out the complete transfer from one neuron to another. The full connected layer contains 120 neurons which are connected to 400 neurons in the input. The number of weights to be learnt in this case are 120x400 + 120 (bias)=48120 weights since the sharing capability is not considered in the fully connected layers. These fully connected layers have maximum receptivity and have the capability of forward as well as backward propagation. These fully connected layers have all the highest features of the images and are again passed through a non-linear activation of leaky Relu. Drop out factors can also be assigned in such cases. The drop out features are very essential as they take into consideration the parameter of over fitting in a model. These outputs are further transferred to the second fully connected hidden layer.

9. FULLY CONNECTED LAYER 2

This layer contains 80 neurons which get connected to the 120 neurons in the previous stage. Hence the total weights associated considering no drop outs in the previous stage= 80x120+80 (bias)=9720 weights. These weights are further passed on to the non-linear activation function of leaky Relu to be further connected to a drop out like the previous case to the final output layer where regression by Softmax is carried out.

10. FULLY CONNECTED OUTPUT LAYER WITH SOFTMAX

This layer contains the output 10 neurons which get connected to the 80 neurons in the previous stage with the weights equal to 810. This is then used for the Softmax calculation which takes into consideration the log soft max and provides the log probabilities to the output. It normalises the outputs to determine the output labels. Cross entropy function is utilised as it shows how the entropy of a particular label is associated with a counter label. Cross entropy is much more robust manner of defining the labels than the L2 norm as it takes into consideration the one hot property of having 100 percent probability for a particular label and 0 for the rest during training. Since such networks are trained under cross entropy regime, we obtain a non-linear variant of multinomial logistic regression. The formula for

Softmax regression is given below, where x is the input neuron. The total number of trainable parameters=61,482.

$$s(x) = \frac{e^x}{\sum_{i=i}^{N} e^i}$$

## DIFFERENCE BETWEEN CNN AND MLP

CNN comprises of two portions which are explained above as a convolutional network followed by a fully connected layer which is similar to the MLP. Sharing feature is available in the convolutional layers whereas the fully connected layers lack that feature and hence we need to carry out more computations and the larger number of neurons need to be addressed.

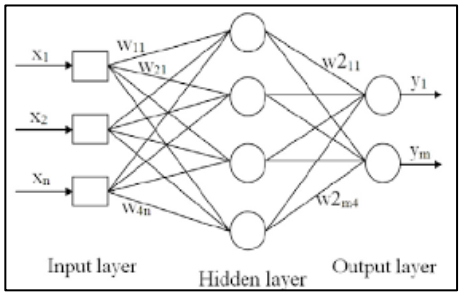



Fig 6: Multi-Layer Perceptron module

MLP is supervised learning algorithm which was better than the previously considered machine learning techniques of SVM, ANN, Structured Forest because of its Backward propagation ability and its ability to work with the neurons in parallel. As compared to the linear analysis systems it always provided a more flexible mapping between the feature space and the decision space as the feature space could be non-convex and irregular.

CNN uses this MLP as a building module with far less computations. The major difference between them lies in the fact that CNN uses the source data which could be image, video, speech, etc. whereas MLP uses features. Since MLP uses features as input it becomes difficult to compute large visual patterns or features as input image size increases. In order to have a successful MLP running we need to have large hidden layers as we need to have a full connection between each of the neurons and this could be in millions which is highly impractical. CNN obtains the features through the convolutions and max pooling whereas MLP needs handcrafted features.

CNN instead of taking into consideration each pixel and a huge feature data set works with the interaction of pixels and takes into consideration image patches which are also known as receptive fields for nodes. In the first convolution, CNN takes small pixel diversities and further expands its receptive fields to larger interactions like textures. As the convolution occurs subsampling is also carried out which eliminates some of the neurons by max pooling technique and takes into consideration global interaction which then gets connected to the fully connected layers which is similar to MLP. Hence the connections which are of the order of millions can be easily brought down to the order of thousands (61,482). Hence CNN shifts from the traditional feature engineering technique to a data driven approach for classification and labelling.

## Why CNN works better than Classical Computer Vision algorithms

CNN works better than most classical computer vision algorithms mainly because of its feature engineering process and the back propagation between layers. In classical computer vision algorithms like SVM, RF, MLP, ANN we need to provide handcrafted features which may or may not fit the data well and may lead to erroneous labelling. CNN demonstrates high level of robustness in performance due to this feature engineering process which gives a larger discriminant power due to BP. Feature extraction subnet and Decision making subnet closely are responsible for better performance. In image classification again similarly labelling is possible for both traditional as well CNN but CNN provides much better feature extraction subnet to outperform the others.

## Loss function and back propagation (BP) in CNN

The loss function in CNN is calculated at the output end of the network. It is generally considered to be the difference between the obtained value and the expected value. In this sense while labelling during training, we expect the output to include one hot vector and hence include 9 zeros and only one 1. The loss function is the error with respect to this one hot vector. Generally, the loss function that is calculated is the L2 norm or also called as the Mean Square Error or the Negative log likelihood function which is essential for finding out the cross entropy in the outputs. Cross Entropy works on the principle of principle of negative log likelihood which is more robust than that of the L2 norm as log considerations are more appropriate for consideration of errors in one hot vectors. For this problem we utilised the concept of categorical cross entropy which was then back propagated from the output to the input.

Back propagation (BP) is the process in which the error function or the loss function is propagated from the output to the input in such a manner so that the net error in the network is minimised. After the feed forward path is done and the error is calculated by either L2 or cross entropy, this error is propagated backwards until a sufficient weight update is no carried out which nullifies the error in the system. BP uses the error values to calculate the gradient of the loss function with respect to the weights which is then optimised using a particular optimiser such as Stochastic Gradient Descent (SGD) or Adam to update the weights in such a manner that the error is minimised. It is important to note that the optimisers play a crucial role in successful BP as they update the weights and the learning rate associated to them should be appropriate enough. The figure below shows how the error is transmitted backwards and the gradients with rest to weights are calculated.

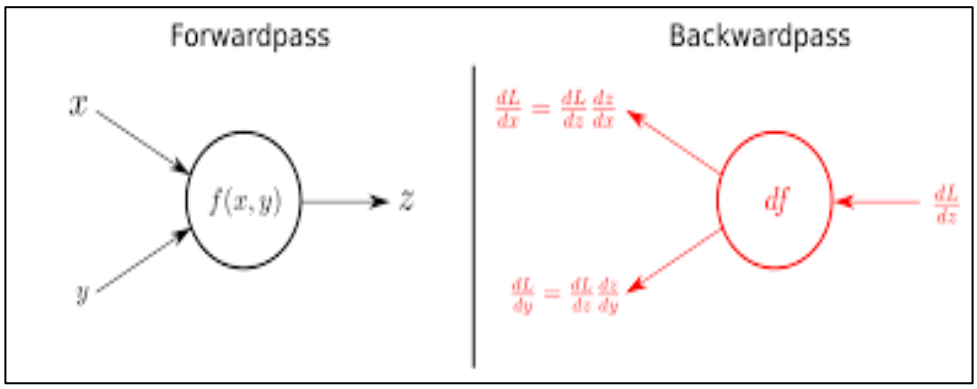



Fig 7: Backward propagation process with gradients in terms of weights to be updated

## 1 (b) Application of the CNN to CIFAR-10 Dataset

The CIFAR 10 data set consists of 50000 training images and 10000 images for validation. The LeNet-5 architecture mentioned above with the parameters and weights mentioned above are utilised for calculation of the accuracy and the precision of the data. Some pre-processing techniques were carried out during this process which are enlisted below. These pre-processing techniques help in achieving a higher accuracy with lesser number of computations. Hence it is essential that such techniques are utilised efficiently in order to attain a higher value of accuracy using the said parameters in the question. Data pre-processing is carried out only on the training data and not on the testing data. It is important to keep the testing data as a sacred gospel of data that should not be altered in any case.

DATA PREPROCESSING

Two important pre-processing steps are carried out at this stage which include the zero centre pixel values of all the training data.

1. Subtracting the mean image which includes obtaining the DC value of all the pixels in the image training data set. This creates only AC values in the training set which show the variation of data. All the pixel values were normalised by 255.0 before the process was initialised.
2. Subtracting mean per channel which includes computation of 3 means which are representative of the means per channel and subtraction from each channel respectively. Normalisation of training input data set is also carried out with respect to the standard deviation.

DATA AUGMENTATION

Training data generally consists of large amounts of noise which can cause overfitting issues. Hence it is important that we augment data when we have limited data sets under test. This is a low cost way to generate more samples. Minor perturbations in input image data set do not generally have a large effect on the class types. It sometimes helps in also eliminating the dataset bias that is present in the input training dataset. There are different methods to augment data which include random cropping of test data, carrying out affine and homographic transforms on the testing data or carrying out colour jittering. In our case, we utilised random flipping the input test image along with addition of rotation in clockwise direction by a 25 degrees. All these operations are carried out on random image test samples and not on all datasets. Without data augmentation, it is possible to have large difference between the training and testing accuracies.

WEIGHT INITIALISATION

It is very essential to consider the weight initialisation parameter in CNN as most of the gradient related operations depend on weight initialisation for reduction in the loss function during BP. There are various random initialisation techniques which include initialisation by zeros, uniform, uniform scaling, normal, truncated normal, Xavier initialisation, variance scaling or k means. If we initialise the weights by zeros throughout then all the neurons will end up computing the same weights and the same gradient for BP. Similarly, if we choose a random number that is normally distributed with zero mean and

standard deviation of 0.01 we end up causing deep networks to fail due to gradient vanishing. Hence we use Xavier initialisation method in our case as it more robust than the other initialisation techniques and helps in updating the weights effectively without any gradient vanishing effects. K means initialisation is considered to outperform all others as it aids in obtaining a higher accuracy. In general, the weight initialisation is dependent inversely on the square root of the receptive field of the previous layer.

Xavier initializer is designed to keep the keep the scale of gradients same throughout all the layers. Generally in uniform distribution it ends up in the range of x = sqrt (6./(in + out)); [-x, x] and normal distribution a stand deviation of x = sqrt (3./(in + out)) is used.

LOSS FUNCTION SELECTION

Loss function calculation is an important feature for BP and hence in this case the loss function selected was cross entropy as it provides a closer approximation than the Mean Square Error. Since negative log likelihood provides a closer approximation of the degree of surprise than MSE, it is wise to choose cross entropy.

LEARNING RATE

Learning rate is essential as it defines how fast or how slow does a network converge. Smaller learning rates converge very slowly whereas larger learning rates may miss the global minima or maxima. Hence, the learning rate decay parameter is essential as it defines this rate. Using brute force, we obtain that 0.001 is the optimal rate for CIFAR-10 training.

BATCH SIZE

Mini batch gradient descent algorithm is used in this process where the weights are constantly updated after an input batch is processed. Size of the batch is important to decipher the convergence of the model. Again using brute force, we need to estimate a good batch size. Larger batch sizes tend to create unstable update vectors which tend to converge even slowly. Usage of batches however helps to increase the variance in the training. Batch sizes used for experimentation were 128 and 256 and the change in accuracy was noted.

DATA SHUFFLING

Overfitting is a common issue that is created when large amounts of data are utilised in CNN. This may lead to poor performance of the training data set. To prevent overfitting of data and addition of noise, the training data set is used in batches and is shuffled regularly to maintain the robust performance of neural networks. After each epoch the training data is randomly shuffled.

DROP OUT

Drop out factors after fully connected layers are essential in maintaining the network at its optimal value and to ensure effectiveness. Drop out factor is important to curb the issue of overfitting in the model. It helps in reducing the effect of stronger features dominating the weaker features. It was seen in the experimentation that as the dropout rate is decreased it more computations are needed but better accuracy is obtained. It is hence a trade-off between the accuracy and the computational complexity. Different drop out factors varying from keeping percentage of 50 to 90 percent were tested for model analysis. It was seen that a large keep percentage produces better results but is computationally expensive and may sometimes lead to overfitting issues.

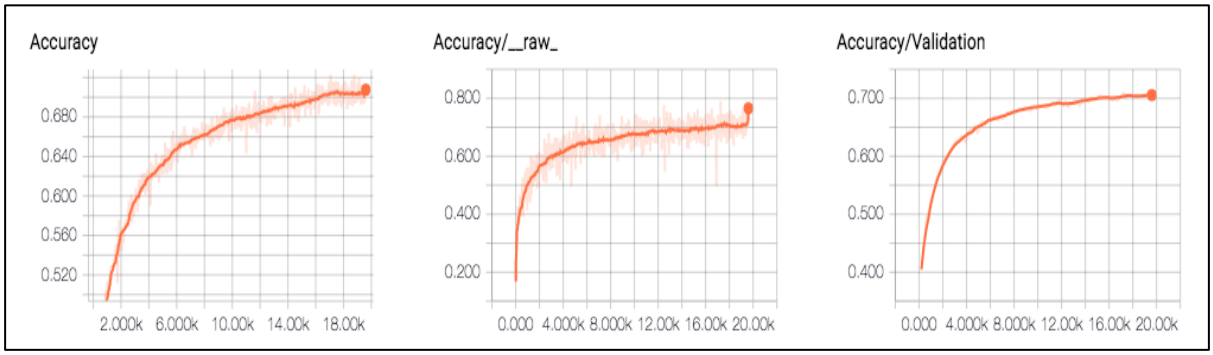
***EXPERIMENTAL RESULTS***

The standard parameters that are taken into consideration while comparing all the results using Tflearn on Tensor-Flow easily available through git-hub are as follows:

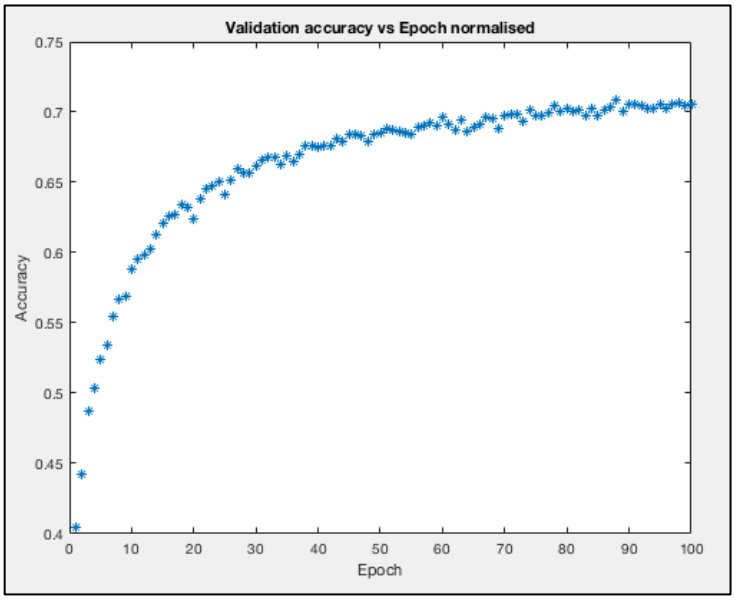
Standard:
**EPOCH**:100, **DROP OUT**:0.9(keeping %), **BATCH SIZE**:256, **OPTIMIZER**: ADAM, **NON LINEAR ACTIVATION**: Leaky Relu, **INITIALISATION**: Xavier, **LOSS FUNCTION**: Cross Entropy by NLL, **DATA AUGMENTATION**: Yes, **LEARNING RATE**: 0.001

The accuracy curves followed by the errors in training and validation are mentioned below in the figures:

```
--
Training Step: 19600  | total loss: 0.82563 | time: 57.916s
| Adam | epoch: 100 | loss: 0.82563 - acc: 0.7077 | val_loss: 0.85000 - val_acc: 0.7056 -- iter: 50000/50000
--
```




NOTE: The X axis shows the epoch mapped to the number of steps linearly and the Y axis shows the accuracy in each case. Instead of having one value as a representative for an epoch, many representatives for an epoch are plotted on the X axis as shown below which is similar as shown below.



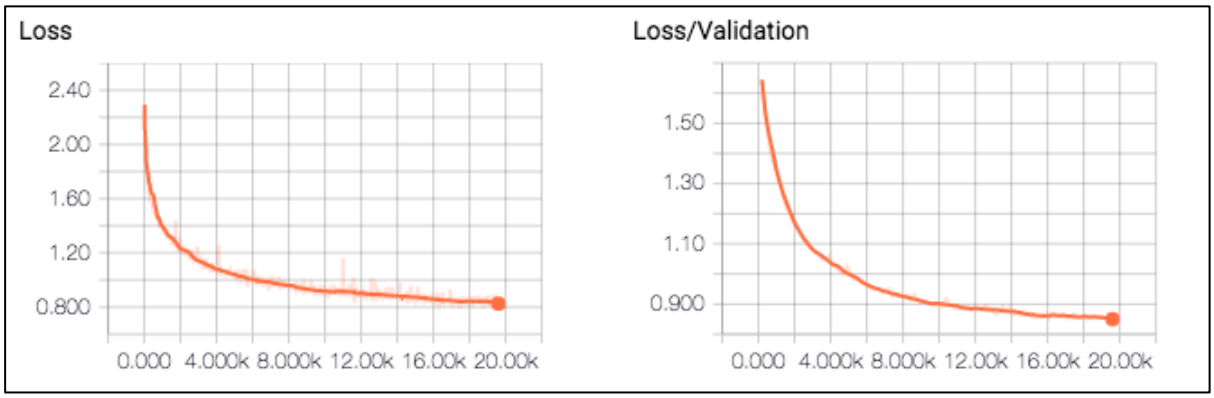



Fig 8: Accuracy and error curves for the case of Size=256 and learning rate=0.001with data augmentation of 25 degrees.

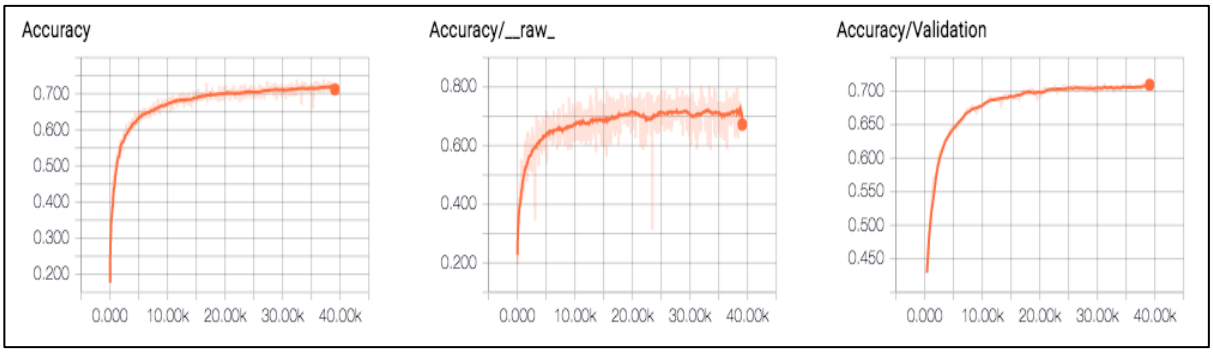
TRAINING ACCURACY:**70.77**%
TESTING ACCURACY:**70.56**%

**Effect of different batch size**

**EPOCH**:100, **DROP OUT**:0.9(keeping %), **BATCH SIZE**:128, **OPTIMIZER**: ADAM, **NON LINEAR ACTIVATION**: Leaky Relu, **INITIALISATION**: Xavier, **LOSS FUNCTION**: Cross Entropy by NLL, **DATA AUGMENTATION**: Yes, **LEARNING RATE**: 0.001

The accuracy curves followed by the errors in training and validation are mentioned below in the figures:

```
--
Training Step: 39100  | total loss: 0.82175 | time: 62.707s
| Adam | epoch: 100 | loss: 0.82175 - acc: 0.7121 | val_loss: 0.84385 - val_acc: 0.7094 -- iter: 50000/50000
--
```




NOTE: The X axis shows the epoch mapped to the number of steps linearly and the Y axis shows the accuracy in each case. Instead of having one value as a representative for an epoch, many representatives for an epoch are plotted on the X axis.


Fig 9: Accuracy and error curves for the case of Size=128 and learning rate=0.001with data augmentation of 25 degrees.

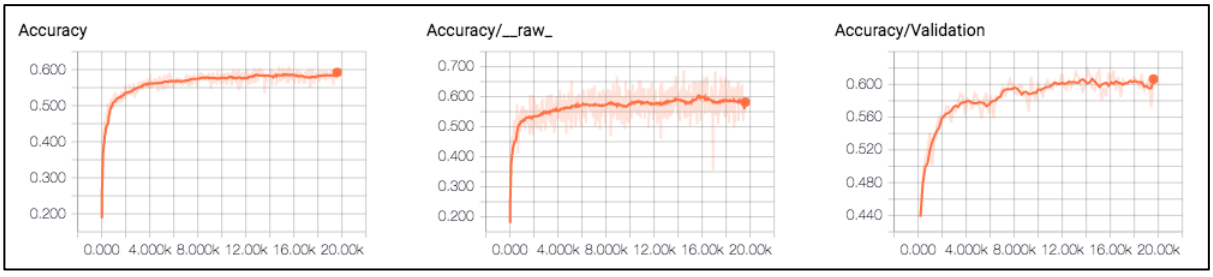
TRAINING ACCURACY:**71.27**%
TESTING ACCURACY:**70.94**%

**Effect of different Learning rate**

**EPOCH**:100, **DROP OUT**:0.9(keeping %), **BATCH SIZE**:256, **OPTIMIZER**: ADAM, **NON LINEAR ACTIVATION**: Leaky Relu, **INITIALISATION**: Xavier, **LOSS FUNCTION**: Cross Entropy by NLL, **DATA AUGMENTATION**: Yes, **LEARNING RATE**: 0.01

The accuracy curves followed by the errors in training and validation are mentioned below in the figures:

```
--
Training Step: 19600  | total loss: 1.20180 | time: 59.246s
| Adam | epoch: 100 | loss: 1.20180 - acc: 0.5923 | val_loss: 1.16755 - val_acc: 0.6065 -- iter: 50000/50000
--
```




NOTE: The X axis shows the epoch mapped to the number of steps linearly and the Y axis shows the accuracy in each case. Instead of having one value as a representative for an epoch, many representatives for an epoch are plotted on the X axis.

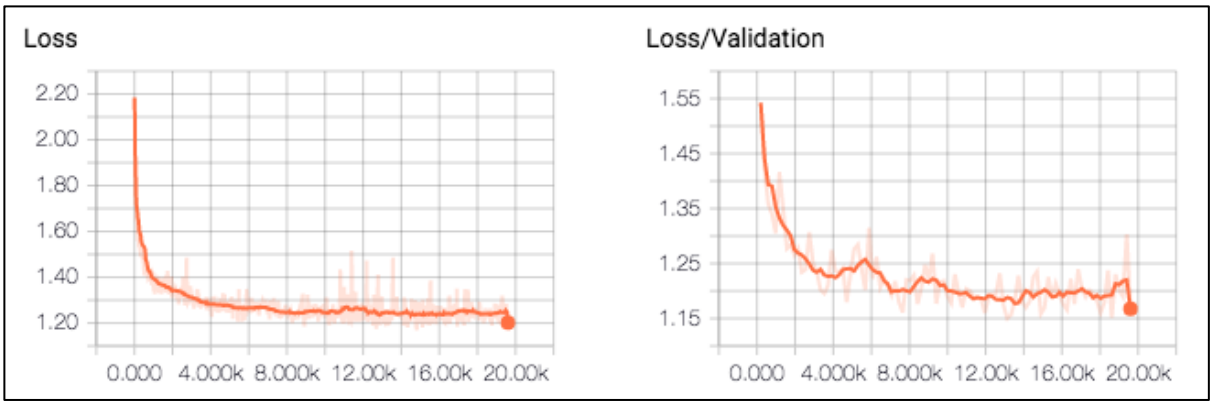



Fig 10: Accuracy and error curves for the case of Size=256 and learning rate=0.01with data augmentation of 25 degrees.

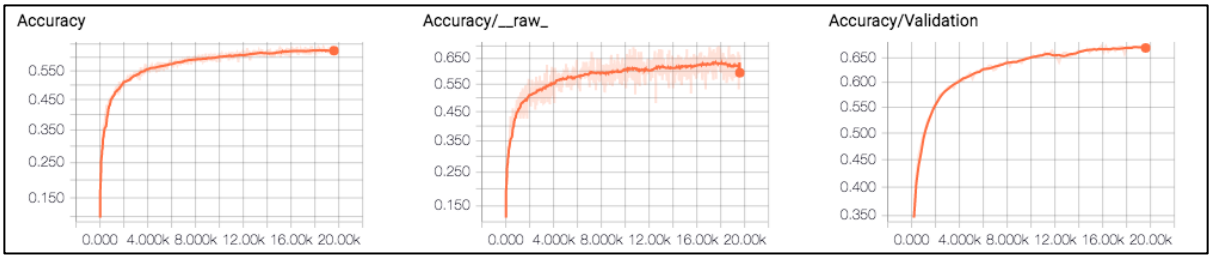
TRAINING ACCURACY:**59.23**%
TESTING ACCURACY:**60.65**%

**Effect of different drop out**

**EPOCH**:100, **DROP OUT**:0.5, **BATCH SIZE**:256, **OPTIMIZER**: ADAM, **NON LINEAR ACTIVATION**: Leaky Relu, **INITIALISATION**: Xavier, **LOSS FUNCTION**: Cross Entropy by NLL, **DATA AUGMENTATION**: Yes, **LEARNING RATE**: 0.001

The accuracy curves followed by the errors in training and validation are mentioned below in the figures:

```
Training Step: 19600  | total loss: 1.08135 | time: 55.552s
| Adam | epoch: 100 | loss: 1.08135 - acc: 0.6245 | val_loss: 0.95419 - val_acc: 0.6710 -- iter: 50000/50000
--
```




NOTE: The X axis shows the epoch mapped to the number of steps linearly and the Y axis shows the accuracy in each case. Instead of having one value as a representative for an epoch, many representatives for an epoch are plotted on the X axis.

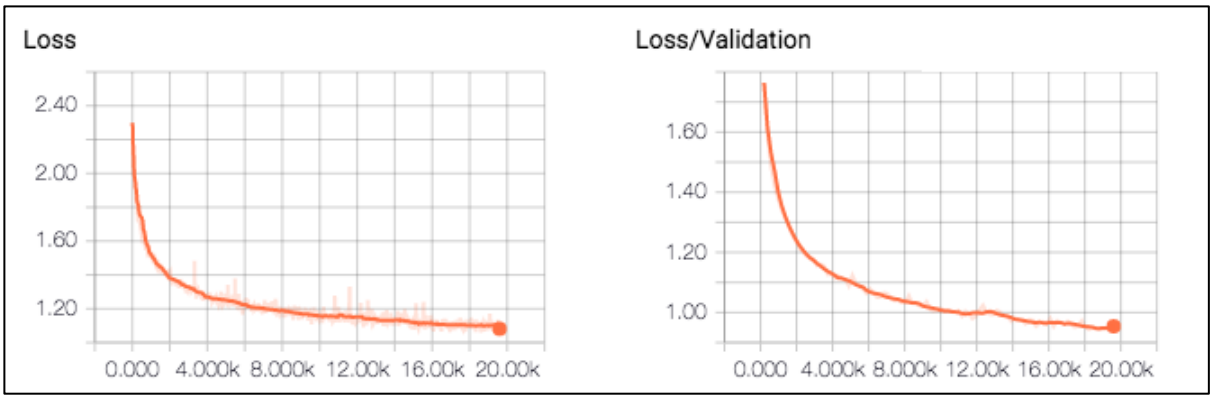



Fig 11: Accuracy and error curves for the case of Size=256 and learning rate=0.01with data augmentation of 25 degrees with drop out=0.5.

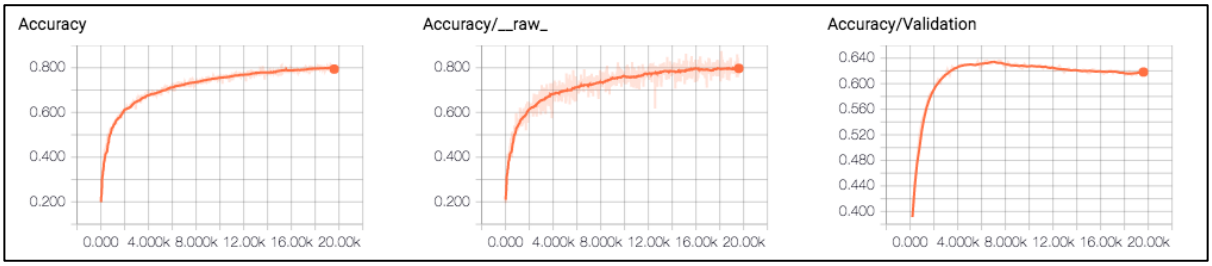
TRAINING ACCURACY:**62.45**%
TESTING ACCURACY:**67.10**%

**Effect without data augmentation**

**EPOCH**:100, **DROP OUT**:0.9(keeping %), **BATCH SIZE**:256, **OPTIMIZER**: ADAM, **NON LINEAR ACTIVATION**: Leaky Relu, **INITIALISATION**: Xavier, **LOSS FUNCTION**: Cross Entropy by NLL, **DATA AUGMENTATION**: NO, **LEARNING RATE**: 0.001

The accuracy curves followed by the errors in training and validation are mentioned below in the figures:

```
--
Training Step: 19600  | total loss: 0.55167 | time: 49.872s
| Adam | epoch: 100 | loss: 0.55167 - acc: 0.7938 | val_loss: 1.38086 - val_acc: 0.6182 -- iter: 50000/50000
--
```




NOTE: The X axis shows the epoch mapped to the number of steps linearly and the Y axis shows the accuracy in each case. Instead of having one value as a representative for an epoch, many representatives for an epoch are plotted on the X axis.

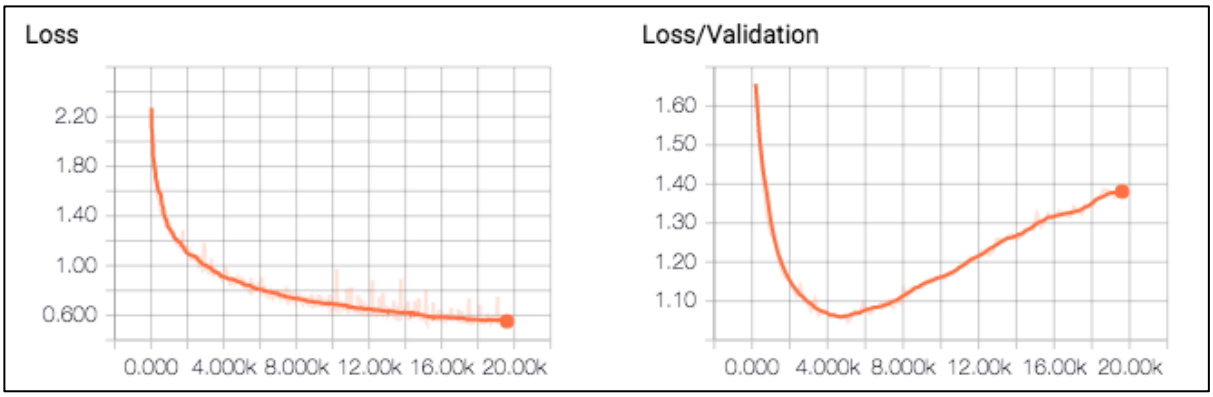


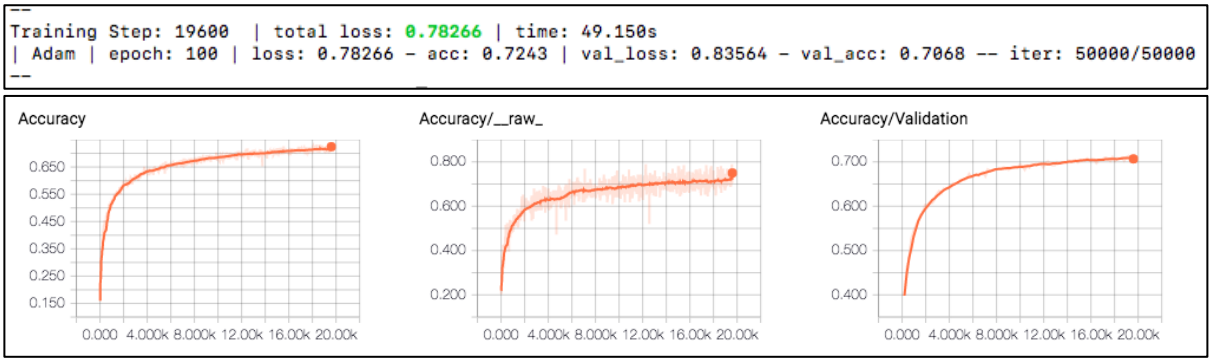

Fig 12: Accuracy and error curves for the case of Size=256 and learning rate=0.01without data augmentation of 25 degrees with drop out=0.9.

TRAINING ACCURACY:**79.38**%
TESTING ACCURACY:**61.82**%

**Effect with Relu activation**

**EPOCH**:100, **DROP OUT**:0.9(keeping %), **BATCH SIZE**:256, **OPTIMIZER**: ADAM, **NON LINEAR ACTIVATION**: Relu, **INITIALISATION**: Xavier, **LOSS FUNCTION**: Cross Entropy by NLL, **DATA AUGMENTATION**: NO, **LEARNING RATE**: 0.001

The accuracy curves followed by the errors in training and validation are mentioned below in the figures:




NOTE: The X axis shows the epoch mapped to the number of steps linearly and the Y axis shows the accuracy in each case. Instead of having one value as a representative for an epoch, many representatives for an epoch are plotted on the X axis.

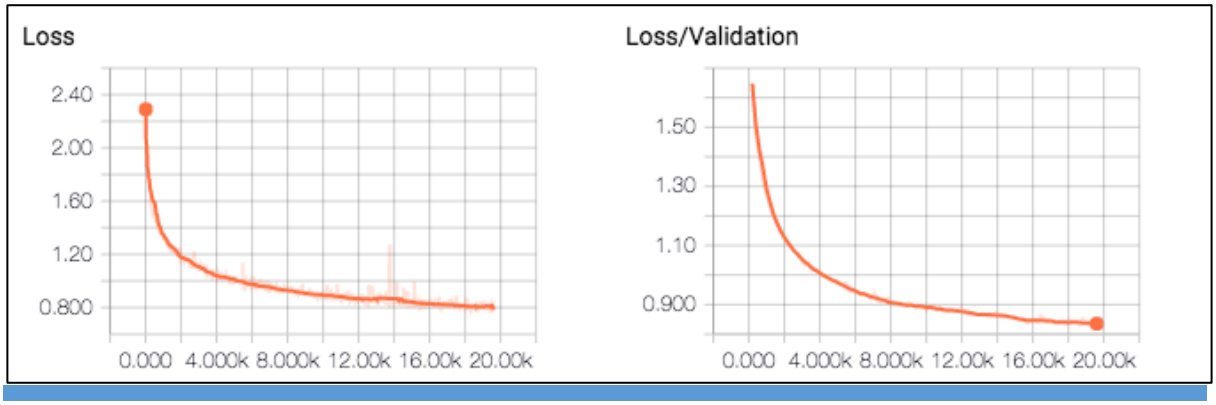


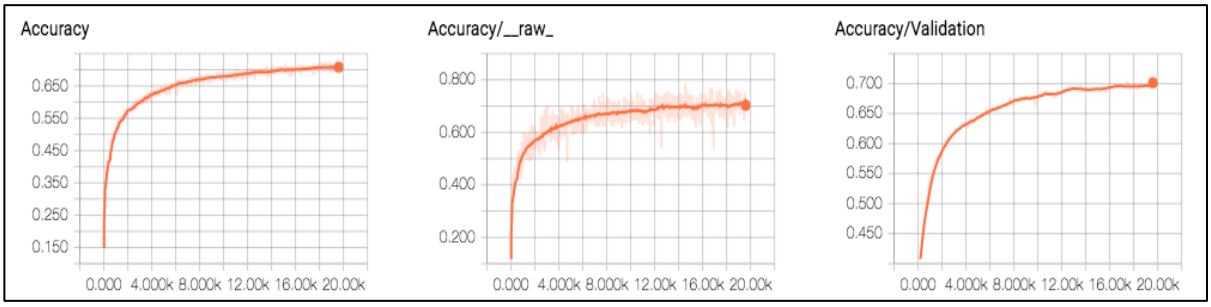
Fig 13: Accuracy and error curves for the case of Size=256 and learning rate=0.01without data augmentation of 25 degrees with drop out=0.9 with Relu activation.

TRAINING ACCURACY:**72.43**%
TESTING ACCURACY:**70.68**%

**Effect with Zero initialisation instead of Xavier**

**EPOCH**:100, **DROP OUT**:0.9(keeping %), **BATCH SIZE**:256, **OPTIMIZER**: ADAM, **NON LINEAR ACTIVATION**: Leaky Relu, **INITIALISATION**: Zeros, **LOSS FUNCTION**: Cross Entropy by NLL, **DATA AUGMENTATION**: Yes, **LEARNING RATE**: 0.001

```
--
Training Step: 19600  | total loss: 0.81933 | time: 55.446s
| Adam | epoch: 100 | loss: 0.81933 - acc: 0.7079 | val_loss: 0.85538 - val_acc: 0.7013 -- iter: 50000/50000
--
```




NOTE: The X axis shows the epoch mapped to the number of steps linearly and the Y axis shows the accuracy in each case. Instead of having one value as a representative for an epoch, many representatives for an epoch are plotted on the X axis.

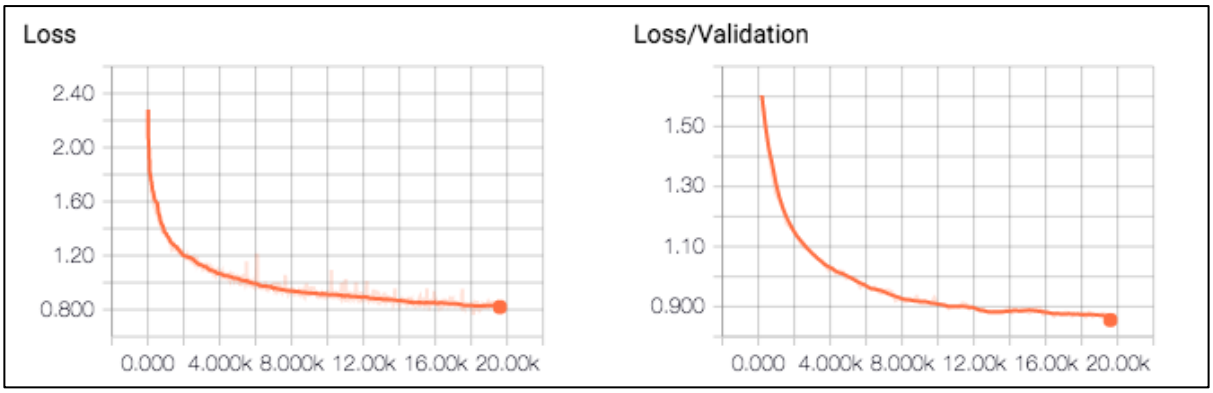



Fig 14: Accuracy and error curves for the case of Size=256 and learning rate=0.01without data augmentation of 25 degrees with drop out=0.9 with Zeros initialisation.

TRAINING ACCURACY:**70.79**%
TESTING ACCURACY:**70.13**%

***CONCLUSION and DISCUSSION***

As seen above from the Figures 8 to 13, the standard setup for the CIFAR-10 which was mentioned in Figure 8 was compared under conditions of different learning rate, different batch size, with and without augmentation, different dropout rates, different activation functions through the various figures following it.

Comparison between Standard size of 256 and Size 128

Figures 8 and 9 are representative of the comparison between the standard size and the experiment with size 128, the accuracies obtained in both the cases are as follows:

SIZE: 256, ACCURACY:70.56%
SIZE: 128, ACCURACY:70.94%

Both the cases show that with all other parameters constant, relatively small change in batch size does not affect the accuracy to a large margin. The computational times obtained in both cases are different as the more batches are needed in case of lower of lower batch size. Hence we can definitely state that with all other parameters constant, until the difference in the batch size is large enough, we will not obtain a huge difference in accuracy.

Comparison between learning rates of 0.001 and 0.01

Figures 8 and 10 are representative of the comparison between the standard learning rate and the experiment with learning rate 0.01, the accuracies obtained in both the cases are as follows:

LEARNING RATE: 0.001, ACCURACY:70.56%
LEARNING RATE: 0.01, ACCURACY:60.65%

As seen in the above comparison, when the learning rate is very large, there is a possibility that the global minima or maxima is missed as it is not able to traverse the ravines due to quick transitions whereas if it is kept too small it will never be able to reach the global minima of the function. Hence the learning rate parameter is crucial in order to have proper traversal in the global regions and maintain the properties uniformly instead of traversing in the local regions. In the above case, learning rate of 0.001 is found to produce better results since it is optimal to traverse the entire region instead of getting stuck in local maxima's or minima's.

Comparison between drop outs with keeping ratio 0.5 and 0.9

Figures 8 and 11 are representative of the comparison between the standard drop out ratio and the experiment with keeping ratio of 0.5, the accuracies obtained in both the cases are as follows:

DROP OUT WITH KEEPING RATIO: 0.9, ACCURACY:70.56%
DROP OUT WITH KEEPING RATIO: 0.5, ACCURACY:67.10%

The keeping ratio or the drop out factors in the fully connected layer provide an important characteristic which helps in preventing overfitting issues due to noise. As seen above if we tend to reduce the keeping ratio by a large margin, most of the neuron activations would be missed and considered absent. This would reduce the total number of weight considerations but may not fit the data well and hence would lead to a lower accuracy. Larger keep ratios

sometimes generate noisy characteristics which sometimes take into consideration the dead neurons even after successful back propagation and weight updation. Accuracy obtained is better in case of 0.9 keeping ratio than 0.5 keeping ratio in the above case since in the latter case larger neuron activations are missed which causes worse performance.

Comparison between data augmentation and no data augmentation

Figures 8 and 12 are representative of the comparison between the standard data with augmentation and the experiment without augmentation, the accuracies obtained in both the cases are as follows:

DATA AUGMENTATION TRAINING ACCURACY:70.77%, ACCURACY:70.56%
NO DATA AUGMENTATION TRAINING ACCURACY:79.38%, ACCURACY:61.28%

As seen above the difference between the training and testing accuracy is very large when data augmentation may not be utilised. The difference in the training and testing accuracy is 18.1% and is 0.21 % for no data augmentation and augmentation cases respectively. Hence this shows that when data augmentation is carried in training data sets with different changes in the parameters, the larger variation in features are present which leads to better feature extraction subnet which increases the accuracy by a large margin and maintains a lower difference between the training and test data accuracies. Sometimes when data augmentation is not carried out, there is a possibility of creation of dataset bias towards a particular class and hence accuracy is hampered.

Comparison between Relu and leaky Relu

Figures 8 and 13 are representative of the comparison between the standard leaky Relu activation and the experiment with Relu, the accuracies obtained in both the cases are as follows:

RELU , ACCURACY:70.68%
LEAKY RELU, ACCURACY:70.56%

The non-linear activation functions are most important in deciding the labels and classification. In the above experiment, it is shown that the difference between RELU and LEAKY RELU activations is not appreciable. If improper learning rates were chosen then the RELU would have created dead RELU units which would cause drastic change in accuracies. Since the learning rates chosen are appropriate, the non-linear activation functions do not have a large difference in the accuracies. Generally, if better accuracies are needed to be obtained, then these functions need to be learned instead of constant functions which is discussed in detail in section 2 (b).

Comparison between Xavier and Zeros

Figures 8 and 14 are representative of the comparison between the standard leaky Relu activation and the experiment with Relu, the accuracies obtained in both the cases are as follows:

INITIALISATION WITH ZEROS, ACCURACY:70.13%
INITIALISATION WITH XAVIER, ACCURACY:70.68%

The nominal increase in accuracy using Xavier initialisation is evident from the above experiment but hardly affects the error and accuracy curves. If larger change through

initialisation is expected, unsupervised learning algorithms like K means can be utilised as seen in section 1 (c) and mentioned by Prof Kuo's paper. Significant change is not seen in both methods because Xavier works well only when the signal input is containing images or input which has very large variance and initial pre-processing and normalisation is not done significantly. Variance scaling can be done by Xavier and hence it works when variance scaling is not done, optimal output can be obtained by choosing Xavier initialisation instead of initialisation of all the weights to zeros. Xavier initialisation ensures that the weights are just set right within a reasonable range of values through many layers of convolution.

The best performance in terms of accuracy were obtained in the case with the parameters: **EPOCH**:100, **DROP OUT**:0.9(keeping %), **BATCH SIZE**:128, **OPTIMIZER**: ADAM, **NON LINEAR ACTIVATION**: Leaky Relu, **INITIALISATION**: Xavier, **LOSS FUNCTION**: Cross Entropy by NLL, **DATA AUGMENTATION**: Yes, **LEARNING RATE**: 0.001 with **TESTING ACCURACY: 70.94%** as shown in figure 9.

## 1 (c) K-means with CNNs

Network initialisation technique is any important aspect that deals with the error propagation and accuracy of any deep neural network. In general, random initialisation techniques are carried out which randomly assign the weights of the layer. Prof Kuo in his paper mentions the need to consider CNN as a two stage process in which we can broadly divide the network into feature extraction(FE) sub band and Decision making sub band. K means provides an unsupervised method of initialisation of the network instead of random initialisation.

After the pre-processing of data is completed with zero mean and unit normalisation considering the standard deviation, k means is applied at each input layer and is carried forward layer by layer. K means initialisation provides 10 centroids at the output node and hence the anchor vectors point to ten different clusters instead of overlapping label creation which is unique about k means.

It is clearly evident that if proper pre-processing is carried out then weight initialisation by standard Xavier, Gaussian and other distributions does not affect the output significantly but unsupervised initialisation changes the accuracy significantly and can better the accuracy to up to 14 percent above standard performance.

***PROCEDURE***

Firstly, extract 10 random patches every 1000 samples for each input data image. This method is computationally expensive and needs multi-core GPU for the processing. In this question, I have applied k means initialisation only for the input layer and random initialisation for the rest of the layers as it cannot be efficiently done on a single core CPU.

The K parameter applied for each layer depends on the number of filters that are being used in each individual layer. For the 1$^{st}$ layer k=6 and for the following layers it is dependent on the number of filters that are utilised. Since it is computationally expensive, it is efficient to take small batches that are shuffled regularly.

It is essential to stack up all the input test images into 1D arrays in order to have efficient k means unsupervised learning parameters. Carrying out such a process helps in creating only 6 centroids for the 1$^{st}$ layer and then corresponding 16 centroids in the hidden

convolutional layers. Hence these centroids are representative for each class repeated over each batch size.

If K means initialisation was applied to each layer as carried out by Prof Kuo, the expected output would be like the figure below:

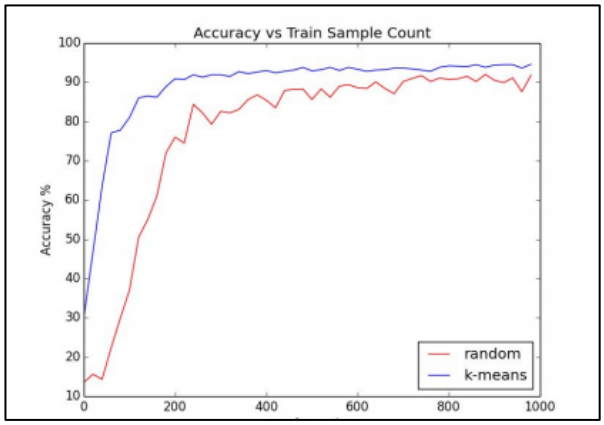



Fig 15: Prof Kuo's experimental results after applying k means at each layer from input to output stage. Drastic change due to unsupervised clustering at the Feature extraction subnet.

## ***EXPERIMENTAL RESULTS***

Since the K means was applied to only the first layer of convolution and not all the stages, I obtained a very small change in the accuracy which is not as drastic as Prof Kuo's output in figure above. The parameters used in standard model with drop out 0.5 that was utilised in figure 8 were utilised with initialisation of zeros represented in blue and then with initialisation only for the 1st layer k means and zeros for the rest of layers in violet.

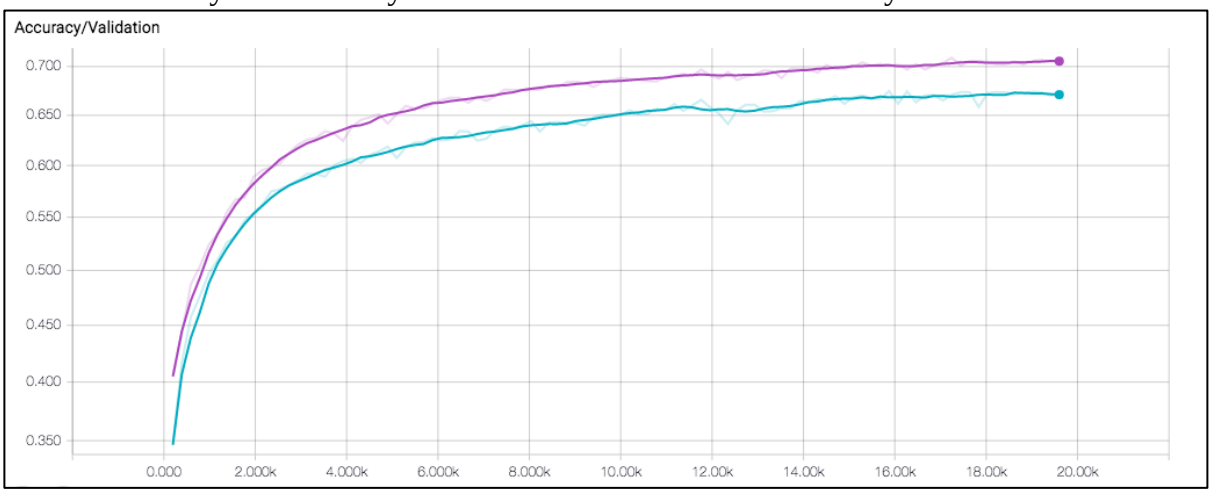



Fig 16: The difference with initialisations with k means for 1st stage and zeros for all other cases represented in violet and the zeros initialisation throughout in blue.

**Standard Initialisation in** blue
**EPOCH**:100, **DROP OUT**:0.5, **BATCH SIZE**:256, **OPTIMIZER**: ADAM, **NON LINEAR ACTIVATION**: Leaky Relu, **INITIALISATION**: Zeros, **LOSS FUNCTION**: Cross Entropy by NLL, **DATA AUGMENTATION**: Yes, **LEARNING RATE**: 0.001

**Initialisation by K means only for input stage and zeros in the following layers in violet**
**EPOCH**:100, **DROP OUT**:0.5, **BATCH SIZE**:256, **OPTIMIZER**: ADAM, **NON LINEAR ACTIVATION**: Leaky Relu, **INITIALISATION**: K means at input stage followed by zeros everywhere, **LOSS FUNCTION**: Cross Entropy by NLL, **DATA AUGMENTATION**: Yes, **LEARNING RATE**: 0.001

ACCURACY for Standard Initialisation in blue: **67.08%**

ACCURACY for Initialisation by K means only for input stage and zeros in the following layers in violet: **71.02%**

## ***CONCLUSION and DISCUSSION***

According to the above results it is pretty evident that k means initialisation is a viable option when we have multiple core GPU for processing since it is very computationally expensive. This process takes into consideration the effectiveness of providing a better initial guess to the system by k means unsupervised feature extraction method. If it is applied to all the layers from input to output, only then a considerable amount of betterment in the accuracy can be obtained. The figure shows a very small difference in the accuracies because weight initialisation of k means was not done to all layers.

# Problem 2: Capability and Limitation of CNN

## 2 (a) Improving Your Network for CIFAR-10 Dataset

The baseline CIFAR-10 dataset yielded a best performance of 70.94% which can be further improved by techniques that were not changed in the problem 1(b). The parameters which were not hampered during different experiments were that of taking different number of filters per stage, adding more convolutional layers, adding normalisation layers, trying different activation functions, etc. These parameters were changed and the experimental results are discussed in the next section.

***EXPERIMENTAL RESULTS***

The outputs which gave optimal results with the parameters related to each graph are mentioned below

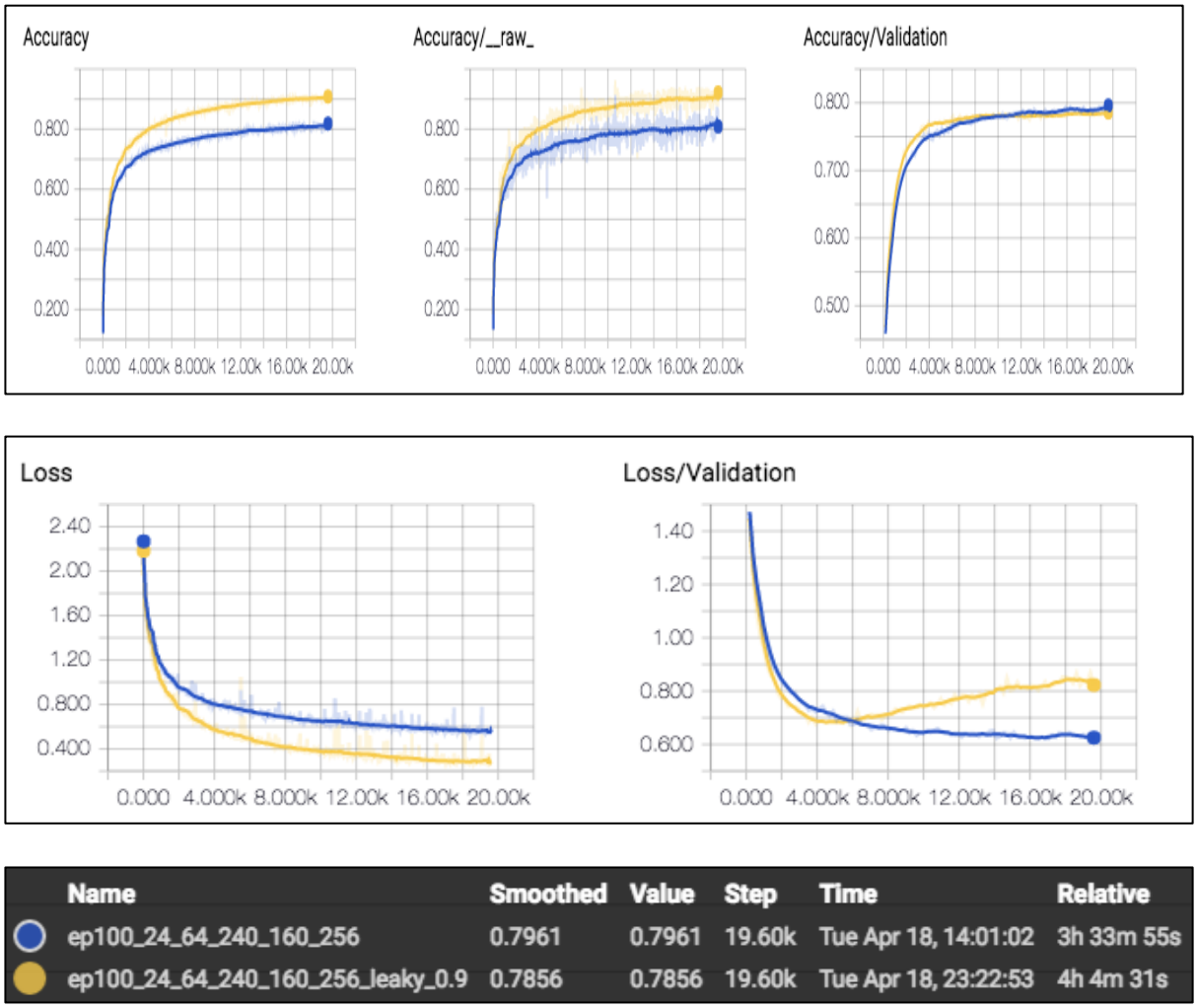



Fig 17: Effect of different filter sizes on the performance.

The parameters utilised for the output in blue are as follows:

**EPOCH**:100, **DROP OUT**:0.9(keeping %), **BATCH SIZE**:256, **OPTIMIZER**: ADAM, **NON LINEAR ACTIVATION**: leaky Relu, **INITIALISATION**: Xavier, **LOSS**

**FUNCTION**: Cross Entropy by NLL, **DATA AUGMENTATION**: Yes, **LEARNING RATE**: 0.001
with **Convolutional layer** 1 with filter size = 24 with stride=1 and no padding,
with max pooling of size=2.
with **Convolutional layer** 2 with filter size = 64 with stride=1 and no padding,
with max pooling of size=2.
with **Fully connected layer 1** of size = 240,
with **Fully connected layer 2** of size = 160.

And the parameters utilised for output in yellow are as follows:

**EPOCH**:100, **DROP OUT**:0.9(keeping %), **BATCH SIZE**:256, **OPTIMIZER**: ADAM, **NON LINEAR ACTIVATION**: Relu, **INITIALISATION**: Xavier, **LOSS FUNCTION**: Cross Entropy by NLL, **DATA AUGMENTATION**: Yes, **LEARNING RATE**: 0.001
with **Convolutional layer** 1 with filter size = 24 with stride=1 and no padding,
with max pooling of size=2.
with **Convolutional layer** 2 with filter size = 64 with stride=1 and no padding,
with max pooling of size=2.
with **Fully connected layer 1** of size = 240,
with **Fully connected layer 2** of size = 160.

The graph utilised for this implementation is as follows:

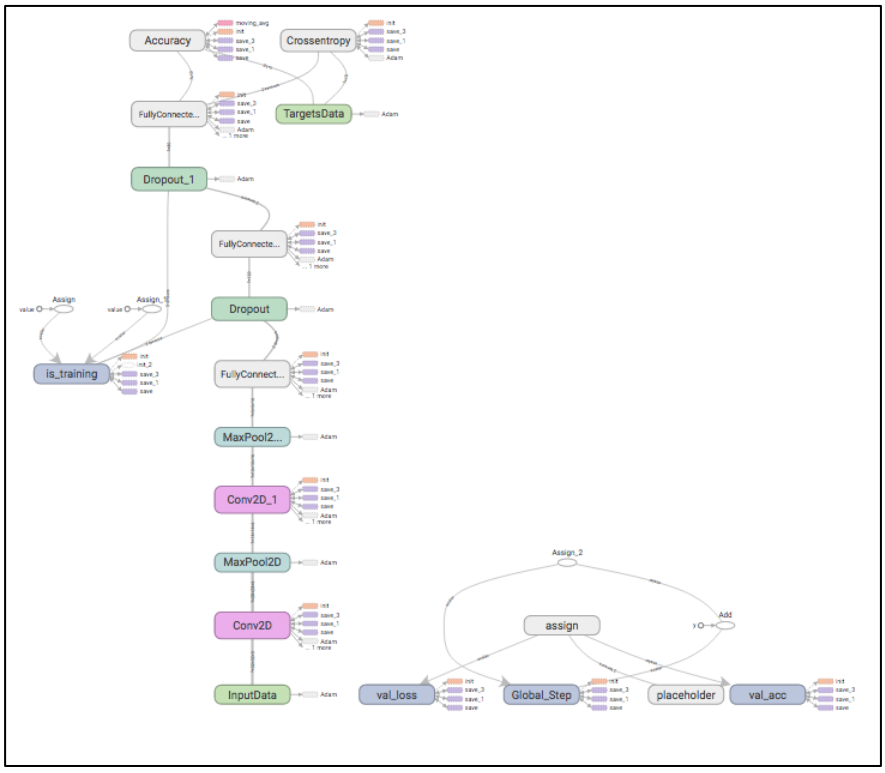



Fig 18: Graph for the implementation of the optimal case with accuracy of **79.61%**

**Accuracy with Relu and architecture mentioned above in yellow: 78.56%**

**Accuracy with Leaky Relu and architecture mentioned above in blue: 79.61%**

***CONCLUSION and DISCUSSION***

As seen above the accuracy was improved by 10 percent by changing the number of filters after the optimal setup was made for the baseline CNN for CIFAR-10 dataset. Better results can be obtained by using K means for initialisation but is computationally expensive.

Since we included more filters after data augmentation more variance in the filters is available for better feature extraction. In the baseline LeNet-5 architecture the 1st convolutional layer had only 6 filters which was replaced by 24 which aided in lower level feature extraction like edges, contours, etc. Max pooling layer is similar to baseline architecture because if larger pooling is carried out the effect of the larger feature extraction would be nullified. Similarly, the second convolutional layer has 64 filters instead of 16 filters which creates more activation maps which are fruitful in extracting the higher level features like textures and better feature extraction subnet is created which is followed by a max pooling layer similar to the previous step.

Since the objects to be classified in each of the datasets are not very closely associated to the boundary edges, we do not carry out zero padding and since stride of 1 leads to smaller neuron responses, we choose stride 1 and no zero padding.

Data augmentation and pre-processing is carried out similar to baseline CNN and provide similar advantages of smaller difference between training and testing accuracies and avoid any kind of dataset bias.

Activation functions tried out for both the cases are Relu and Leaky Relu which show that Leaky Relu and Relu provided hardly significant change with Leaky Relu providing a better accuracy than Relu of 79.61%. Better accuracy could be obtained if these activation functions for each neuron was not fixed constant but if these are also learnt iteratively as mentioned in the section 2 (b).

Drop out factor is kept to 0.9 similar to baseline CNN and Fully connected layers that are responsible for decision making similar to MLP also have been increased to twice their values. In this manner more fully connected neuron sets are created and the computation time which was 1 hr 12 m 32 s is changed to 4 h 4 m 31 s. Better results are obtained as there are more connections for each neuron and the decision making capability is enhanced with better back propagation and weight update process in larger neural net.

The learning rate is kept to 0.001 which was found optimal for the baseline case and the optimiser used is still ADAM and the loss functions are still Softmax by categorical cross entropy which is optimal since it deals with obtaining the higher discriminant power while taking into consideration the one hot vectors in the training.

The comparisons done in section 1 (b) are been fruitful in deciding the parameters for this section. Better results can be obtained if

1. Better multi core GPUs are used with more convolutional layers instead of 2 convolutional layers.
2. Usage of activation functions which learn differently for each neuron and update after each back propagation.
3. Initialisation of weights is done by unsupervised learning algorithm of k means or similar to provide weights that are better oriented according inputs.
4. Using K NN supervised learning algorithm in training and choice of the top k parameter for validation but this is not considered robust.

The output accuracy can be enhanced drastically to about 97 % if the learning of the training data is carried out on multi core GPU with the above mentioned parameters and since in this case a CPU was used for training with processor limitation, around 80 % accuracy was obtained.

## 2 (b) State-of-the-Art CIFAR-10 Implementation

The State of the Art CIFAR- 10 implementation that fetches and accuracy of 92.49 % is that of the Learning activation functions to improve deep neural networks (https://arxiv.org/pdf/1412.6830.pdf). This paper is credited to Forest Agostinelli, Matthew Hoffman and Peter Sadowski, Pierre Baldi at the University of California at Irvine(UCI).

OUTLINE

The paper intends to utilise the piecewise linear activation function which is learned independently by each neuron using gradient descent. Using this deep neural network design, which is composed of static rectified linear units, it is able to achieve performance on CIFAR-10 (7.51%) and 30.83% on the CIFAR-100 dataset. Generally, a non-linear activation function is fixed for all the layers of convolution, max pooling and fully connected layers but in this approach this approach the activation functions are also learned during the training process itself. The piecewise linear activation function can learn independently for each neuron using SGD, and can hence be used to represent convex as well as non-convex functions.

ADAPTIVE PIECEWISE LINEAR UNITS

Rectified linear units or RELU is commonly used non-linear activation function which is non zero and has positive slope only for positive values which is fixed. But the adaptive piecewise rectified linear units (APL) tend to learn from each neuron and change according to the characteristics of the learned parameters. The general form of RELU is appended with a learning parameter which is stated below where a and b are learning parameters:

$$h_i(x) = \max(0, x) + \sum_{s=1}^{S} a_i^s \max(0, -x + b_i^s)$$

S is the hyper parameter that is learned during SGD and a determines the slope of the activation APL units. The total number of additional parameters that need to be learnt in this process are equal to 2*S*M where M is the total number of hidden units in the network. The figure below demonstrates the various APL units for different values of a and b with focus on how S parameter update can change the activation function for each of the neurons as these activation functions clearly adversely affect the performance of the feed forward as well that Backward propagation.

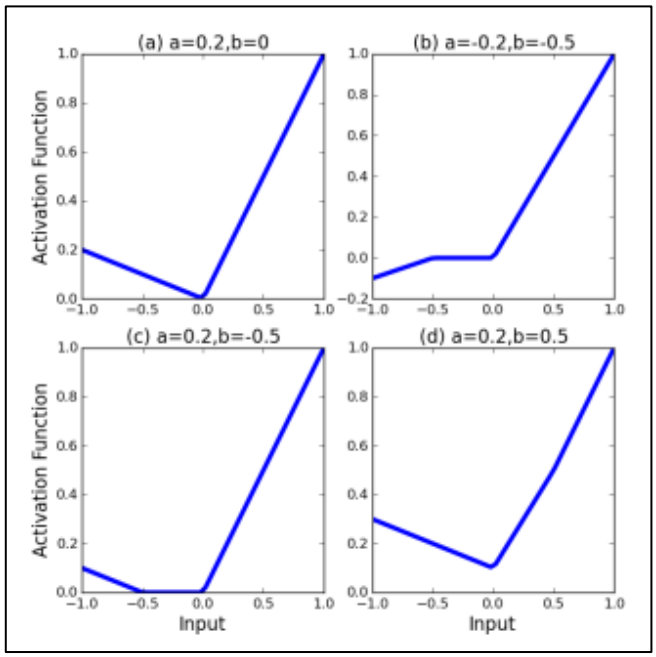



Fig 19: Change in the learning parameters a and b on the APL.

The APL outperforms the Maxout which takes a series of multiple linear functions and returns the largest among them. Maxout functions heavily depend on the weight parameters of each neuron. The order of complexity is higher than that of the APL. Since the expressive power of APL is sufficiently large enough, it would be a waste of computational and modelling power to use Maxout.

Similarly, the Network in Network although is a better approach to be used that just fixed activation functions, a hand in hand club of both APL and Network in Network can be used to attain larger values of accuracy at lower number of epochs successfully.

COMPARISON BETWEEN TRADITIONAL CNN AND APL

The traditional method of non-linear activation along with the most recent approaches of Maxout and Network in Network are compared for both the CIFAR-10 dataset and the CIFAR 100 data set and the results of the drop in error are shown below:

| **Method** | **CIFAR-10** | **CIFAR-100** |
|---|---|---|
| Without Data Augmentation | | |
| CNN + ReLU (Srivastava et al., 2014) | 12.61% | 37.20% |
| CNN + Channel-Out (Wang & JaJa, 2013) | 13.2% | 36.59% |
| CNN + Maxout (Goodfellow et al., 2013) | 11.68% | 38.57% |
| CNN + Probout (Springenberg & Riedmiller, 2013) | **11.35%** | 38.14% |
| CNN (Ours) + ReLU | 12.56 (0.26)% | 37.34 (0.28)% |
| CNN (Ours) + Leaky ReLU | 11.86 (0.04)% | 35.82 (0.34)% |
| CNN (Ours) + APL units | 11.38 (0.09)% | **34.54 (0.19)%** |
| NIN + ReLU (Lin et al., 2013) | 10.41% | 35.68% |
| NIN + ReLU + Deep Supervision (Lee et al., 2014) | 9.78% | 34.57% |
| NIN (Ours) + ReLU | 9.67 (0.11)% | 35.96 (0.13)% |
| NIN (Ours) + Leaky ReLU | 9.75 (0.22)% | 36.00 (0.36)% |
| NIN (Ours) + APL units | **9.59 (0.24)%** | **34.40 (0.16)%** |
| With Data Augmentation | | |
| CNN + Maxout (Goodfellow et al., 2013) | 9.38% | - |
| CNN + Probout (Springenberg & Riedmiller, 2013) | 9.39% | - |
| CNN + Maxout (Stollenga et al., 2014) | 9.61% | 34.54% |
| CNN + Maxout + Selective Attention (Stollenga et al., 2014) | **9.22%** | **33.78%** |
| CNN (Ours) + ReLU | 9.99 (0.09)% | 34.50 (0.12)% |
| CNN (Ours) + APL units | 9.89 (0.19)% | 33.88 (0.45)% |
| NIN + ReLU (Lin et al., 2013) | 8.81% | - |
| NIN + ReLU + Deep Supervision (Lee et al., 2014) | 8.22% | - |
| NIN (Ours) + ReLU | 7.73 (0.13)% | 32.75 (0.13)% |
| NIN (Ours) + APL units | **7.51 (0.14)%** | **30.83 (0.24)%** |

It can be seen that as a combination of Network in Network (NIN) is utilised with the ALP the best results are obtained in both the cases of CIFAR-10 and CIFAR 100 which shows that the activation functions also need to be learnt in the process of CNN as well as NIN deep learning.

## CONCLUSION

It is visible evidently that CNN as well NIN need to have learning based activation functions for better accuracy and lower errors in the system at fewer epochs. It also showcases that an independent piecewise linear function for each neuron activation is far better than fixed approach. Gradient descent along with weights in consideration help attain this task without largely affecting the baseline parameters of the CNN architecture. It can also be concluded that we cannot have a single activation function that can perform well for all neural network layers instead should focus primarily on finding the function which fits that particular net optimally.

## References

[1] LeNet-5 http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf
[2] CIFAR-10 https://www.cs.toronto.edu/~kriz/cifar.html
[3] ReLU https://en.wikipedia.org/wiki/Rectifier_(neural_networks).
[4] Softmax https://en.wikipedia.org/wiki/Softmax_function
[5] Prof.'s Paper https://arxiv.org/abs/1701.08481
[6]CIFAR-10 Leaderboard
http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html#43494641522d3130.
[7]Tensorflow, Tflearn CIFAR-10 tutorial git-hub
https://github.com/tflearn/tflearn/blob/master/examples/images/convnet_cifar10.py
[8] Learning Activation Functions to Improve Deep Neural Networks
https://arxiv.org/pdf/1412.6830.pdf