CSE583 Project 4: Training a Convolutional Neural Network

Sree Sai Teja Lanka

March 31, 2019

Abstract

The goal of the projec is to train a Convolutional Neural Network (CNN) on the wallpaper images consisting of images which are 256x256 and 1 channel (grayscale). The main purpose of this project is to get familiar with the inbuilt CNN training functions (because of their ease) from the Matlab and then further learn how to augment the data, build a CNN, and train on the data.

Contents

1 Introduction

When it comes to Machine Learning, Artificial Neural Networks perform really well. Artificial Neural Networks are used in various classification task like image, audio, words. Different types of Neural Networks are used for different purposes, for example for predicting the sequence of words we use Recurrent Neural Networks more precisely an LSTM, similarly for image classification we use Convolution Neural Network. In this blog, we are going to build basic building block for CNN.

Before diving into the Convolution Neural Network, let us first revisit some concepts of Neural Network. In a regular Neural Network there are three types of layers:

Input Layers: Its the layer in which we give input to our model. The number of neurons in this layer is equal to total number of features in our data (number of pixels incase of an image).

Hidden Layer: The input from Input layer is then feed into the hidden layer. There can be many hidden layers depending upon our model and data size. Each hidden layers can have different numbers of neurons which are generally greater than the number of features. The output from each layer is computed by matrix multiplication of output of the previous layer with learnable weights of that layer and then by addition of learnable biases followed by activation function which makes the network nonlinear.

Output Layer: The output from the hidden layer is then fed into a logistic function like sigmoid or softmax which converts the output of each class into probability score of each class. The data is then fed into the model and output from each layer is obtained this step is called feedforward, we then calculate the error using an error function, some common error functions are cross entropy, square loss error etc. After that, we backpropagate into the model by calculating the derivatives. This step is called Backpropagation which basically is used to minimize the loss.

Now, CNN-Convolution Neural Networks or covnets are neural networks that share their parameters. Imagine you have an image. It can be represented as a cuboid having its length, width (dimension of the image) and height (as image generally have red, green, and blue channels).

Now imagine taking a small patch of this image and running a small neural network on it, with say, k outputs and represent them vertically. Now slide that neural network across the whole image, as a result, we will get another image with different width, height, and depth. Instead of just R, G and B channels now we have more channels but lesser width and height. his operation is called Convolution. If patch size is same as that of the image it will be a regular neural network. Because of this small patch, we have fewer weights.

Now lets talk about a bit of mathematics which is involved in the whole convolution process.

Convolution layers consist of a set of learnable filters (patch in the above image). Every filter has small width and height and the same depth as that of input volume (3 if the input layer is image input). For example, if we have to run convolution on an image with dimension 34x34x3. Possible size of filters can be axax3, where a can be 3, 5, 7, etc but small as compared to image dimension. During forward pass, we slide each filter across the whole input volume step by step where each step is called stride (which can have value 2 or 3 or even 4 for high dimensional images) and compute the dot product

between the weights of filters and patch from input volume. As we slide our filters well get a 2-D output for each filter and well stack them together and as a result, well get output volume having a depth equal to the number of filters. The network will learn all the filters. Layers used to build ConvNets

A covnets is a sequence of layers, and every layer transforms one volume to another through differentiable function. Types of layers: Lets take an example by running a covnets on of image of dimension 32 x 32 x 3.

Input Layer: This layer holds the raw input of image with width 32, height 32 and depth 3. **Convolution Layer:** This layer computes the output volume by computing dot product between all filters and image patch. Suppose we use total 12 filters for this layer well get output volume of dimension 32 x 32 x 12. Activation Function Layer: This layer will apply element wise activation function to the output of convolution layer. Some common activation functions are $RELU : max(0, x), Sigmoid : 1/(1 + e^{-}x), Tanh, LeakyRELU, etc.$ The volume remains unchanged hence output volume will have dimension 32 x 32 x 12. **Pool Layer:** This layer is periodically inserted in the covnets and its main function is to reduce the size of volume which makes the computation fast reduces memory and also prevents from overfitting. Two common types of pooling layers are max pooling and average pooling. If we use a max pool with 2 x 2 filters and stride 2, the resultant volume will be of dimension 16x16x12. **Fully-Connected Layer:** This layer is regular neural network layer which takes input from the previous layer and computes the class scores and outputs the 1-D array of size equal to the number of classes. This data is referenced from [1] and [2].

2 Approach

2.1 Data

Wallpaper dataset: The dataset consists of 17,000 images, 1,000 images per group, and each image containing a wallpaper pattern. The images are 256x256 and 1 channel (grayscale). You will be training networks to discover these patterns. The datasets are located in the "data/wallpapers/¡train,test¿/¡group¿/" folders. The train and test images are in separate folders and within them, there are folders for each wallpaper group's images.
Data Augmentation: As per the project description all the images in train and test folders within the groups should be augmented and transformed to into formats of scale, reflect, rotate, translate for an original image. If we perform these we get 4 new images for a single image and each group has $1000x5 = 5000$ images. So we have total images as $17x5000 = 85000$ images in the trainaug folder same thing happens with the test aug folder with 85000 images.
The **The visualisations of random 5 example images augmentation is shown below:**

2.2 Network Implementation

2.2.1 Without data augmentation

This step is to get you familiar with the CNN interface and to train your first network. The starter code comes complete with an example of a convolutional neural network. The data is not augmented so this data is very separable (as seen by your last project). You can adjust the setting if it doesn't run on your computer (such as decrease the batch size if your GPU doesn't have enough memory for the images). This network should train quickly (getting above 10 % accuracy after the second epoch). You should train this for at least 10 epochs (though you are free to train it more if you like).

2.2.2 With data augmentation

Based on the given description the original train and test data is now augmented using the four scopes: scaling, rotating, reflecting, translating.
The explanation for the augmentation based on these is as below:
Scaling:
1. Scaling an image is zooming in or out the original image.
2. As we are asked for less than 100 till 50 percentage, it says that it means zoom out the image.
3. Now, resize or crop to 128x128size based on the convenience.
Rotating:
1. Rotate the image based on random angle from 1 to 359 which results in black background and increases size of original image as there is extra part of black for each image.
2. Now I have cropped the image from center for image size 128x128 using the rectangle minimum and maximum dimensions.
Reflecting:
1. Reflection is nothing but the mirroring of the image.
2. I have reflected the image in horizontal direction first and then I again reflected in vertical direction.// 3. After that I have resized the image to 128x128 size.
Translating: 1. Translation of the image means that the original image is now moved from place with x and y axis.
2. I have used randomised scaling for images.
3. Then I have cropped the scaled image for removing the black background.
4. In the end I have resized the image to 128x128

2.2.3 Skinny Network

A skinny network has more layers and but not many filters at each layer. This should at least be the length of the example at the bottom of the starter code.
I have used 15 epochs with a bath size of 400 per iteration.
The layers are based on the sequence described in project description. I have used the following layer sequence.

$$INPUT \rightarrow [CONV \rightarrow RELU \rightarrow POOL] * 5 \rightarrow FC \rightarrow RELU \rightarrow FC \rightarrow Softmax$$

The filter size is 5x5 and 3x3 based on the requirement with number of filters to be 20 and 10 respectively. With 25 and 17 fully connected layers and the dropout to be 0.25.

2.2.4 Wide Network

A wide network has fewer layers but many filters. You should at least double the number of filters in the starter code network. You can also make it longer as well, just make the skinny network even longer if you do.
I have used 20 epochs with a bath size of 400 per iteration.

$$INPUT \rightarrow [CONV \rightarrow RELU \rightarrow POOL] \rightarrow FC \rightarrow RELU \rightarrow FC \rightarrow Softmax$$

With the filter size equal to 5x5 and number of filters equal to 100, with 25 and 17 fully connected layers and the dropout to be 0.25.

3 Results

Based on the data augmentation results the visualisation of loss and accuracy leads to the bad skinny and a better wide network. But the un-augmented data has a very good accuracy the results are best visualised in the below sections for different tests and the networks based on statistics and visualisations.
While I perform the networking trains I have visualised that the main network which is given as an example stays as a good task for the original data. But when the data is augmented the results changed.
The augmented data results in less accuracy than the original data network does. As the loss learning for network for high learning rate doesn't change more the model doesn't have good accuracy.
Among the both skinny and wide the wide results in better results compared to the skinny. The results are not as good as for the original data but they reach a 70 percentage which is far better than the skinny accuracy.
Finally, comparison on skinny and wide with Alex network showed a drastic change in the networking results which are far better than wide and are not a way equal to original data network but the accuracy reaches to at atleast 85 percentage which is better.
The comparison ith ranking based on accurcy of test for a network is as follows:
rank1: AlexNet with original data
rank2: given exmple network with original data
rank3 : AlexNet with augmented data
rank4: Wide network with augmented data
rank5: Skinny network with augmented data

3.1 Augmented data

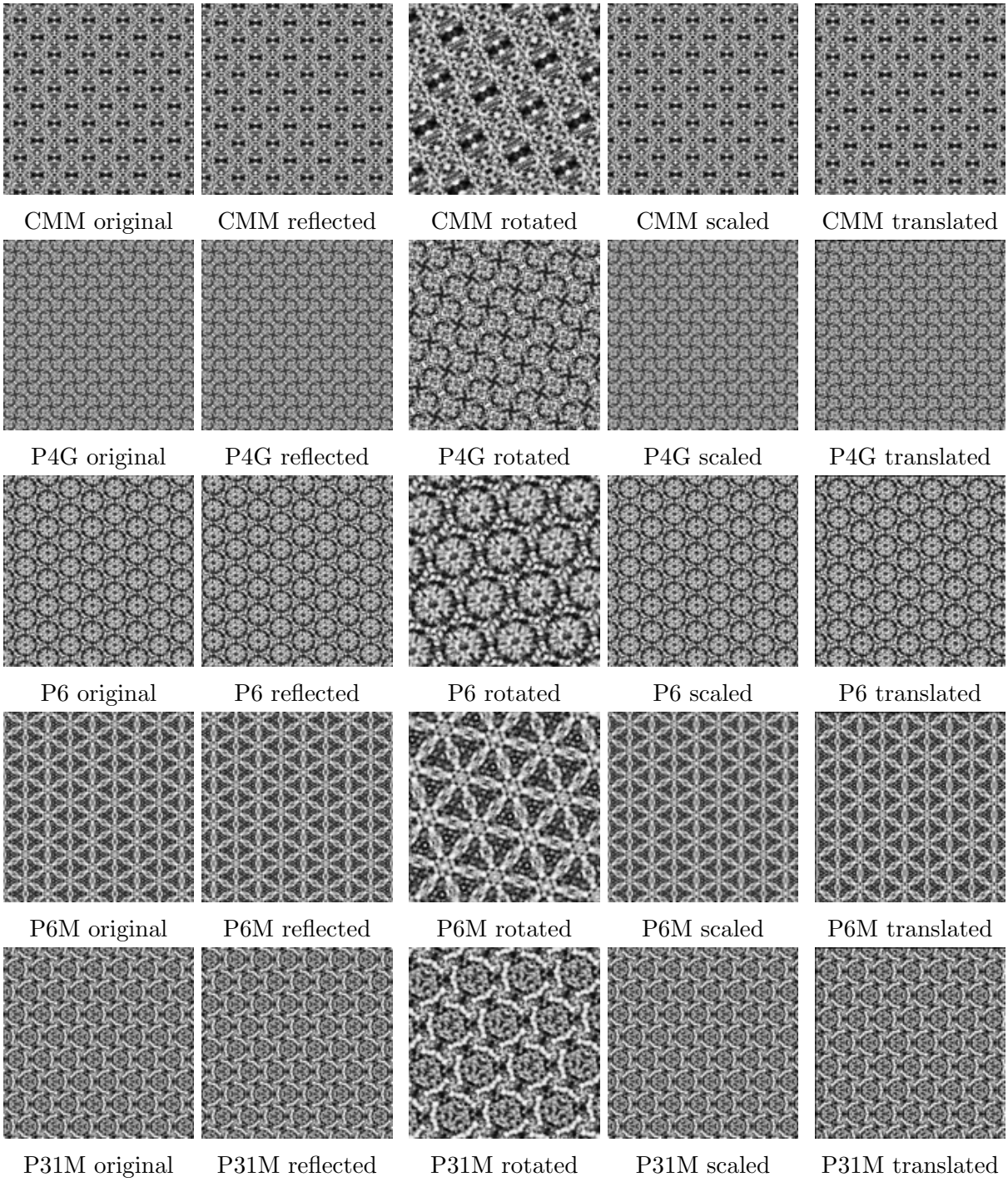


Figure 1: Augmented Visualisations of 5 random images each from 5 groups CMM, P4G, P6, P6M, P31M

3.2 Main Example Network

The given code involves the example network which holds the training for 256x256 images (original dat).
The layers and filters of the network are as below:

$$INPUT \rightarrow [CONV \rightarrow RELU \rightarrow POOL] \rightarrow FC \rightarrow RELU \rightarrow FC \rightarrow Softmax$$

The number of filters are 20 and the filter size is 5x5 with the dropout rate of 0.25.

3.2.1 Statistics

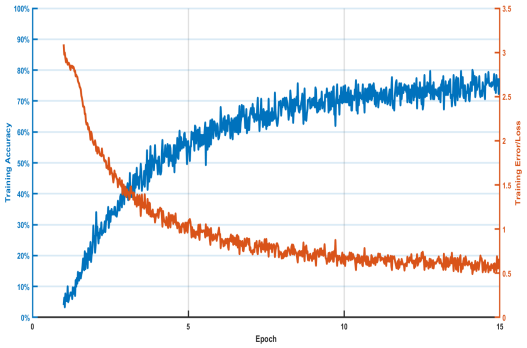
Epoch	Iteration	Time Elapsed	Mini-batch Loss	Mini-batch Accuracy	Base Learning Rate
1	1	1.50	3.0914	4.00%	0.0005
1	50	14.96	2.0818	20.33%	0.0005
2	100	29.21	1.5050	39.00%	0.0005
3	150	43.22	1.1676	50.67%	0.0005
4	200	57.28	1.0109	58.00%	0.0005
5	250	71.25	1.0517	49.33%	0.0005
6	300	85.23	0.8330	64.33%	0.0005
7	350	99.41	0.8239	61.67%	0.0005
8	400	113.20	0.8011	65.33%	0.0005
9	450	127.17	0.7764	67.33%	0.0005
10	500	141.11	0.6435	74.33%	0.0005
11	550	155.82	0.6254	72.00%	0.0005
12	600	169.67	0.5911	75.00%	0.0005
13	650	183.41	0.6148	70.67%	0.0005
14	700	197.11	0.5521	77.67%	0.0005
15	750	210.84	0.5996	74.67%	0.0005
15	765	214.89	0.5789	77.33%	0.0005

Example network statistics on non augmented data for high learning rate 0.0005 for 15 epochs

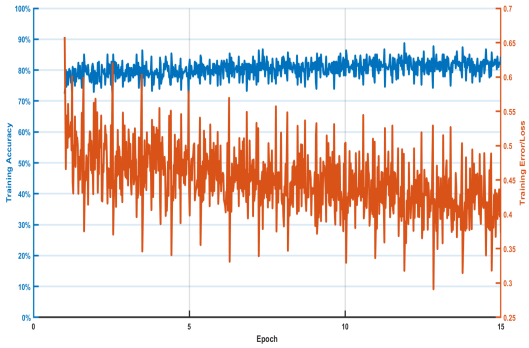
Epoch	Iteration	Time Elapsed	Mini-batch Loss	Mini-batch Accuracy	Base Learning Rate
1	1	0.41	0.6583	72.33%	1.00e-04
1	50	13.39	0.4677	82.00%	1.00e-04
2	100	26.91	0.4285	81.00%	1.00e-04
3	150	41.76	0.5004	78.67%	1.00e-04
4	200	55.41	0.4592	79.00%	1.00e-04
5	250	71.91	0.4882	81.00%	1.00e-04
6	300	85.75	0.4700	80.33%	1.00e-04
7	350	99.13	0.4233	84.33%	1.00e-04
8	400	112.73	0.4695	79.33%	1.00e-04
9	450	126.32	0.3904	85.00%	1.00e-04
10	500	139.73	0.4919	79.33%	1.00e-04
11	550	153.35	0.4293	81.00%	1.00e-04
12	600	166.60	0.4197	82.33%	1.00e-04
13	650	180.06	0.3884	82.33%	1.00e-04
14	700	193.59	0.3942	82.67%	1.00e-04
15	750	207.13	0.3759	82.00%	1.00e-04
15	765	211.06	0.3732	84.33%	1.00e-04

Example network statistics for high learning rate 1.00e-04 for 15 epochs

3.2.2 Visualisations



plot result for the high learning rate



Plot result for for less learning rate

Figure 2: Train accuracy vs loss plots for both lesser and higher learning rates

	Validation Accuracy	Test Accuracy	Execution time (seconds)
high learning rate	0.7300	not the best	257.92
low learning rate	0.7718	0.7678	224.12

Table 1: Results for the accuracy for the main example network

3.3 Wide Network

A wide network has fewer layers but many filters. You should at least double the number of filters in the starter code network. You can also make it longer as well, just make the skinny network even longer if you do.
I have used 20 epochs with a bath size of 400 per iteration.

$$INPUT \rightarrow [CONV \rightarrow RELU \rightarrow POOL] \rightarrow FC \rightarrow RELU \rightarrow FC \rightarrow Softmax$$

With the filter size equal to 5x5 and number of filters equal to 100, with 25 and 17 fully connected layers and the dropout to be 0.25.

	train accuracy	Validation Accuracy	Test Accuracy	Execution time
high learning rate	0.8119	0.4296	not the best	7432.41 secs
low learning rate	0.9062	0.4495	0.6297	7839.36 secs

Table 2: Results for the accuracy for the wide network

3.3.1 Statistics

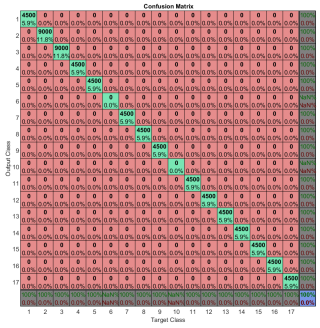
Epoch	Iteration	Time Elapsed	Mini-batch Loss	Mini-batch Accuracy	Base Learning Rate
1	1	3.11	2.9447	5.25%	0.0005
1	50	76.65	2.7677	9.25%	0.0005
1	100	169.21	2.6622	14.50%	0.0005
1	150	261.84	2.448	23.00%	0.0005
2	200	354.35	2.1208	31.00%	0.0005
2	250	447.39	1.9533	37.25%	0.0005
2	300	577.85	1.8616	36.75%	0.0005
2	350	664.58	1.6712	46.00%	0.0005
3	400	757.78	1.7619	42.00%	0.0005
3	450	846.66	1.7495	44.50%	0.0005
3	500	939.49	1.4996	51.25%	0.0005
3	550	1029.28	1.5883	45.50%	0.0005
4	600	1122.63	1.4621	50.25%	0.0005
4	650	1210.86	1.4254	49.50%	0.0005
4	700	1304.09	1.5128	50.00%	0.0005
4	750	1426.21	1.3304	51.50%	0.0005
5	800	1516.55	1.3289	55.75%	0.0005
5	850	1608.88	1.3835	54.50%	0.0005
5	900	1700.96	1.336	52.50%	0.0005
5	950	1790.55	1.1821	57.75%	0.0005
6	1000	1914.08	1.2744	52.25%	0.0005
6	1050	2005.81	1.4902	51.75%	0.0005
6	1100	2094.97	1.191	59.00%	0.0005
7	1150	2190.27	1.302	58.00%	0.0005
7	1200	2278.26	1.2378	58.50%	0.0005
7	1250	2391.13	1.2075	57.50%	0.0005
7	1300	2491.58	1.1968	58.75%	0.0005
8	1350	2581.26	1.172	57.25%	0.0005
8	1400	2672.6	1.1994	63.25%	0.0005
8	1450	2761.01	1.1979	58.75%	0.0005
8	1500	2852.76	0.9662	65.00%	0.0005
9	1550	2941.85	1.1524	60.50%	0.0005
9	1600	3032.97	1.0218	64.75%	0.0005
9	1650	3122.07	1.0727	58.75%	0.0005
9	1700	3214.01	0.9855	67.50%	0.0005
10	1750	3303.84	1.0814	65.75%	0.0005
10	1800	3395.24	1.1477	58.00%	0.0005
10	1850	3483.67	1.0771	64.25%	0.0005
10	1900	3575.02	1.0685	61.75%	0.0005
11	1950	3664.46	1.0486	63.00%	0.0005
11	2000	3757.9	0.8702	67.75%	0.0005
11	2050	3846.97	1.0778	65.25%	0.0005
11	2100	3958.8	0.8705	67.25%	0.0005
12	2150	4050.08	0.9433	66.25%	0.0005
12	2200	4139	0.9353	66.75%	0.0005
12	2250	4231.32	0.8931	68.50%	0.0005
13	2300	4320.38	0.9075	68.50%	0.0005
13	2350	4412.39	0.8114	73.25%	0.0005
13	2400	4501.26	0.9283	68.50%	0.0005
13	2450	4593.14	0.8446	70.75%	0.0005
14	2500	4684.99	0.8519	67.25%	0.0005
14	2550	4776.52	0.7989	70.75%	0.0005
14	2600	4865.17	0.8062	70.25%	0.0005
14	2650	4957.13	0.7703	70.50%	0.0005
15	2700	5046.29	0.7882	69.75%	0.0005
15	2750	5138.85	0.7685	71.25%	0.0005
15	2800	5227.36	0.8748	68.00%	0.0005
15	2850	5319.67	0.7151	72.25%	0.0005
16	2900	5407.85	0.8346	73.25%	0.0005
16	2950	5497.92	0.7944	70.50%	0.0005
16	3000	5587.04	0.7964	69.75%	0.0005
16	3050	5678.4	0.7924	70.25%	0.0005
17	3100	5791.57	0.916	68.50%	0.0005
17	3150	5880.22	0.7932	71.50%	0.0005
17	3200	5972.14	1.015	68.25%	0.0005
18	3250	6061.6	0.7002	73.25%	0.0005
18	3300	6155.26	0.7398	73.50%	0.0005
18	3350	6244.64	0.8975	68.50%	0.0005
18	3400	6336.46	0.7833	71.50%	0.0005
19	3450	6425.9	0.6556	76.75%	0.0005
19	3500	6518.5	0.6815	74.75%	0.0005
19	3550	6607.39	0.6441	75.75%	0.0005
19	3600	6700.76	0.7068	74.00%	0.0005
20	3650	6790.92	0.77	71.75%	0.0005
20	3700	6883.32	0.6512	74.25%	0.0005
20	3750	6971.7	0.586	80.50%	0.0005
20	3800	7063.71	0.7728	69.50%	0.0005
20	3820	7100.79	0.6728	75.50%	0.0005

Wide network statistics for high learning rate 0.0005

Epoch	Iteration	Time Elapsed	Mini-batch Loss	Mini-batch Accuracy	Base Learning Rate
1	1	4.2	0.6646	75.25%	1.00E-04
1	50	160.77	0.4257	83.50%	1.00E-04
1	100	267.5	0.5997	78.50%	1.00E-04
1	150	355.88	0.4225	83.00%	1.00E-04
2	200	446.45	0.5226	79.25%	1.00E-04
2	250	534.34	0.5187	75.50%	1.00E-04
2	300	625.72	0.5502	79.00%	1.00E-04
2	350	714.04	0.5181	81.00%	1.00E-04
3	400	807.36	0.538	80.50%	1.00E-04
3	450	895.51	0.5155	77.00%	1.00E-04
3	500	987.24	0.5034	81.25%	1.00E-04
3	550	1075.44	0.4903	80.25%	1.00E-04
4	600	1188.47	0.5236	78.00%	1.00E-04
4	650	1280.55	0.5374	80.00%	1.00E-04
4	700	1369.33	0.4986	82.50%	1.00E-04
4	750	1465.81	0.513	81.50%	1.00E-04
5	800	1555.46	0.5303	81.50%	1.00E-04
5	850	1647.62	0.5368	79.50%	1.00E-04
5	900	1737.05	0.4989	81.75%	1.00E-04
5	950	1829	0.468	81.50%	1.00E-04
6	1000	1918.46	0.4653	82.25%	1.00E-04
6	1050	2010.99	0.4648	80.25%	1.00E-04
6	1100	2099.61	0.4949	79.00%	1.00E-04
7	1150	2195.98	0.4225	85.25%	1.00E-04
7	1200	2287.98	0.4606	80.25%	1.00E-04
7	1250	2376.72	0.4606	81.00%	1.00E-04
7	1300	2469.55	0.4307	83.75%	1.00E-04
8	1350	2561.98	0.482	81.75%	1.00E-04
8	1400	2653.23	0.4937	81.00%	1.00E-04
8	1450	2740.56	0.458	80.50%	1.00E-04
8	1500	2832.75	0.435	83.50%	1.00E-04
9	1550	2939.08	0.4372	83.75%	1.00E-04
9	1600	3043.6	0.4388	85.25%	1.00E-04
9	1650	3136.42	0.4475	80.50%	1.00E-04
9	1700	3224.59	0.5025	82.50%	1.00E-04
10	1750	3317.69	0.4216	84.50%	1.00E-04
10	1800	3406.17	0.4499	81.75%	1.00E-04
10	1850	3529.65	0.5148	80.00%	1.00E-04
10	1900	3618.46	0.4452	82.50%	1.00E-04
11	1950	3707.69	0.4869	83.50%	1.00E-04
11	2000	3799.58	0.4592	81.75%	1.00E-04
11	2050	3891.34	0.4557	82.50%	1.00E-04
11	2100	3983.87	0.4149	83.75%	1.00E-04
12	2150	4107.91	0.4402	83.75%	1.00E-04
12	2200	4193.5	0.3655	85.50%	1.00E-04
12	2250	4282.8	0.4665	81.50%	1.00E-04
13	2300	4370.63	0.4649	80.75%	1.00E-04
13	2350	4461.17	0.4296	82.00%	1.00E-04
13	2400	4549.79	0.4771	82.00%	1.00E-04
13	2450	4641.98	0.3991	84.00%	1.00E-04
14	2500	4752.62	0.4192	83.50%	1.00E-04
14	2550	4840.84	0.4258	82.75%	1.00E-04
14	2600	4932.38	0.4376	82.25%	1.00E-04
14	2650	5020.71	0.4322	84.25%	1.00E-04
15	2700	5113.31	0.4306	81.50%	1.00E-04
15	2750	5201.11	0.4606	82.75%	1.00E-04
15	2800	5323.84	0.4046	85.50%	1.00E-04
15	2850	5411.55	0.4104	83.75%	1.00E-04
16	2900	5498.94	0.3866	85.50%	1.00E-04
16	2950	5589.32	0.4259	85.75%	1.00E-04
16	3000	5676.74	0.4552	81.75%	1.00E-04
16	3050	5768.22	0.4295	83.25%	1.00E-04
17	3100	5857.44	0.3972	84.50%	1.00E-04
17	3150	5949.01	0.4126	87.00%	1.00E-04
17	3200	6036.94	0.3795	83.50%	1.00E-04
18	3250	6160.34	0.3912	85.50%	1.00E-04
18	3300	6250.48	0.4192	82.00%	1.00E-04
18	3350	6338.85	0.3602	86.50%	1.00E-04
18	3400	6431.24	0.3341	87.75%	1.00E-04
19	3450	6534.74	0.4145	86.00%	1.00E-04
19	3500	6643.81	0.4047	84.00%	1.00E-04
19	3550	6736.28	0.4444	83.00%	1.00E-04
19	3600	6823.75	0.4137	85.75%	1.00E-04
20	3650	6915.86	0.395	84.00%	1.00E-04
20	3700	7003.37	0.4286	81.25%	1.00E-04
20	3750	7127.23	0.3593	86.25%	1.00E-04
20	3800	7217.34	0.3435	86.00%	1.00E-04
20	3820	7251.21	0.4207	82.25%	1.00E-04

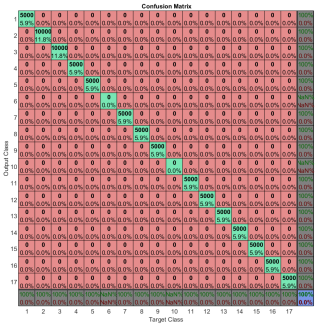
Wide network statistics for low learning rate 1.00E-04

3.3.2 Visualisations

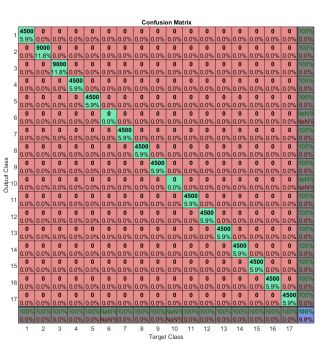


Confusion matrix for 17 groups of train augmented data for the high learning rate

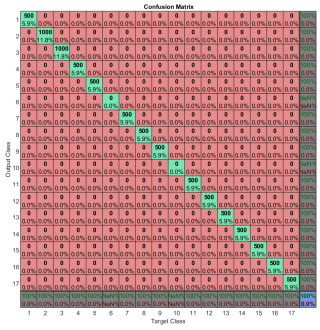
Confusion matrix for 17 groups of validation augmented data for the high learning rate



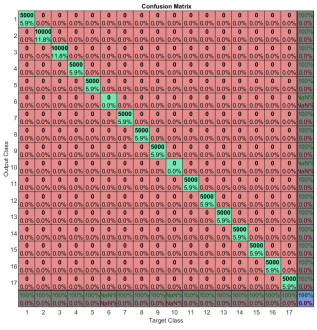
Confusion matrix for 17 groups of test augmented data for the best learning rate which is low here



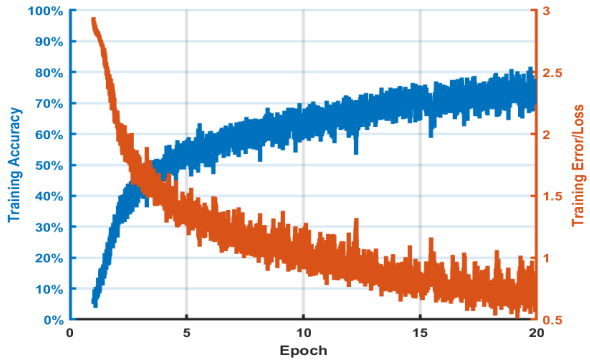
Confusion matrix for 17 groups of train augmented data for the less learning rate



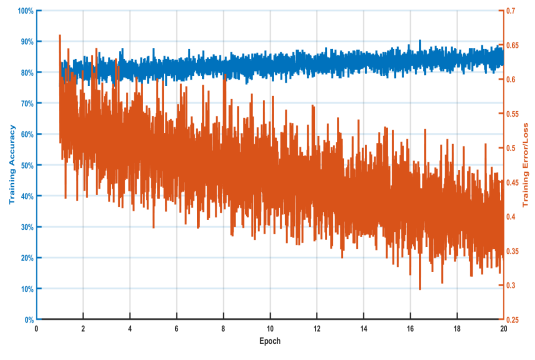
Confusion matrix for 17 groups of validation augmented data for the less learning rate



Confusion matrix for 17 groups of test augmented data for the best learning rate which is low here



plot for high learning rate



plot for low learning rate

Figure 3: Augmented Visualisations of confusion matrix for train, validation and test (for best learning rate)

3.4 Skinny Network

3.4.1 Statistics

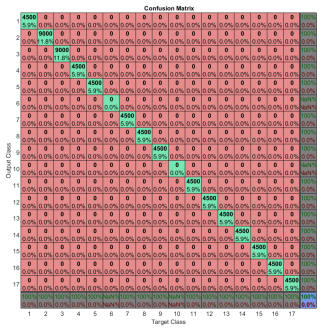
Epoch	Iteration	Time Elapsed	Mini-batch Loss	Mini-batch Accuracy	Base Learning Rate
1	1	9.23	2.8332	5.50%	0.0005
1	50	121.01	2.8332	5.00%	0.0005
1	100	215.3	2.8332	6.25%	0.0005
1	150	309.01	2.8332	5.50%	0.0005
2	200	402.81	2.8332	6.00%	0.0005
2	250	496.24	2.8332	7.25%	0.0005
2	300	630.64	2.8332	6.00%	0.0005
2	350	725.11	2.8332	4.75%	0.0005
3	400	818.4	2.8332	6.00%	0.0005
3	450	911.71	2.8332	6.75%	0.0005
3	500	1004	2.8332	6.00%	0.0005
3	550	1097.84	2.8332	4.50%	0.0005
4	600	1293.08	2.8332	6.50%	0.0005
4	650	1491.68	2.8332	7.00%	0.0005
4	700	1583.85	2.8332	5.75%	0.0005
4	750	1677.26	2.8332	3.25%	0.0005
5	800	1770.25	2.8332	6.00%	0.0005
5	850	1864.91	2.8332	6.00%	0.0005
5	900	1957.38	2.8332	4.50%	0.0005
5	950	2050.94	2.8332	5.00%	0.0005
6	1000	2144.32	2.8332	5.75%	0.0005
6	1050	2239.6	2.8332	5.00%	0.0005
6	1100	2334.84	2.8332	7.50%	0.0005
7	1150	2428.2	2.8332	4.25%	0.0005
7	1200	2524.19	2.8332	7.50%	0.0005
7	1250	2616.83	2.8332	6.75%	0.0005
7	1300	2710.3	2.8332	5.50%	0.0005
8	1350	2824.69	2.8332	5.25%	0.0005
8	1400	2937.88	2.8332	6.75%	0.0005
8	1450	3030.55	2.8332	6.00%	0.0005
8	1500	3124.05	2.8332	5.25%	0.0005
9	1550	3218.01	2.8332	5.00%	0.0005
9	1600	3312.01	2.8332	8.00%	0.0005
9	1650	3406.04	2.8332	6.25%	0.0005
9	1700	3498.64	2.8332	7.25%	0.0005
10	1750	3593.5	2.8332	4.75%	0.0005
10	1800	3685.66	2.8332	6.00%	0.0005
10	1850	3820.17	2.8332	5.75%	0.0005
10	1900	3913.12	2.8332	3.50%	0.0005
11	1950	4007.41	2.8332	6.50%	0.0005
11	2000	4118.02	2.8332	4.50%	0.0005
11	2050	4233.59	2.8332	4.75%	0.0005
11	2100	4329.05	2.8332	5.00%	0.0005
12	2150	4422.11	2.8332	5.00%	0.0005
12	2200	4515.47	2.8332	6.25%	0.0005
12	2250	4609.27	2.8332	5.25%	0.0005
13	2300	4743.77	2.8332	4.00%	0.0005
13	2350	4836.21	2.8332	5.50%	0.0005
13	2400	4930.28	2.8332	4.50%	0.0005
13	2450	5023.25	2.8332	6.50%	0.0005
14	2500	5117.35	2.8332	5.25%	0.0005
14	2550	5210.99	2.8332	7.25%	0.0005
14	2600	5303.6	2.8332	4.25%	0.0005
14	2650	5438.08	2.8332	6.25%	0.0005
15	2700	5531.43	2.8332	5.00%	0.0005
15	2750	5624.31	2.8332	5.75%	0.0005
15	2800	5718.17	2.8332	5.25%	0.0005
15	2850	5811.81	2.8332	5.75%	0.0005
15	2865	5839.86	2.8332	5.00%	0.0005

Skinny network statistics for high learning rate 0.0005

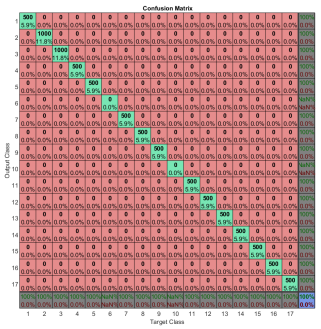
Epoch	Iteration	Time Elapsed	Mini-batch Loss	Mini-batch Accuracy	Base Learning Rate
1	1	2.38	2.8332	5.50%	1.00E-04
1	50	87.91	2.8332	7.50%	1.00E-04
1	100	182.18	2.8332	4.75%	1.00E-04
1	150	275.05	2.8332	5.75%	1.00E-04
2	200	369.41	2.8332	7.00%	1.00E-04
2	250	462.18	2.8332	8.00%	1.00E-04
2	300	555.92	2.8332	5.75%	1.00E-04
2	350	648.45	2.8332	4.00%	1.00E-04
3	400	756.51	2.8332	4.75%	1.00E-04
3	450	877.59	2.8332	4.25%	1.00E-04
3	500	971.46	2.8332	6.00%	1.00E-04
3	550	1064.26	2.8332	6.50%	1.00E-04
4	600	1158.43	2.8332	6.00%	1.00E-04
4	650	1251.12	2.8332	5.00%	1.00E-04
4	700	1348.4	2.8332	3.75%	1.00E-04
4	750	1441.97	2.8332	3.50%	1.00E-04
5	800	1535.26	2.8332	6.25%	1.00E-04
5	850	1670.32	2.8332	5.75%	1.00E-04
5	900	1763.21	2.8332	5.50%	1.00E-04
5	950	1855.93	2.8332	5.50%	1.00E-04
6	1000	1950.13	2.8332	4.75%	1.00E-04
6	1050	2085.23	2.8332	5.75%	1.00E-04
6	1100	2177.9	2.8332	6.75%	1.00E-04
7	1150	2271.16	2.8332	6.00%	1.00E-04
7	1200	2364.79	2.8332	5.50%	1.00E-04
7	1250	2457.45	2.8332	4.75%	1.00E-04
7	1300	2593.19	2.8332	5.75%	1.00E-04
8	1350	2685.91	2.8332	5.50%	1.00E-04
8	1400	2780.05	2.8332	5.25%	1.00E-04
8	1450	2873	2.8332	4.75%	1.00E-04
8	1500	2966.74	2.8332	6.25%	1.00E-04
9	1550	3061.47	2.8332	4.25%	1.00E-04
9	1600	3196.91	2.8332	6.50%	1.00E-04
9	1650	3289.32	2.8332	7.25%	1.00E-04
9	1700	3383.48	2.8332	7.50%	1.00E-04
10	1750	3476.79	2.8332	6.00%	1.00E-04
10	1800	3570.28	2.8332	7.50%	1.00E-04
10	1850	3664.01	2.8332	5.00%	1.00E-04
10	1900	3798.35	2.8332	4.50%	1.00E-04
11	1950	3891.94	2.8332	6.25%	1.00E-04
11	2000	3985.64	2.8332	4.00%	1.00E-04
11	2050	4079.46	2.8332	5.00%	1.00E-04
11	2100	4172.2	2.8332	5.00%	1.00E-04
12	2150	4266.86	2.8332	5.50%	1.00E-04
12	2200	4359.91	2.8332	5.25%	1.00E-04
12	2250	4453.42	2.8332	6.50%	1.00E-04
13	2300	4546.9	2.8332	7.50%	1.00E-04
13	2350	4640.91	2.8332	7.75%	1.00E-04
13	2400	4733.9	2.8332	5.75%	1.00E-04
13	2450	4828.13	2.8332	4.75%	1.00E-04
14	2500	4925.77	2.8332	6.50%	1.00E-04
14	2550	5060.05	2.8332	4.50%	1.00E-04
14	2600	5152.33	2.8332	4.75%	1.00E-04
14	2650	5246.39	2.8332	7.00%	1.00E-04
15	2700	5340.93	2.8332	5.25%	1.00E-04
15	2750	5433.91	2.8332	5.25%	1.00E-04
15	2800	5528.33	2.8332	8.75%	1.00E-04
15	2850	5621.31	2.8332	5.00%	1.00E-04
15	2865	5650.24	2.8332	5.25%	1.00E-04

Skinny network statistics for high learning rate 1.00E-04

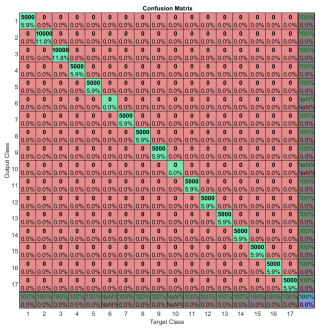
3.4.2 Visualisations



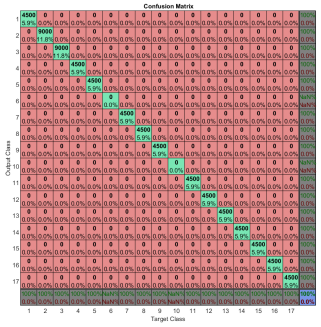
Confusion matrix for 17 groups of train augmented data for the high learning rate



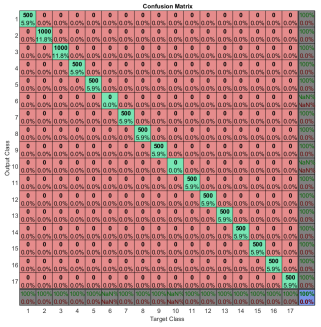
Confusion matrix for 17 groups of validation augmented data for the high learning rate



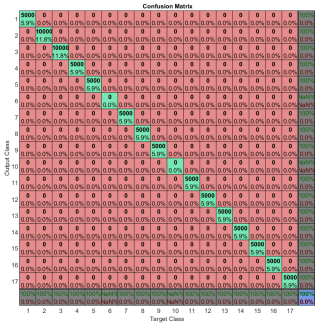
Confusion matrix for 17 groups of test (same as for both rates) augmented data for the high learning rate



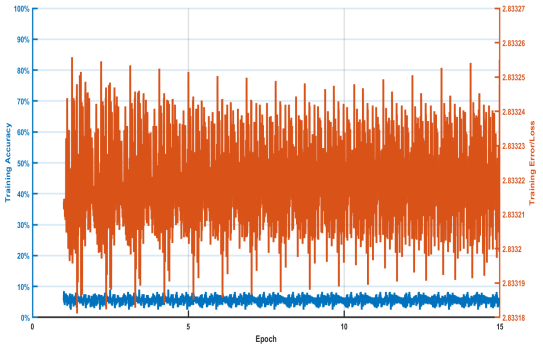
Confusion matrix for 17 groups of train augmented data for the less learning rate



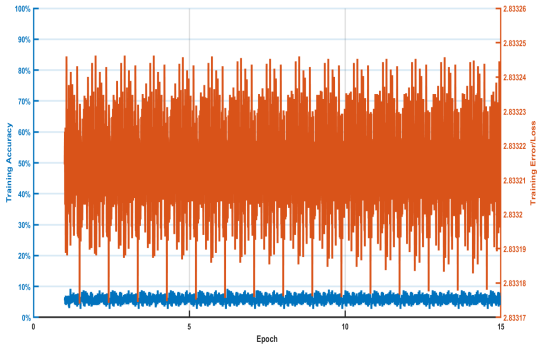
Confusion matrix for 17 groups of validation augmented data for the less learning rate



Confusion matrix for 17 groups of test (same for both rates) augmented data for the best learning rate which is low here



plot for high learning rate



plot for low learning rate

Figure 4: The accuracy vs train loss plots for the skinny network for both lesser and higher leanring rates

Skinny Network Accuracy Results: The results are not that great as the loss rate stops changing ad the validation and train accuracies are same for both learning rates.

	train accuracy	Validation Accuracy	Test Accuracy	Execution time
high learning rate	0.0588	0.0588	0.0588	6181.74 secs
low learning rate	0.0588	0.0588	0.0588	5972.76 secs

Table 3: Results for the accuracy for the skinny network

3.5 Extra credit work

3.5.1 Transfer Learning (AlexNet)

Explanation: I have installed the alexnet module in matlab and saved the workspace for alexnet and performed the training for that .mat file generated in the work space of matlab.
The accuracy results for the pre-trained alex network of convolution neural network for original data (unaugmented) are as below:

	train accuracy	Validation Accuracy	Test Accuracy	Execution time
high learning rate	0.8454	0.7783	not the best	5567.45 secs
low learning rate	0.8972	0.9213	0.8796	6754.76 secs

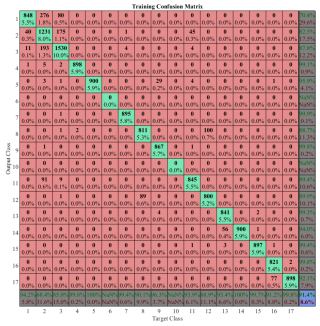
Table 4: Results for the accuracy for the Alex network for unaugmented (original) data

The accuracy results for the pre-trained alex network of convolution neural network for transformed data (augmented) are as below:

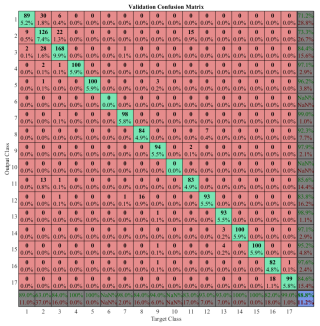
	train accuracy	Validation Accuracy	Test Accuracy	Execution time
high learning rate	0.834	0.856	0.807	8098.4 secs
low learning rate	0.945	0.7985	0.8653	7472.96 secs

Table 5: Results for the accuracy for the Alex network for augmented data

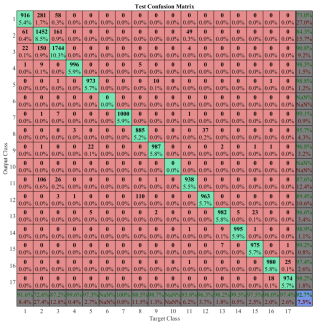
The training results for the pre-trained alex network of convolution neural network for original data (unaugmented) are as below:



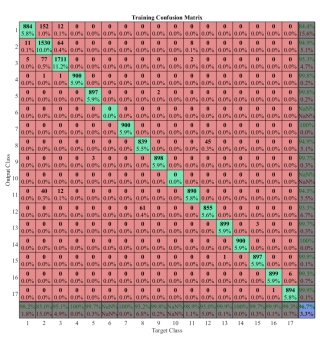
Confusion matrix for 17 groups of train augmented data for the high learning rate



Confusion matrix for 17 groups of validation augmented data for the high learning rate

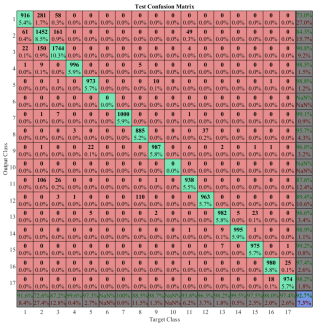


Confusion matrix for 17 groups of test (same for both) augmented data for the high learning rate

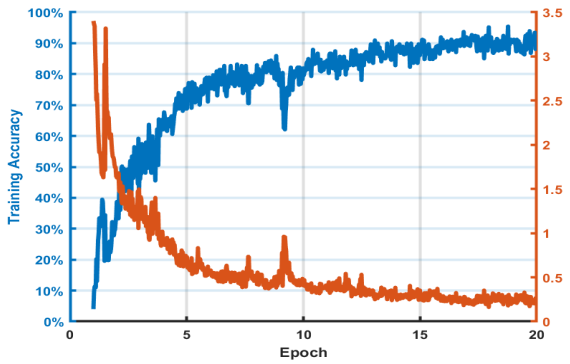


Confusion matrix for 17 groups of train augmented data for the less learning rate

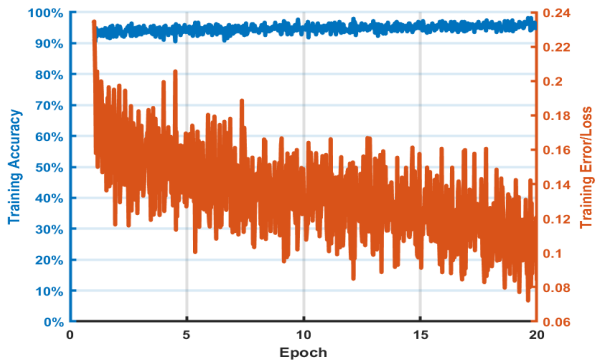
Confusion matrix for 17 groups of validation augmented data for the less learning rate



Confusion matrix for 17 groups of test augmented data for the best learning rate which is low here



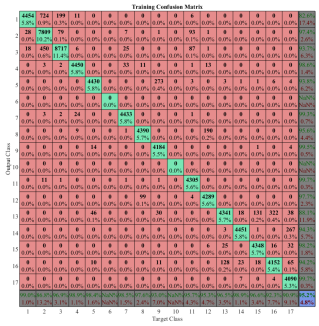
plot for high learning rate



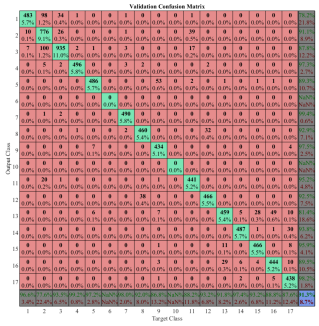
plot for low learning rate

Figure 5: train accuracy vs loss plots for both less and high learning rates

The training results for the pre-trained alex network of convolution neural network for transformed data (augmented) are as below:



Confusion matrix for 17 groups of train augmented data for the high learning rate



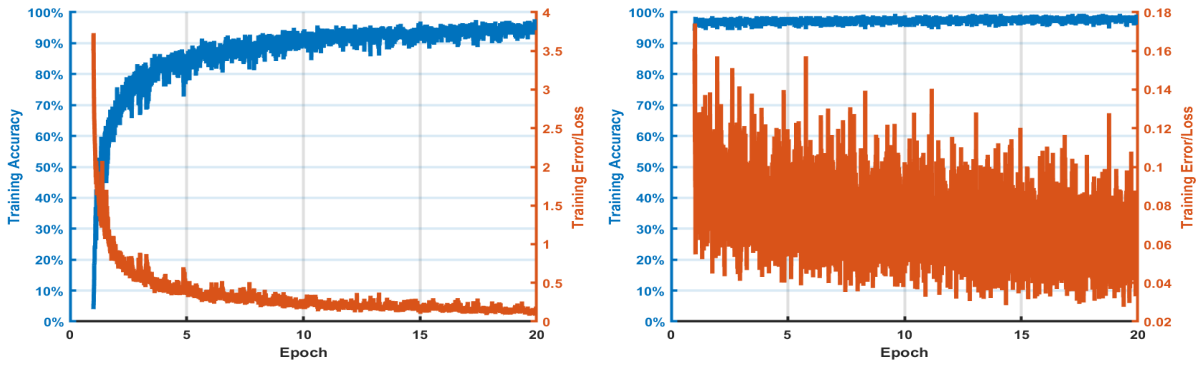
Confusion matrix for 17 groups of validation augmented data for the high learning rate

Confusion matrix for 17 groups of test (same for both) augmented data for the high learning rate

Confusion matrix for 17 groups of train augmented data for the less learning rate

Confusion matrix for 17 groups of validation augmented data for the less learning rate

Confusion matrix for 17 groups of test augmented data for the best learning rate which is low here



plot for high learning rate plot for low learning rate

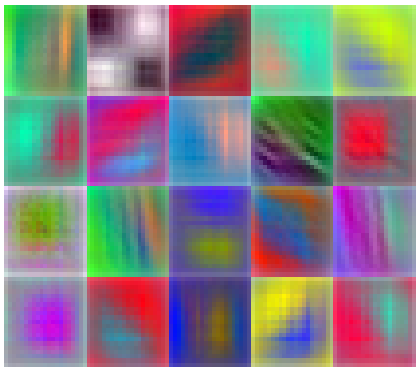
Figure 6: train accuracy vs loss plots for both less and high learning rates

3.5.2 First layer of filters visualisation

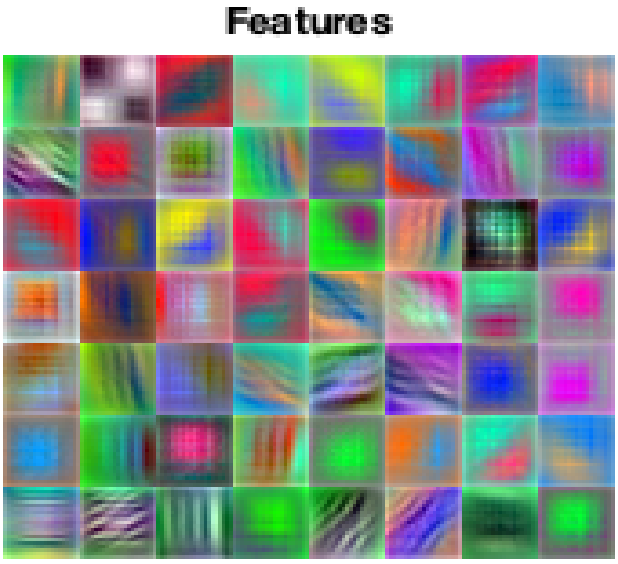
The first layers of convolutional neural networks often have very human interpretable values, as seen in these example plots. Visually, these filters are similar to other filters used in computer vision, such as Gabor filters.

Notice that the first-layer weights are very nice and smooth, indicating nicely converged network. The color/grayscale features are clustered because the AlexNet contains two separate streams of processing, and an apparent consequence of this architecture is that one stream develops high-frequency grayscale features and the other low-frequency color features. The later CONV layer weights are not as interpretable, but it is apparent that they are still smooth, well-formed, and absent of noisy patterns.

Here are two visualisations for the AlexNet model which are as below:
1. tsne for AlexNet for the original data
2. tsne for AlexNet for the augmented data



Visualization of the first layer of filters for AlexNet for the unaugmented data

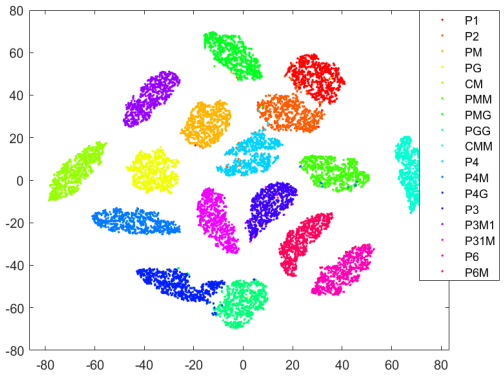


Visualization of the first layer of filters for AlexNet for the augmented data

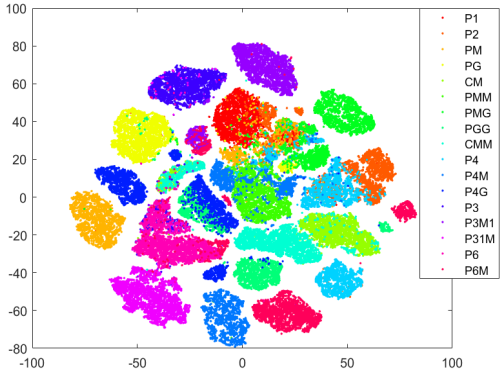
Figure 7: First layer of filters visualisation of AlexNet for both original and augmented data

3.5.3 t-SNE multidimensional reduction visualisation

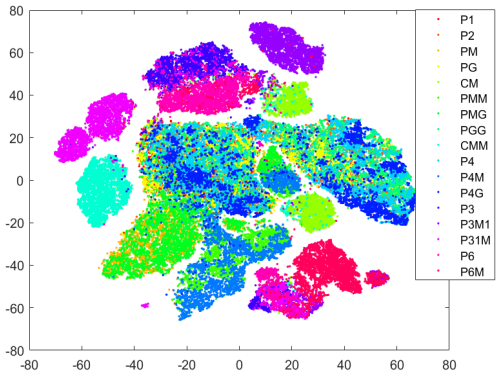
t-SNE (tsne) is an algorithm for dimensionality reduction that is well-suited to visualizing high-dimensional data. (Generally, it is impossible to match distances exactly between high-dimensional and low-dimensional spaces.) The tsne function creates a set of low-dimensional points from high-dimensional data.
Bases on the python point of view, t-SNE [1] is a tool to visualize high-dimensional data. It converts similarities between data points to joint probabilities and tries to minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data. t-SNE has a cost function that is not convex, i.e. with different initializations we can get different results.
It is highly recommended to use another dimensionality reduction method (e.g. PCA for dense data or TruncatedSVD for sparse data) to reduce the number of dimensions to a reasonable amount (e.g. 50) if the number of features is very high. This will suppress some noise and speed up the computation of pairwise distances between samples.



Visualization of t-SNE multidimensional reduction on the fully connected layer activations of wide network for unaugmented data



Visualization of t-SNE multidimensional reduction on the fully connected layer activations of wide network for augmented data



Visualization of t-SNE multidimensional reduction on the fully connected layer activations of alex network for augmented data

Figure 8: tsne visualisations for wide and alex networks which shows an accuracy (skinny not involved)

4 Conclusion

- While I perform the networking trains I have visualised that the main network which is given as an example stays as a good task for the original data. But when the data is augmented the results changed.
- The augmented data results in less accuracy than the original data network does. As the loss learning for network for high learning rate doesn't change more the model doesn't have good accuracy.
- Among the both skinny and wide the wide results in better results compared to the skinny. The results are not as good as for the original data but they reach a 70 percentage which is far better than the skinny accuracy.
- Finally, comparison on skinny and wide with alex network showed a drastic change in the networking results which are far better than wide and are not a way equal to original data network but the accuracy reaches to at atleast 85 percentage which is better.

References

[1] J. J. Instructor: Fei-Fei Li, Serena Yeung, *CS231n: Convolutional Neural Networks for Visual Recognition*. Gardent et al., August, 2018. [Online]. Available: http://cs231n.stanford.edu/ 4

[2] A. P. Mishra, *Introduction to CNN*. Geeks for geeks, May, 2017. [Online]. Available: https://www.geeksforgeeks.org/introduction-convolution-neural-network/ 4