Name: Shreya Kate

Email: shreyak@usc.eu

USC ID: 2334973997

Homework 5

Problem 1:

(a) 1.

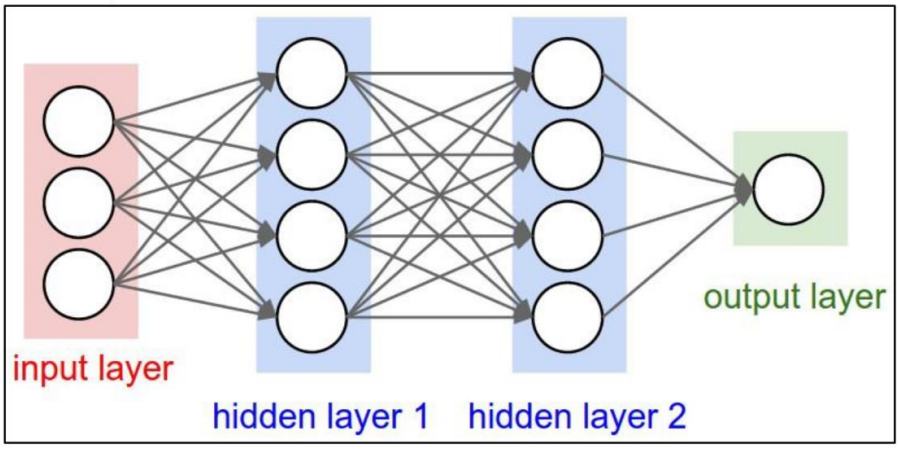
1) fully connected layer

We apply the convolution and max pooling to the original image. After that we apply the activation function to get the features of the image. After doing this, we get the fc layer. We get the fc layer by extracting the most important features from the set of features. The most important features are the ones that correlate the most to a class. We assign weights so that we get the correct probabilities for the different classes when we take the product of weights and features.

There are 120 filters in the 1st fc layer. 84 filters in the second fc layer. This is in the LeNet5 CNN. These filters mean different weighting schemes. The fully connected layer merges the spatial and spectral feature map into feature dimensions and thus we get 120D feature vector from the 16 5x5 feature maps. The 84D feature vector (which is the 2nd fc layer) is obtained by dimension reduction on the 120D feature vector.

As the name suggests, this layer is fully connected to all the activation functions in the previous layer. It takes input (from the layer preceding it) and output is an N dimensional vector (N is the no. of classes). The weights are assigned to each element of the vector and the probabilities for each class are computed by multiplying the weight vector and the previous layer.

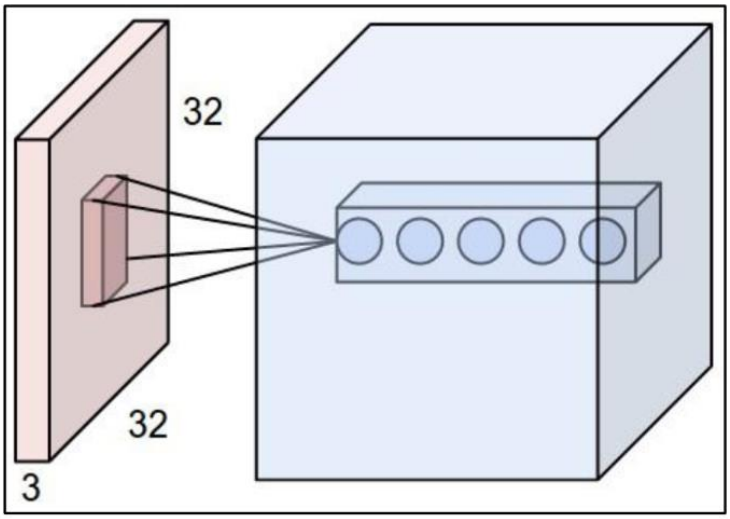
The fc layer provides the classification decision as the output.



2) convolutional layer

The convolution layer has the most computationally heavy task. There are filters of size 5x5 in the CNN that we discussed in the lecture. These filters are used on the input image. The dot product of the filter and the image pixels is taken. The parameters of the convolutional layer include 'stride' which means the number pf pixels the filter is shifted by during the convolutions. Filter size and number of filters used are other parameters. Zero padding is done too so the edge pixels of the image can also be used to extract features.

As we slide the filter over the height and width of the image 2D activation map is generated which represents the response map and represents the features extracted by using that particular filter. The number of activation maps or response maps equals the number of filters used. These response maps are stacked along the depth. Thus, this is how the output image volume is created.

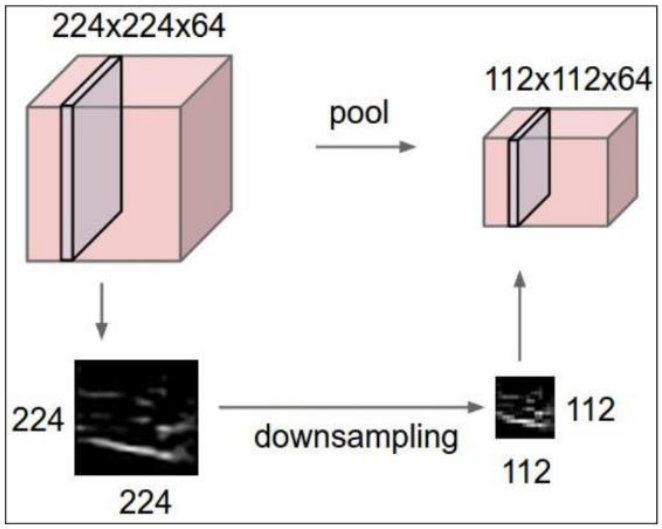


This is a representation of the 1st convolutional layer. The filter is applied to the 32x32 image and the output for that patch is shown as the first blue circle. The other blue circles represent the outputs from other filters.

3) the max pooling layer

This layer is used to reduce the size (spatial size) of the output from the convolutional layers. So, this layer is present after each convolutional layer. This layer has parameters like window size and stride. The window size is 2x2 in our case. So, it reduces the 4 pixels to one pixel by taking the maximum of the 4 values in each window i.e. by keeping the maximum value. This layer works on all the outputs obtained from the convolutional layer. That is, it works through the depth of the convolutional layer outputs independently. There is no learning in this layer.

The image below demonstrates the function of this layer.

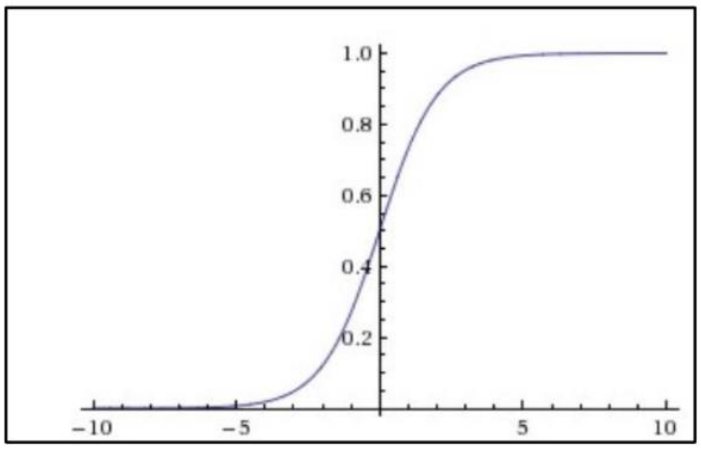


4) the activation function

This function is a nonlinear function. It decides which features are selected and which features need to be discarded for converting the output of the convolutional layer to an N dimension feature vector. It introduces non-linearity to the network. The output of the convolutional layer is the product of the inputs and the weights plus a bias term. Some of the popular activation functions are the sigmoid function, the tanh function, ReLU function and leaky ReLU function.

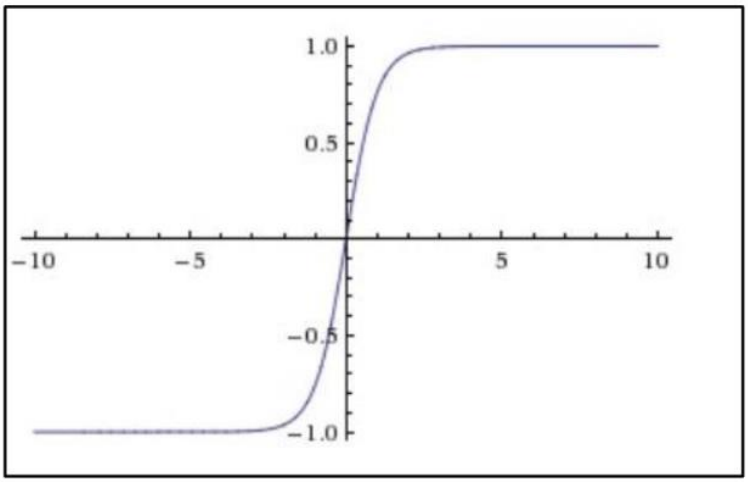
(i) sigmoid function

This function makes the range of numbers between 0 and 1. It has slow convergence and it kills the gradient so it is not favourable.



(ii) tanh function

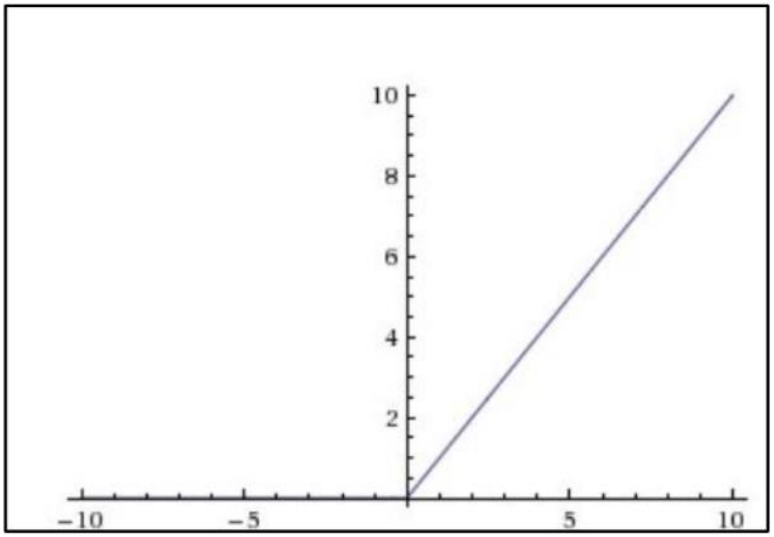
This function makes the range between 1 and -1. The convergence is faster than sigmoid. But it also kills the gradient.



(iii) ReLU

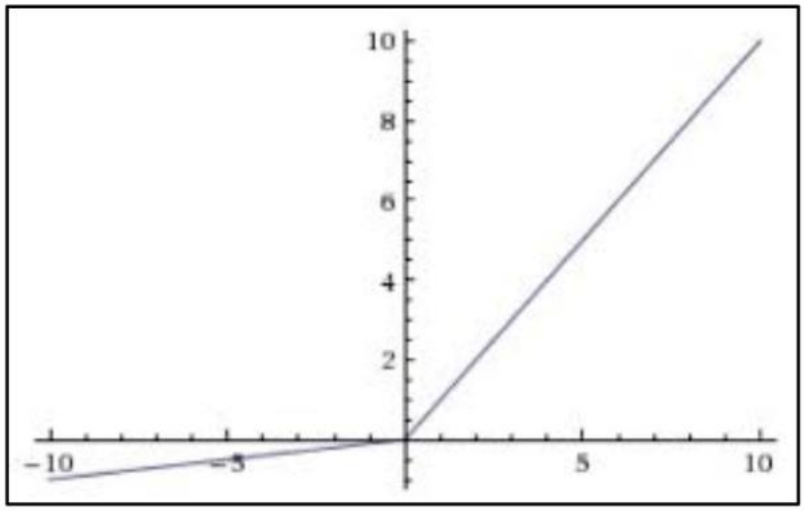
It is the Rectified Linear Unit.

The threshold of the activation function is 0. Therefore, all values below 0 are clipped. It does not saturate, and it has faster convergence as compared to sigmoid and tanh functions. No zero center for output values. In ReLU if the learning late is not set properly, there is a problem of 'Dying ReLU'. In this, the ReLU gives the same output for all the inputs.



(iv) Leaky ReLU

This solves the problem of dead ReLU. In this, if the input is less than 0, the output will have a small negative value. This does not saturate and also has a faster convergence than sigmoid and tanh functions.



5) the softmax function

This function is like a generalized version of logistic regression.It can be explained mathematically. It converts the k dimensional vector of arbitrary real values into a k dimensional function of values ranging between 0 and 1. The sum of thesevalues in each of these vectors is 1. These function are trained under a cross-entropy regime and they give the output as a non-linear variant of multinomial logistic regression.

The functions are described above along with their explanations. The function is image classification and the different functions to achieve this are carried out by the different components mentioned above.

2. Overfitting

Overfitting in model learning is when there are more training parameters than there are data samples in the training. It is when we fit the parameters to a very granular level to fit the training data. In this case, the model works very well for training data but it performs very badly for testing data. This is due to lack of generalization of the model.

To avoid overfitting in CNN the following things can be done.

(i) Cross Validation

This is done to achieve generalization of data. The training samples are split into 2 parts: training and validation set. This split is done randomly and changes for every iteration. So when we train the data on the training set, we can test it on the validation set. We tune the parameters based on the training set, but because the selection of train and validation data is random, we are able to achieve a generalized fit for the data.

(ii) The other method that can be used is to increase the data so that the data is more than the number of parameters. We can generate data synthetically. We can augment the data and

another way to generate more data is by using geometric modifications on the existing data so that it adds to the data.

3.

CNNs are used for computer vision problems widely nowadays. This is because they provide good results as compared to all other models available now or that existed earlier or were more popular earlier. CNNs provide very good results for problems like object detection, object recognition, classification etc.

In any image classification problem there are 4 steps, namely, pre-processing of data, feature extraction, classification and post-processing of data. The feature extraction step is tricky because there is no clear definition of which features are to be extracted or how many features are to be extracted. We need to choose features having a high discriminant power which will help us correctly classify the images. The classification step uses different classifiers like linear classifiers, nearest means classifiers, Support Vector Machines etc. for classification. The weights are updated using the loss function in order to correctly classify the data/images.

In traditional methods, the steps of feature extraction and image classification are separate steps. But, in CNN these two steps are combined.

CNN works much better because:

(1) We know the number of features to be extracted since it is equal to the number of filters used in the convolutional layers.

(2) The max pooling layers that are present after each convolutional layer help us to extract the best features i.e. features with the highest discriminant power.

(3) The softmax function is used which helps us get a good loss function which provides us with the weights that need to be updated for accurate classification. Back propagation and chain rule is used.

(4) It is faster and gives higher accuracy than any traditional method. It extracts features and classifies the data simultaneously in one step. It is self-learning.

4.

Loss Function:

The loss function is used in CNN to gauge the gap between actual labels and the predicted labels.

The loss function used in CNN is softmax function which calculates the weights of the filter and data based on how wrongly the data is classified in each iteration. These weights are assigned in order to classify the data correctly and increase the accuracy of classification. The updated weights in each iteration are done so that the loss function is minimized.

The aim is to minimize the loss function which will in turn maximize the gain/accuracy of the testing as well as training data.

Optimization methods like stochastic GD, batch GD, sequential GD can be used to minimize the loss function. Back propagation and chain rule are used to update model parameters based on the loss function. This is done in every iteration

Back Propagation:

CNN combines the feature extraction and classification steps and is therefore a self-learning model. Back propagation and chain rule are the main ideas behind CNN being a self-learning model. For the minimization of loss function gradient descent is used since it is the most robust and simple method for learning. The performance of the model depends on backpropagation.

The back propagation is needed to update weights to minimize the loss function at each iteration.

First, the loss function is calculated by the random initialization of the weight and bias term. Then the gradient of the loss function is calculated, and the weights are updated at iteration so that the weights are updated along the negative gradient descent. The chain rule is used to achieve this.

The new gradient and the local gradient are multiplied to obtain the gradient descent of each weight. This is what is done in back propagation. This process is done till we reach the desired input from the output that we started with. This is the reason it is called back propagation.

The loss function is computed with the updated weights using forward propagation and then the GD od the loss function is propagated backwards to get new weights. This process is repeated till the loss function is minimized i.e. when it converges.

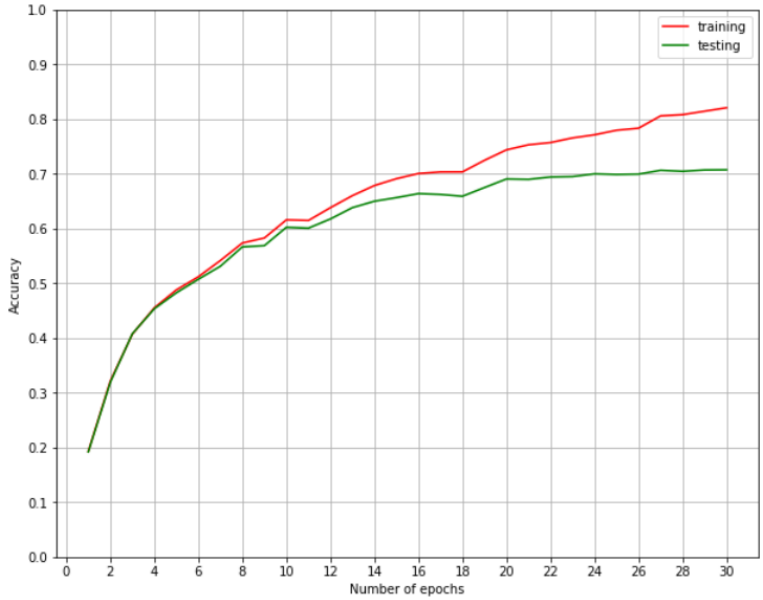
Back-propagation is less complex computationally. It saves memory and the computational blocks are cascaded without any hassle (chain rule). It is also self-learning because it knows which features to extract on its own as and when the weights are updated and it learns which features are more important.

(b) 1.

I tried different values of learning rate and kept other parameters constant. I got the best test accuracy for learning rate 0.001. Learning rate of 0.005 overfit the data and learning rate of 0.0009 was very slow as its convergence was slow.

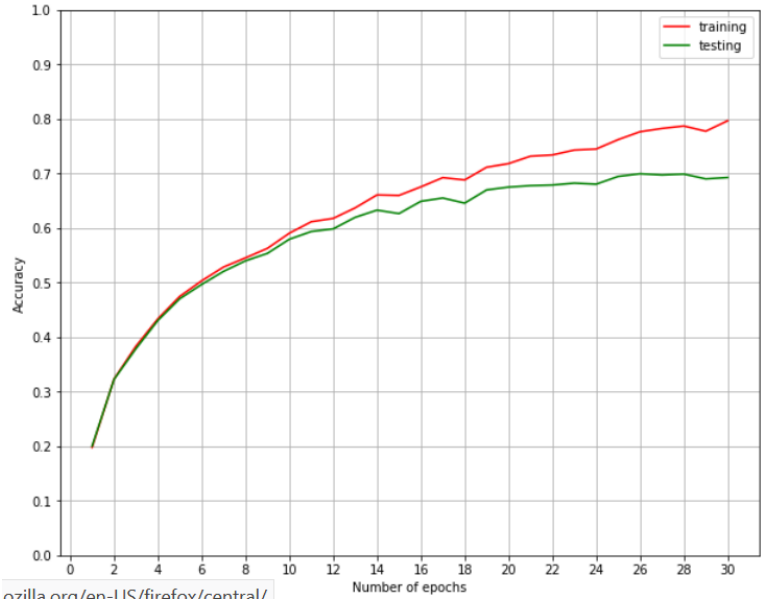
Learning rate = 0.001
Momentum = 0.9
Number of epochs = 30
Batch size = 64
CrossEntropyLoss, SGD criterion function

```
Training accuracy: 0.82094
Test accuracy: 0.7075
```



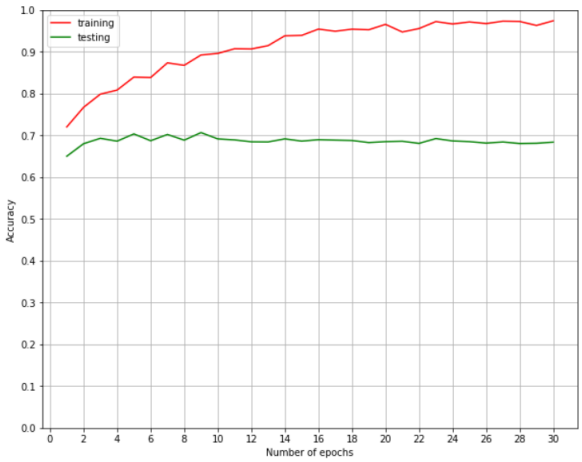
Learning rate = 0.0009
Momentum = 0.9
Number of epochs = 30
Batch size = 64
CrossEntropyLoss, SGD criterion function

```
Training accuracy: 0.79686
Test accuracy: 0.6928
```



Learning rate = 0.005
Momentum = 0.9
Number of epochs = 30
Batch size = 64
CrossEntropyLoss, SGD criterion function

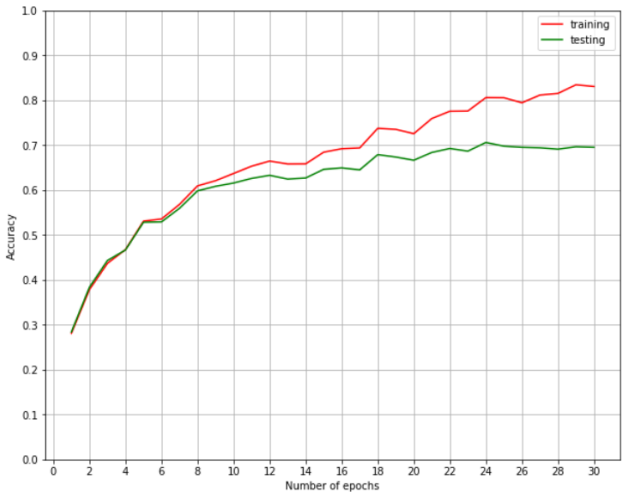
```
Training accuracy: 0.97436
Test accuracy: 0.6839
```



Next, I varied the values of momentum and found that momentum = 0.9 gave the best result.

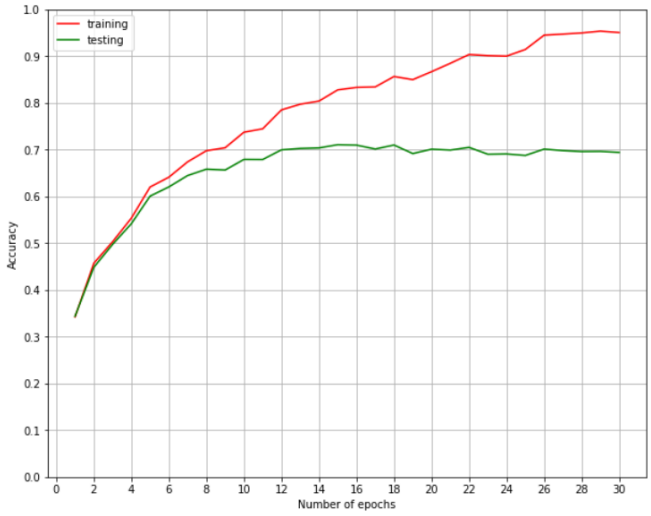
Learning rate = 0.001
Momentum = 0.6
Number of epochs = 30
Batch size = 64
CrossEntropyLoss, SGD criterion function

```
Training accuracy: 0.83088
Test accuracy: 0.6954
```

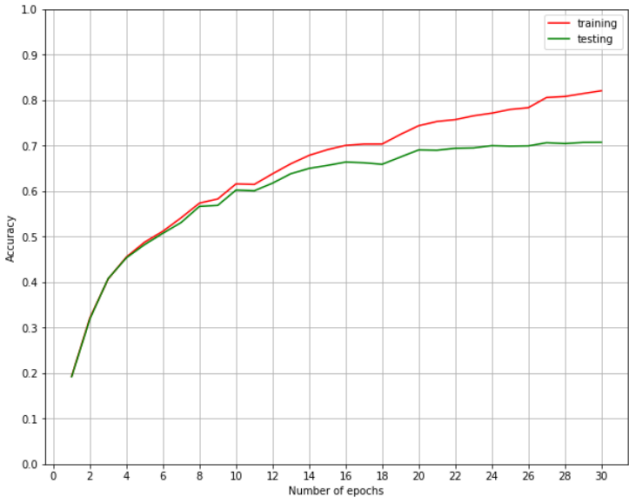


Learning rate = 0.001
Momentum = 0.8
Number of epochs = 30
Batch size = 64
CrossEntropyLoss, SGD criterion function

```
Training accuracy: 0.95072
Test accuracy: 0.6943
```



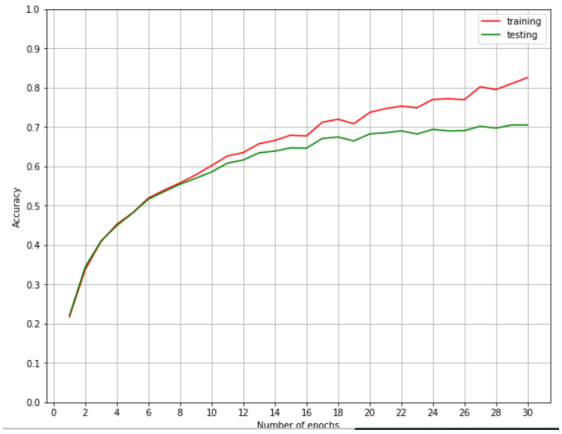
Learning rate = 0.001
Momentum = 0.9
Number of epochs = 30
Batch size = 64
CrossEntropyLoss, SGD criterion function
Training accuracy: 0.82094
Test accuracy: 0.7075



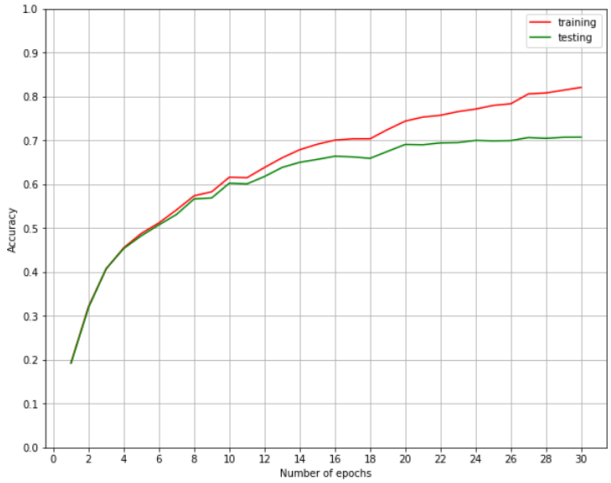
I varied the batch size and found that the smaller batch size gave the best test accuracy. This is because the data is divided into smaller batches and gets classified more accurately.

Learning rate = 0.001
Momentum = 0.9
Number of epochs = 30
Batch size = 32
CrossEntropyLoss, SGD criterion function

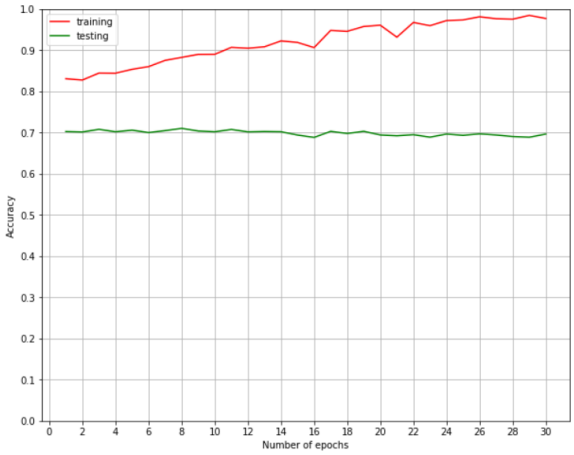
```
Training accuracy: 0.8261
Test accuracy: 0.7075
```



Learning rate = 0.001
Momentum = 0.9
Number of epochs = 30
Batch size = 64
CrossEntropyLoss, SGD criterion function
Training accuracy: 0.82094
```
Test accuracy: 0.7054
```

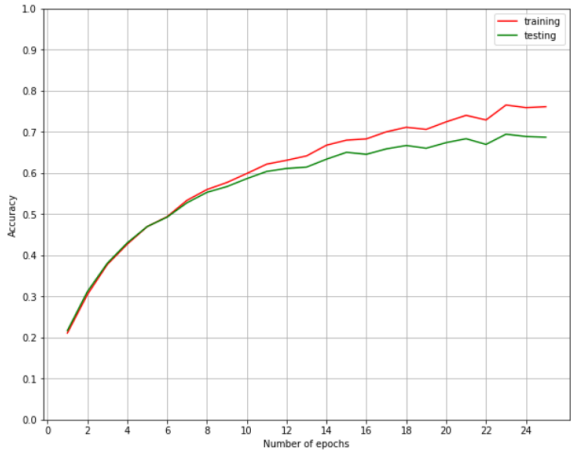


Learning rate = 0.001
Momentum = 0.9
Number of epochs = 30
Batch size = 128
CrossEntropyLoss, SGD criterion function
`Training accuracy: 0.97716`
`Test accuracy: 0.6965`

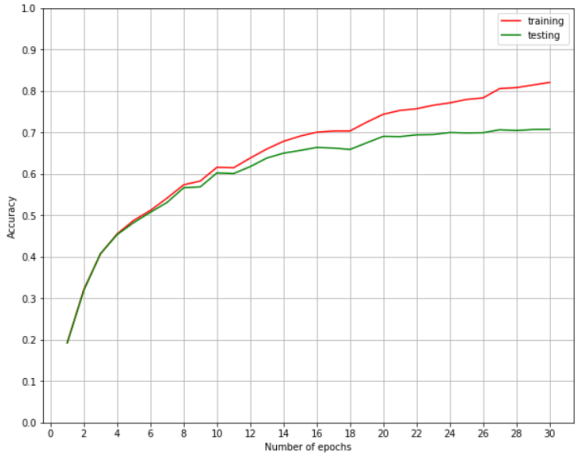


Learning rate = 0.001
I found out that the number of epochs affect the test accuracy. 25 epochs were less and gave a low accuracy and 35 epochs tend to overfit the data and thus reduce the accuracy of test data. I found that 30 epochs were giving the maximum test accuracy.

Momentum = 0.9
Number of epochs = 25
Batch size = 64
CrossEntropyLoss, SGD criterion function
`Training accuracy: 0.76142`
`Test accuracy: 0.6873`

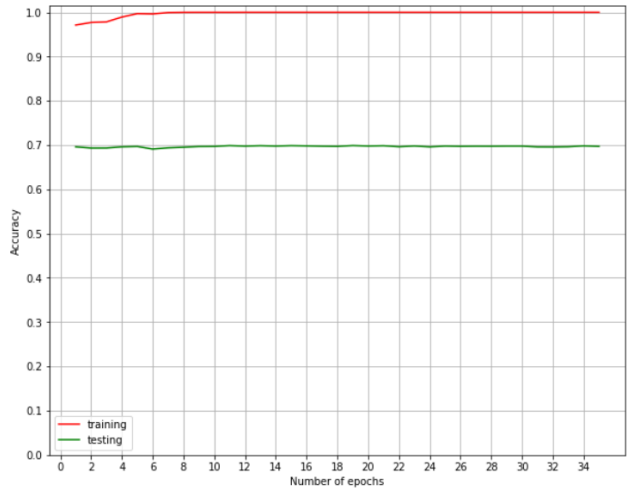


Learning rate = 0.001
Momentum = 0.9
Number of epochs = 30
Batch size = 64
CrossEntropyLoss, SGD criterion function
Training accuracy: 0.82094
Test accuracy: 0.7075



Learning rate = 0.001
Momentum = 0.9
Number of epochs = 35
Batch size = 64
CrossEntropyLoss, SGD criterion function

Training accuracy: 1.0
Test accuracy: 0.6971



5.
I also found that the CrossEntropyLoss function gives a much higher accuracy than the MSE loss function.

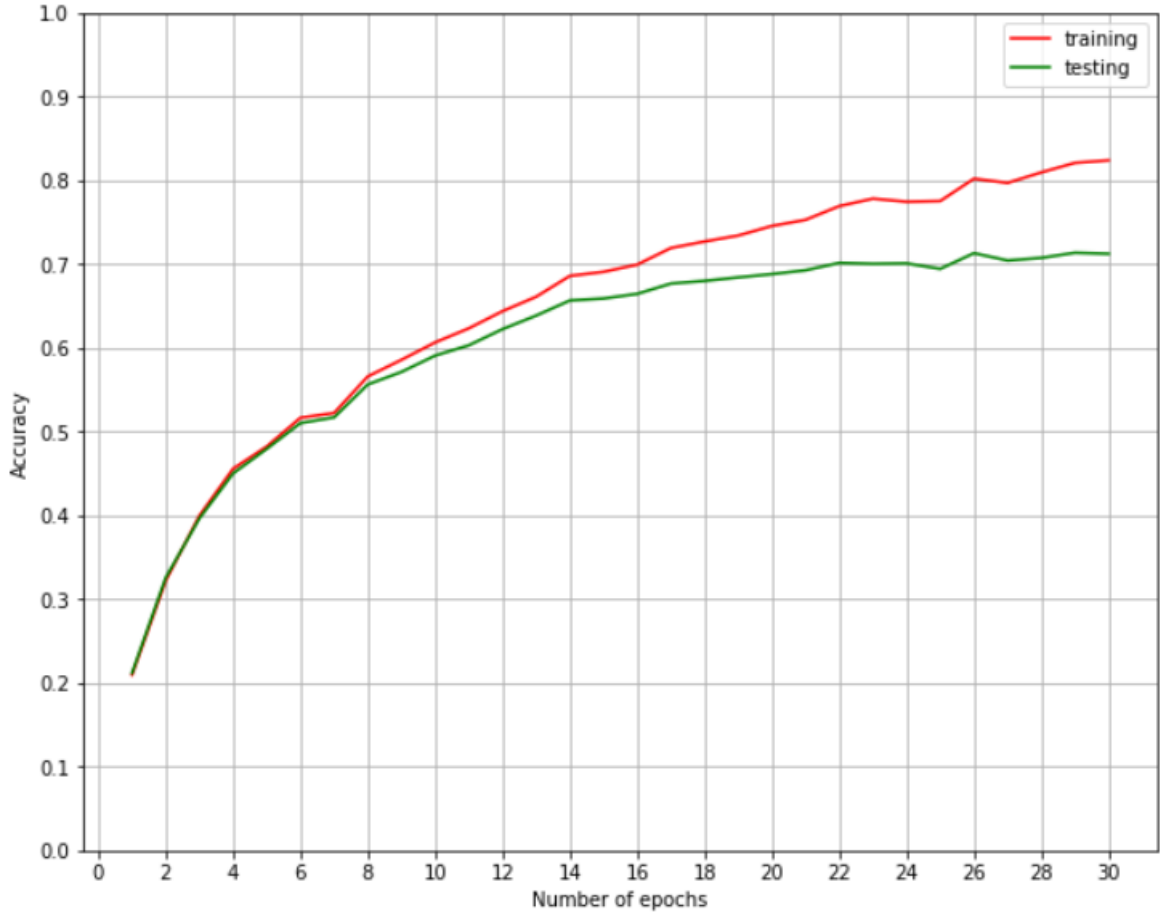
2.

I found that the following parameter settings gave me the best test accuracy.

Learning rate = 0.001
Momentum = 0.9
Number of epochs = 30
Batch size = 32
CrossEntropyLoss, SGD criterion function

Training accuracy: 0.82408
Test accuracy: 0.7125

With these parameter settings I got 71.25 % test accuracy. The training accuracy is 82.408 %.

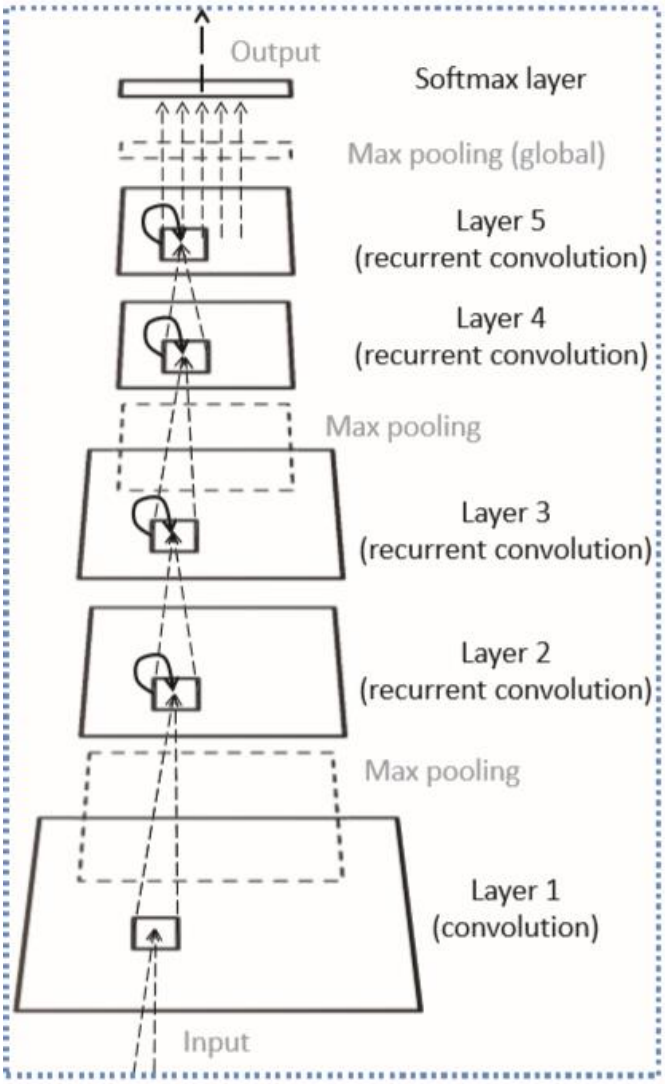


(c)

I have chosen the following paper [1] . Their network gives 92.91% accuracy for CIFAR10 dataset.

1.
The main idea of the authors behind RCNN was to include recurrent connections in the CNN network. This idea came from the fact that in the visual system in our bodies, recurrent connections are present in large numbers, which makes us recognize objects so well. RCNN proves that recurrent structure in the network is far superior to the purely feed-forward system.

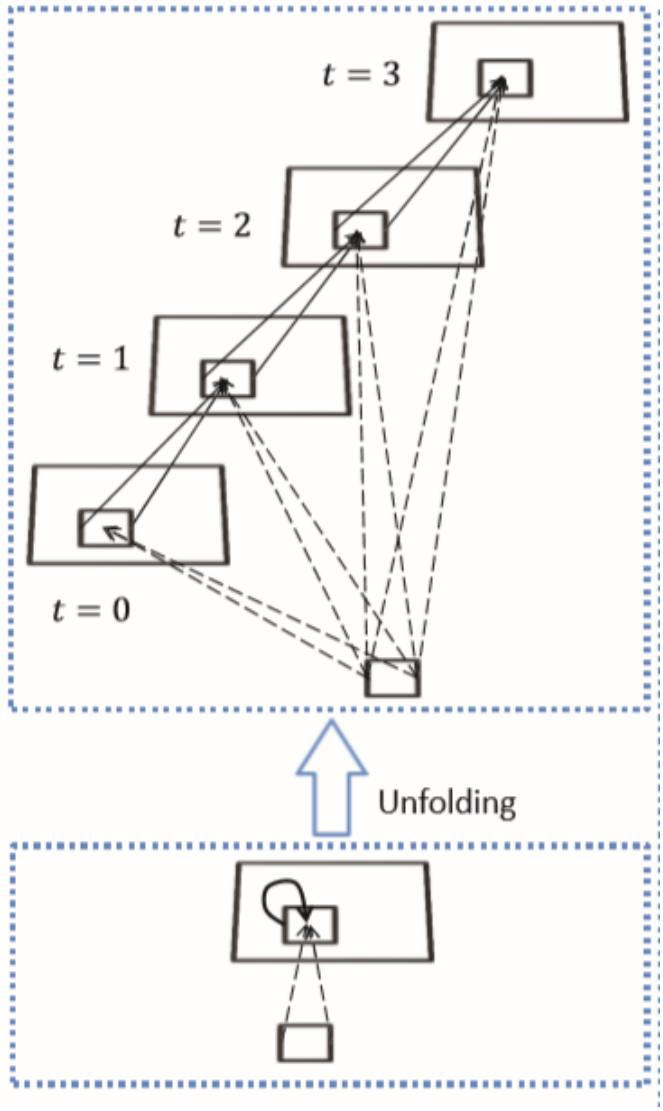
The block diagram of the RCNN network that the authors used in [1] is shown below:



This image has been taken from [1].

The RCNN has 1 convolutional layer as the first layer. Then it has 4 recurrent convolutional layers (RCL) and max pooling layers after each RCL. Then, it has 1 softmax layer after which the output layer is present.

Each RCL has a subnetwork of 4 layers. This is a feed-forward subnetwork. You can visualize it as follows:



The block at the bottom represent the recurrent convolutional layer and the block at the top represents how this layer unfold to form the feed-forward subnetwork. It has a depth of 4.

Only feed-forward system is present between two RCLs. The max pooling layers have a window size of 3x3 and the stride is 2. The max pooling layer after the last RCL is a global max pooling layer which guves the output as a feature vector representing the image. It gives out the maximum of each feature map. After this, the softmax layer is there which is used to classify the objects into the 10 classes of CIFAR10.

In classification, for minimization of the loss function, the Cross Entropy Loss function is used. This is done by using the back propagation through time algorithm (BPTT) and the criterion function was SGD (Stochastic Gradient Descent).

In their implementation, they kept the number of feature maps in layers 1-5 to be the same i.e. K feature maps. They kept the filter size in the first convolutional layer as 5x5 and then the filter size in RCLs as 3x3. They set the other parameters such as momentum to 0.9. The network was tested for several iterations and when the accuracy stopped improving they set the learning rate to 1/1000 of the original value that it was set to. The weight decay terms were kept the same for all weights.

For CIFAR10 dataset, the images were preprocessed. This preprocessing was done by subtracting the mean of each image from the image. This was done only for the training data. Increasing the K value i.e. the number of feature maps gave better results.

The recurrent structure enabled the units in one layer to be able to be modified by the results of the same layer. The depth of the network increased in RCNN as compared to CNN, at the same time keeping the number of parameters constant and weight sharing between layers was also constant. It was found that if the number of parameters in RCNN was increased, it would perform even better.

2.

The recurrent convolutional layers in RCNN are advantageous.

(i) In RCNN because the recurrence in the subnetwork is there, the units in layers are affected by the other units in the same layer. This enable the RCNN to be able to scan/watch over a larger window which in turn helps in object recognitions. For eg. If the window size is smaller, you will only be able to see a small portion of the object which might not be sufficient to classify the object. A larger window size overcomes this problem.

In CNN, since recurrence is not present, the window size is smaller and thus the classification accuracy is lesser than that of RCNN. In, CNN a larger window is possible in the higher layers only.

(ii)
The depth of the network in RCNN is more and the number of parameters and weight sharing is the same.
CNN has a lesser depth than RCNN with the same number of parameters and weight sharing. Thus, RCNN is more effective. The depth of the CNN can be increased, but by experiments it has

been proven that even if the depth is increased, the performance achieved through recurrence is superior.
This is because it is difficult for the learning algorithm to learn a very deep network. It becomes computationally very complex.

(iii)
The RCNN can be visualized as a CNN with larger number of paths between input and output. This enhances the learning. It enables the network to learn complex features.

CNN can have a slight advantage because they have shorter path between input and output. This can help the back-propagation algorithm while training the data.

Reference:

[1] Ming Liang, Xiaolin Hu “**Recurrent Convolutional Neural Network for Object Recognition**”