HOMEWORK #5

Issued: 03/23/2020 Due: 04/07/2020

Problem 1: CNN Training on LeNet-5

I. Abstract and Motivation:

LeNet-5 Architecture: The Artificial Neural Networks (ANN) previously called as Multi-Layer Perceptron (MLP) were popular for the raw data samples. But for images, it was a tedious process to achieve convergence on such a huge dimension image data samples. So, the idea is to represent an image as a single feature vector and then make the neural network to learn on these feature vectors. But the idea is to extract the features from the image automatically without the intervention of humans. So, the need of an end-to-end architecture where the features are automatically extracted was much needed. LeCun et.al proposed a state of the art neural network called Convolution Neural Network (CNN) [6]. The architecture showed best performance on MNIST data set which are gray level handwritten number images. Further a new era of modern computer vision prevailed over all the contemporary methods when AlexNet achieved a staggering 92% accuracy on ImageNet data set.

Figure 1. shows the LeNet 5 architecture.

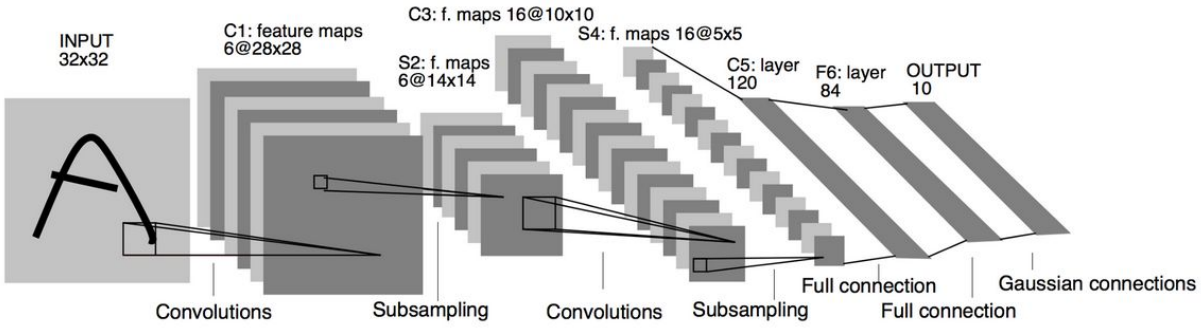

Figure 1: LeNet-5 Architecture

Totally there are 60000 learnable parameters in the network. It has 3 convolution layers, 2 max-pooling layers, and 1 Fully connected layer followed by the output layer. The architecture is primitive compared to today's CNNs where we have millions of parameters with 100s of layers.

Figure 2 shows the architectural overview of LeNet-5

Layer	Layer Type	Feature Maps	Size	Kernel Size	Stride	Activation
Input	Image	1	32x32	-	-	-
1	Convolution	6	28x82	5x5	1	tanh
2	Average Pooling	6	14x14	2x2	2	tanh
3	Convolution	16	10x10	5x5	1	tanh
4	Average Pooling	16	5x5	2x2	2	tanh
5	Convolution	120	1x1	5x5	1	tanh
6	Fully Connected	-	84	-	-	tanh
Output	Fully Connected	-	10	-	-	softmax

Figure 2: Architectural Overview of LeNet- 5

LeNet-5 achieved an accuracy of 98% on MNIST dataset. But the accuracy of LeNet-5 on modern data sets like ImageNet, CIFAR-10 is not so good.

Function and operation of Individual layers are explained briefly in the Results and Discussion section (Question - Answers).

II. Approach and Implementation:

1. The implementation is done in PyTorch (a python framework built on tensor flow).
2. Anaconda is used as a package manager.
3. Jupyter notebook is used as a code editor.
4. CIFAR-10 data is loaded and the data is split into 50000 train images and 10000 test images.
5. Normalized the dataset by subtracting each pixel with mean and dividing each pixel by standard deviation.
6. Normalization ensures faster convergence.
7. Stochastic gradient descent method is used for optimization and a batch size of 64 is chosen for train data and 1000 is chosen for test data.
8. Check whether the image classes are balanced, i.e., if there is any skew in the number of images in each class.
9. Build a CNN by defining each layer and their inputs and outputs.
10. Train the neural network using the training images by setting suitable parameters.
11. Obtain the train and test accuracy.
12. Obtain accuracy in each class.
13. Plot the accuracy vs number of epoch for test and train data.
14. Repeat the process for different parameter values.
15. Observe and comment on the result.

III. Results and Discussion:

A. CNN Architecture: (Question and Answer)

1) CNN Components:

i) Fully Connected Layer: Fully connected layers are the fundamental components in traditional neural networks (also referred as Multi-Layer Perceptron). It takes an n dimensional array as an input and gives classification results in the output layer. The input passes through a series of layers called hidden layers. The neuron in each layer is connected to every other neuron in a subsequent layer. Figure 1 shows a fully connected layer.

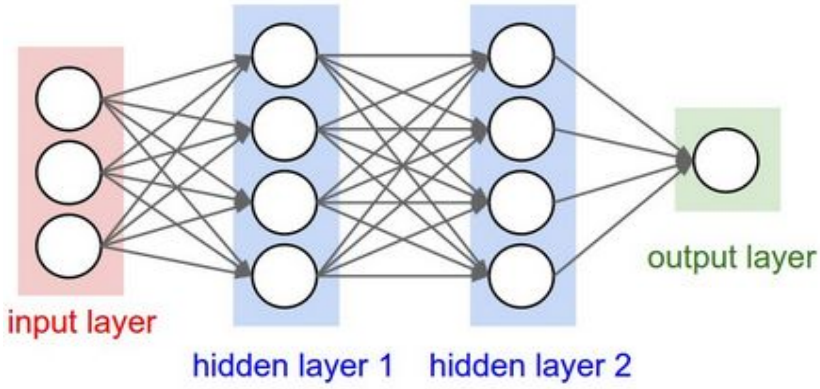

Figure 1: A fully connected layer.

Each neuron in a fully connected layer takes the input from the previous layer's activation function along with weights and a bias bias term. It calculates the inner product between the weight vector and the input vector and adds a required bias. Figure 2 shows a single neuron

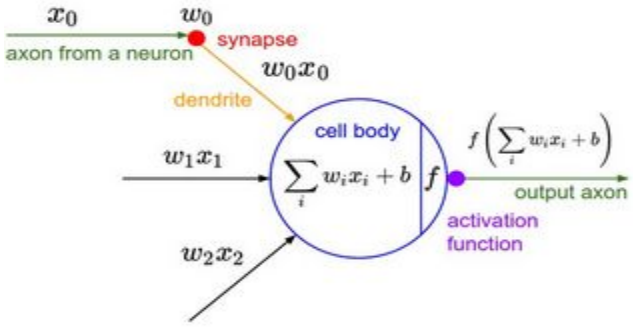

Figure 2: A single Neuron

ii) **Convolution Layer**: The convolutional layers are the principal blocks in CNN architecture. Convolutional layer performs most of the heavy computation to extract features from an input volume through a set of learnable filters. Conventionally convolution in terms of signal processing means, the filter should undergo mirror operation before convolution. However, in this case we are skipping this step. So, convolution can be more specifically referred to as **cross correlation.** When we convolve an image of size $n * n$ with a filter of size $f * f$ which are k in number, we will get a 3D output volume where the spectral component corresponds to the feature map of each filter. The output volume obtained as a result of convolution is called a **feature-map**. Based on the spatial dimension of the output volume, we have three types of convolution.

Valid Convolution: In this type of convolution, the spatial dimension of the output volume will reduce as there will be no padding. So the output dimension will be $(n-f+1) * (n-f+1)$.

Same Convolution: In this we usually pad the input volume, so that the output size remains the same. So the output dimension will be $(n+2p-f+1) * (n+2p-f+1)$ with $p = \frac{f-1}{2}$.

Strided Convolution: Conventionally, we move the filter by one pixel while performing convolution. However, we can move the filter by more than 1 block at a time. This parameter is called ***stride.*** The dimension of the output volume will be,

$$(\frac{n+2p-f}{s}+1) * (\frac{n+2p-f}{s}+1)$$

Hyperparameters: While specifying a convolution layer, we describe following hyperparameter,

$$f: filter\ size,\ s: stride,\ p: padding,\ k: number\ of\ filters$$

Therefore, the output volume will be,

$$(\frac{n+2p-f}{s}+1) * (\frac{n+2p-f}{s}+1) * k$$

Figure 1. Shows the convolution operation and the resulting feature map.

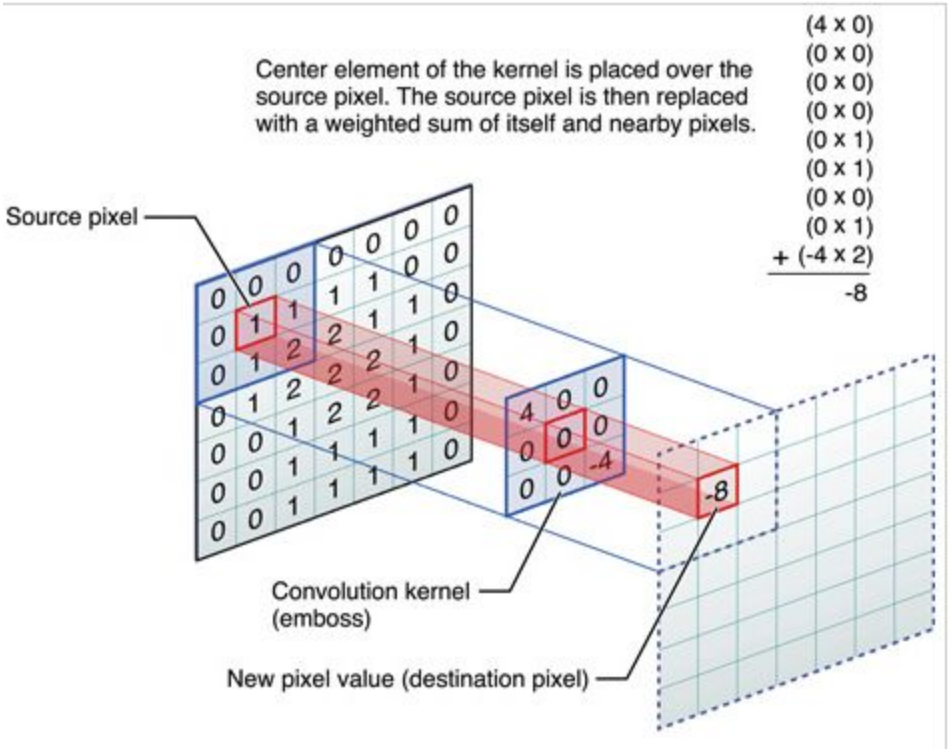

Figure 1. Convolution operation and the resulting feature map.

iii) Max pooling Layer: Pooling layer is usually added after a convolution layer followed by non-linear activation. Pooling layer drastically reduced the spatial dimension of the feature map obtained after convolution. Employing pooling layers will reduce the number of parameters and computations in the CNN. Also, it helps to avoid overfitting problems to some extent. In signal processing notion, pooling can be considered as downsampling of a signal. We perform either max-pooling or average pooling but the previous is more efficient than the latter.

The hyperparameters of a pooling layer are,

$$f: \text{spatial extent}, \; s: \text{stride}$$

Figure 2 shows the max-pooling operation with a spatial extent of 2 and with stride 2.

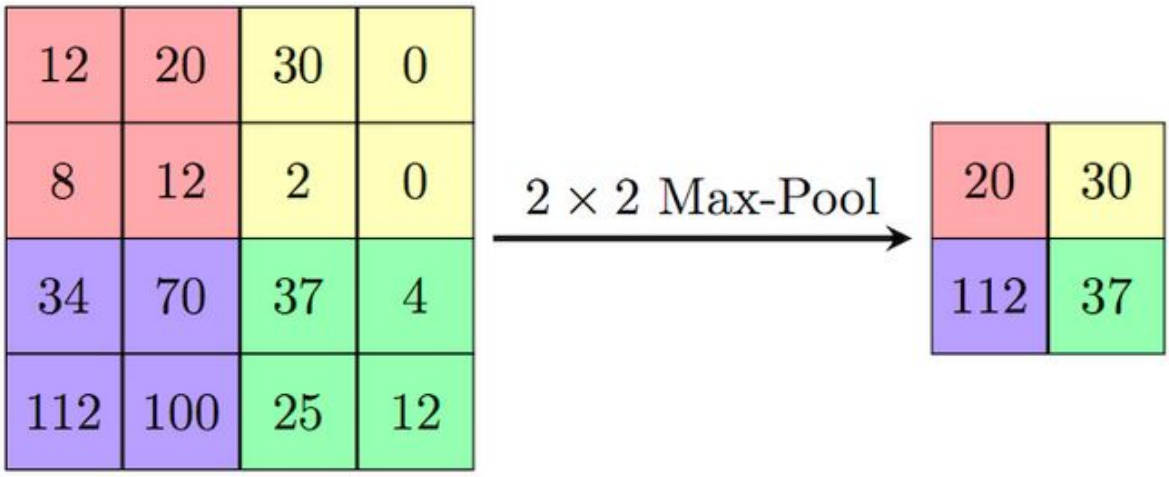

Figure 2 Max-pooling operation

However few works have proposed to completely remove the pooling layer and employ repeated convolution layers with higher stride value. This caters the need of downsampling and also provides better results in Generative Adversarial Networks (GANs) [3].

iv) **Activation Function:** After calculating the inner product between the weight vector and the input vector, we give the result to an activation function which provides activation for the next layer. The activation is also called non-linearity. There are many activation functions we can use.

Sigmoid: Sigmoid nonlinearity takes a real number as input and wraps the output in between 0 and 1. The mathematical representation of sigmoid function is, $\sigma(x) = 1/(1 + e{-}x)$. But the major drawback of sigmoid activation is that it kills the gradient at the extremas which is undesirable

Tanh: tanh activation function is very similar to sigmoid activation, but it wraps the real valued input in between -1 and 1. The mathematical representation of tanh is $\tanh(x)=2\sigma(2x)-1$.

ReLu: Nowadays, ReLu (Rectified Linear Unit) has become popular because of its robustness and gradient won't saturate for large input values. The mathematical representation of ReLu is

$f(x) = max(0, x)$. The main advantage of using ReLu non-linearity is that it speeds up the convergence of Stochastic Gradient Descent. Figure 3 shows the ReLu activation function.

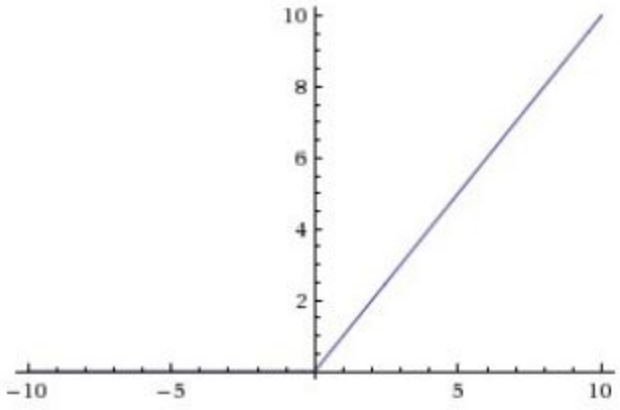

Figure 3: ReLu activation function.

Need for Non linearity in Neural Networks: As we have seen, taking inner product between weight vector and inputs is a linear operation. If we don't add a non linearity, then it doesn't matter how many layers we add, it will be typically considered as a single layer neural network because inputs in subsequent layers can be reconstructed as the linear combination. This is undesirable in back propagation as it requires non linear mapping.

v) **The Softmax function:** The numerical values of the last layer of the neural network aren't probabilities (it doesn't sum up to 1). To make them probabilities, we use a softmax layer. The mathematical representation of a softmax layer is $y' = \frac{e^{(y)}}{\sum e^{y}}$. Softmax functions are also called **multi-class sigmoids** because the output-layer vectors to probabilities at a time. The softmax layer becomes costly if there are a large number of classes. In this case, we use **candidate sampling** where the softmax calculates probabilities only to a subset of classes. Figure 4 shows the softmax layer in a neural network.

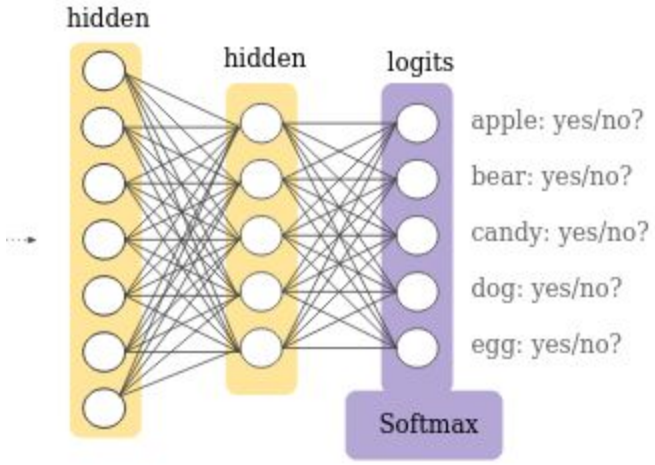

Figure 4: Softmax Layer

2) Overfitting in Model Learning: In supervised learning, we train the data with the known labels (target values) to learn the model parameters. Then we use the model parameters to predict/classify the new data. So, the efficiency of a machine learning algorithm is measured by its ability to generalize the unseen data (test data). Therefore, **Generalization** is the key metric here.

Overfitting happens when the model learns minute details of the training data like noise and distortions, which can hamper the performance on test data. Overfitting refers to the condition when the model performs extremely well on training data but miserably fails to generalize on test data. Similar to overfitting there is something called **underfitting** where a model's performance will be abysmal on both training and test data.

Regularization to avoid Overfitting: Inorder to prevent overfitting, one straightforward approach is to increase the training data. But, it is very hard to get more training data especially while training with medical images. So, we need to employ a mathematical technique to avoid overfitting. L2 Regularization is a common way in which we add a regularization term to the Loss (Objective function). Therefore, the objective function after adding regularization term will be,

$$J(w^{(1)},\ b^{(1)},.....w^{(l)},\ b^{(l)}) \ = \ \frac{1}{m}\sum_{i=1}^{n} L(y^1 - y) \ + \ \frac{\lambda}{2m}\sum_{l=1}^{L} ||w^{(l)}||^2$$

Where, λ is the regularization parameter. While updating the weights we differentiate the objection function, so the weight update will be decreased because of the regularization term. Hence regularization is also called the **"weight decay"** process.

Basically, the regularization term penalizes the weight vectors which have large magnitude. This makes the activation to be in the linear region which normally yields linear decision boundaries instead of complicated boundaries which overfits the training data.

Dropout Regularization: In recent times, this technique is widely used to prevent overfitting. Dropout regularization was proposed by Nitish et. al [4]. In this method we set a probability of a node which determines whether we drop the particular node or not. We can also use a threshold probability to make the decision. This eliminates few nodes in the layer which makes the computation simpler avoiding the overfitting. Figure 5 shows the result of using dropout regularization.

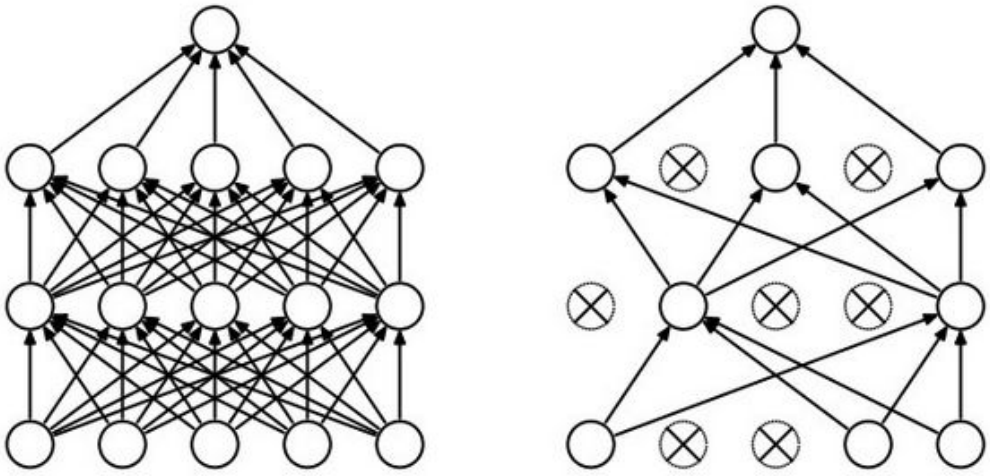

Figure 5: Dropout Layer

3) Reason for the success of CNN: The advancement in deep neural networks (DNNs) has altered the paradigm of computer vision domain by pushing the boundaries of the problems solved using computer vision. Before exploring the depth of neural networks, we need to understand the **descriptive analysis** and **predictive analysis**. In descriptive analysis, a mathematical model is developed

by observing the patterns of the domain and that model is expected to encapsulate all the inherent properties of the phenomenon. This kind of approach was very complicated to understand and often became futile. On the other hand predictive analysis is similar to what we do in classical pattern recognition and machine learning.

Before deep learning, the traditional approaches like SIFT/SURF feature extractors, Bag of Visual words were used for vision problems where the role and expertise of a CV engineer was to decide the important features and design filter weights. This type of methodology has a serious flaw when we want to incorporate object recognition systems on portable devices like mobiles. But with CNNs it's **end-to-end learning** where the role of CV engineer was shifted from designing feature extractors to building CNN architectures. All we need is an image dataset and labels. It is ensured that deep learning models outperforms the classical computer vision algorithms. The trade off is computation time, GPU resource and the need of a huge dataset. Figure 6 shows the transition of computer vision engineers' roles.

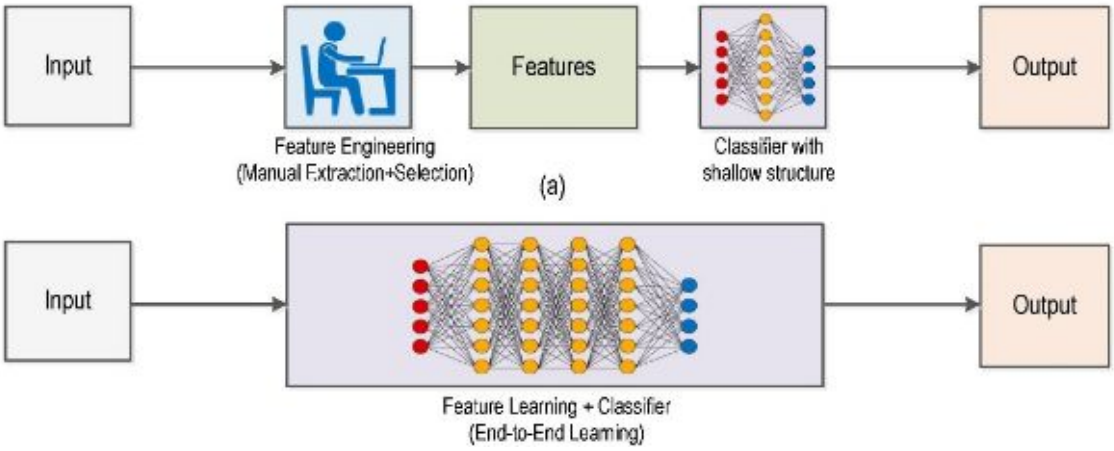

Figure 6: Traditional Computer Vision (a) vs Deep Learning (b)

The results obtained from CNN on computer vision problems image classification easily surpasses the results of classical computer vision algorithms. The only trade off is deep learning needs a large data set, high computational resources and time. Following are the reasons why CNN performs better,

a) CNN is a **data driven** approach (big data) where traditional CV algorithms restrict the ability to extract features from a single image.

b) **Back propagation** is the backbone of deep learning. Unlike tradition CV, the filter weights are learnt by the neural network by automatically adjusting the weights to reduce the error.
c) Some researchers attribute the success of CNNs to it's similarity with the **workflow of the brain.** This is still a contentious topic where debates are still going on
d) The automatic features learnt from the huge data set were more efficient than the features designed by CV engineers.

However, CNNs can't be used for all the vision problems. Some of the Image processing tasks can be more efficiently done using traditional CV algorithms. For example Image Stitching, Denoising, 3D Vision, Segmentation can be more efficiently done using traditional algorithms.

4) Loss function and Back propagation: Loss function/Objective function is used in traditional machine learning to train the classifier or regressor. The main objective of the algorithm is to minimize the loss function (convex optimization) to obtain the model which fits the train data. We can use Mean squared error between the predicted value and the target value as a loss function or we can use cross entropy. MSE loss function is given by,

$$J(w) = \frac{1}{2M} \sum_{i=0}^{M} [(w_i x_i + w_0) - b_i]^2$$

Similarly cross entropy loss function is given by,

$$J(w) = -\sum_{i=0}^{M} [(w_i x_i + w_0)\, log(\, b_i)]$$

Basically training is associated with minimizing the loss function using gradient descent.

Back Propagation refers to **backward propagation of error** from output layer to input layer through hidden layers. Similar to optimizing the cost function by calculating the gradient of the loss function, we propagate the negative gradient of the error function to the previous layer. The weight update rule is shown below,

$$w^{t+1} : w^t - \alpha \frac{\delta J(w)}{\delta w}$$

where, $\alpha\frac{\delta J(w)}{\delta w}$ is the gradient of the error function. The error δ^{l}_{i} from layer l is dependent on the error δ^{l+1}_{i} in layer l+1. So, the error propagates from output layer to input layer and all we need to do is, calculate the error in the last layer and then use the error to find and adjust the weights of the previous layer until it reaches the input layer. Figure 7 shows the back propagation schematic.

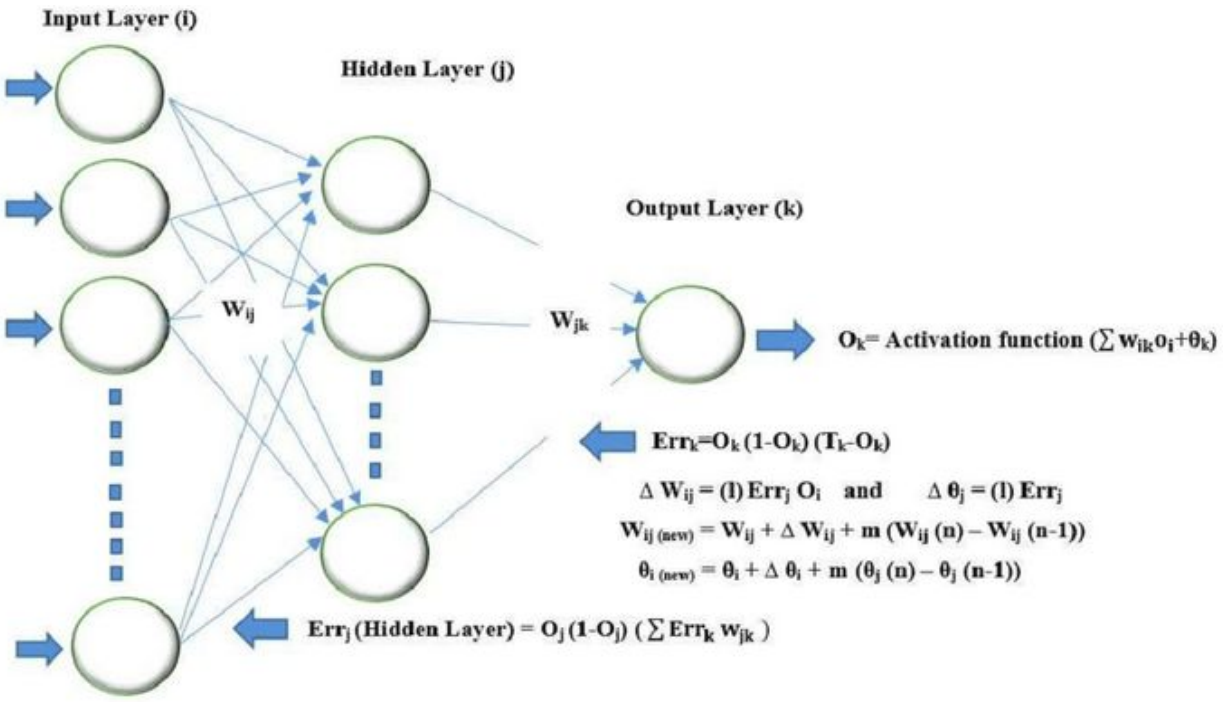

Figure 7: Back Propagation

Basically the process of optimizing the error/loss function using back propagation comes back to the problem of optimization. We usually use Stochastic Gradient Descent to optimize the error. In CNN, the learnable filter weights and biases are tuned so that the error is minimum with respect to the target values using back propagation. This is done in batches of images. So, the final filter weights minimizes the loss function and the learned filter parameters are used on test data sets.

B. CIFAR-10 Classification:

In this section, CNN architecture based on LeNet-5 is implemented and the accuracy is noted for different parameter values. CIFAR-10 data set has 50,000 training images and 10,000 test images of 10 different classes.

1. **Using SGD with Cross Entropy as criterion:**

Optimizer: SGD
Criterion: Cross Entropy Loss
Epochs: 60
learning rate: 0.01
Momentum: 0.7

Test Accuracy: 60.56%
Train Accuracy: 89.26%

Figure 8 shows the accuracy vs number of epoch plots for first parameters

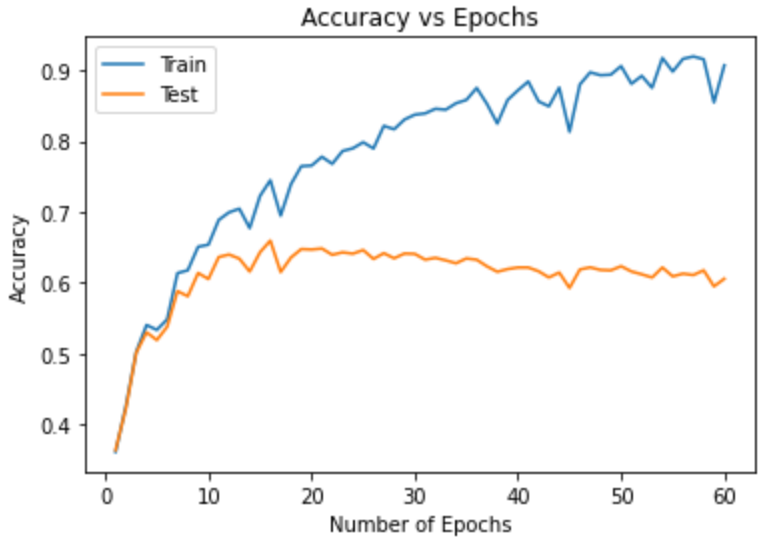

Figure 8: Accuracy vs number of epoch plots for first set of parameters

Figure 9 shows the accuracy of individual classes

```
Accuracy of plane : 42 %
Accuracy of   car : 50 %
Accuracy of  bird : 25 %
Accuracy of   cat : 33 %
Accuracy of  deer : 50 %
Accuracy of   dog : 66 %
Accuracy of  frog : 66 %
Accuracy of horse :  0 %
Accuracy of  ship : 100 %
Accuracy of truck : 75 %
```

Figure 9: Accuracy of each class for Parameters 1

2. **Decreasing the learning rate and increasing the momentum for SGD:**
   Optimizer: SGD
   Criterion: Cross Entropy Loss
   Epochs: 60
   learning rate: 0.001
   Momentum: 0.9

   Test Accuracy: 63.53%
   Train Accuracy: 81.63%

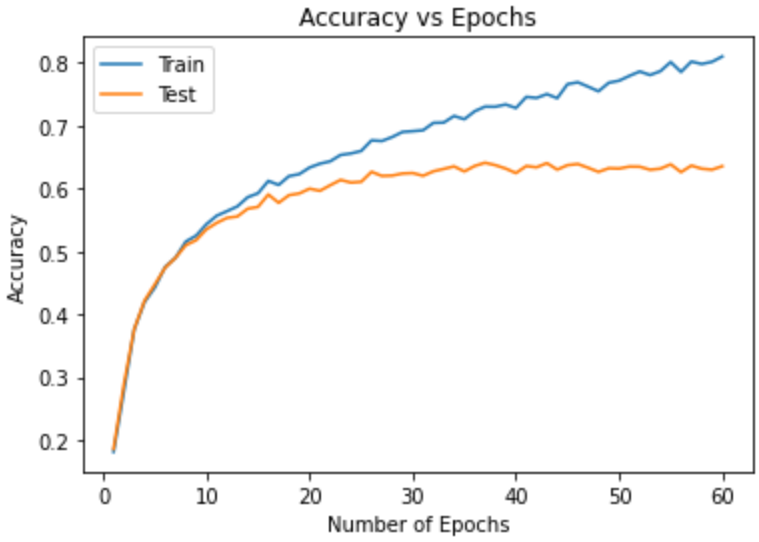

Figure 10:Accuracy vs number of epoch plots for second set of parameters

Figure 11 shows the accuracy of each class.

```
Accuracy of plane : 80 %
Accuracy of   car : 75 %
Accuracy of  bird : 16 %
Accuracy of   cat : 25 %
Accuracy of  deer : 50 %
Accuracy of   dog : 50 %
Accuracy of  frog : 33 %
Accuracy of horse : 66 %
Accuracy of  ship : 60 %
Accuracy of truck : 100 %
```

Figure 11: Accuracy of each class for parameters 2.

3. **Using ADAM (Adaptive Moment Estimation):**

Optimizer: Adam
Criterion: Cross Entropy Loss
Epochs: 60
learning rate: 0.001
Betas: (0.9, 0.999)

Test Accuracy: 59.43%
Train Accuracy: 92.75%

Figure 12 shows the plot of accuracy vs number of epochs for 3rd set of parameters

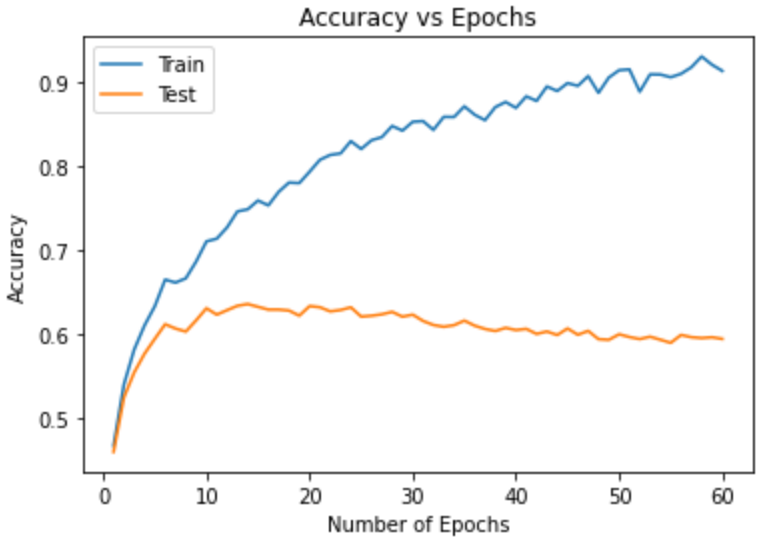

Figure 12: Accuracy vs number of epochs for 3rd set of parameters

Figure 13 shows the accuracy of each class for 3rd set of parameters

```
Accuracy of plane : 20 %
Accuracy of   car : 75 %
Accuracy of  bird : 100 %
Accuracy of   cat : 100 %
Accuracy of  deer :  0 %
Accuracy of   dog :  0 %
Accuracy of  frog :  0 %
Accuracy of horse : 85 %
Accuracy of  ship : 100 %
Accuracy of truck : 66 %
```

Figure 13: Accuracy of each class for 3rd set of parameters

4. **Increasing the learning rate of Adam Optimizer:**

Optimizer: Adam
Criterion: Cross Entropy Loss
Epochs: 60
learning rate: 0.01
Betas: (0.9, 0.999)

Test Accuracy: 51.2%
Train Accuracy: 54.5%

Figure 14 shows the accuracy vs number of epochs for the 4th set of parameters.

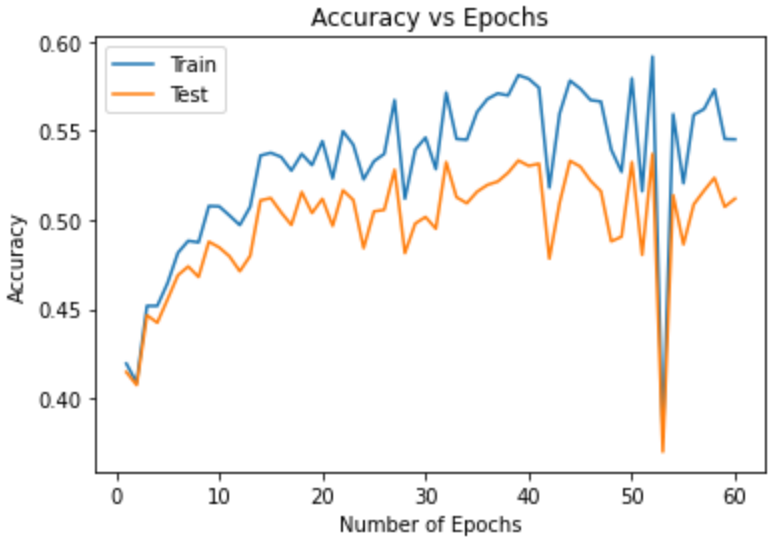

Figure 14: Accuracy vs number of epochs 4th set of parameters.

Figure 15 shows the accuracy of each class for the 4th set of parameters.

```
Accuracy of plane : 33 %
Accuracy of   car : 75 %
Accuracy of  bird : 25 %
Accuracy of   cat : 25 %
Accuracy of  deer : 42 %
Accuracy of   dog : 33 %
Accuracy of  frog : 100 %
Accuracy of horse : 50 %
Accuracy of  ship : 33 %
Accuracy of truck : 62 %
```

Figure 15: Accuracy of each class for the 4th set of parameters.

5. **Increasing the epochs of SGD.**
   Optimizer: SGD
   Criterion: Cross Entropy Loss
   Epochs: 120
   learning rate: 0.001
   Momentum: 0.9

Test Accuracy: 60.98%
Train Accuracy: 96.26%

Figure 16 shows the accuracy vs number of epochs for 5th set of parameters

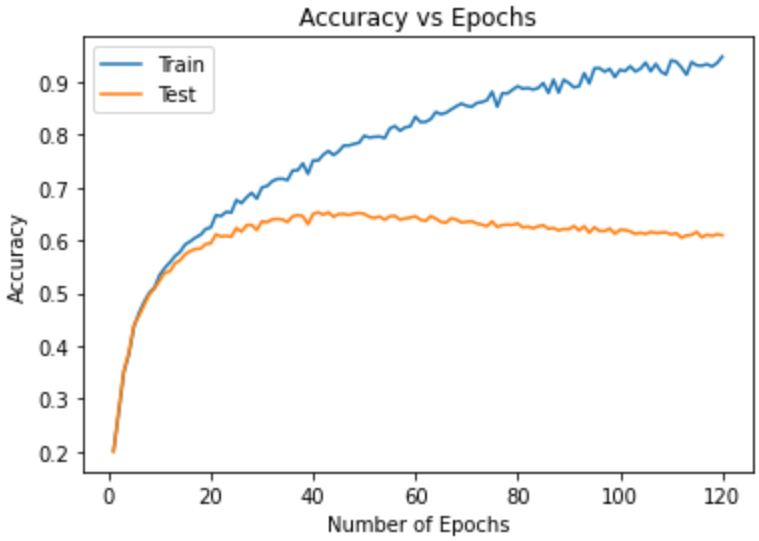

Figure 16: Accuracy vs number of epochs for 5th set of parameters

Figure 17 shows the accuracy of each class for 5th set of parameters

```
Accuracy of plane : 66 %
Accuracy of   car : 25 %
Accuracy of  bird : 40 %
Accuracy of   cat : 50 %
Accuracy of  deer : 100 %
Accuracy of   dog : 100 %
Accuracy of  frog : 100 %
Accuracy of horse : 25 %
Accuracy of  ship : 75 %
Accuracy of truck : 50 %
```

Figure 17: Accuracy of each class for 5th set of parameters

6. **Introducing weight_decay parameter for SGD.**
   Optimizer: SGD
   Criterion: Cross Entropy Loss
   Epochs: 60
   learning rate: 0.001
   Momentum: 0.9
   weight_decay : 1

Train and Test Accuracy: 10%

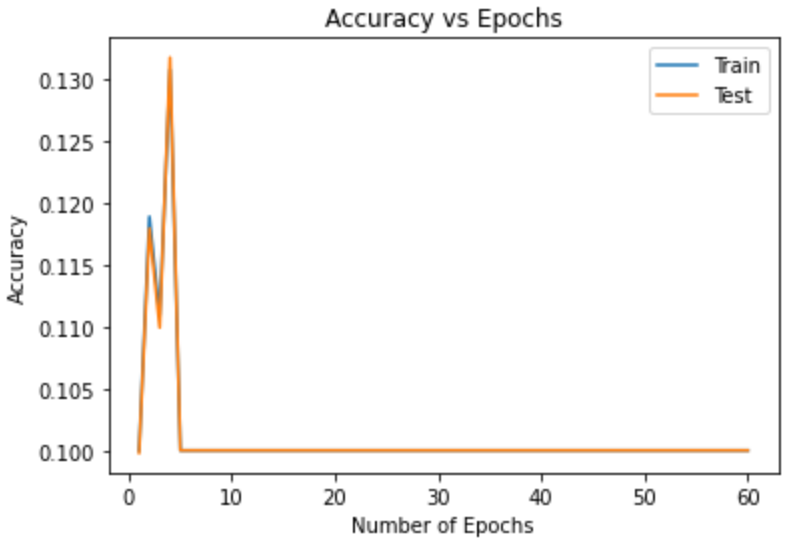

Figure 18: Accuracy vs number of epochs for 6th set of parameters

```
Accuracy of plane :  0 %
Accuracy of   car :  0 %
Accuracy of  bird :  0 %
Accuracy of   cat :  0 %
Accuracy of  deer :  0 %
Accuracy of   dog :  0 %
Accuracy of  frog :  0 %
Accuracy of horse :  0 %
Accuracy of  ship : 100 %
Accuracy of truck :  0 %
```

Figure 19: Accuracy of each class for 6th set of parameters

7. **Using RMSProp algorithm**

Optimizer: RMSProp
Criterion: Cross Entropy Loss
Alpha: 0.99 (Smoothing Constant)
Learning rate: 0.01

Test Accuracy: 49.9%
Train Accuracy: 46.66%

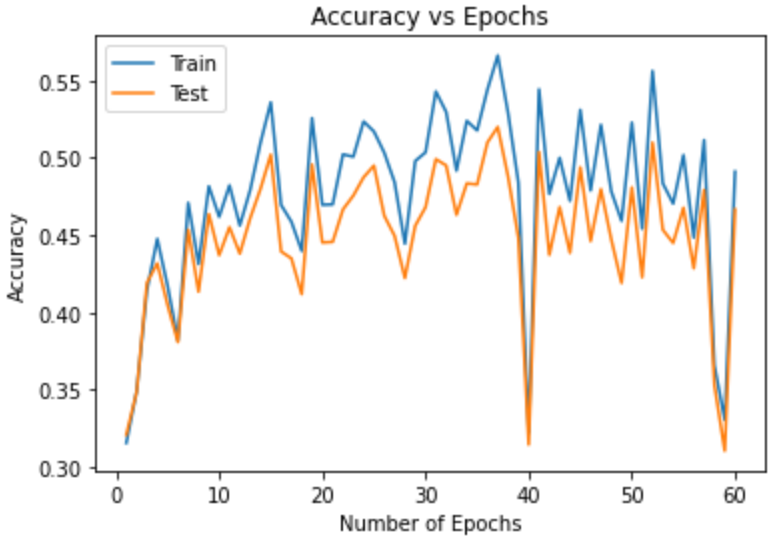

Figure 20: Accuracy vs number of epochs for 7th set of parameters

```
Accuracy of plane : 40 %
Accuracy of   car : 100 %
Accuracy of  bird :  0 %
Accuracy of   cat :  0 %
Accuracy of  deer : 50 %
Accuracy of   dog : 25 %
Accuracy of  frog : 100 %
Accuracy of horse : 33 %
Accuracy of  ship :  0 %
Accuracy of truck : 75 %
```

Figure 21: Accuracy of each class for 7th set of parameters

Discussion and comments on results:

I. The CNN architecture based on LeNet-5 is trained using different optimizers namely **Stochastic Gradient Descent, Adam and RMSProp**. Also the parameters like learning rate, epochs, weight_decay has been changed and the results are noted.

II. **Change in Optimizer:** Among all the optimizers SGD has achieved highest accuracy compared to others. Adam Optimizer gave accuracy almost similar to that of SGD. Adam optimizer is an extended implementation of SGD but with adaptive change in momentum. Among the 3 optimizers, RMSProp got least accuracy for both train and test data. RMPProp optimizers are mostly efficient for RNNs.

III. **Effect of Learning rate:** Decreasing the learning rate gives a smooth accuracy curve and also got maximum accuracy. Increasing the learning rate increased the training accuracy, while decreasing the accuracy on test data. This is because of overshooting problems.

IV. **Effect of Epochs:** We chose 60 epochs as standard value for each experiment. However, we increased the epochs to 120 for the parameters with maximum accuracy. The observation is that the accuracy becomes stagnated after a certain number of epochs and accuracy gets saturated.

V. **Effect of Weight decay:** This is the addition of regularization term as we discussed in previous section. The accuracy has gone down

to 10% because setting weight_decay to 1 means setting λ value to 1. This means while updating the weights, the final weight vector will be set to almost zero. So, if the model is overfitting the data then introducing a low weight_decay of 0.004 will help.

Best Parameter Setting to Achieve the Highest Accuracy:

The highest accuracy is achieved for the following parameters:

Optimizer: SGD
Criterion: Cross Entropy Loss
Epochs: 60
learning rate: 0.001
Momentum: 0.9

Test Accuracy: 63.53%
Train Accuracy: 81.63%

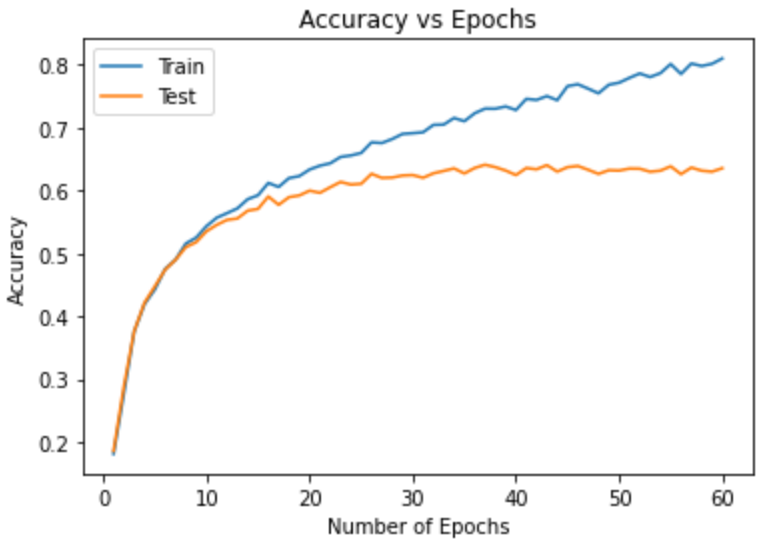

Figure 22: Highest accuracy result

C. State of the Art CIFAR-10 Classification:

Benjamin Graham in 2015 proposed a new pooling method called **Fractional Max Pooling** for the LeNet-5 architecture which improved the accuracy on CIFAR-10 dataset from 60% to **96%** [5]. The method is proved to be efficient on the CIFAR-100 dataset as well. In this section we discuss the implementation and result of this paper.

Main Idea: The author says that the max-pooling layer after the couple of convolution layers, although decreases the spatial size drastically, decreases the performance of CNN due to disjoint nature because of stride 2 which inturn reduces the generalization. In this paper, the author proposed fractional max-pooling operation where the spatial size of the input layer will be decreased in fractional numbers. This new pooling operation not only improves the performance but also reduces the problem of overfitting without even adding the drop out layers.

Author considers the drastic reduction in the spatial dimension as a loss of information. Instead of reducing the spatial dimension by 2, if we reduce it by $\sqrt{2}$,we can achieve the main purpose of pooline i.e., to reduce the spatial dimension and also it gives additional information in different scales. Also fractional max pooling introduces a certain degree of randomness. So, the reduction in spatial dimension will be $1 < \alpha < 2$.

The main gist of fractional pooling is finding the pooling region. This can be done in a random or pseudo random way. This results in pooling regions which may be disjoint or overlapping.

Let, $N_{in} * N_{in}$ be the spatial dimension of input volume and $N_{out} * N_{out}$ be the spatial dimension of output volume. The according the fractional max pooling operation, $\frac{N_{out}}{N_{in}} = \sqrt[n]{2}$ i.e, nth root of 2. This reduces the rate of reduction in the spatial size by n times. So, typically we choose $\frac{N_{out}}{N_{in}} \in (1, 2)$.

Figure 23 shows the Fractional pooling layer after convolution layers.

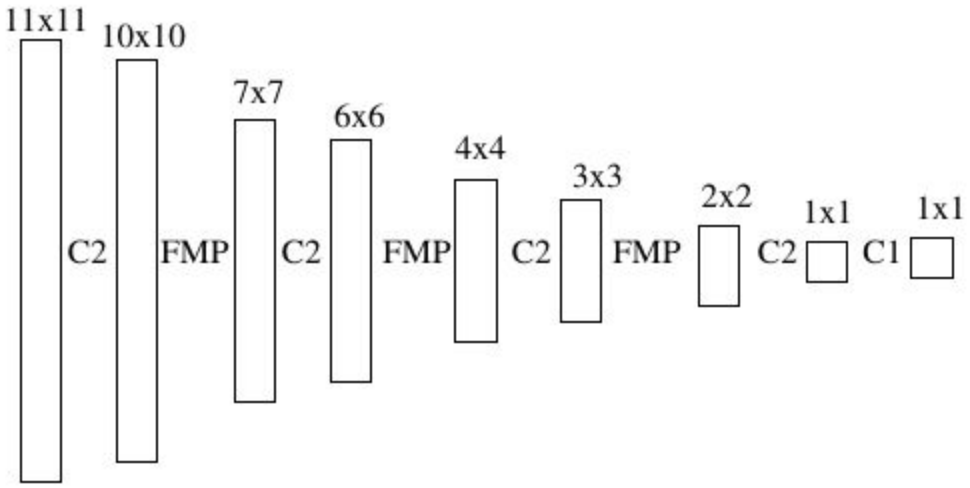

Figure 23: Fractional Max-Pooling layers

Now, the problem comes down to choosing a pooling regions (P_{ij}) given (N_{in}, N_{out}).

Generating Pooling Regions (Stochastic Process): Let us define two increasing sequences $(a_i)^{N_{out}}{}_{i=0}$ and $(b_i)^{N_{out}}{}_{i=0}$. The starting number of both sequences will be 1 and ending with N_{in}. The increments will be 1 or 2, i.e, $(a_{i+1} - a_i) \in \{1, 2\}$. Now the pooling regions can be defined as,

$$P_{i,j} = [a_{i-1}, a_i] * [b_{j-1}, b_j]$$

The integer sequences are generated using random or pseudo random variables. However, experiments proved that pseudo random variables yield better results.

Figure 24 shows the results of random pooling by generating pseudo random sequences for selecting pooling regions.

Figure 24: Results of fractional max pooling ($\sqrt{2}$) by generating random sequences. It shows the result of 5 layers of fractional max pooling.

The above figure shows that the result obtained using random sequences have elastic distortions as the number of pooling layers increases.

Model Averaging: In order to increase the accuracy, we perform fractional max pooling operation several times using different pseudo random sequences. The results are averaged to get the final accuracy.

Figure 25 shows the effect of model averaging on test data of MNIST data set.

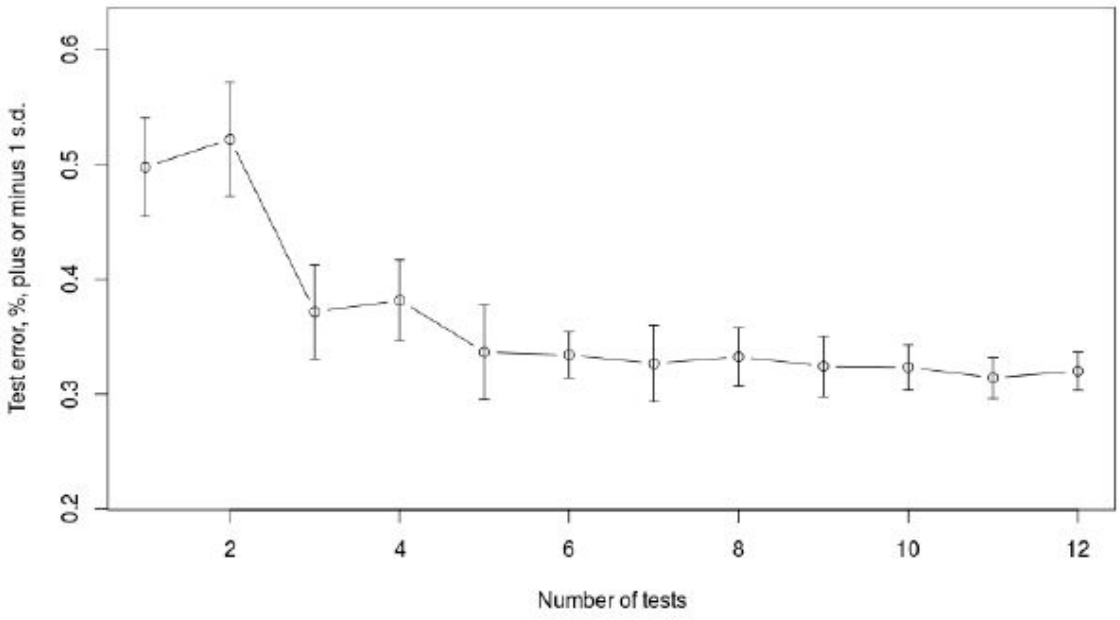

Figure 25: Error vs Number of experiments on MNIST data set.

Results on CIFAR-10:

While implemented on CIFAR-10 dataset, dropouts are used and also the training images are augmented using affine transformations like translation, rotation, shearing, reflection and stretching. For the final 10 epochs, the affine transformations are not added. The results obtained are tabulated below.

# of Experiments	**1 Test**	**12 - Tests**	**100-Tests**
Error	4.50%	3.67%	3.47%

This is the highest accuracy obtained on the CIFAR-10 dataset so far.

Comparison with LeNet -5:

(+) : Pros and (-): Cons

LeNet-5	Fractional-Max-Pooling
(+)The pooling layer is simple to implement, fast and the spatial dimension is reduced quickly	(-) The pooling layer is complicated and the spatial dimension is reduced in fractions and it requires more pooling layers which increases the training time.
(-) The rapid reduction in the spatial dimension hampers the performance sometimes.	(+) The random selection of pooling regions based on the random sequences generated will avoid the overfitting problem and increases the performance
(-) The max- pooling is usually done using spatial size 2 and stride 2 which makes the pooling disjoint. This often limits the generalization of test data. While the accuracy on train data was 82%, the accuracy on test data is around 60%.	(+) The degree of randomness induced based on the random selection of pooling regions and also the overlapping case will ensure better performance on unseen data. The accuracy on test data has increased to 96%.
(-) As the spatial dimension reduces rapidly, we lose some important features.	(+) As we scale down the spatial dimension in fractions we obtain more intermediate results which allow us to visualize the features and it's easy to interpret what's going on in the network.

References:

1. https://cs231n.github.io/convolutional-networks/.
2. https://www.tinymind.com/learn/terms/relu.
3. https://arxiv.org/abs/1412.6806
4. http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf
5. https://arxiv.org/pdf/1412.6071.pdf
6. LeCun, Yann, et al. "Gradient-based learning applied to document recognition." Proceedings of the IEEE 86.11 (1998): 2278-2324.