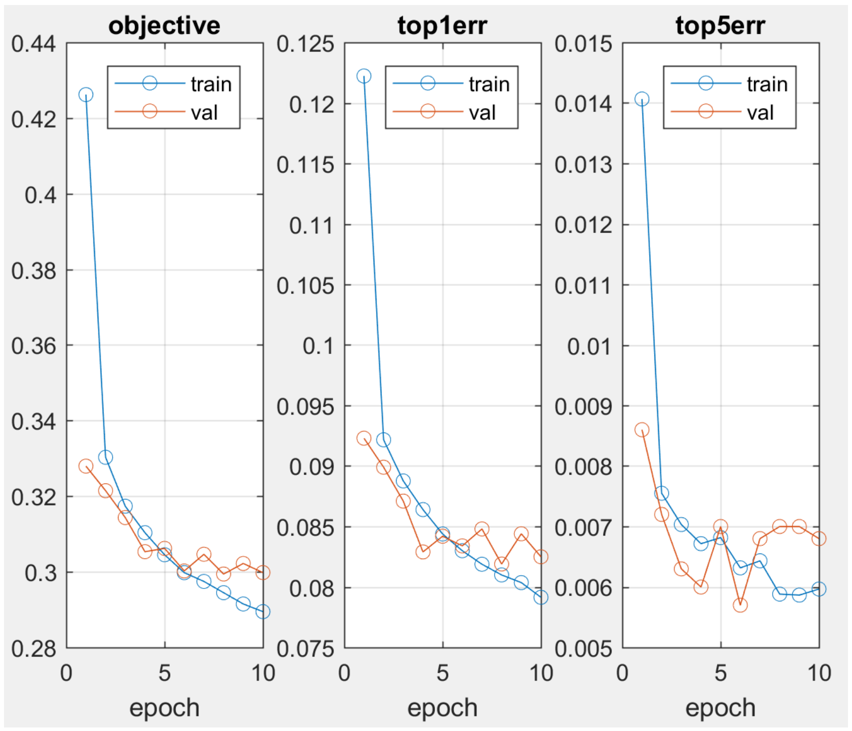
Assignment 5
Muhammad Basit Iqbal Awan
2020-10-0153

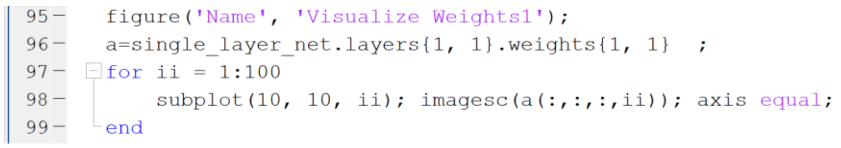
Problem 0

I started the code and it compiled. I changed the number of epochs in the ‘create_single_layer_nn’ to 10 epochs to get a better idea of the trend and better accuracy as shown below:



The accuracy came out to be 91.75%

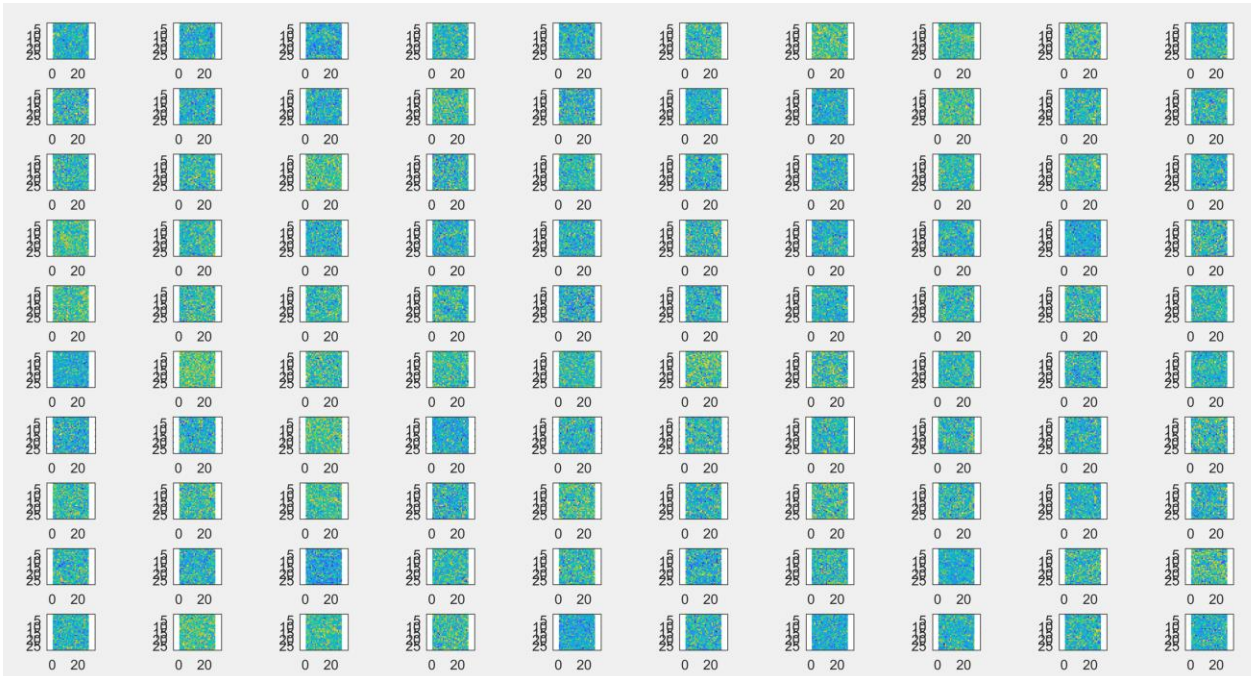
a) I extracted the weight from the first layer to the second and at first displayed all of them in a 10x10 grid. Using the code below:



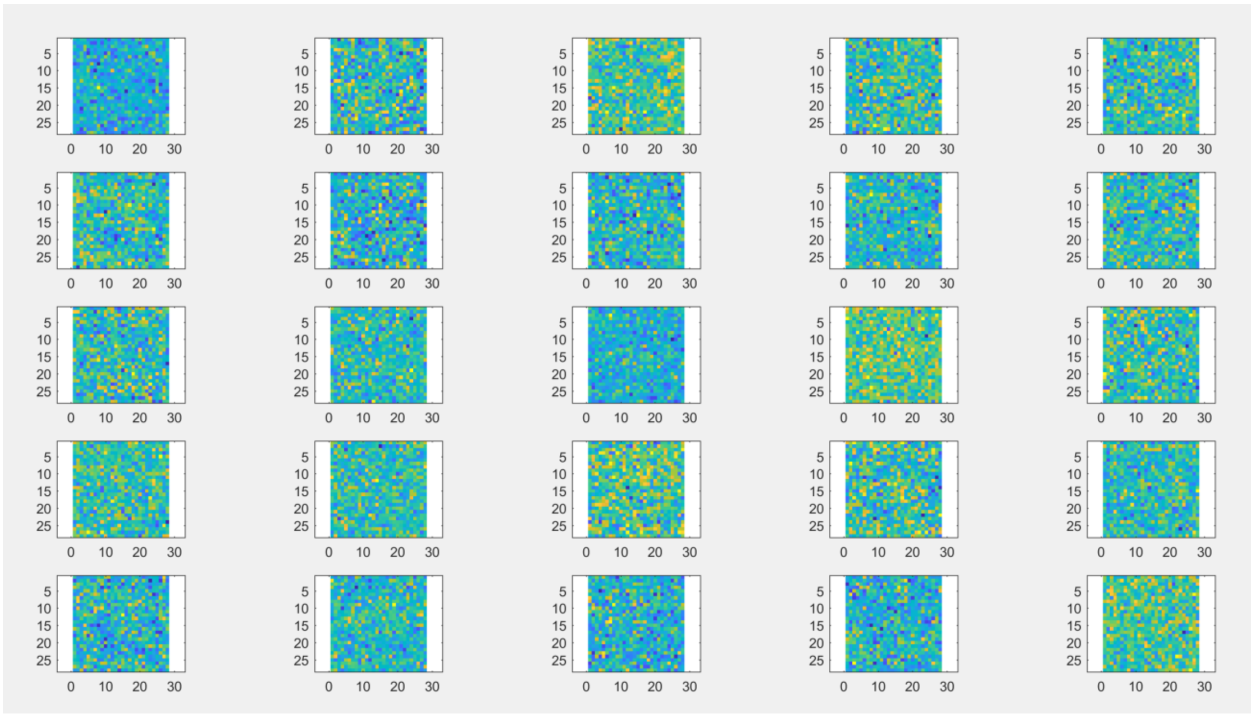
```
95 -    figure('Name', 'Visualize Weights1');
96 -    a=single_layer_net.layers{1, 1}.weights{1, 1}  ;
97 -    for ii = 1:100
98 -        subplot(10, 10, ii); imagesc(a(:,:,:,ii)); axis equal;
99 -    end
```

As seen, I stored the data in variable ‘a’ and then in like 98 indexed it in the loop.

The result came out as shown below:



But then I took 25 weights somewhere in the middle to give a better look, it came out like this:



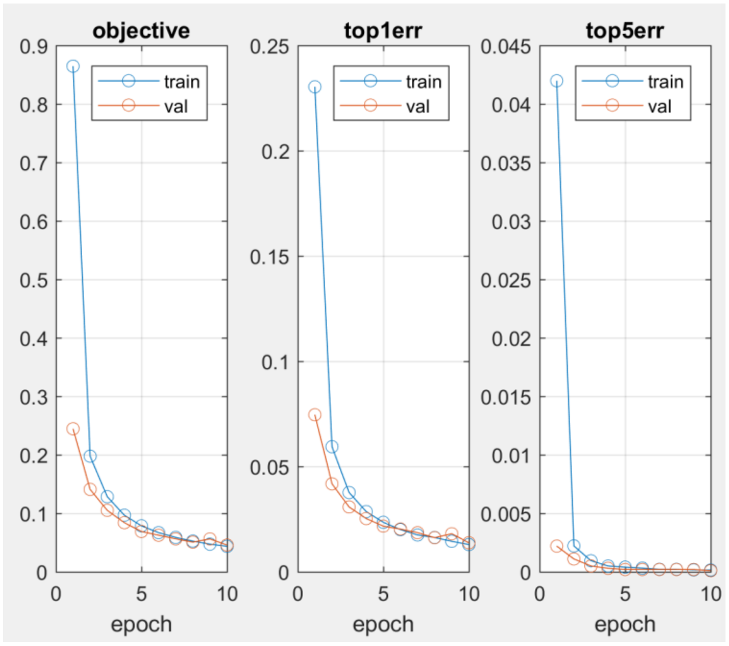
b)

The FC1000 layer has 28*28*1000 parameters
The FC10 layer has a total of 1000*10 parameters

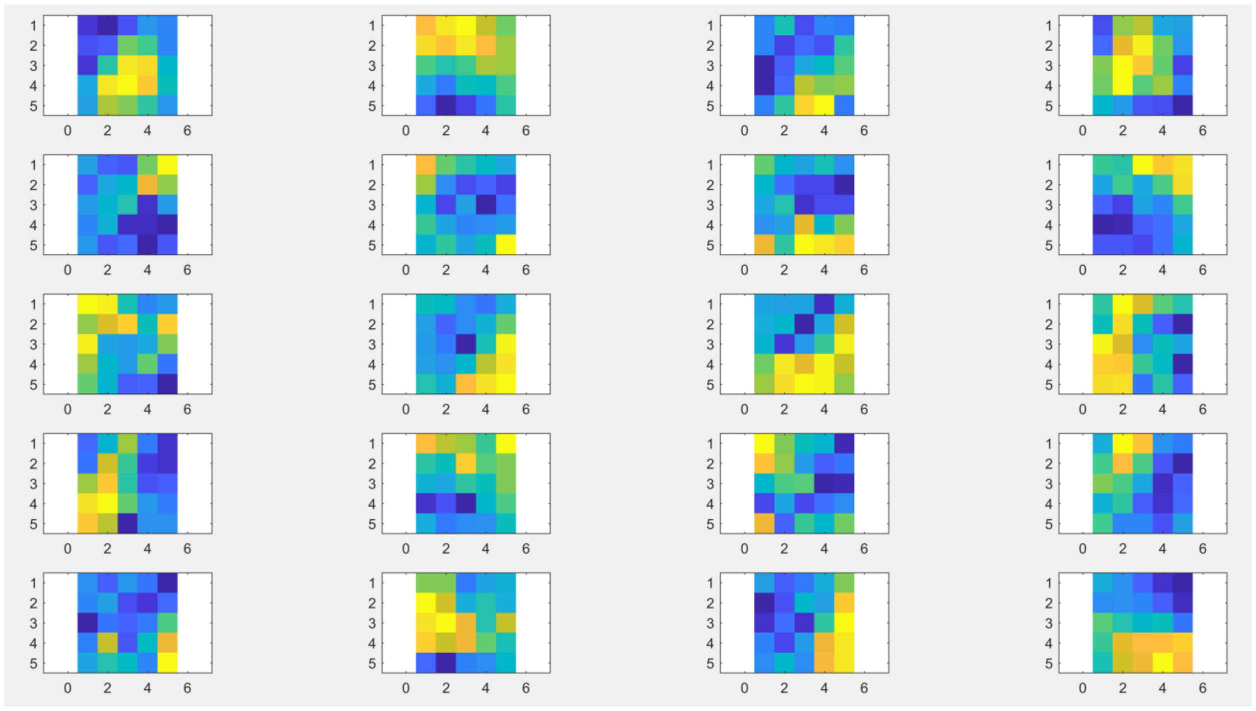
These add up to = 28*28*1000 + 1000*10 = 794000 parameters

Problem 1:

I ran the code for 10 epochs and got the following trend, which showed that the top 5 error went down to 0 at the 6th epoch and the top1err was slowly decreasing but getting steady at the 10th epoch, the accuracy was 98.62%:



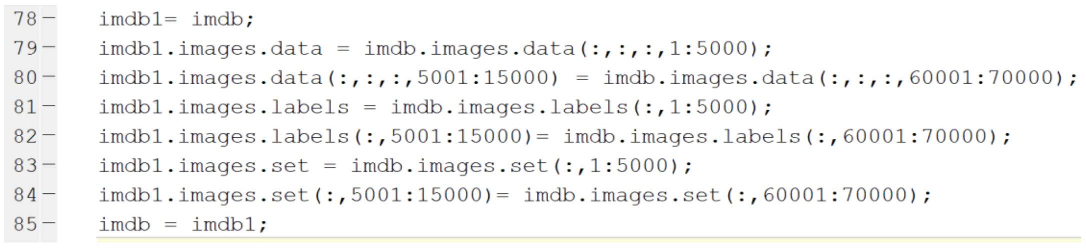
a) I used the previous code I used for the problem 0 to output the weights but with small modifications on the size of the subplot as there were 20 weights in the first layer. The weights represented in image form came out to look like this:



Just attaching the code snippet just incase you want to see it to see the small changes:

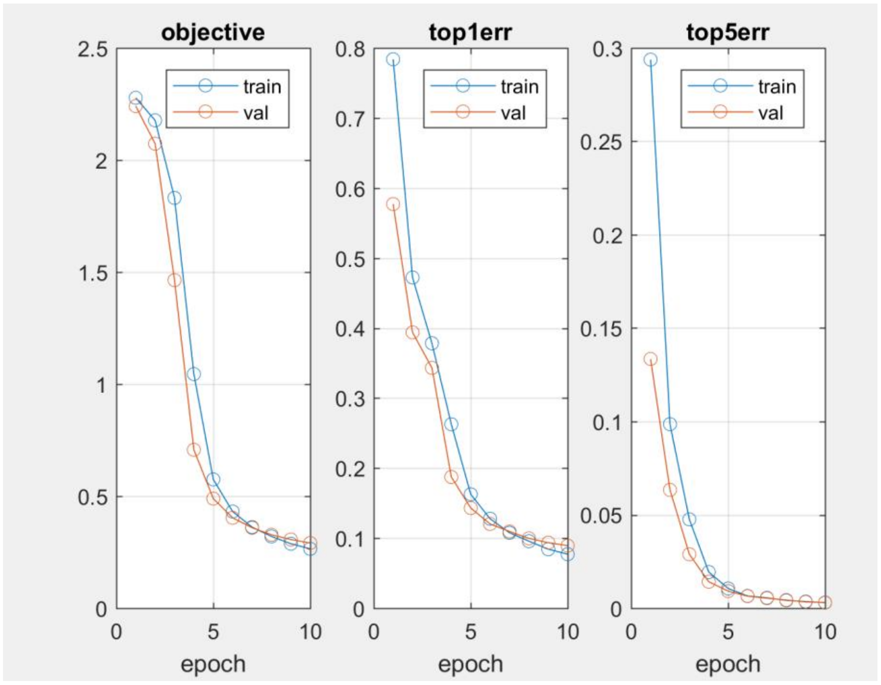
```
figure('Name', 'Visualize Weights2');
a=multi_layer_net_mnist.layers{1, 1}.weights{1, 1}  ;
for ii = 1:20
    im_ = imdb.images.data(:,:,:,ii);
    subplot(5, 4, ii); imagesc(a(:,:,:,ii)); axis equal;
end
```

b) To reduce the size of the batch, I took out the first 5000 images as the data is arranged randomly it had all the classes, along with their labels and set information and placed it in a new imdb file and placed the testing data after it. As shown in the code below:



```
imdb1= imdb;
imdb1.images.data = imdb.images.data(:,:,:,1:5000);
imdb1.images.data(:,:,:,5001:15000) = imdb.images.data(:,:,:,60001:70000);
imdb1.images.labels = imdb.images.labels(:,1:5000);
imdb1.images.labels(:,5001:15000)= imdb.images.labels(:,60001:70000);
imdb1.images.set = imdb.images.set(:,1:5000);
imdb1.images.set(:,5001:15000)= imdb.images.set(:,60001:70000);
imdb = imdb1;
```

After which I trained the network again and got the following results:

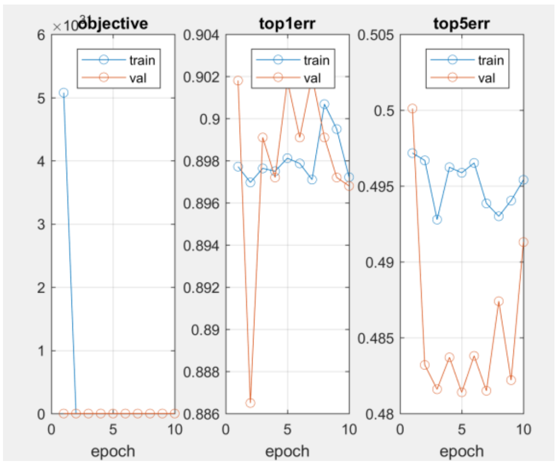


The network is training well for top5 errors but for top1 errors it's accuracy is stabilizing at 91 percent as can be shown by the trend. So the reduction in training examples makes the model a bit inaccurate and also requires it to train on more epochs than a with more training examples. The accuracy at the end came out to be 91.06%

c)

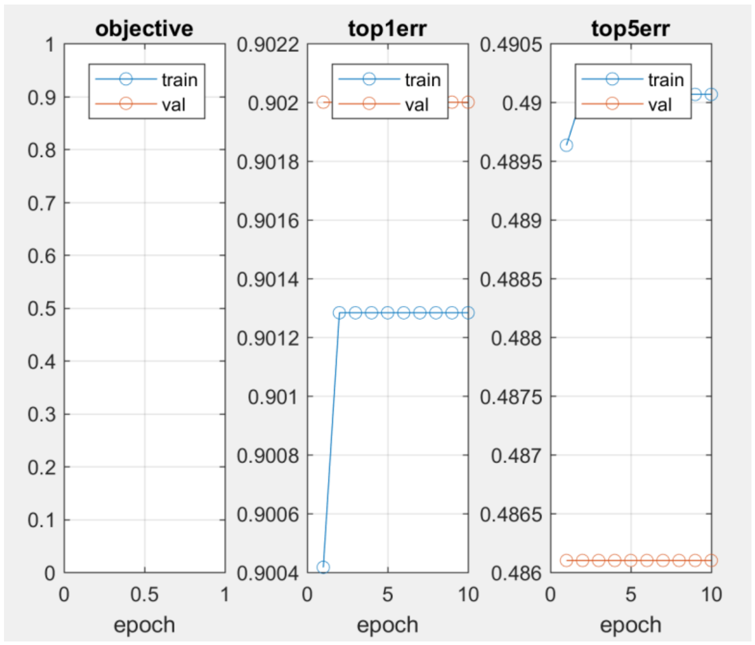
I tried 6 learning rates from 1 to 1e^{-5} and the results are shown below

Learning rate = 1:



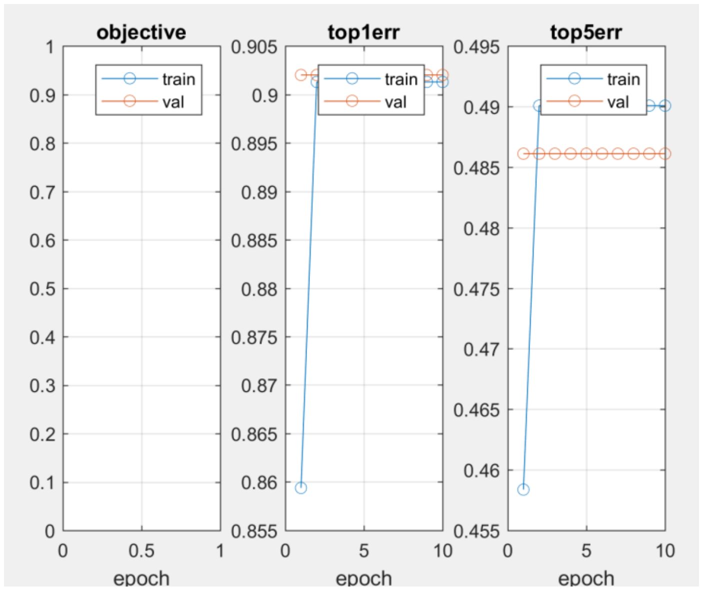
As we can see the error is shooting up and down because of the high learning rate, it is definitely faster but with this learning rate it's unlikely that it will converge ever.

Learning Rate = 0.1:



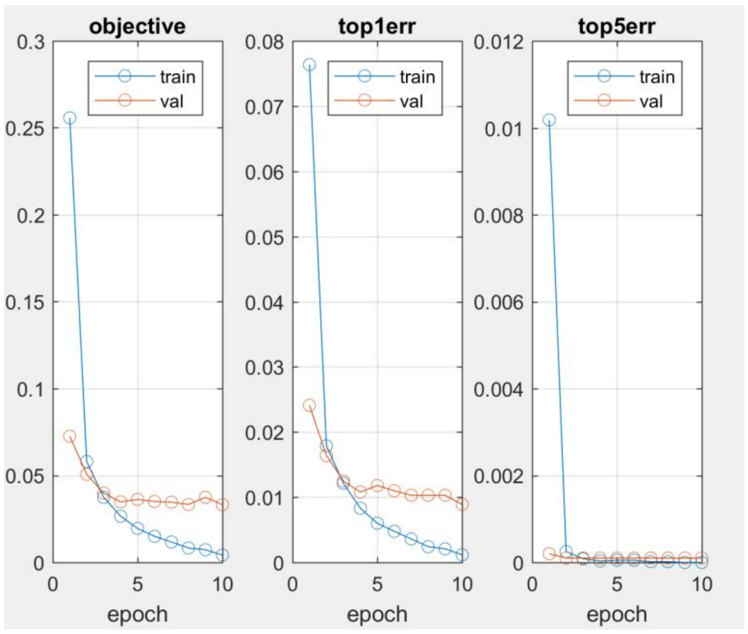
At first it seems like the network isn't training at all but I saw the results and according to them 0.1 training rate was the perfect thing to make the error slightly go up and get it back down slightly with the same amount, making it stuck in a loop and not really learning anything.

Learning Rate = 0.01:



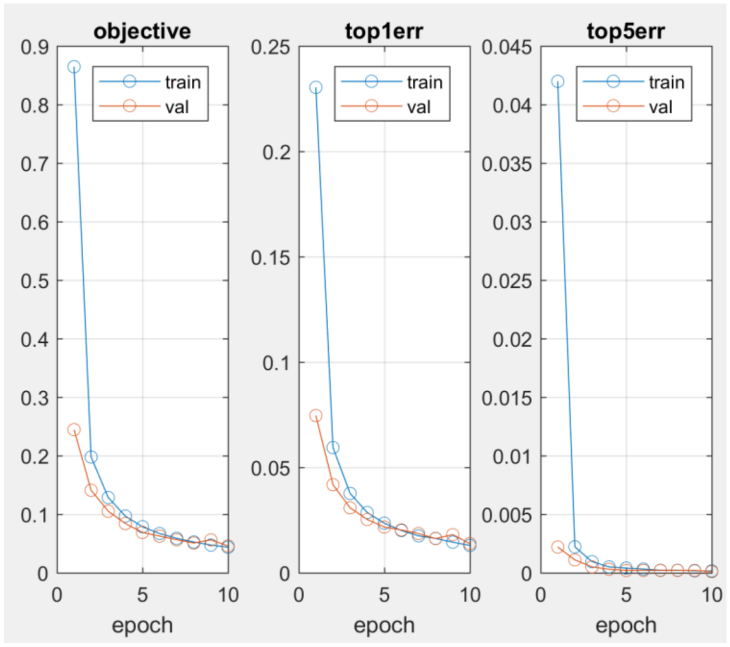
This is a horrible learning rate as well, as it shows the trends of the last one.

Learn Rate – 0.001:



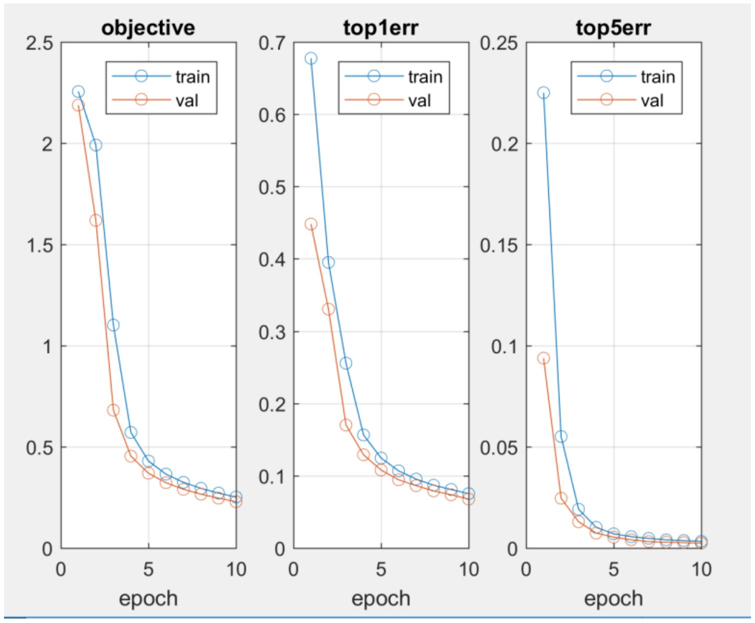
It is better than the previous one as the accuracy is going up it has found a point where it can't converge more and the error has stabilized. The learning is faster but less accurate.

Learning Rate = 0.0001(default):



The learning trend is a nice smooth curve, diverging into the minima and isn't taking too long as well

Learning Rate = 0.00001:



With this learning rate, the network is a bit slow to learn, but it does eventually converge to the right result.

Overall the trend is that, if you increase learning rate the network learns faster but it can also learn incorrectly or not learn at all. If we decrease it, it gives accurate results but take longer to learn. So, the key here is to get a sweet spot in the middle.

d)

The first Conv layer had 5*5*20 weights.

The second Conv layer had 5*5*20*50 weights.

The FC500 layer had 4*4*50*500 weights.

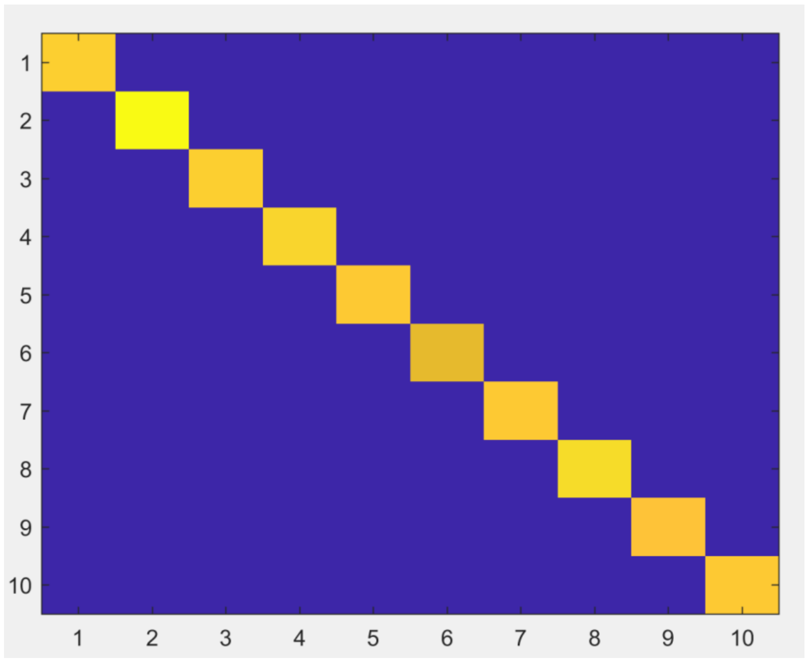
The FC10 layer had 500*10 weights.

So, a total of = 5*5*20+ 5*5*20*50+4*4*50*500+500*10= 430500.

e) I first turned the last later into softmax and then found the predicted value by forward loop and getting the highest score's index. Then I made the predicted list and compared it with the labels using the 'confusionmat' function to get the confusion matrix and after that just printed it. As shown in the code below:

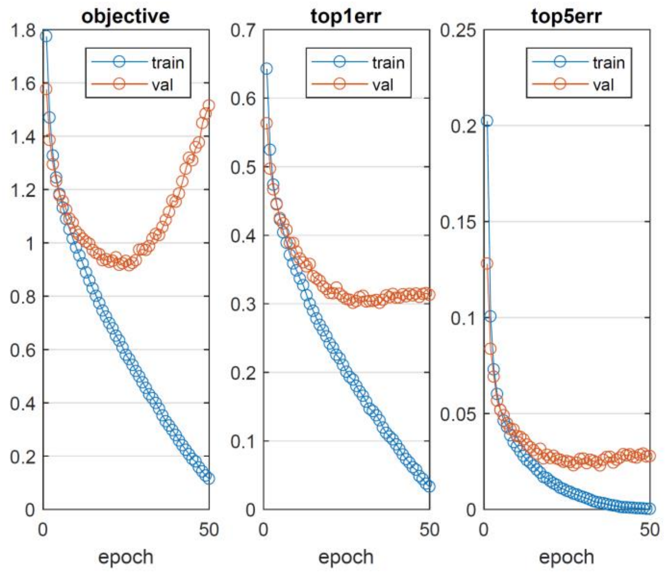
```
name = multi_layer_net_mnist;
name.layers{end} = struct('type', 'softmax') ;
predict = zeros(1,size(imdb.images.data,4));
known = imdb.images.labels(:,:);
for i=1:size(imdb.images.data,4)
    Original=imdb.images.data(:,:,:,i);
    res = vl_simplenn(name,Original);
    scores = squeeze(gather(res(end).x));
    [~,ind] = max(scores);
    predict(i) = ind;
end
C = confusionmat(known,predict);
imagesc(double(C))
```

The image I got is shown below, the diagonals are brighter signifying the high accuracy rate of the network as there are very few false negatives and positives.

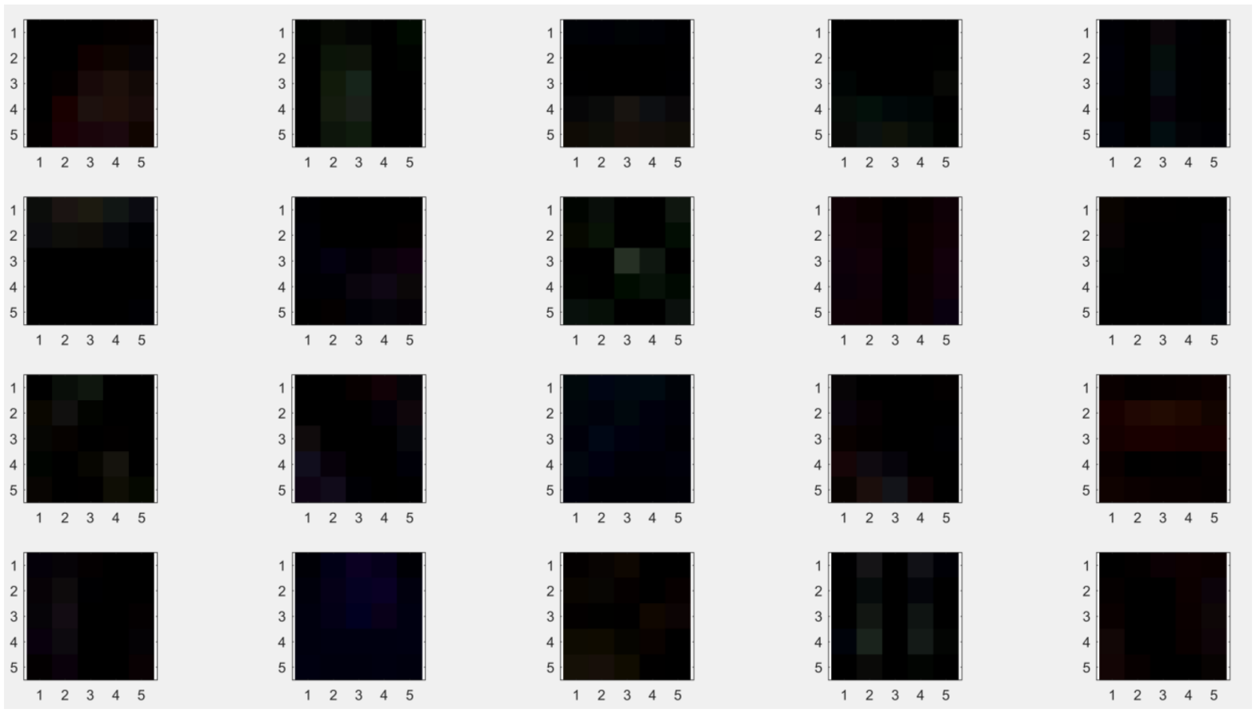


Problem 2

I ran the code with the default epoch number and got an accuracy of 68.72% and a learning rate which stabilized at the 30th epoch or so, as shown below:

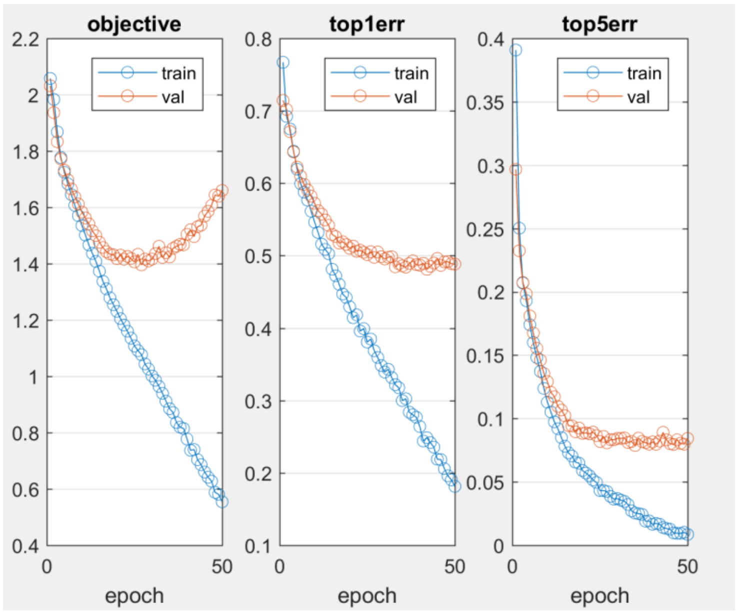


a) I have used the same code to get the weight data and represent them as I did in problem 1 a), as I have already used too much space with the images I don't think that sort of repetition is necessary. The weight represented I got after 50 epochs was as shown below:



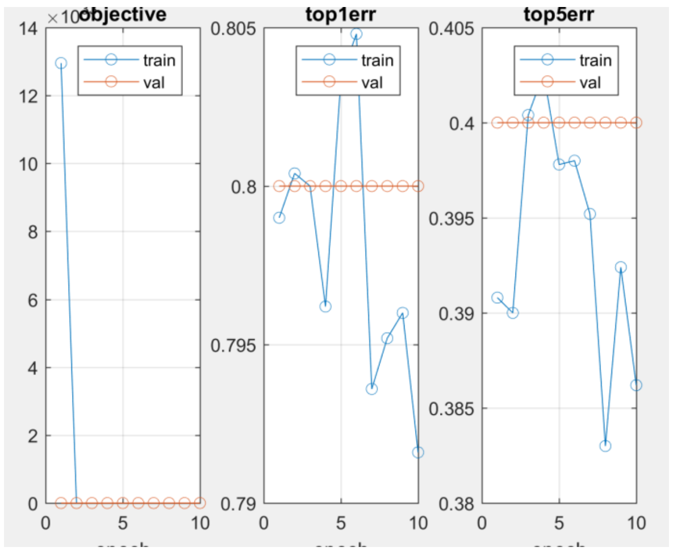
I think because of the addition of multiple layers and also the more complicated nature of CIFAR10 dataset, the weights come out to be this way.

b) Again I did this the way I did problem 1 b) at least the data extract and in this case I just had to change some indices because the test data was in 50000 to 60000 rather than 60000 to 70000 before. Other than that it was the same code. The accuracy with the fewer images was terrible, the accuracy came out to be 51.17%, although it was faster if that matters, which I don't it does.



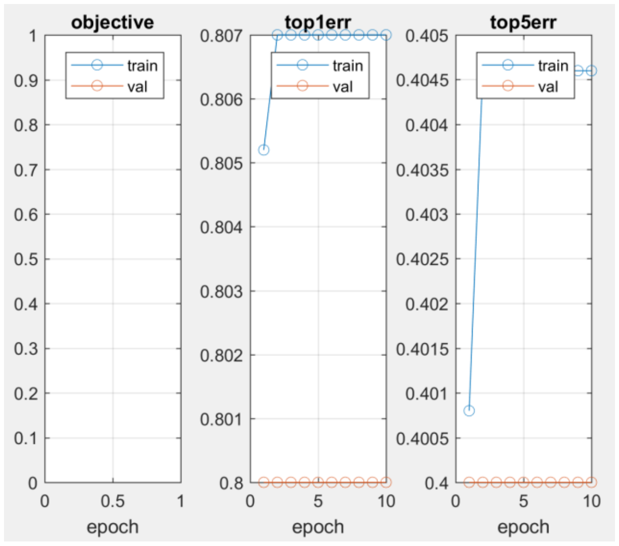
c) I learnt from my mistake last time and I tried 5 learning rates from 1 to 1e^{-5} and the results are shown below as there were a total of 5 trends.

Learning Rate =1:



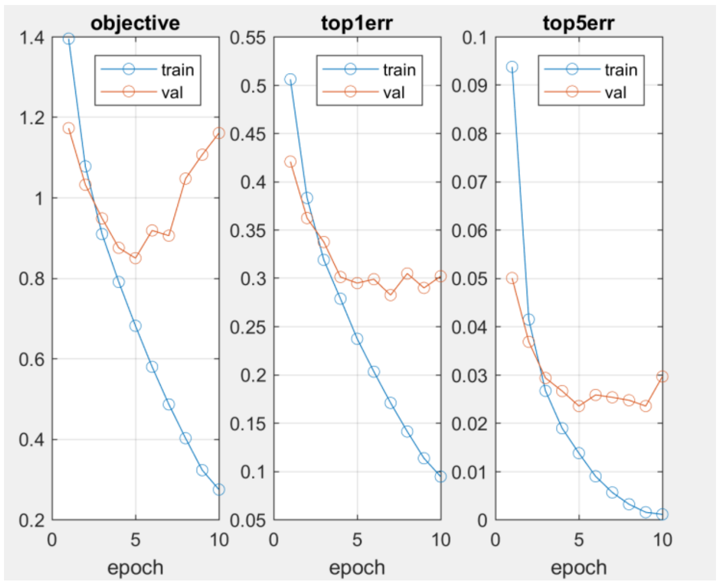
As with the MNIST dataset, we can see the error is shooting up and down because of the high learning rate, it is definitely faster but with this learning rate it's unlikely that it will converge ever. So in other words it's a horrible learning rate.

Learning Rate = 0.1:



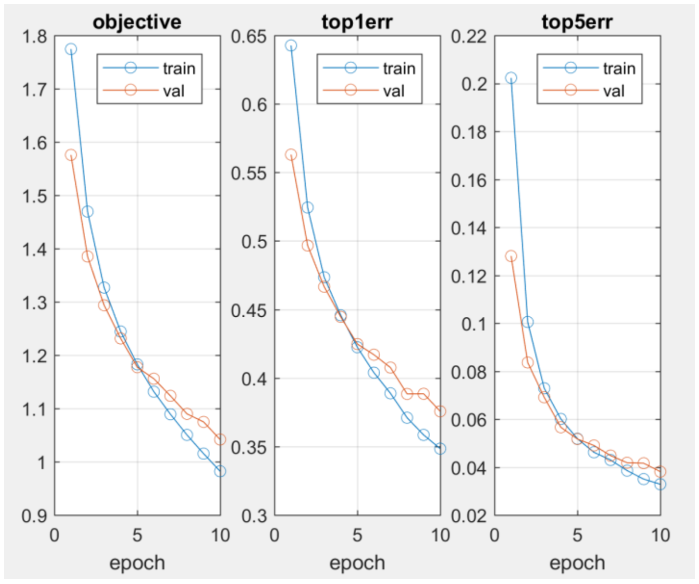
As with MNIST it is giving the same trend of not learning as it's getting stuck in the loop and not going up or down significantly, so it seems stagnated.

Learning Rate = 0.001:



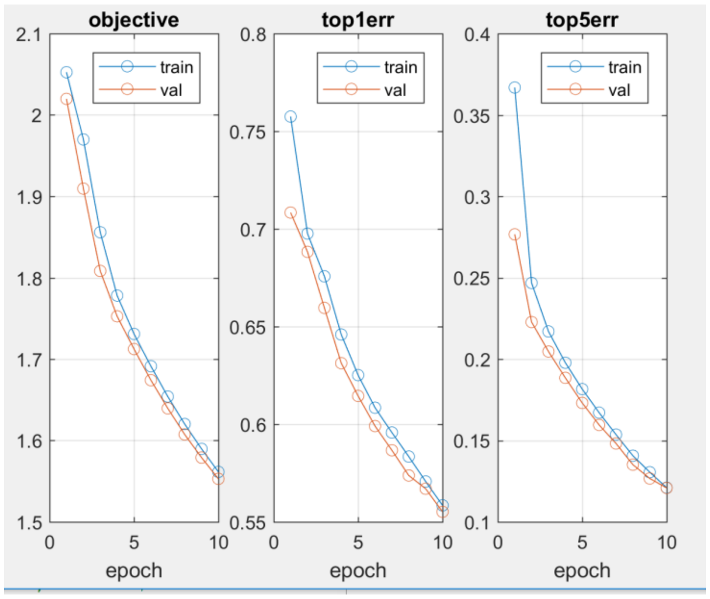
The network is now actually converging on this learning rate, but it is still giving the ups and downs because the learning rate is large. It’s fast but isn’t that accurate.

Learning Rate = 0.0001(Default)



The default as in MNIST gave a very nice curve stabilizing at the end and giving nice accuracy, and it’s speed is optimal as well.

Learning Rate = 0.00001:



In this it’s a bit slower but it’s converging just like the default one.

From these I have made the same conclusion I made previously it’s about getting the sweet spot, lower learning rate results in more accuracy but takes more time, and higher learning rate means faster, but will be less accurate.

d)

The first Conv layer had 5*5*3*20 weights.

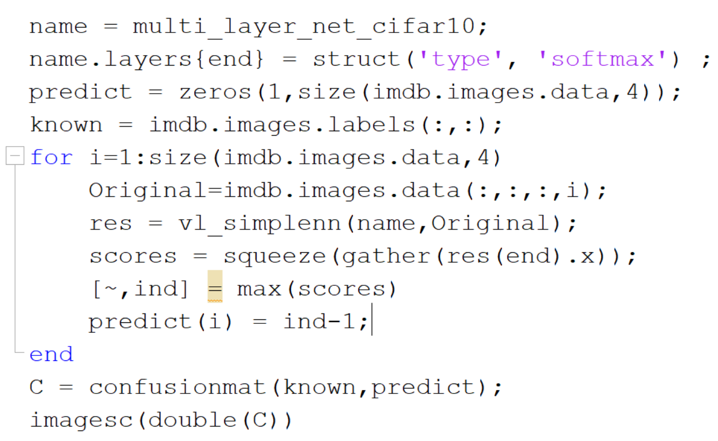
The second Conv layer had 5*5*20*50 weights.

The FC500 layer had 5*5*50*500 weights.

The FC10 layer had 500*10 weights.

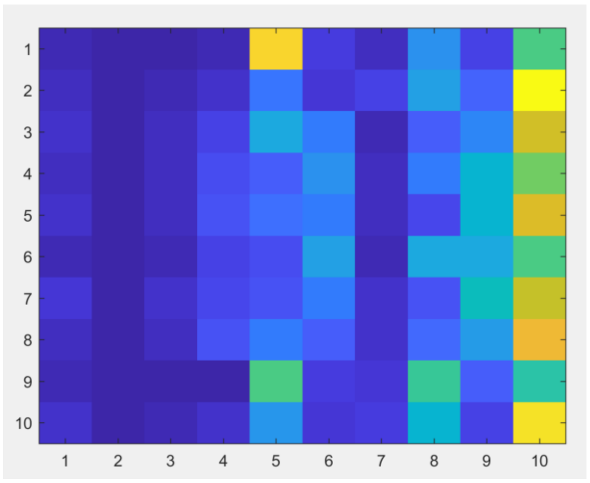
So, a total of = 5*5*3*20+ 5*5*20*50+5*5*50*500+500*10= 656500.

e) The code I used is similar to one before, it is as shown below:

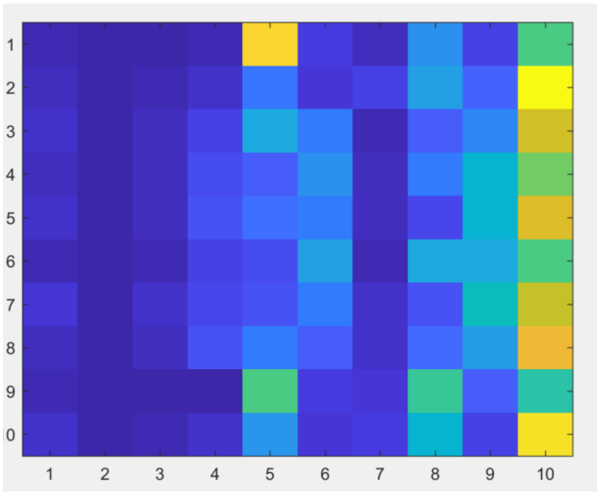


```
name = multi_layer_net_cifar10;
name.layers{end} = struct('type', 'softmax') ;
predict = zeros(1,size(imdb.images.data,4));
known = imdb.images.labels(:,:);
for i=1:size(imdb.images.data,4)
    Original=imdb.images.data(:,:,:,i);
    res = vl_simplenn(name,Original);
    scores = squeeze(gather(res(end).x));
    [~,ind] = max(scores)
    predict(i) = ind-1;
end
C = confusionmat(known,predict);
imagesc(double(C))
```

The confusion matrix I got is this, it signifies that the network classifies a lot of images as 10 and 9, and very few as the lower number. I think the problem is that I trained the network with the last layer being ‘SoftmaxLoss’ and after which changed it to Softmax and did forward pass, as Softmaxloss was giving an “Assertion error”.. But this didn’t seem to be a problem in the last example. I am going to try and train the network on ‘Softmax’ and then see the result.

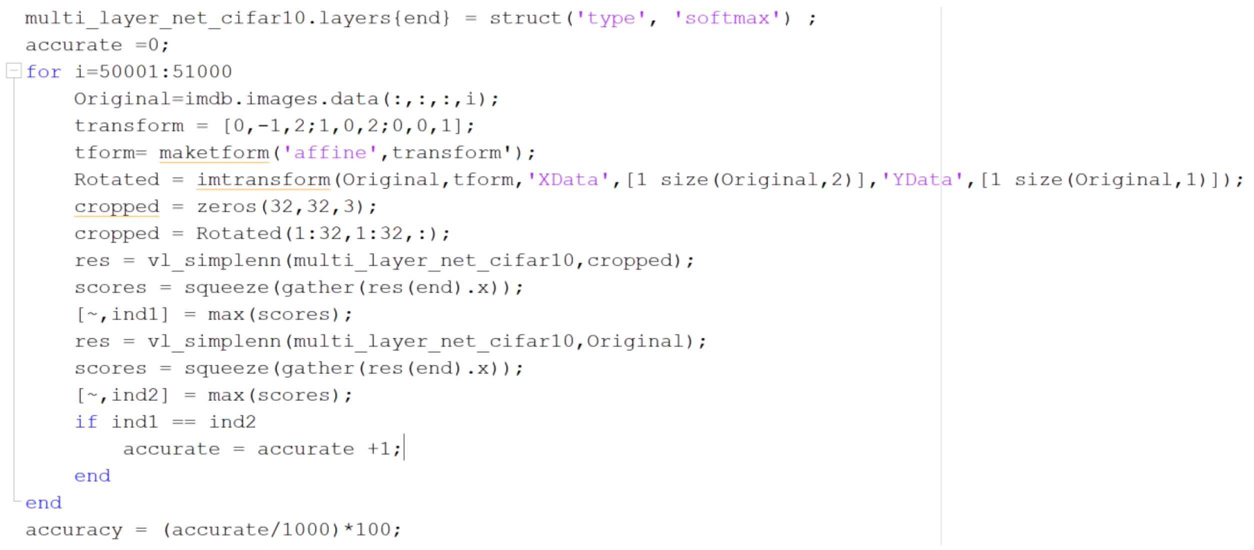


And it’s almost the same, I don’t know what to do I guess it is the indeed the correct confusion matrix.



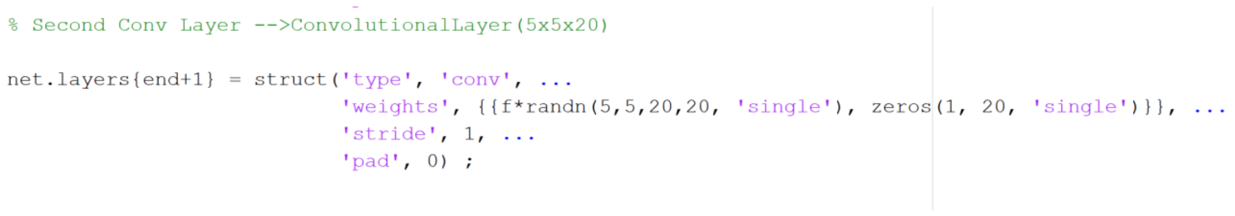
This confusion matrix has made me very confused.

f) For this part I took a 1000 images from the test data, as the images were randomly distributed I took images from 50001 to 51000, I then put them in a loop and applied a transformation on the image, got the prediction of the transformed image and compared it with the prediction of the original image. If they were same I would increase a counter and at the end get the accuracy by dividing it by 1000 and multiplying by hundred. I did this for 3 different transformations, for my first sample I only rotated the image at an angle of 90 degrees, and the accuracy came out to be 33.1%. I then did it with a pure translation of 2, it gave me 54.6% accuracy. After which I did it with a transformation of rotation of 90 degrees and translation of 2 as shown in the code below and got an accuracy of 32.9%. Hence it can be seen that the output of the network does indeed change with different transformations.



```
multi_layer_net_cifar10.layers{end} = struct('type', 'softmax') ;
accurate =0;
for i=50001:51000
    Original=imdb.images.data(:,:,:,i);
    transform = [0,-1,2;1,0,2;0,0,1];
    tform= maketform('affine',transform');
    Rotated = imtransform(Original,tform,'XData',[1 size(Original,2)],'YData',[1 size(Original,1)]);
    cropped = zeros(32,32,3);
    cropped = Rotated(1:32,1:32,:);
    res = vl_simplenn(multi_layer_net_cifar10,cropped);
    scores = squeeze(gather(res(end).x));
    [~,ind1] = max(scores);
    res = vl_simplenn(multi_layer_net_cifar10,Original);
    scores = squeeze(gather(res(end).x));
    [~,ind2] = max(scores);
    if ind1 == ind2
        accurate = accurate +1;
    end
end
accuracy = (accurate/1000)*100;
```

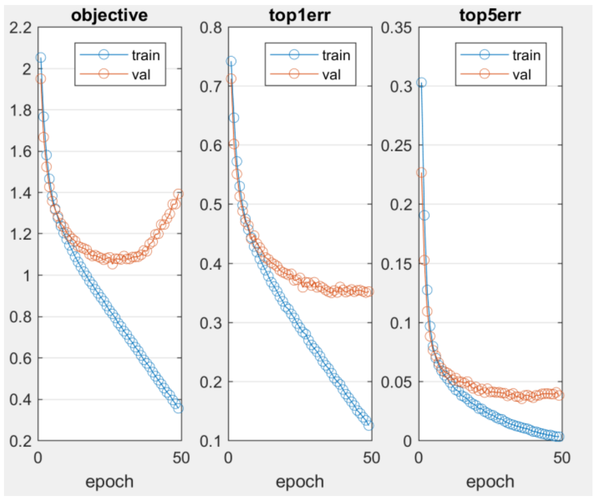
g) I made the new network by just adding a new layer after the first layer and changing layers accordingly. The code for the new layer is as shown below:



```
% Second Conv Layer -->ConvolutionalLayer(5x5x20)

net.layers{end+1} = struct('type', 'conv', ...
                           'weights', {{f*randn(5,5,20,20, 'single'), zeros(1, 20, 'single')}}, ...
                           'stride', 1, ...
                           'pad', 0) ;
```

I trained the neural network for 50 epochs as it was training really slowly and it took about 45 mins to end. I got the following result:

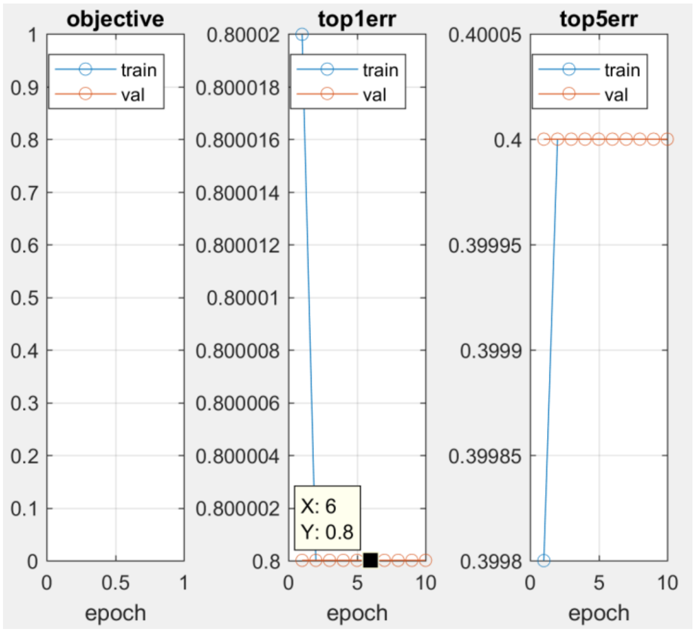


The accuracy was 64.19% after 50 epochs as compared to the last network's 68.72% hence this network performed poorly. It took a lot more time than the previous network as well as a layer was added and at the end of the day the architecture was not worth it.

Testing Learning Rates:

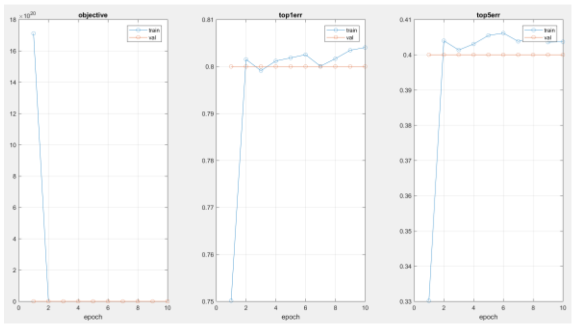
I tested 3 learning rates from 1 to 1.0xe^{-6} to get an idea on how it effects the performance, as it is slower I chose less number of testing rates.

Learning Rate=1:



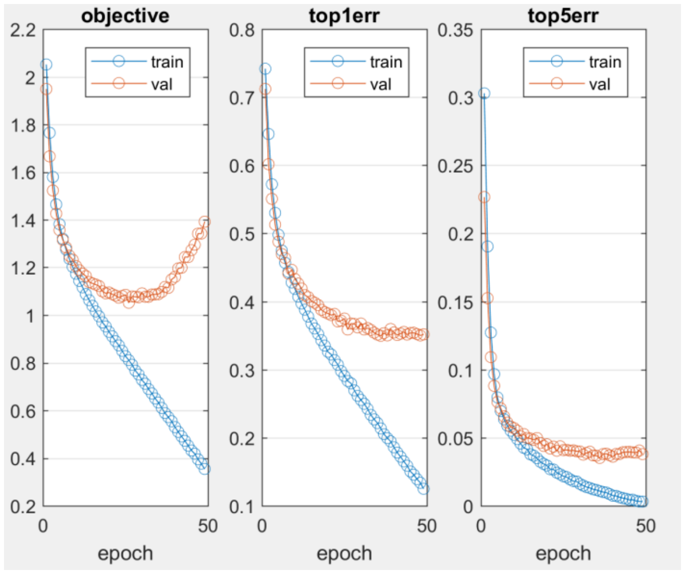
The learning rate is showing similar results to what the learning rate of 0.1 was showing in the last network. It is giving the same trend of not learning as it's getting stuck in the loop and not going up or down significantly, so it seems stagnated. Also both the errors are the same in this example.

Learning Rate = 0.01:



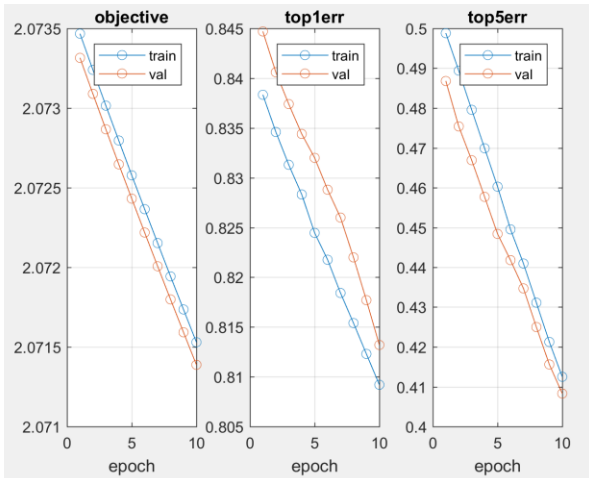
This is quite similar to learning rate of 1 as it has not really learnt anything as can be seen by the val plot being straight.

Learning Rate = 0.0001(default):



I am going to use the one from 50 epoch. It can be seen that it is actually converging to lower error although it takes a bit of time but it’s still working unlike the others, also it is definitely going to be faster than the lower learning rates, so it’s the perfect mix.

Learning Rate = 0.000001:



Although with this learning rate the network converges pretty nicely but it's very slow as it only got to 0.81 error after 10 epochs which is really slow hence is reliable but slow.

Breakdown of weights:

The first Conv layer had 5*5*3*20 weights.

The second Conv layer had 5*5*20*20 weights.

The third Conv layer had 5*5*20*50 weights.

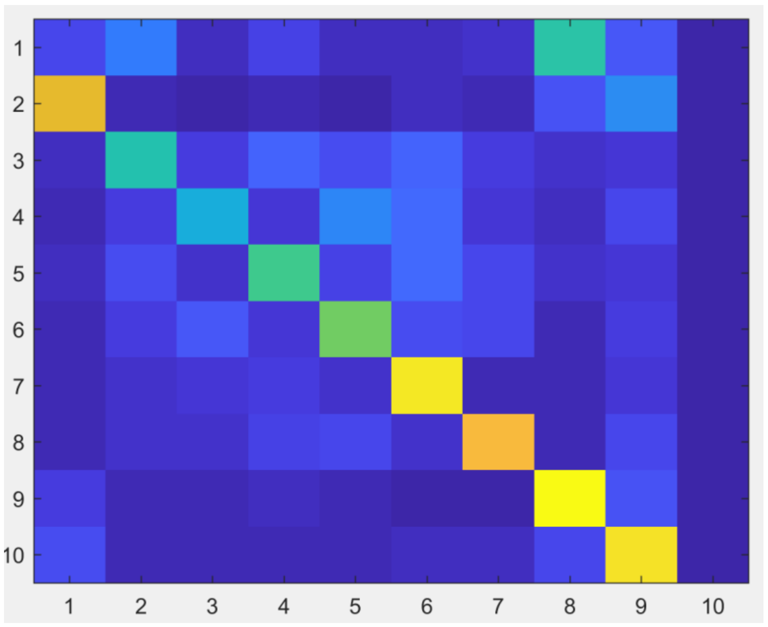
The FC500 layer had 4*4*50*500 weights.

The FC10 layer had 500*10 weights.

So, a total of = 5*5*3*20+ 5*5*20*20+ 5*5*20*50+4*4*50*500+500*10= 441500.

Confusion Matrix:

I used the same code as before and got this confusion matrix as shown below:



This shows that the network is never classifying any image as class 10, also it is for some reason mostly predicting the class before it, which shows very low top5 error but high top1 error. This confusion matrix has indeed again made me very confused.