EE569: Introduction to Digital Image Processing

ASSIGNMENT #5

Sonali B Sreedhar

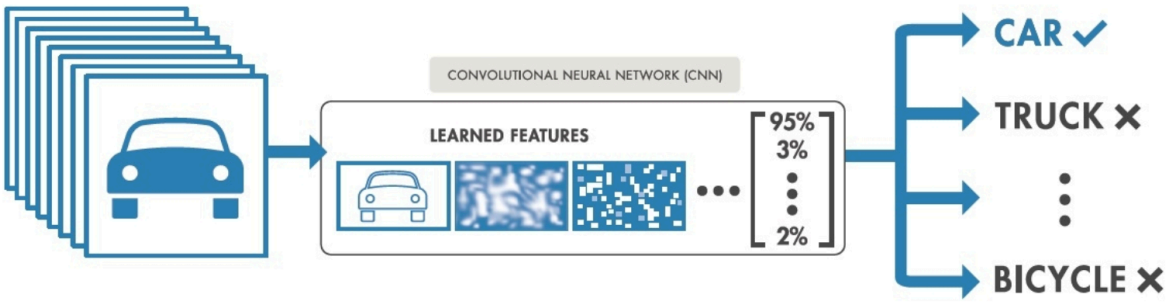
1783668369

sbsreedh@usc.edu

Problem 1: **CNN Training on LeNet-5**

Abstract and Motivation:

CNN is one of the most popular deep learning architectures being in use these days. The effectiveness of the convnets has given CNN its popularity. It is widely used for image classification, segmentation, object recognition and identification. By using various filter, it finds the pattern in an image and classifies almost accurately to its respective class that eliminates the manual efforts in feature extraction of images to be classified.
Here in this assignment we will learn more about CNN and experiment on training CNN model on CIFAR-10 image dataset to obtain good accuracy on classification.



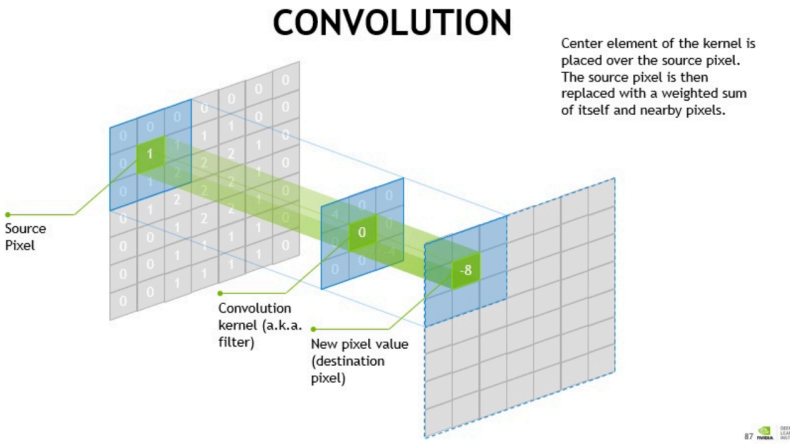
Theory:

In this section lets understand the about CNN and overview of it's architecture.
CNN can be viewed as neural network. The neurons are the basic units they are made up of. The neurons are the one that feed on inputs, through hidden layers and learns the distinct pattern of each image to classify the images.

The main four components in a CNN are:

1. Convolution layer
2. Max pooling layer
3. Activation function
4. Fully connected layer
5. Softmax function

1. Convolution layer:
   This layer is the main layer that is found as the first layer in every architecture. The various filters are used on this first layer to identify different pattern sin an image. A mask is created to detect a certain pattern in the image and is scanned across the image to obtain the convolution product of the pattern to be detected and the masked image. In this way the images fed as input to this layer are calculated for the convolution product with all the filters defined. These filters are designed as such to extract certain features of the images. Thus, a feature map is obtained that tells us about certain pattern or feature present in the given input image.



Convolutional layer (source: https://www.embedded-vision.com)

Figure 1: Convolution Layer

Here, the features are self learnt by the models and are not defined before hand. These filter kernels can be referred as weights of each layer. These weights are initialized near to zero but not zero and is updated by backpropagation using certain methods discussed in the following section.
The hyperparameters involved in the convolution layers are padding, stride and number of filters. Let us discuss how these parameters are chosen and affect the size of the output image.

- The number of filters in a convolution layer directly relates to the depth of the convolution layer. As mentioned earlier each filter correspond to a particular feature of an image to be extracted. The stacked-up filters giving the activation map can be said to be equal to the depth of the output.
- The padding of an image is done in order to perform the convolution at the pixels present in the boundary. By doing so, the size of the input image is preserved. This padding can be done either by doing a zero padding or reflection of the pixels across the boundary. Zero padding does not change the output image size as it will be retained to be of same size as the input. So, here we can say that the size of this padding can be considered as one of the hyperparameters.
- The parameter stride indicates how the image must be scanned for convolution purpose. If we want the mask to move one pixel by one then stride is set to 1 and if it is set to 2, for every two pixels, the mask moves and calculates the convolution product. Basically, higher the stride parameter, the mask is moved over larger number of pixels and smaller output volume is obtained.

Below is the formula to calculate the output volume dimensions:
Considering W to be size of the input volume, S to be the stride parameter, N to be the size of filters, P be the padding amount. The output spatial size could be calculated as (W-N+2P)/S+1.

2. POOLING LAYER:

Pooling layer is usually used in between two convolution layers. Basically, it tries to reduce the size of the input images without altering any of its features or characteristics of each image.
It basically reduces the spatial size of the input. This helps in reducing over fitting and hence increases the efficiency.
Using MAX operation, every input is operated upon independently by the pooling layer. Here a mask is defined and is scanned across with a stride value of 2. The maximum value obtained replaces the pixel values under the mask, thus reducing the spatial size of the input.
To summarize the pooling layer takes an input as a dimension of W1xH1XD1.
The hyper parameters used are:

- F, the spatial extent
- S, the stride parameter

The output volume size would be W2xH2XD2, here,

- W2= (W1−F)/S+1W2
- H2=(H1-F)/S+1
- D1=D2

Zero padding is commonly not used here.

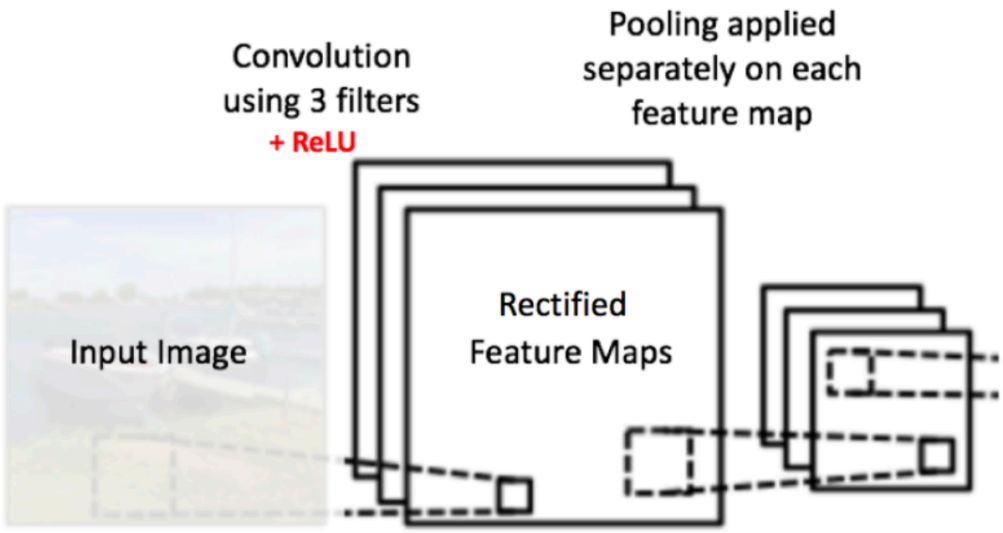


Figure 2: Pooling Layer

Activation functions:

Activation functions are used to get an input node of a neural network mapped into an output signal. It brings out the relationship between the input and the output response. The non-linear properties are introduced to the network by them. Without an activation function , neural network would be just functioning as a linear regression model. The model would be simpler as it would be performing on the transformation that is linear on weights and the images given as input. But definitely would not be a stronger model failing to learn the complex features from the given input dataset. Hence an activation function introduces the non-linearities into a model thus enhancing the learning of the model towards complex datasets.

There are various types of activation functions used in deep learning. Let us discuss a few of them in the following section:

1. **Sigmoid Function:**
   This is one of the widely used activation function among all. This transforms input to output space between 0 and 1. We have to notice that a sigmoid function is a non-linear function and hence the transformed output is also expected to be a non-linear one.

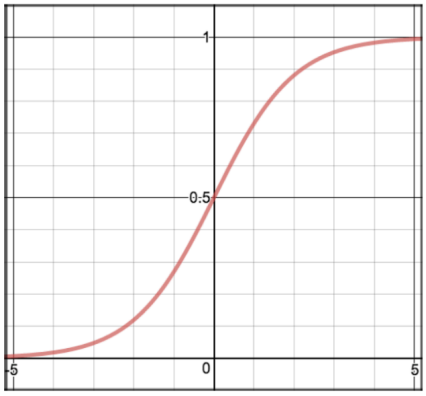


Figure 3: Sigmoid Activation function

2. **Tanh Function:**
   This can be said to be similar to sigmoid function except that it is kind of symmetrical around the origin. The transformed output space is between 1 and -1. Otherwise, all the properties of tanh and sigmoid match with each other. The tanh function is also differentiable and continuous at all points as in sigmoid function.
   Since tanh activation function is zero centered and has no restrictions on gradient with respect to the direction, tanh is preferred over sigmoid activation function.

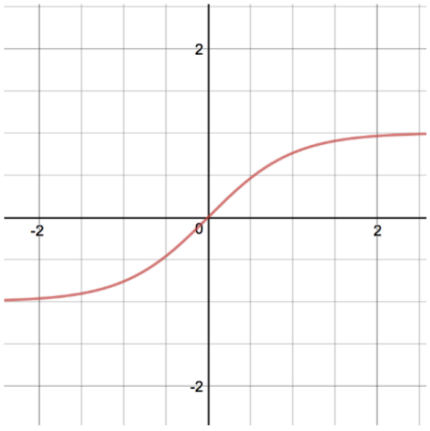


Figure 4: Tanh activation function

3. **ReLu Function:**
   ReLu activation function is one of the widely used one in deep learning domain. Unlike other activation functions, all the neurons here are not activated simultaneously.

This implied that any neuron would be de activated if it is assigned 0 at the output. Here the result might be set to 0 if the values of the input are less than 0 but that does not deactivate the neuron. Here, we have to observe the increase in the efficiency of the model upon using ReLu function when compared to sigmoid and tanh function.

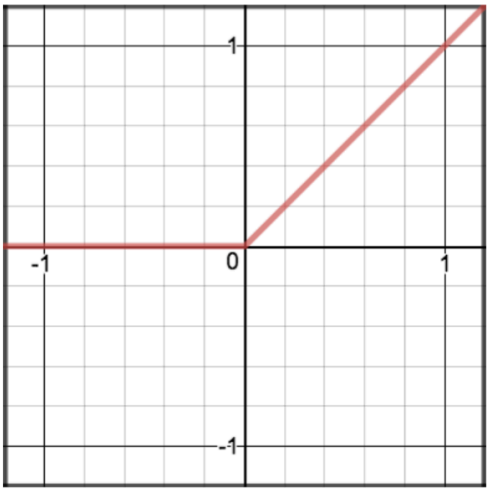


Figure 5: ReLu activation function

4. **Leaky ReLu:**

To mitigate the drawbacks of the ReLu activation function, we use Leaky ReLu function where a small linear component is defined, instead of setting 0 for the negative values. So we can see a non-zero gradient here, hence can avoid the deactivation of neurons in the region of negative range.

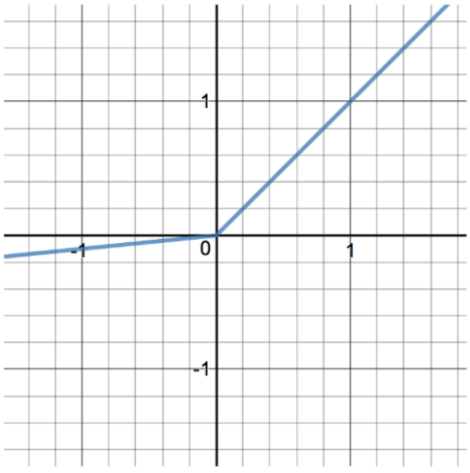


Figure 6: Leaky ReLu activation function

5. **Fully connected layer:**
   At this stage, we are able to identify all the features at a higher level. The fully connected layer is referred to be one of the traditional Multi-layer perceptron. We usually see a softmax function being used in the output layer.
   The name itself here self explains that all the neurons in the previous layer is connected to the ones on the layer next to it. Once we obtain the high level features from the previous layer ,to mention: the convolution layer and Relu layer, are used for classifying the test images to their respective classes according to the model trained on training set. This layer also proves to be a cheap way to learn the non- linearities in a model.Though the convolution and ReLu layer bring out the non-linearities in the input images, it is better to have a combination of those non linearities together which will yield better results.
   Here the output layer gives out probabilities of the dataset belonging to certain classes, sum up to 1 as we use Softmax function as the activation function in the output layer.

OVERFITTING AND TECHNIQUES TO REDUCE IT IN CNN:

With the help of a plethora of parameters, we are able to deal with the complex datasets and obtain good accuracies which would be a difficult task without these traditional machine learning models in hand. But also, these parameters themselves sometimes prove to be our weakness. This may either lead our model to undergo overfitting, where the model is very closely fitted to the training set and fails to generalize to make accurate predictions on the test data or it may also lead to underfitting, where the model performs bad and is less flexible with the new data.

To avoid overfitting we can indulge in some the following steps. They might not permanently help to eradicate them but will definitely reduce the model suffering from overfitting or underfitting.

1. **More data can be added:**
   More data means the model gets more variations of the circumstances or dataset it is trained for. This will help the model to generalize more and helps to be more flexible towards the new test data.
   But considering the cases where we might be have limited access to the dataset or collecting more data would be an expensive idea, we can move to our next method called data augmentation.
2. **Data Augmentation:**
   Data augmentation helps to increase the number of samples in a training set by introducing variations for a given image itself by doing some geometrical modifications. This modification may include scaling of the images, rotation of the images, zooming in and out of the images, blurring or adding filters to the input images.
   We should ensure that this is done only on the training set and not on the validation or the test set.

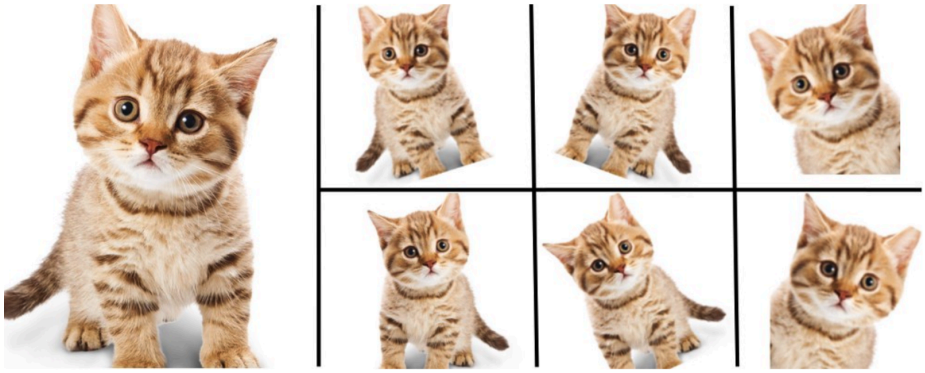
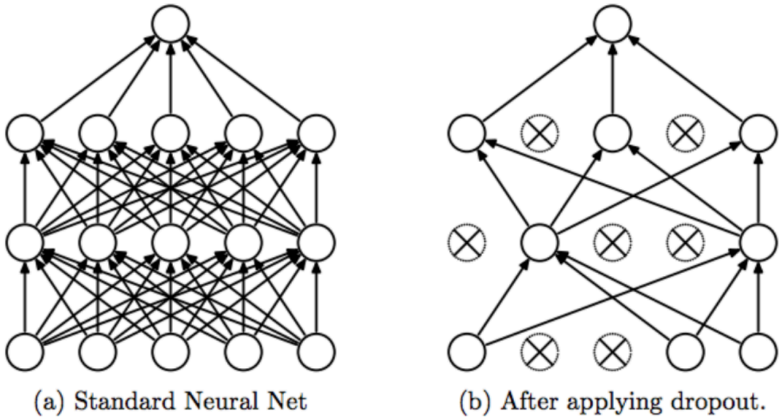


Figure 7: Data augmentation example

3. Architectures that generalize well can be used:
   There is no guarantee that we could overcome the overfitting or underfitting using a particular architecture. It mainly depends on the types of parameters such as initialization, loss, optimizers being used and the complexity of the dataset. So there is no harm in trying out various architectures which may well generalize to get better accuracy on both validation as well as test set.

4. Regularization:
   There are mainly three types of regularizations used namely:
   1. Dropout
   2. L1 Regularization
   3. L2 Regularization

   a) Dropouts: It is one of the techniques which randomly selects neurons and ignores them while performing training of a model. This implies that they would not be participating in contributing to activating downstream neurons in the next layers, and also if there are any updates in the weights while back propagation, these dropped out neurons are not considered as well.



Visualization of dropout

Figure 8: Dropout

b) **L1 Regularization**: This is referred as LASSO regularization. Here the model is penalized by adding an absolute value of magnitude coefficient to the loss function.

$$\sum_{i=1}^{n}(Y_i - \sum_{j=1}^{p} X_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} |\beta_j|$$

c) **L2 Regularization**: It is also referred as Ridge regularization. Here the model is penalized by adding a squared magnitude of coefficients to the loss function.

$$\sum_{i=1}^{n}(y_i - \sum_{j=1}^{p} x_{ij}\beta_j)^2 + \lambda \sum_{j=1}^{p} \beta_j^2$$

4. Reduction in the complexity of the architecture**:** This method could also be used to reduce the overfitting of neural networks.

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

In the present days, we see a lot of real time projects are being dealt using Deep Nets. In that too, especially CNN is being widely used in all the computer vision projects, thus replacing the traditional methods of solving it.
Deep Nets learns about a particular dataset across layers of a neural network, more efficiently and accurately. All this is being carried out without supervision, that's the main advantages of using Deep Nets and hence, used by many of the engineers and scientists.
Needless to say, the raising improvements in the capabilities of device in-terms of power consumption, sensor resolution, and computing power has elevated the advantages of using Deep Nets in more efficient and quickened procedure in comparison to CV methods.

Few of the reasons why Deep Nets out rate traditional methods are discussed below:

- As mentioned before using Deep Nets enable engineers and scientists achieve greater accuracy. This includes the domain of image classification or segmentation, object detection and many more. This is mainly because, the Deep Nets are not programmed but are trained. The applications that are supposed to be used for this does not need an expert hand.
- CNN is more flexible in comparison of the traditional methods as we can re-train the models with customized datasets for a specific use case. But where as in CV, the problem needs to domain specific and hence does not offer the flexibility.
- Considering the task of image processing, below are few differences we find while using traditional methods and while using CNN.

a. A well established techniques in computer vision are used to extract the features namely SIFT, SURF etc,. So for any image classification or detection task, we need to carry a step where we extract the features. These features extracted act as information descripting the characteristics of particular images. These extracted features form a bag of words for particular images and help in classifying them into their respective classes.
b. Here, in the traditional methods, the difficulty lies in selecting the right features that are important for that particular image. Also, we need to notice that as the number of classes increases, the task of extracting the features becomes more tedious and cumbersome. This task is more of a manual task which also needs the engineer to choose the best parameters for feature extraction and should be fine tuned .
c. CNN offers a machine end-to-end learning, as here only the customized image dataset is provided, with annotation of the classes they belong to. The CNN model itself discovers the patterns underlying and learns the linear and non-linearities in the image dataset provided.
d. From the above we can easily see how well established our Deep Nets is and how well they perform out rating the traditional algorithms. CV engineer's workflow has changed in employing these methodologies, as the manual job of feature extracting is being replaced by expertise of deep learning iterations through its sophisticate architectures.

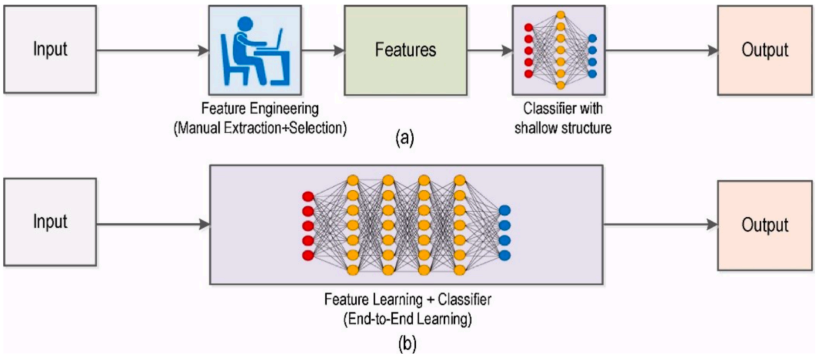


Figure 9: Traditional CV workflow vs DL workflow

Explain the loss function and the classical backpropagation (BP) optimization procedure to train such a convolutional neural network.

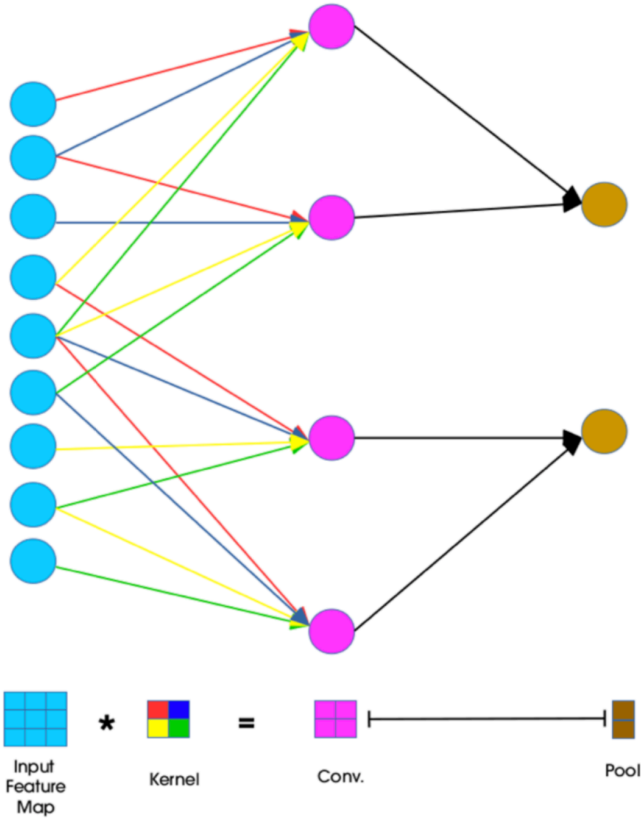
Loss Function:

The inconsistency between the label predicted on the test set and the label annotated in the training dataset is measured using loss function. Loss function is one of the important parameter to be considered while training a model. It can be used in CNN as softmax. It is given by:

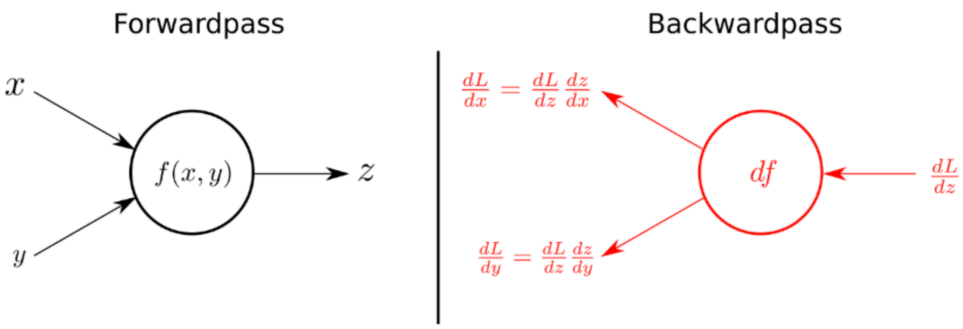
$$L_i = \log\left(\frac{e^{sk}}{\sum_j e^{sj}}\right)$$

Here the weights are updated in each epochs so as that the loss is reduced. Since, a minimum loss ensures high accuracy, we should always try to reduce the loss as much as possible. In order do this, we have many optimization techniques that is used to reduce the loss namely, batch optimization, SGD optimization, adaptive moment estimation.

CNN as we already know is a inspiration derived from Multilayer Perceptron. Unlike in Multilayer Perceptron all the neurons are assigned with separate weights, here the weights are shared in CNN. This helps in reducing the complexity while training a model.
In the earlier section we saw that, on completion of every epoch we see that the weights get updated. This is done by Backpropagation.
In every epoch, the algorithm performs both backward and forward weight propagation in order to optimize the weights assigned to each neurons in CNN. As we have seen in earlier section, we scan the image in convolution step where the kernel is flipped and according to the input for number of strides the input feature map is scanned for.
In the below image we can see the forward propagation is inputted with the weight values which is used while performing convolution.



The filters that are used in the process of convolution is flipped. It will be considered as cross-correlation if the filters are flipped.
In back propagation, we find the output's gradients. Back propagation can be viewed as forward propagation in reverse order ie, from output to input.



∂h_{ij} represents $\frac{\partial L}{\partial h_{ij}}$

∂w_{ij} represents $\frac{\partial L}{\partial w_{ij}}$

$$\partial W_{11} = X_{11}\partial h_{11} + X_{12}\partial h_{12} + X_{21}\partial h_{21} + X_{22}\partial h_{22}$$

$$\partial W_{12} = X_{12}\partial h_{11} + X_{13}\partial h_{12} + X_{22}\partial h_{21} + X_{23}\partial h_{22}$$

$$\partial W_{21} = X_{21}\partial h_{11} + X_{22}\partial h_{12} + X_{31}\partial h_{21} + X_{32}\partial h_{22}$$

$$\partial W_{22} = X_{22}\partial h_{11} + X_{23}\partial h_{12} + X_{32}\partial h_{21} + X_{33}\partial h_{22}$$

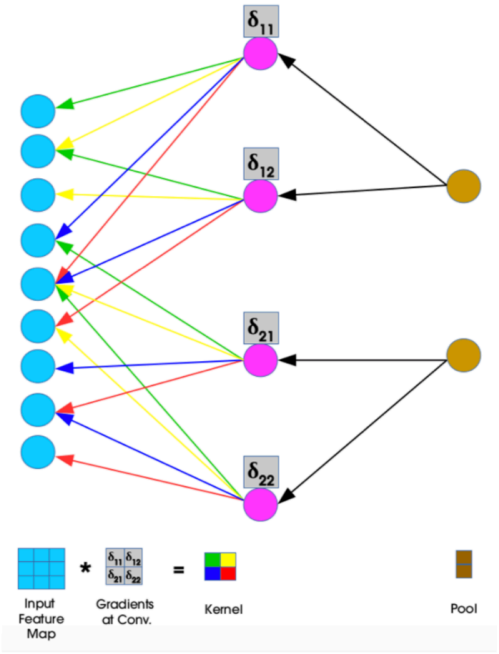


Figure 10: Back propagation

In CNN, the neurons sharing the weights is a very important strategy. This reduces the parameters that is to be learnt by the model. We need to also notice that the lies difference in the derivation taken during forward and backward propagations. This depends on the layer where the

propagation is happening. Thus, the weight matrix is flipped as kernel in forward propagation and for kernels in backward propagation we use the gradient matrices.

Problem 1b:
Abstract and Motivation:

LeNet-5 architecture is one of the few CNN architectures that has been used in a large scale to classify images to their respective classes. CNN forms the foundation for many of the state-of-art deep learning projects.
In this assignment, we are training a CNN model with LeNet -5 architecture on Cifar-10 dataset.

Approach and Procedure:

LeNet-5 architecture is very similar to the convolutional neural networks. This consists of

- Convolutional layers
- Max-pooling layers
- Fully connected layer

The output is obtained from the last layer ie, fully connected layer. This will be the layer that is responsible for the classification of all the 10 classes of the images.

- The input images are of the size 32x32, RGB images.
- Through the convolutional layers , these input images are passed.
- The first convolution layer has 6 filters and the second one will have 16 filters with no padding, and stride is 1.
- Each convolution layer is followed by a max pooling layer with filters sizes of 5x5 , stride as 2 and no padding.
- From the max pooling layer, the outputs are sent to fully connected layer having 120 and 80 neurons respectively.
- The activation function used in all the layers is ReLu except for the last layer where softmax function is used.
- The final layer will have 10 neurons. This computes the probabilities of each of the images belonging to the 10 classes.

Algorithm and implementation:

1. The Cifar -10 dataset is downloaded from the keras dataset.
2. Import all the necessary libraries such as Sequential, Dense, Maxpooling, Conv2D and Dropout.
3. Initialize the parameters such as epochs, learning rate, batch size, etc.
4. Add the convolution layer followed by max pooling with the above mentioned parameters.
5. Ensure to use proper loss function and activation function suitable for this dataset.

6. So for each epochs here, we take
   Each batch for training
   - Perform a feed forward operation on it.

- Loss is calculated.
- Then we update the weights by performing back propagation.
    - Loss and accuracy on the training and testing dataset is calculated after each epoch.

7. Plot the accuracy and loss observed on the training and testing dataset.
8. At the end, evaluate the performance of the model by predicting on the test dataset.
9. Calculate the test accuracy and loss observed.

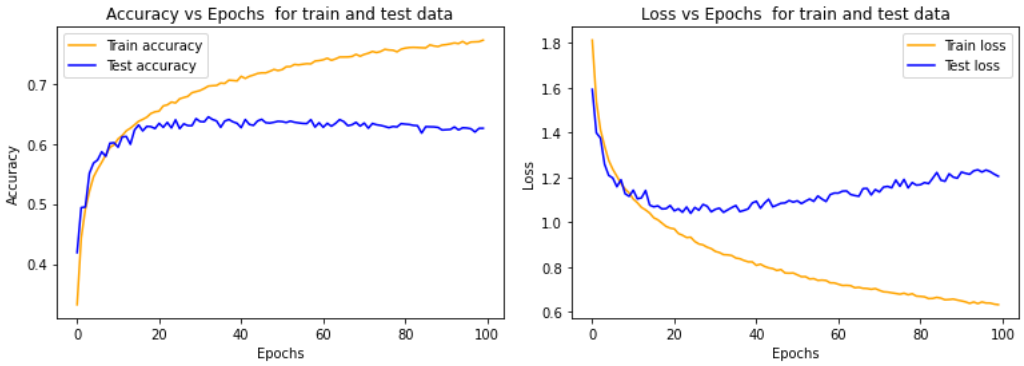
Experimental results and Discussion:

Here we have trained the LeNet model on Cifar-10 dataset. A variety of parameter settings has been tried out. The results for various permutation and combination of these parameter settings has been carried out and the results have been observed for the same.

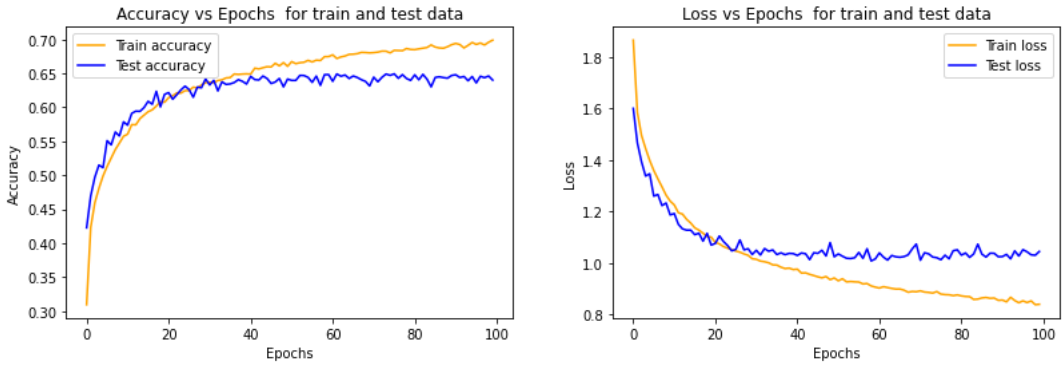
- The parameter batch_size was kept at 128.
- I tried training the model first with a epoch as 50. But the accuracy was observed to never go above than 47-50%.
- Then I tried with epoch set as 100, where I observed the accuracy hitting 60+ accuracy.
- Moving forward, I fixed epoch as 100 and varied other parameter.
- For kernel initializer I used two setting, that is glorot_normal and glorot_uniform namely.
- Also I tried changing the bias_initializer to RandomUniform from the default settings of default setting.
- For learning rate I used two settings namely 0.001 and 0.01.
- For Dropout used after Dense layer I used two settings as well, namely, 0.25 and 0.5.
- For all the above mentioned parameters, I tried different optimizers namely, adam, SGD, and RMSprop.

  The table below gives the observations done on various parameter setting on kernel initializer, bias initializer, optimizer, Dropout and learning rate.

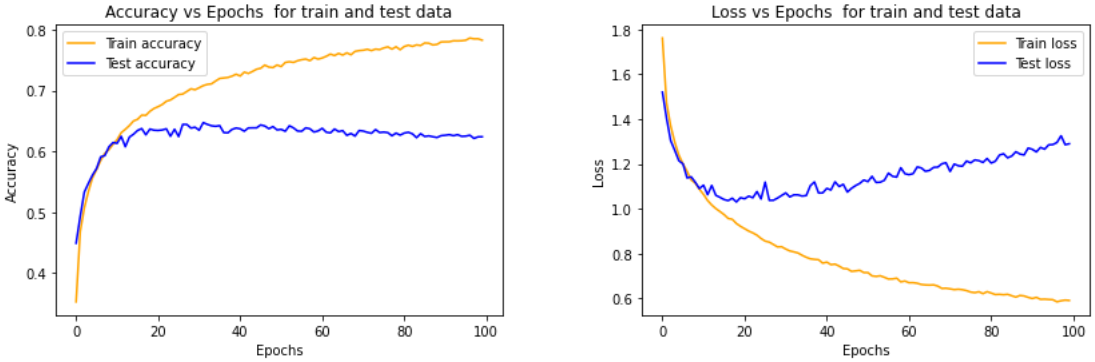
Case	Optimizer	Learning rate	Kernel initialiser	Dropout	train_loss	train_accu	test_loss	test_accu
1	ADAM	0.001	Glorot_normal	0.25	0.9006246328	0.6862800121	1.537936568	0.5109999776
2				0.5	0.6597203016	0.7779600024	1.080383658	0.621900022
3			Glorot_uniform	0.25	0.4156943858	0.8620600104	1.27575779	0.6148999929
4				0.5	0.589414537	0.8027399778	1.023481727	0.6463000178
5		0.01	Glorot_normal	0.25	1.178703547	0.5857999921	1.497235537	0.4911000133
6				0.5	2.303011417	0.1000000015	2.303002357	0.1000000015
7			Glorot_uniform	0.25	0.898647666	0.898647666	1.39475131	0.5770000219
8				0.5	2.303419352	0.1000000015	2.303436279	0.1000000015
9	SGD	0.001	Glorot_normal	0.25	0.8375932755	0.70976	1.018990649	0.6472
10				0.5	1.006748048	0.64432	1.116533163	0.6034
11			Glorot_uniform	0.25	0.8713513356	0.69536	1.046063524	0.6374
12				0.5	1.046100824	0.6307	1.140220582	0.5961
13		0.01	Glorot_normal	0.5	0.6640006536	0.76592	1.08065037	0.6306
14				0.25	0.3850344819	0.87292	1.206063737	0.6421
15			Glorot_uniform	0.25	0.4098230678	0.86394	1.231377974	0.619
16				0.5	0.7352031279	0.75368	1.094865777	0.6169
17	RMSprop	0.001	Glorot_normal	0.25	0.6163	0.7807	1.3401	0.6222
18				0.5	0.857	0.6943	1.11836	0.6234
19			Glorot_uniform	0.25	0.6074	0.782	1.3285	0.615
20				0.5	0.8829	0.6857	1.100379	0.6293
21		0.01	Glorot_normal	0.25	1.7474	0.3841	1.70775	0.3869
22				0.5	1.8657	0.3559	2.130554	0.2223
23			Glorot_uniform	0.25	2.0648	0.397	1.811547	0.3999
24				0.5	1.8639	0.323	1.878065	0.2974



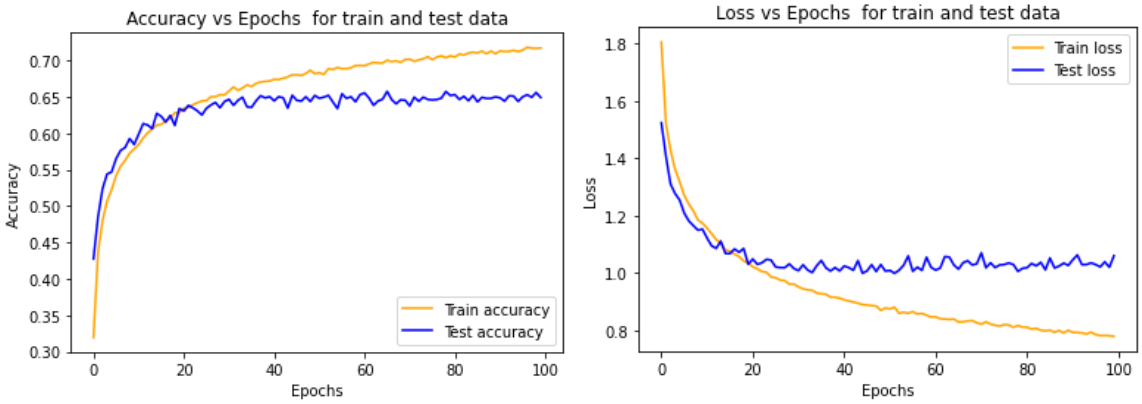
Accuracy and Loss plots each epochs for Case 1



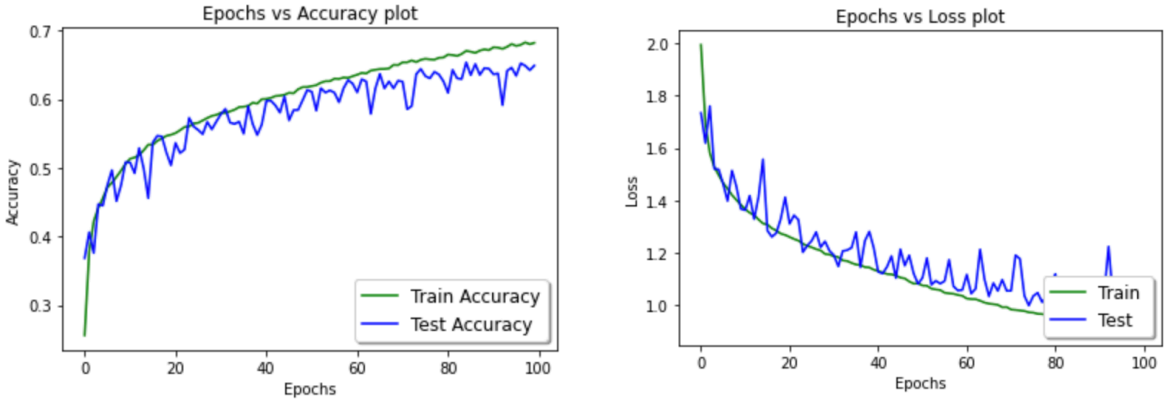
Accuracy and Loss plots each epochs for Case 2



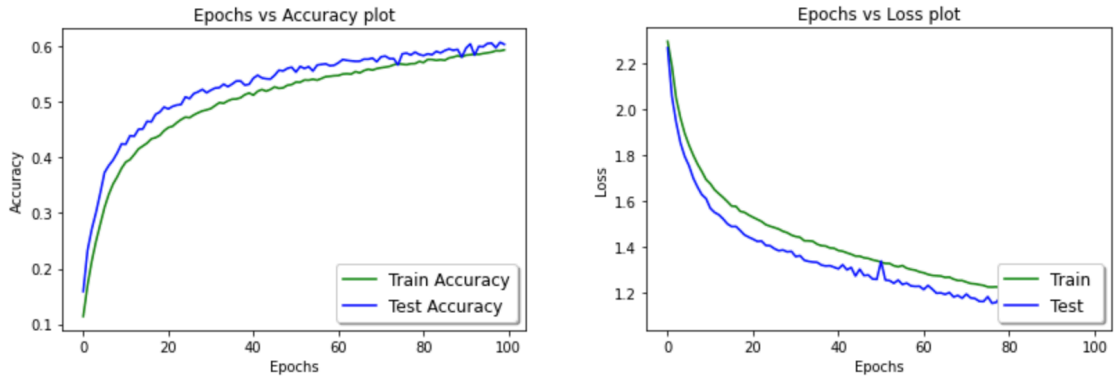
Accuracy and Loss plots each epochs for Case 3



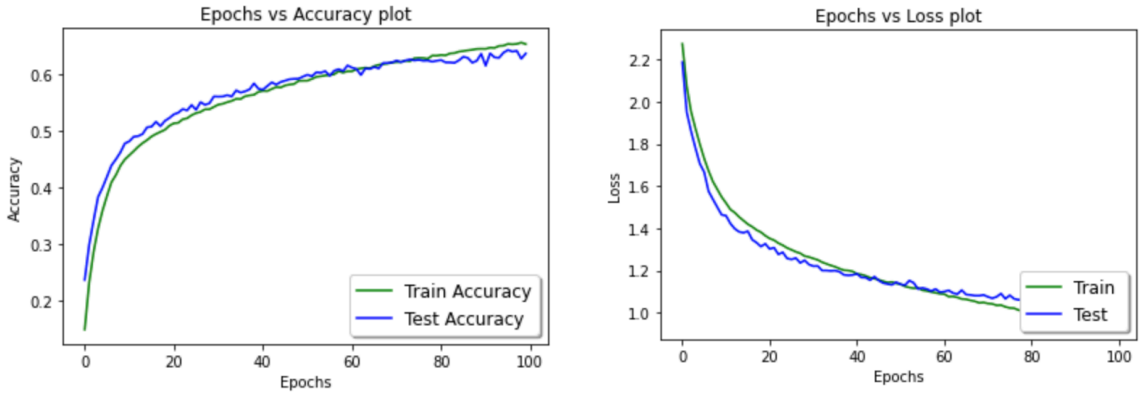
Accuracy and Loss plots each epochs for Case 4



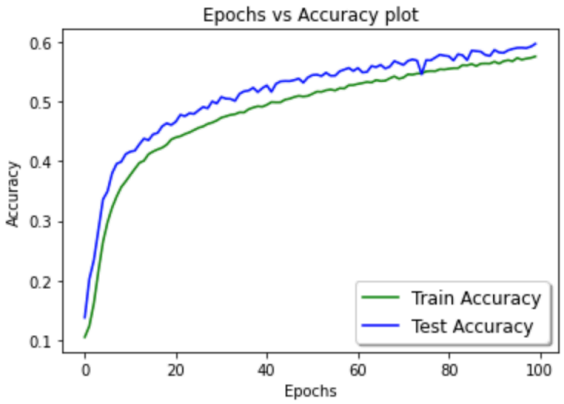
Accuracy and Loss plots each epochs for Case 9

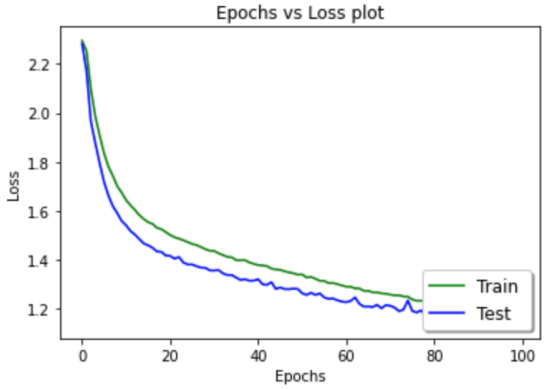


Accuracy and Loss plots each epochs for Case 10

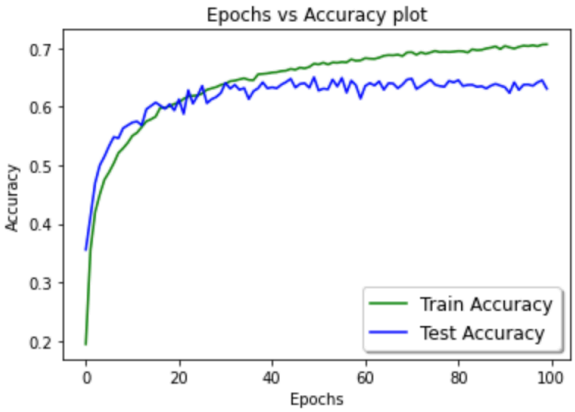


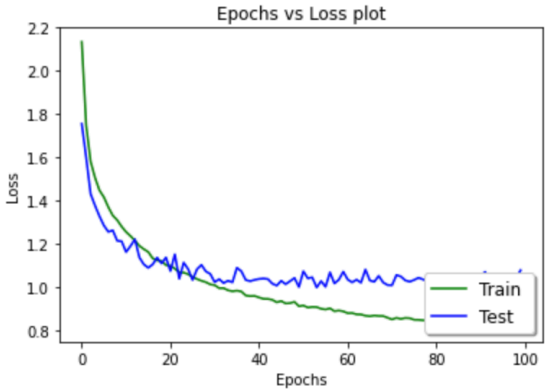
Accuracy and Loss plots each epochs for Case 11



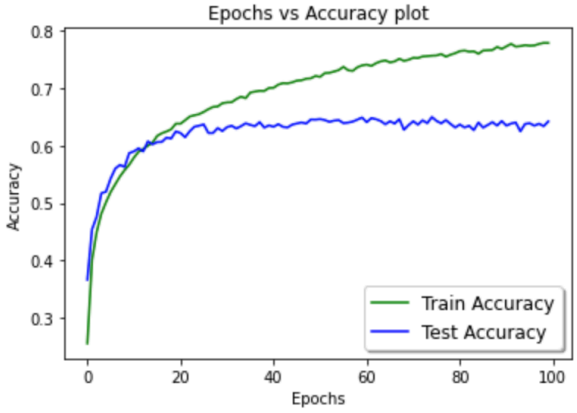


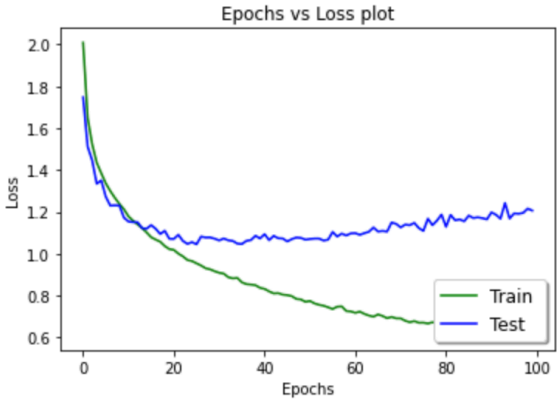
Accuracy and Loss plots each epochs for Case 12



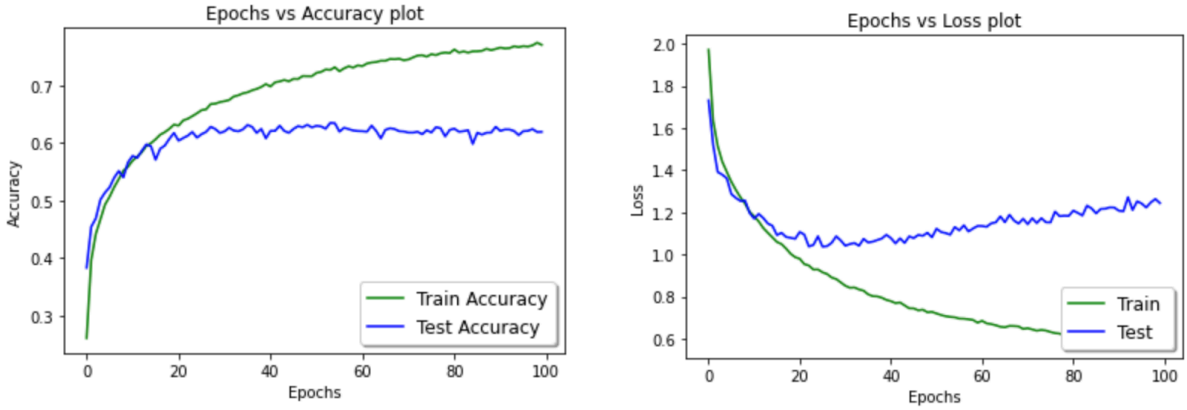


Accuracy and Loss plots each epochs for Case 13

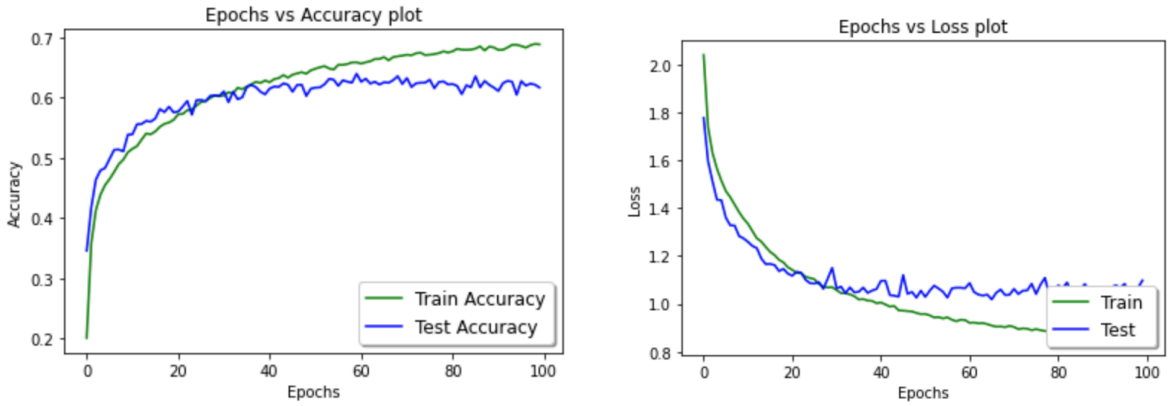




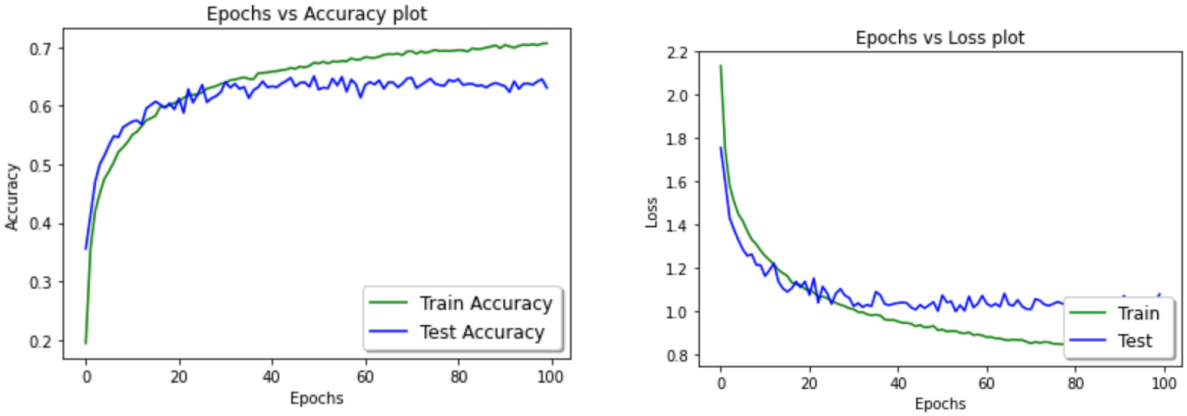
Accuracy and Loss for each epochs for Case 14



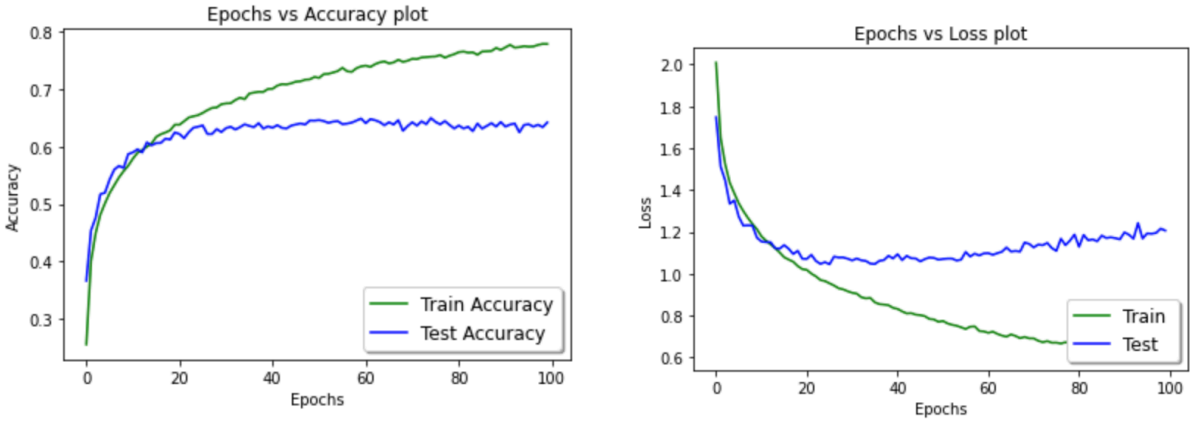
Accuracy and Loss for each epochs for Case 15



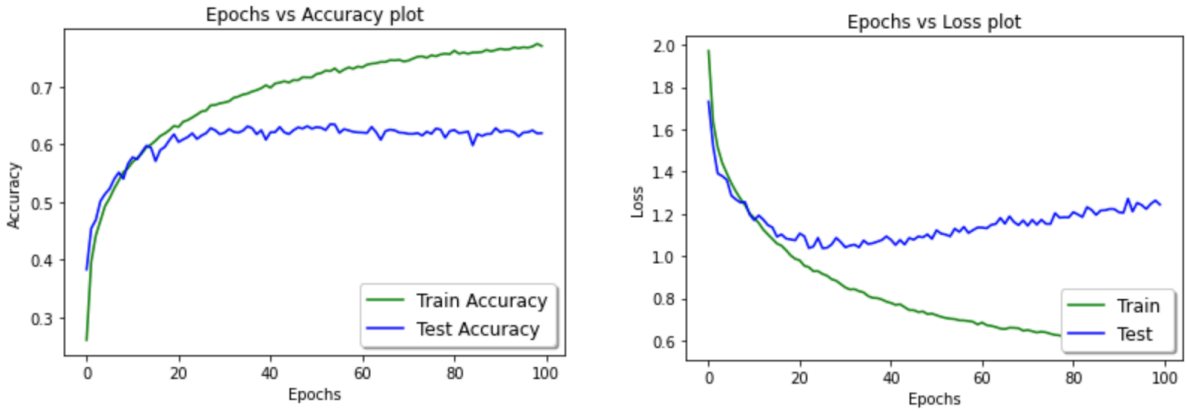
Accuracy and Loss for each epochs for Case 16



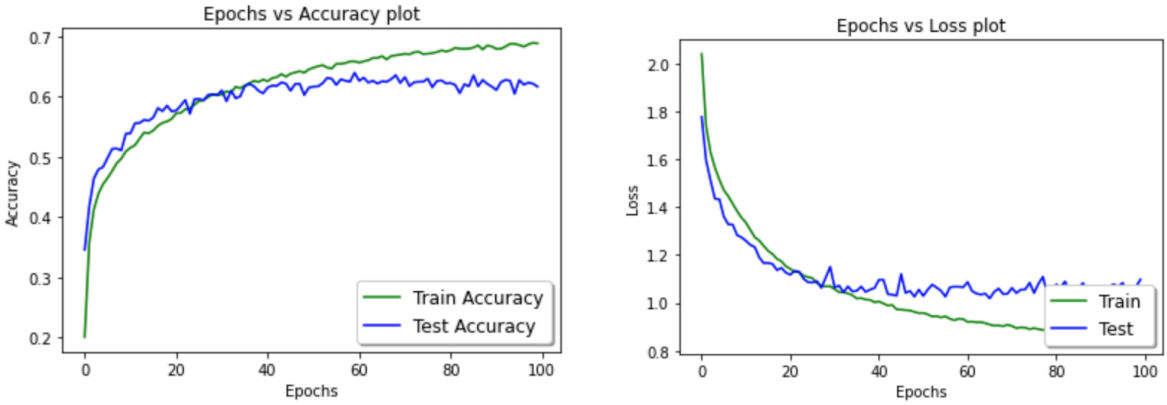
Accuracy and Loss plots each epochs for Case 17



Accuracy and Loss for each epochs for Case 18



Accuracy and Loss for each epochs for Case 19



Accuracy and Loss for each epochs for Case 20

Case 1: Learning rate

- A learning rate of 0.01 and 0.001 was used to train the model.
- The weights get updated in each iteration in small increments when the learning rate is smaller. This ensures that it takes more time for the gradient descent to converge.
- From the above observations we can see that the smaller the learning rate higher the accuracy obtained on the model trained.
- Hence we can conclude a learning rate of 0.01 is the best parameter to be used for this dataset while using SGD optimizer.

Case 2: Batch size

- A batch size of 32, 128 was tried.
- It can be observed that with the increase in the batch size , we observe an increase in the accuracy of the model.
- With a smaller batch size, the weights have to be updated very frequently, which makes the gradient optimizer to take less time to converge and thus does not converge properly.
- Thus, I chose 128 to be an optimum number for the batch size for this particular dataset.

Case 3: Optimizer

- The model was trained using three different types of optimizers namely adam, SGD and RMSprop.
- SGD performs better than ADAM optimizer.
- Thus, we can conclude that SGD is the optimum choice here.

Case 4: Dropout

- Dropout helps in reducing overfitting
- It helps in removing some of the neurons at each update. Then onwards those neurons are not considered for further update in the weights. Thus, helps in reducing overfitting.

Case 5: Bias initializer

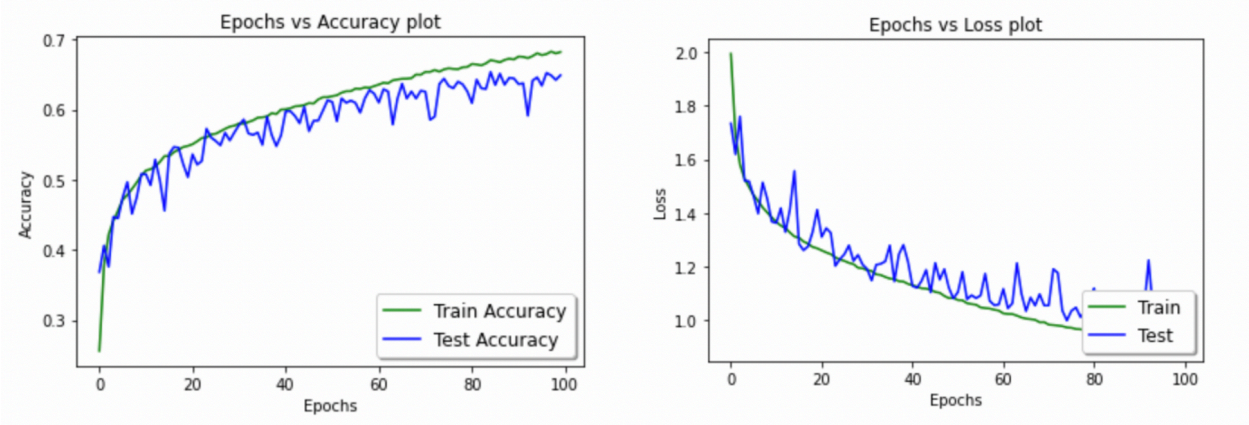
- The default bias initializer will be set to zeros.
- It is always recommended to initialize the weights near to zero but not zeros.
- After the bias initializer set to Random uniform, the accuracy on the training set and test set was observed to increase.

Case	Optimizer	Learning rate	Kernel Initializer	Dropout	Train loss	Train accuracy	Test loss	Test accuracy
1	SGD	0.001	Glorot_normal	0.25	0.76	0.72	0.972	0.6579
2	SGD	0.001	Glorot_uniform	0.25	0.893	0.682	1.080	0.6244
3	SGD	0.01	Glorot_normal	0.5	0.753	0.731	1.04	0.643

The best parameter setting for best accuracy obtained is as follows:

- Optimizer: SGD
- Learning rate:0.01
- Kernel Initializer: glorot_uniform
- Bias Initializer: Random uniform
- Epochs:100
- Batch size:128

Plot of the training and test dataset with best accuracy obtained:



Accuracy and loss plot for optimum parameters

With above configurations I have obtained training accuracy as 72% and test accuracy as **65.79%.**

Problem 1c.
Describe what the authors did to achieve such a result.

The main idea projected in this paper by the author is that:

1. A max pooling layers in CNN could be replaced by a convolutional layer that works good for the same scenario. This might be done without degrading the performance of the Cnn model. Infact, this might increase the accuracy obtained in the previous experimental set up.
2. A De-convolutional layer is much easier in an all CNN model. This gives a better visualization of the weights as proposed by Zeiler and Furgus in their paper[REFERECE]

Here the author that, the simplest model of CNN which just consists of convolution layer with a stride parameter of 2 that is trained using SGD with moment 0.9 gives an state of art performance. This is done without making the architecture too complicated by adding many layers and Dropouts.

The experimental setup set by the author is tabulated below. He has experimented on three different base models.

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

Points to be noted from Table 1:

1. The depth and the number of parameter is observed to increasing gradually form Model A to model C.
2. Here a 1 by 1 convolution layer is considered in the base networks at the top in order to produce 10 outputs whose probabilities of them belonging to a particular classes is computed by using softmax function.
3. Experiments were done with fully connected layers instead of 1 by 1 convolution layer. But these yielded worst results.
4. Model B uses 1 by 1 convolution layer after each normal convolution layer.
5. Model C uses simple 3x3 convolution layers which replaces all 5x5 convolution layer. This ensures that the architecture consists only layers operating on 3x3 spatial neighborhoods.

Then the author later tried an additional three variants from the above base model C mentioned above. By doing so, he has tried to evaluate the importance of pooling for this particular dataset.

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

Above table describes the additional variants derived from model C.

Points to be observed in these are:

1. The max pooling layers were removed.
2. The stride parameter of the convolution layer that precedes the max pooling layer is reduced to. This was done to ensure the spatial region of the input layer is same as the next layer.
3. In All-CNN C model the max pooling layers are replaced by convolution layer.
4. In ConvPool-CNN-C model, the convolution layer -dense is placed before each max pooling layer. This was done to ensure the model will have same spatial region.

To summarize the main claims of the author is as follows:

1. In order to obtain a good accuracy rate, dimension reduction plays a very crucial role.
2. In convolution adds could be used to replace the pooling layer. One has to make sure of the inter -feature dependencies here. This can be viewed as learning rather than fixing it.
3. Reduce overfitting by using less parameters. 1x1 conv layers were used to replace fully connected layer.
4. For regularization, small conv layer could be used.

Compare the solution with LeNet-5 and discuss pros and cons of the two methods.

LeNet-5 Architecture:

- With LeNet-5 architecture the CNN model on Cifar-10 dataset we get the accuracy around 63-65% maximum.
- The loss is observed to be decreasing with the number of epochs but tends to increase again if it crosses certain limit.
- To extract the features learnable filters and pooling layers are used here.
- By using dropout with a various combination of convolution layer we achieve good accuracy on this dataset.

The below section discusses about the pros and cons of LeNet-5 architecture used to train CNN model on Cifar-10 dataset:

Pros:

1. Simple architecture, easy to understand and analyze.
2. Improved performance in the model is observed without performing feature engineering.
3. Restricted number of layers to be used give a base to compare the results while changing the other parameters like stride, epochs, batch size.

Cons:

1. With LeNet-5 architecture, we could hardly reach an accuracy of 65%.
2. We have to move on to incorporate other architecture to achieve state-of-art results.

CIFAR-10 classification error

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

The above table shows the comparison of the base model and the model derived from it.

Observation on the results:

- A good accuracy was obtained without data augmentation with a proper schedule of learning rate that was determined based on results obtained from base model.
- Dropout was used for regularization after each pooling layer
- Pooling is removed and convolution is used instead.
- To analyze and visualize the concepts learnt by the layers at higher level, Deconvolution method is used.
- In higher layers the sharp features cannot be visualized properly.

The below section discusses about the pros and cons of author's experimental setup used to train CNN model on Cifar-10 dataset:

Cons:

1. As mentioned by the author himself, this accuracy obtained with the customized architecture does not assure us of best accuracy on all other datasets.
2. The analysis seems apt only for a relatively a smaller dataset.
3. Pooling layers has been in the use since long and in many other papers we see that the state-of-art results were achieved without removing the pooling layer. This might be applicable only for this case.
4. Also, without pooling, there might be some introduction of rotational invariance missing. This might affect the performance of deeper CNN.

Pros:

1. Provides good accuracy by removing pooling layer.
2. Can achieve state-of-art results by just stacking convolution layers.
3. Smaller convolution helps in regularization.
4. As the conv layers replace the pooling layer, we get less number of parameters that the model needs to learn.
5. Dimensionality reduction plays an important role in increasing the accuracy.

References:

1. Class notes and Discussion