Name: Remaldeep Singh
UID: u1143744
Email: u1143744@utah.edu

The main.m file in the project directory is used to call the functions for different parts of this project. The main method starts by setting the setting the environment for the matconvnet library by calling the "vl_setupnn.m" method.

Experiments of MNIST data

- *Step1.1* contains the code for training and testing on the mnist dataset. From the main.m file, cnn_mnist.m method is called. In the cnn_mnist method, the output directory "expDir" and the number of GPUs are set in the opts structure. Inside the cnn_mnist method, the neural network is initialized with the cnn_mnist_init method and the MNIST dataset is made in the form of imdb matlab structure. The trained network and the image mean is returned from the cnn_mnist method.

  Inside the cnn_mnist_init method, the random number generator is set to default 0 because of the pseudo-random number generation around the mean. The neural network has 8 layers in their configuration shown below:

layer	0	1	2	3	4	5	6	7	8
type	input	conv	mpool	conv	mpool	conv	relu	conv	softmxl
name	n/a	layer1	layer2	layer3	layer4	layer5	layer6	layer7	layer8
support	n/a	5	2	5	2	4	1	1	1
filt dim	n/a	1	n/a	20	n/a	50	n/a	500	n/a
filt dilat	n/a	1	n/a	1	n/a	1	n/a	1	n/a
num filts	n/a	20	n/a	50	n/a	500	n/a	10	n/a
stride	n/a	1	2	1	2	1	1	1	1
pad	n/a	0	0	0	0	0	0	0	0
rf size	n/a	5	6	14	16	28	28	28	28
rf offset	n/a	3	3.5	7.5	8.5	14.5	14.5	14.5	14.5
rf stride	n/a	1	2	2	4	4	4	4	4
data size	28	24	12	8	4	1	1	1	1
data depth	1	20	20	50	50	500	500	10	1
data num	100	100	100	100	100	100	100	100	1
data mem	306KB	4MB	1MB	1MB	312KB	195KB	195KB	4KB	4B
param mem	n/a	2KB	0B	98KB	0B	2MB	0B	20KB	0B

  The learning rate is set to 0.001 and the number of epochs to 20 with a batch size of 100. After visualizing the feature map of the first layer it was clear that the neural network will capture low-level details first and consequently go on to capture high-level details later in the network. Hence after the first layer, I did the pooling with a stride of 2 to reduce the number of pixels and effectively encode the information to a higher level. Consequently, I applied pooling once again after the second convolution to get another high-level representation. Then there is a convolution layer followed by relu layer. Intuitively I applied relu layer to have a bit more non linearity in the

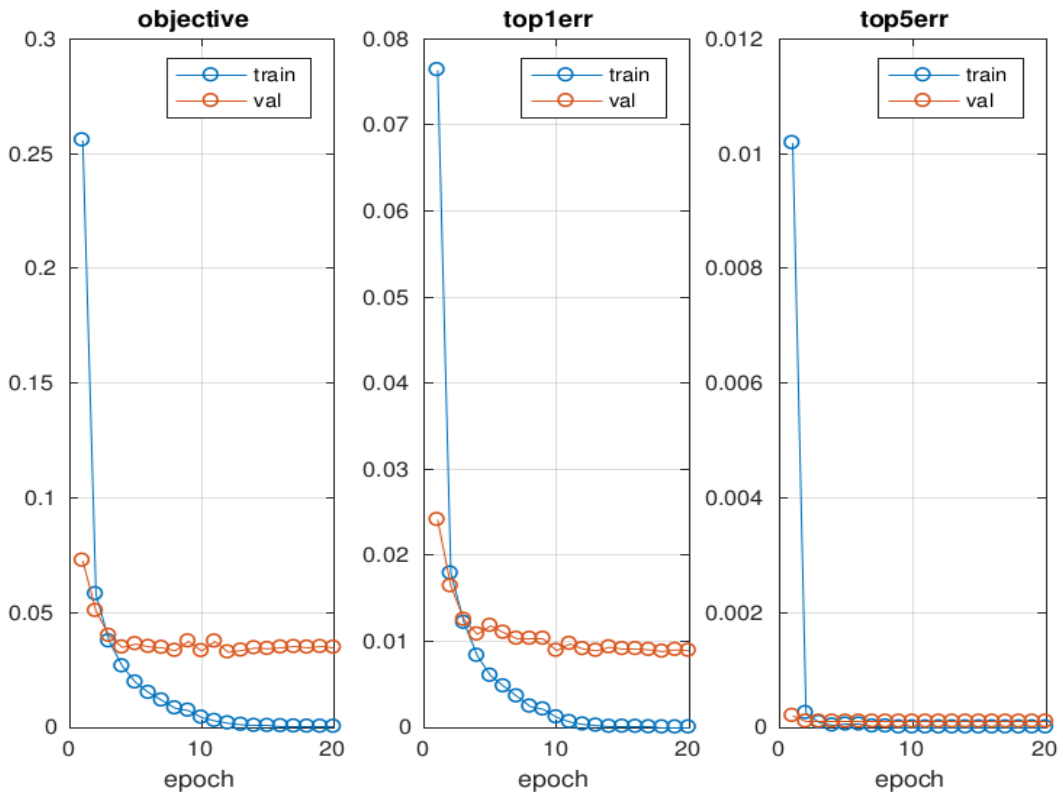
network. And then the last convolution layer is the fully connected layer followed by softmaxloss layer.

The neural network is trained using the cnn_train() method form the matconvnet library. It uses batching, momentum and gradient descent internally for convergence and then produces a graph for the training and validation error at the end. The trained model is then stored in the "Outputs/model.mat" file and then reused after re-loading for recognizing zip codes in Step 2.1.

getNNBactch is the method used for taking random batched samples with replacement from the imdb data structure.

getMNISTData is the method used for making the Matlab imdb structure of the MNIST images. If you pass 1 to this function then it converts all the even labels to 2 and the odd labels to 1. This is used in **Step 1.2** later for recognizing the odd-even digits. This method loads the files MNIST images from the "Inputs" directory and makes 4-D imdb structure out of it. With each image in the form of "28x28x1" and the fourth dimension with the number of images (70000). So the total dimension is 28x28x1x70000. Also as a step for preprocessing the data mean is subtracted from each image. There are 3 sets made 'train', 'val' and 'test', although the train and val test are used to learn hyperparameter they are basically the same. In total there are 10 classes or 2 classes depending upon if you selected 10 digit classification or odd/even classification.

The final results for the neural network on MNIST database are:



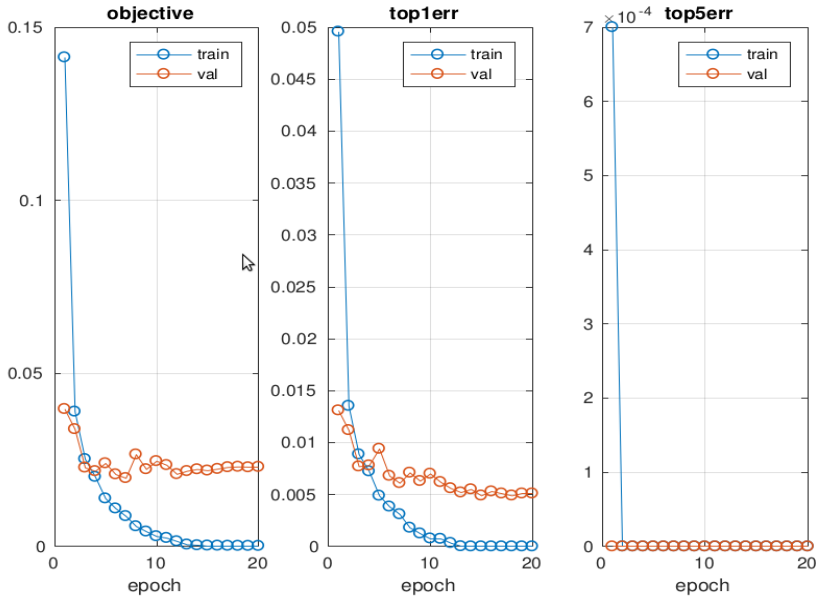
Ignore the top1error and top5error. They are just a graph for the top 1 and top 5 errors in the

epoch. The main graph is the leftmost which show how the training and validation performed for each epoch. From the graph, it can be seen that the validation error became stagnant after 4-5 epochs and after although the training error kept on decreasing. I am guessing the model did overfit in the end.

- *Step1.2* calls the cnn_mnist_odd_even method. The method cnn_mnist_odd_even is basically same as the cnn_mnist except the fact that it recognized odd and even digits only. The output model is in the directory "Outputs/oddEvenModel/". The second last layer of the previous mnist network was modified to contain only 2 nodes. Node 1 is for odd and node 2 is for even. Also from the cnn_mnist_odd_even method, the getMNISTData(1) method is called with 1 as a parameter for odd-even labeling. I explained this in the previous section for the getMNISTData method.
  Here is the final network for odd-even labeling:

layer	0	1	2	3	4	5	6	7	8
type	input	conv	mpool	conv	mpool	conv	relu	conv	softmxl
name	n/a	layer1	layer2	layer3	layer4	layer5	layer6	layer7	layer8
support	n/a	5	2	5	2	4	1	1	1
filt dim	n/a	1	n/a	20	n/a	50	n/a	500	n/a
filt dilat	n/a	1	n/a	1	n/a	1	n/a	1	n/a
num filts	n/a	20	n/a	50	n/a	500	n/a	2	n/a
stride	n/a	1	2	1	2	1	1	1	1
pad	n/a	0	0	0	0	0	0	0	0
rf size	n/a	5	6	14	16	28	28	28	28
rf offset	n/a	3	3.5	7.5	8.5	14.5	14.5	14.5	14.5
rf stride	n/a	1	2	2	4	4	4	4	4
data size	28	24	12	8	4	1	1	1	1
data depth	1	20	20	50	50	500	500	2	1
data num	100	100	100	100	100	100	100	100	1
data mem	306KB	4MB	1MB	1MB	312KB	195KB	195KB	800B	4B
param mem	n/a	2KB	0B	98KB	0B	2MB	0B	4KB	0B

The final graph for odd-even training is:



As you can see the training and the test error are very low.

Zip Code recognition system

This part of the project is under Step 2.1 of the main.m file. recognizeZipCodes() method is called with the trained neural network for MNIST database as the input. Although before passing the neural network I replaced the last softmaxloss layer with the softmax layer since we are no longer doing back propogation and forward pass only.

This method has a for loop that reads two files from the "Input/ ZipCodes-Test" folder iteratively. Each image goes through a preprocessing step with the help of the function preprocessImage(). Inside the preprocessImage method, each method is first converted to grayscale and then thresholded using **otsu threshold**. Then connected component labeling is applied to the thresholded image, followed by **bounded boxes** for the regions. This should give us around 5 regions per image. For each of these regions first, the area is checked and if it is less that 50 those regions are discarded. Consequently, these regions are cropped out of the image and returned from the function. The final regions are plotted and stored in the Outputs directory as "image_number.png". There was a minor error I found that the code was recognizing and additional noisy region that's why I had to put in area 50 constraint. It is shown in **Wrong_Digits.png**

Each of the returned images was rescaled to 28x228 and then passed through the previously trained neural network to find out the label for the image. Vl_simplennn() method is used for feed forward prediction. After all the sub-images are processed the final number is printed on the screen.

Some of the outputs for this system:

84882 for 

87802 for 

I could notice that the outputs are not perfect as compared to a testing set of MNIST Data. I guess the network did overfit the dataset and is not performing well on real-time data.

Experiment on CIFAR dataset

This experiment is under step 3.1 of the main.m file. Main.m calls the cnn_cifar() method for this experiment. I will be using the lenet structure and modify it for this experiment. CIFAR dataset has 60000 examples and is divided into 10 classes. The export directory for the network and the final graph is "**Outputs/cifarModel**".

cnn_cifar_init() method is called to initialize the neural network. This neural network has 13 layers with the following structure:

layer	0	1	2	3	4	5	6	7	8	9	10	11	12	13
type	input	conv	mpool	relu	conv	relu	mpool	conv	relu	mpool	conv	relu	conv	softmxl
name	n/a	layer1	layer2	layer3	layer4	layer5	layer6	layer7	layer8	layer9	layer10	layer11	layer12	layer13
support	n/a	5	3	1	5	1	3	5	1	3	4	1	1	1
filt dim	n/a	3	n/a	n/a	32	n/a	n/a	32	n/a	n/a	64	n/a	64	n/a
filt dilat	n/a	1	n/a	n/a	1	n/a	n/a	1	n/a	n/a	1	n/a	1	n/a
num filts	n/a	32	n/a	n/a	32	n/a	n/a	64	n/a	n/a	64	n/a	10	n/a
stride	n/a	1	2	1	1	1	2	1	1	2	1	1	1	1
pad	n/a	2	0x1x0x1	0	2	0	0x1x0x1	2	0	0x1x0x1	0	0	0	0
rf size	n/a	5	7	7	15	15	19	35	35	43	67	67	67	67
rf offset	n/a	1	2	2	2	2	4	4	4	8	20	20	20	20
rf stride	n/a	1	2	2	2	2	4	4	4	8	8	8	8	8
data size	32	32	16	16	16	16	8	8	8	4	1	1	1	1
data depth	3	32	32	32	32	32	32	64	64	64	64	64	10	1
data num	100	100	100	100	100	100	100	100	100	100	100	100	100	1
data mem	1MB	12MB	3MB	3MB	3MB	3MB	800KB	2MB	2MB	400KB	25KB	25KB	4KB	4B
param mem	n/a	10KB	0B	0B	100KB	0B	0B	200KB	0B	0B	256KB	0B	3KB	0B

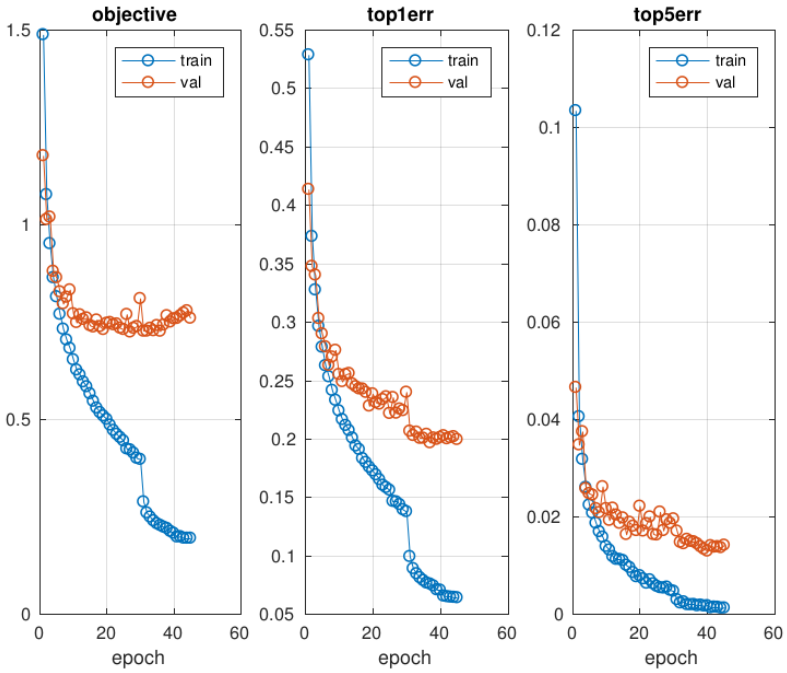
The imdb database is created with the getCIFARData method. The getCIFARData method takes 5 input files from the directory ‘Inputs/cifar-10-batches-mat/’. The method subtracts the average mean from each image and does contrast normalization. File ‘Inputs/ cifar-10-batches-mat/batches.meta.mat’ has the information about which label number corresponds to which category. The imdb structure made contains all the data i.e 32x32x1x60000 images and the meta information as well for each label. The data is divide into 3 sets ‘train’, ‘val’ and ‘test’ with train and val being the same sets used for cross-validation internally. The test set is the final evaluation set. The test set has only 10000 images while training has 50000.

The final training is done with the cnn_train method and it takes the input as a neural network, batched data and export directory among others. The getBatch() method provides the random with replacement batch data to the trainer.

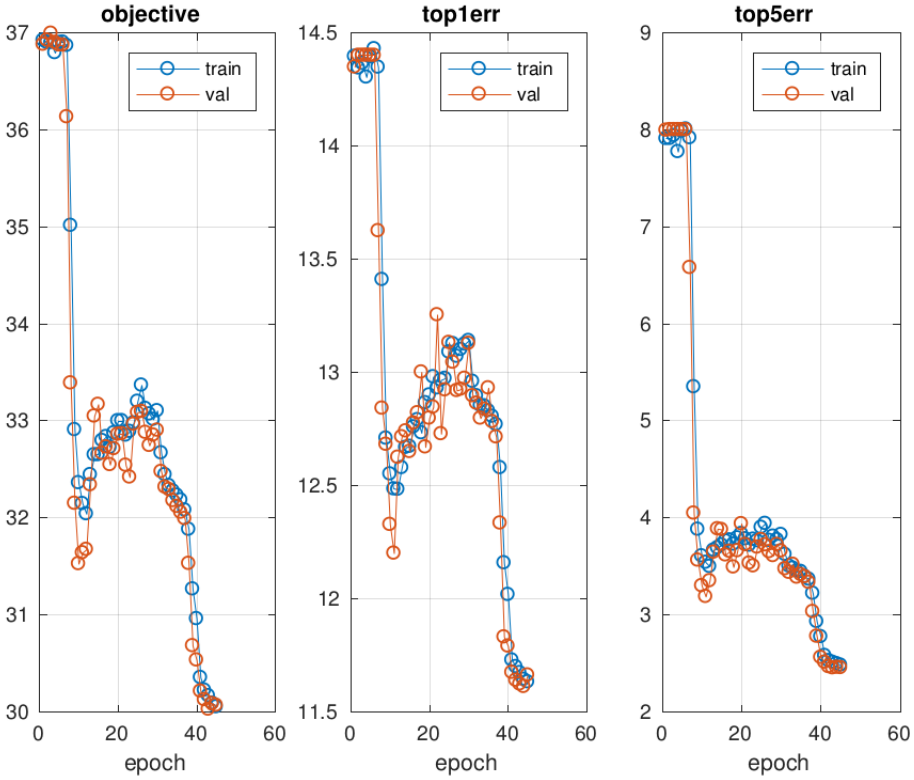
Experiments are on next page.

I tried various experiments on the CIFAR data set. The figures are shown below:

1. The model with default parameters shown in the previous neural network figure.

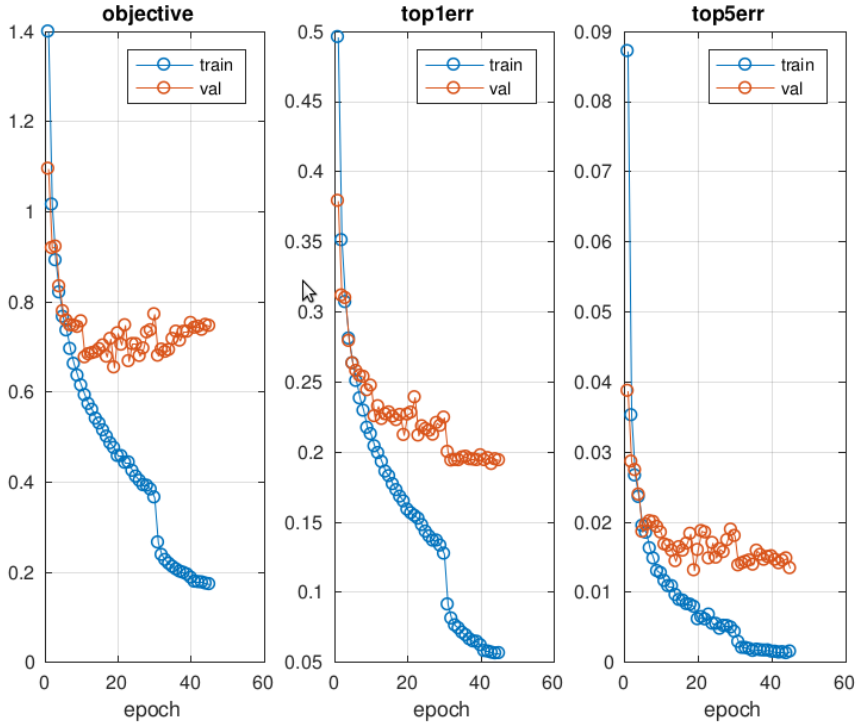


2. The model with one less convolution layer that was the layer number 7.



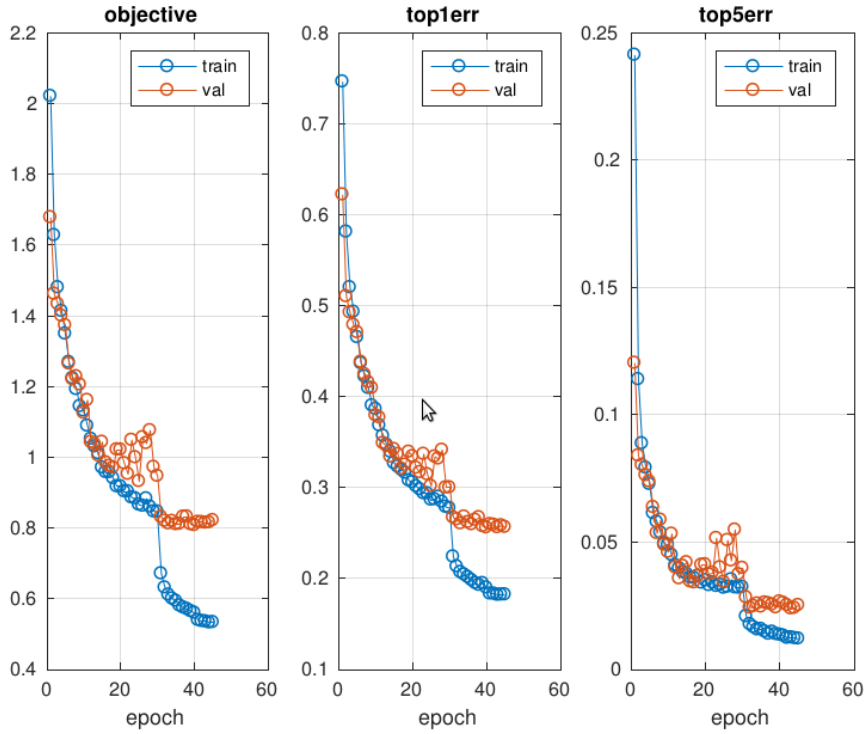
The error on training and validation set is pretty high because my neural network is not expressive enough. It doesn't have the right variance to do the classification.

3. The model with max pooling instead of average pooling:



The difference between this model and the first model is negligible even after using max pooling instead of average pooling.

4. The model with the same network as the default one but with learning rates of [0.1 2].



In this model due to higher learning rate, the model did not converge to the minima fast enough. Maybe with more number of iterations, it could have converged to better minima.

Resources used:

1) http://www.vlfeat.org/matconvnet/training/
2) https://www.mathworks.com/help/matlab/matlab_prog/pass-a-function-to-another-function.html
3) https://stackoverflow.com/questions/34644779/how-to-use-the-network-trained-using-cnn-mnist-example-in-matconvnet
4) https://www.cc.gatech.edu/~hays/compvision/proj6/
5) https://groups.google.com/forum/#!topic/matconvnet/_58_MURJD0Q
6) https://www.mathworks.com/matlabcentral/fileexchange/47811-vlfeat-matconvnet