Convolution Neural Network

Tejas Mahale

March 2018

Abstract :

Neural nets are set of algorithms that are designed to recognise the pattern. They recognise sensor's data obtained by machine perception, labelling or clustering raw input data. Convolution neural network are similar to ordinary neural network[2]. In convolution neural net, we try to extract features using filters before passing them to fully connected neural layers[2]. In this project we are going to see the effect of convolution layers on wallpaper dataset. We will check multiple settings of convolution layer which includes activation(ReLu) and pooling on skinny and wide network. In industry CNN, mostly used on image datasets because of its ability to extract low-level features like edges, as well as the higher level features like detecting the eye that indicates a face or a cat or human[3]. In this project, we will look out for effects data augmentation, different settings of hyper-parameters on both skinny and wide network and try to generalise best setting for wallpaper dataset.

Topic	page Number
1 Introduction	2
1.1 Neural Network	2
1.2 Convolution Neural Network:	4
1.3 ConvNet Layers	5
2 ConvNet Architecture	9
2.1 Basic ConvNet	9
2.2 Skinny Convolution Network	10
2.3 Wide Convolution Network	11
2.4 Alexnet	12
2.5 Data Augmentation	13
3 Results and Comaparisons	14
3.1 Results (Basic, Skinny, Wide network on original and augmented data)	14
3.2 Comparisons between models and datasets used	47
3.3 Extra Credit (Alexnet, First Layer Visualisation, t-SNE)	48
4. Conclusion	60
5. References	61

1. Introduction:

1.1 Neural Network:

For convolution neural net, we need to properly understand deep neural network architecture. In deep neural net, multiple neural networks with connected neurons attached together. Initial layer is input layer which is fed to first hidden neuron layer as per fig 1.1.1

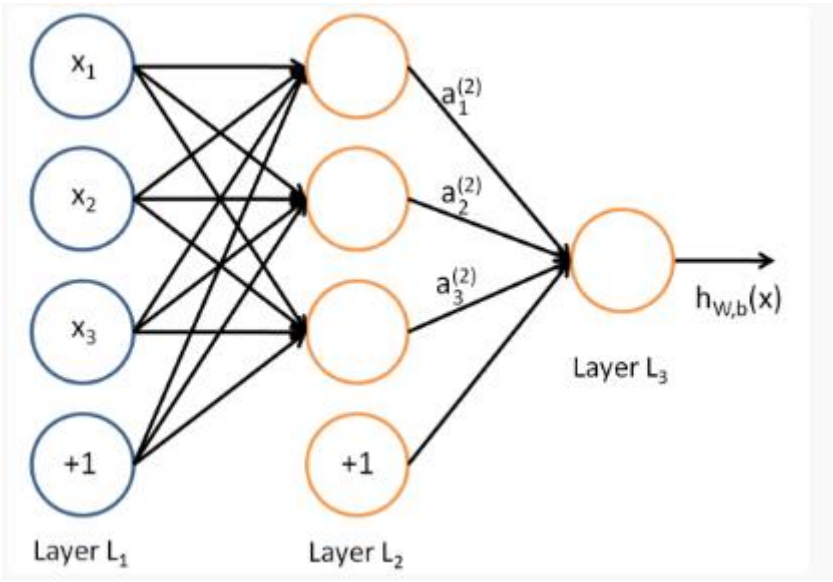


fig 1.1.1

This is one hidden layer neural network. Layer L_1 as input layer, layer L_2 is hidden layer giving output to output layer layer L_1. Every layer has assigned weights and dot product of inputs and weights given to activation function which servers as input to next hidden layer. We will denote activation of unit i in layer l as $a_i^{(l)}$.

The computation of neural network for layer 2 is given as:

$$a_1^{(2)} = f(\, W_{11}^{(1)}\, x_1 + \, W_{12}^{(1)}\, x_2 + \, W_{13}^{(1)}\, x_3 + \, b_1^{(1)})$$

$$a_2^{(2)} = f(\, W_{21}^{(1)}\, x_1 + \, W_{22}^{(1)}\, x_2 + \, W_{23}^{(1)}\, x_3 + \, b_2^{(1)})$$

$$a_3^{(2)} = f(\, W_{31}^{(1)}\, x_1 + \, W_{32}^{(1)}\, x_2 + \, W_{33}^{(1)}\, x_3 + \, b_3^{(1)})$$

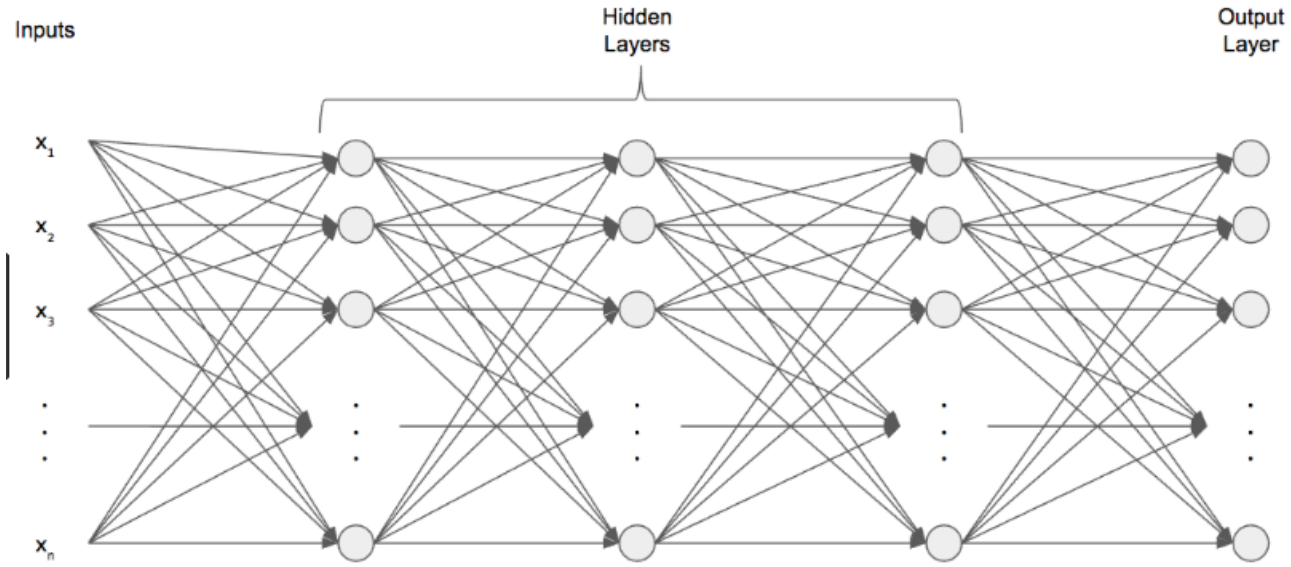
$$h_{w,b}(x) = a_1^{(3)} = \, f(\, W_{11}^{(2)}\, x_1 + \, W_{12}^{(2)}\, x_2 + \, W_{13}^{(2)}\, x_3 + \, b_1^{(2)})$$

here $f\,()$ is called as activation function. In general for forward propogation,

$$z^{(l+1)} = \, W^{(l)} a^{(l)} + b^{(l)}$$

$a^{(l+1)} = f(z^{(l+1)})$

l is current layer. We can generalise this result to multi-layer neural network.



Back propagation:

For m example we will try to find cost function which is difference between actual output label and internally generated label by neural network[4].

$$J(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m}\left(\frac{1}{2}\left\|h_{W,b}(x^{(i)}) - y^{(i)}\right\|^2\right)\right] + \frac{\lambda}{2}\sum_{l=1}^{n_i-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_l+1}(W_{ji}^{(l)})^2$$

We need to find values of parameters W, b such that above cost function has minimum value.

here we can use derivatives to achieve global minima

$$\frac{\partial(J(W,b))}{\partial W_{ij}^{(l)}} = \left[\frac{1}{m}\sum_{i=1}^{m}\frac{\partial J(W,b;x^{(i)},y^{(i)})}{\partial W_{ij}^{(l)}}\right] + \lambda W_{ij}^{(l)}$$

$$\frac{\partial(J(W,b))}{\partial b_i^{(l)}} = \left[\frac{1}{m}\sum_{i=1}^{m}\frac{\partial J(W,b;x^{(i)},y^{(i)})}{\partial b_{ij}^{(l)}}\right]$$

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha\frac{\partial J(W,b)}{\partial W_{ij}^{(l)}}$$

$$b_i^{(l)} = b_i^{(l)} - \alpha\frac{\partial J(W,b)}{\partial b_i^{(l)}}$$

here α is learning rate

1.2 Convolution Neural Network:

Natural images have the property of being "'stationary"', meaning that the statistics of one part of the image are the same as any other part. This suggests that the features that we learn at one part of the image can also be applied to other parts of the image, and we can use the same features at all locations[4].

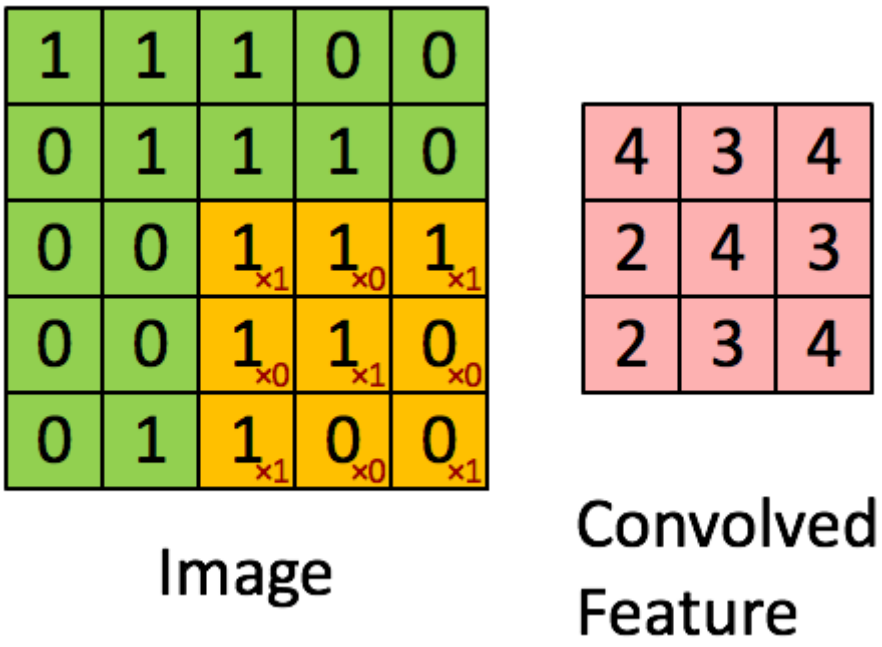


fig 1.1.3

In convolution layer we take image convolve with filter which represents particular feature of image. CovNet is made up layers. Each layer transforms input 3D volume to output 3D volume with some differentiable function with or without patameters[3].

Suppose image has size n x n x 3 convolve with 20 filters of size f x f x 3 we get output as n-f +1 x n-f+1 x 3. But this is general case we never use such dimension to compute output. Because when we use filter with convolution, it won't work for outer layer of image which reduced dimension of image ultimately it reduces information which can't be ignored. We need to understand that 3rd dimension of output is number of filter used rather than related to 3rd dimension of input. Here we introduce concept of padding and stride selection. Lets discuss more on this in ConvNet Layers section

1.3 ConvNet Layers:

1.3.1 Convolution layers:

As discussed earlier, in this layer we implement information extraction using convolution. We can use randomly generated weight filters or custom filters like edge detection filter.

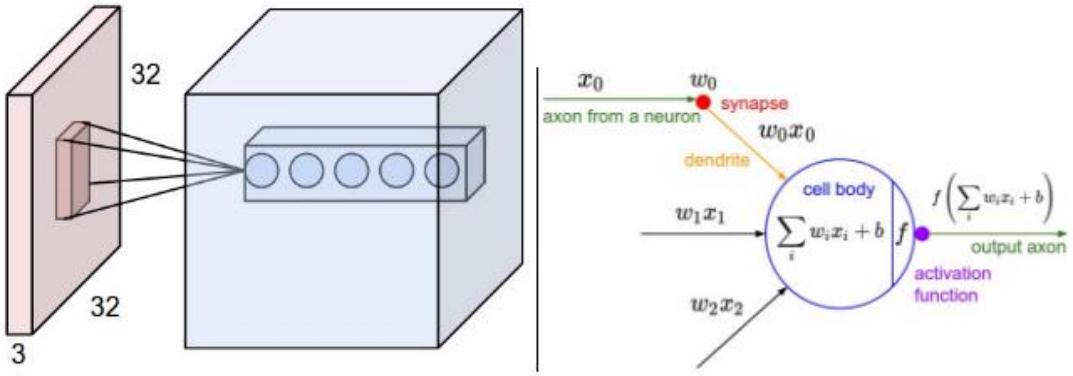


fig 1.3.1

Padding:

To take care of outer layer while filtering, we use padding. We pads image on both dimensions so filter can work on boundaries of image. We usually use zero padding.

Stride:

Stride number shows number of pixels that our filter will move while filtering. When the stride is 1 then we move the filters one pixel at a time. stride value can be 2 or more but higher value of stride will reduce efficiency of filter.

So number of neurons fits in layer is given by formula:

$$(n - f + 2P)/S + 1$$

n x n → Size of filter

f x f → filter size

P → Padding

S → Stride

1.3.2 ReLu activation:

As suggested in 1.1 section, neural net has activation function. Likewise in CovNet we use ReLu activation before pooling. It computes the function f(x)= max(0, x). In other word, activation is simply threshold to zero.

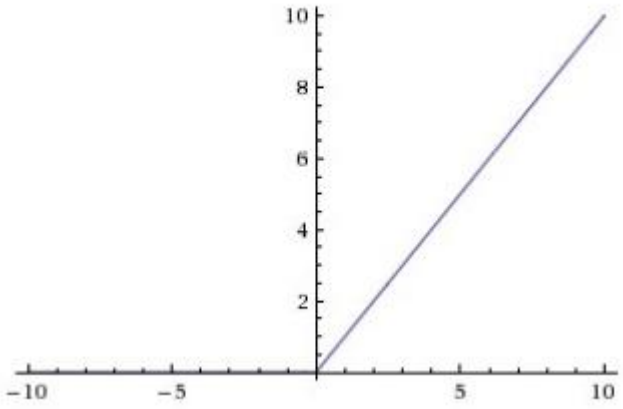


fig 1.3.2.1

ReLu came out to be best activation function in era of deep learning. It the convergence of stochastic gradient descent compared to other activation function[3].

1.3.3 Pooling Layer:

It is common to periodically insert a Pooling layer in-between successive Conv layers in a ConvNet architecture[3]. Advantage of using pooling layer is it furthur reduces number of unnecessary weights by averaging or maxing out number of pixels. The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2 downsamples every depth slice in the input by 2 along both width and height, discarding 75% of the activations[3]. We hardly use zero padding for pooling layer.

To reduce the extensive load from network, we usually use pooling before fully connected layers.

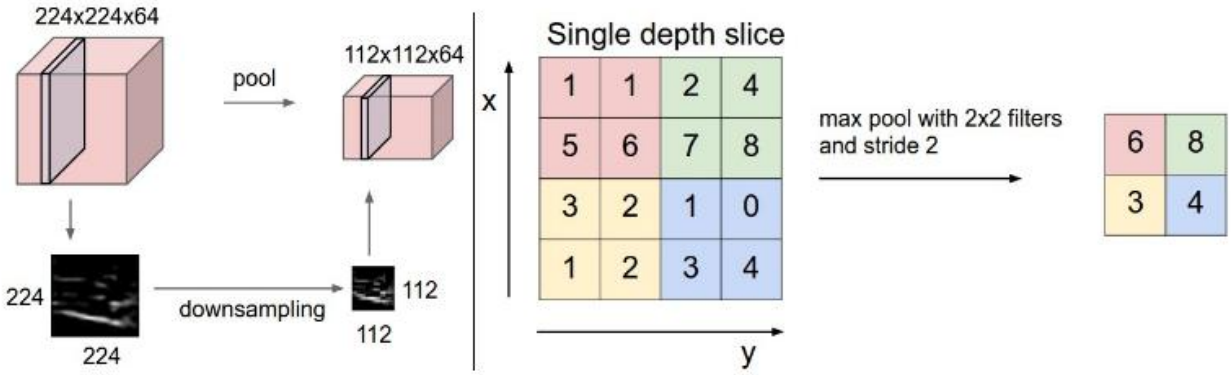


fig 1.3.3 Pooling Example

1.3.4 Fully-connected layer:

At end of convolution layer (including max pooling and ReLu), we use fully connected layer for high level reasoning. Fully connected layer is same as we saw in section 1.1 multiple neural network.

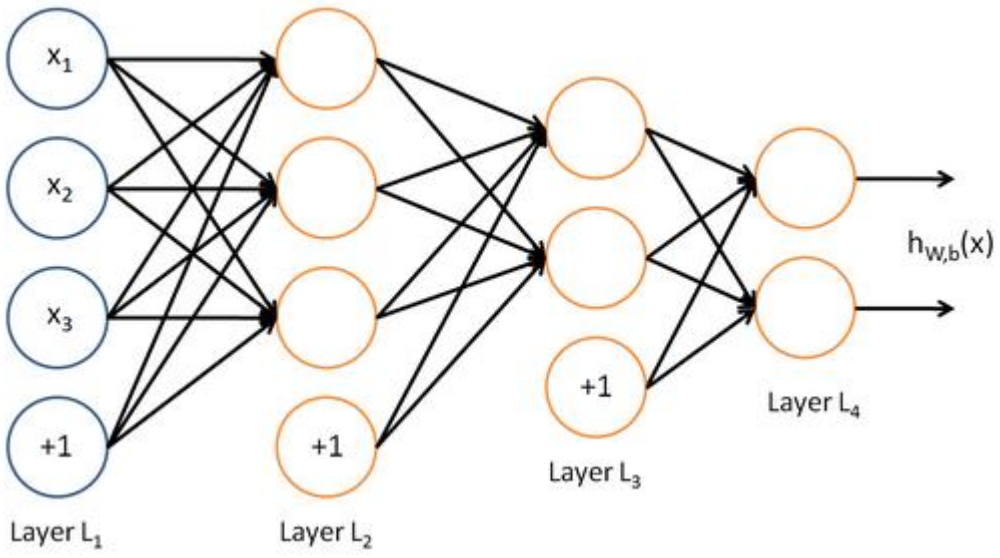


fig 1.3.4

Number of hidden layer are depend on dataset size. If we have large datasets, we can certainly use high number of fully connected layers. Being said that, precaution should be taken on number of fully connected layers just before output layer.

Dropout: Dropout is added between certain fully connected layers to inactivate some neurons in random order. It is one of the method of regularization. It basically avoids dependency of single neuron in layer having large cost which causes over-fitting.

1.3.5 Softmax Layer and Classification layer:

At output layer, we are not going to use Relu because we don't want to threshold output. For this purpose we use softmax layer. Softmax is generalisation of logistic regression which handles multiple classes. Softmax regression allows us to handle Y(i)∈{1,…,K} where K is the number of classes[3].

Softmax function is given as :

$$f_j(z) = \frac{e^{zj}}{\sum_k e^{zk}}$$

In this case loss function is given by

$$L_i = -\log\left(\frac{e^{f_{yi}}}{\sum_k e^{f_i}}\right)$$

Classification layer takes input from softmax layer and decide the labeling on basis of that layer.

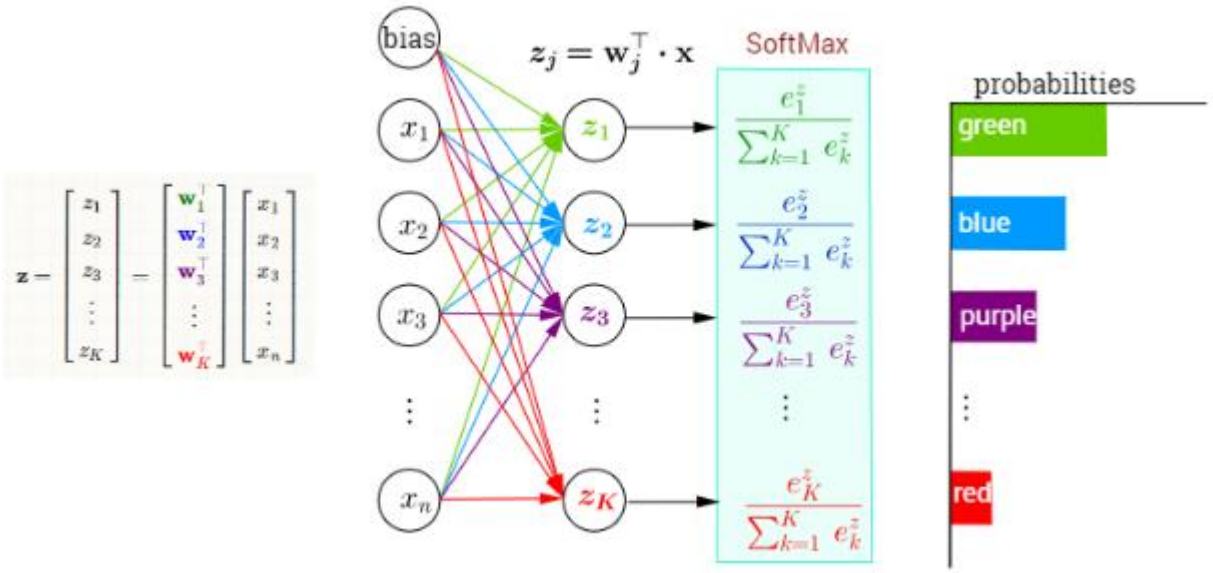


fig 1.3.5

2 .ConvNet Architecture:

What could be the best ConvNet architecture is active area of research. It is difficult to come with best architecture which will work efficiently on every dataset.

In this project we are using 3 convolution layer architectures.

2.1 Basic ConvNet:

To start with this project, we will work on basic ConvNet given below:

CONV -> ReLU -> POOL -> FC -> DROPOUT -> FC -> SOFTMAX

Layers =

[imageInputLayer([256 256 1]); % Input to the network is a 256x256x1 sized image

-convolution2dLayer(5,20,'Padding',[2 2],'Stride', [2,2]); % convolution layer with 20, 5x5 filters

-reluLayer(); % ReLU layer

-maxPooling2dLayer(2,'Stride',2); % Max pooling layer

-fullyConnectedLayer(25); % Fullly connected layer with 50 activations

-dropoutLayer(.25); % Dropout layer

-fullyConnectedLayer(17); % Fully connected with 17 layers

-softmaxLayer(); % Softmax normalization layer

-classificationLayer(); % Classification layer

]

2.2 Skinny ConNet:

In skinny architecture, we are increasing number of convolution layers and fully connected layers to stretch the network. In every ConvNet we are adding max pool and relu.

CONV -> ReLU -> POOL -> CONV -> ReLU -> POOL -> CONV -> ReLU -> POOL -> FC -> DROPOUT -> FC -> SOFTMAX

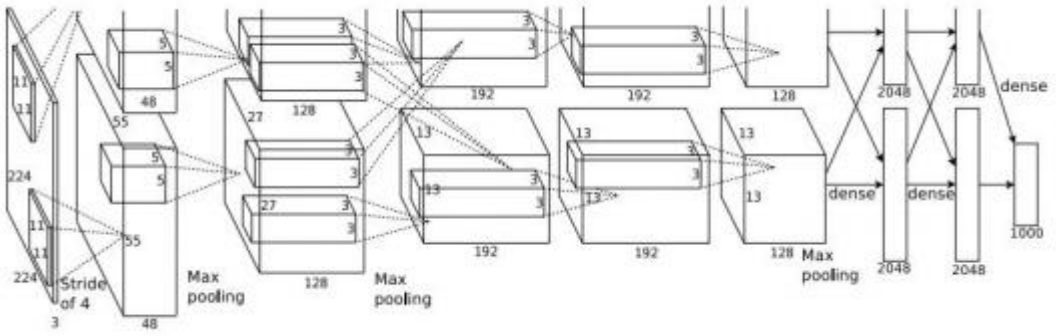
```
layers = [
- imageInputLayer([128 128 1]); % Input to the network is a 256x256x1 sized image
-convolution2dLayer(5,40,'Padding',[2 2],'Stride', [1,1]);  % convolution layer with 20, 5x5 filters
-reluLayer();  % ReLU layer
-maxPooling2dLayer(2,'Stride',2); % Max pooling layer
-convolution2dLayer(5,40,'Padding',[1 1],'Stride', [1,1]);  % convolution layer with 20, 5x5 filters
-reluLayer();  % ReLU layer
-maxPooling2dLayer(2,'Stride',2); % Max pooling layer
-convolution2dLayer(3,80,'Padding',[1 1],'Stride', [1,1]);  % convolution layer with 20, 5x5 filters
-reluLayer();  % ReLU layer
-maxPooling2dLayer(2,'Stride',2); % Max pooling layer
-fullyConnectedLayer(100); % Fullly connected layer with 50 activations
-dropoutLayer(.25); % Dropout layer
-fullyConnectedLayer(17); % Fully connected with 17 layers
-softmaxLayer(); % Softmax normalization layer
-classificationLayer(); % Classification layer
  ];
```

2.3 Wide ConvNet:

In wide network, we are increasing number of filters rather than increasing number of layers.

```
 [
-imageInputLayer([128 128 1]); % Input to the network is a 256x256x1 sized image
-convolution2dLayer(5,80,'Padding',[2 2],'Stride', [1,1]);  % convolution layer with 20, 5x5 filters
 -reluLayer();  % ReLU layer
-maxPooling2dLayer(2,'Stride',2); % Max pooling layer
-convolution2dLayer(3,120,'Padding',[1 1],'Stride', [1,1]);  % convolution layer with 20, 5x5 filters
-reluLayer();  % ReLU layer
-maxPooling2dLayer(2,'Stride',2); % Max pooling layer
-fullyConnectedLayer(100); % Fullly connected layer with 50 activations
-dropoutLayer(.25); % Dropout layer
-fullyConnectedLayer(17); % Fully connected with 17 layers
-softmaxLayer(); % Softmax normalization layer
-classificationLayer(); % Classification layer
   ];
```

2.4 Alexnet



Alexnet returns a pretrained AlexNet model. This model is trained on a subset of the ImageNet database, which is used in ImageNet Large-Scale Visual Recognition Challenge (ILSVRC)[5]. The model is trained on more than a million images and can classify images into 1000 object categories[5].

25x1 Layer array with layers:

1	'data'	Image Input	227x227x3 images with 'zerocenter' normalization
2	'conv1'	Convolution	96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]
3	'relu1'	ReLU	ReLU
4	'norm1'	Cross Channel Normalization	cross channel normalization with 5 channels per element
5	'pool1'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
6	'conv2'	Convolution	256 5x5x48 convolutions with stride [1 1] and padding [2 2 2 2]
7	'relu2'	ReLU	ReLU
8	'norm2'	Cross Channel Normalization	cross channel normalization with 5 channels per element
9	'pool2'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
10	'conv3'	Convolution	384 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]
11	'relu3'	ReLU	ReLU
12	'conv4'	Convolution	384 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
13	'relu4'	ReLU	ReLU
14	'conv5'	Convolution	256 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]
15	'relu5'	ReLU	ReLU
16	'pool5'	Max Pooling	3x3 max pooling with stride [2 2] and padding [0 0 0 0]
17	'fc6'	Fully Connected	4096 fully connected layer
18	'relu6'	ReLU	ReLU
19	'drop6'	Dropout	50% dropout
20	'fc7'	Fully Connected	4096 fully connected layer
21	'relu7'	ReLU	ReLU
22	'drop7'	Dropout	50% dropout
23	'fc8'	Fully Connected	1000 fully connected layer
24	'prob'	Softmax	softmax
25	'output'	Classification Output	crossentropyex with 'tench', 'goldfish', and 998 other classes

2.5 Data Augmentation:

While dealing with deep learning applications like computer vision, we require larger datasets. If you don't have bigger dataset, one method to generate it, is Data Augmentation.

Data Augmentation Techniques:

1. Mirroring
2. Random multiple cropping
3. Rotation
4. Shearing
5. Local Wrapping
6. Adding and subtracting R,G,B values in some proportion

We need to be careful while data augmentation that it's distribution must be same as test data distribution.

In this project we used wallpaper dataset.

No of Images = 17000

No of groups =17

Image size = 256 x 256

No. of feature =17

3. Results and comparisons

In this section we will compare different result generated from non augmented dataset, augmented dataset, skinny layer network and wide layer network.

3.1 Results:

1) Single convolution layered model on base dataset:

CONV -> ReLU -> POOL -> FC -> DROPOUT -> FC -> SOFTMAX

Parameters	Values
Dataset size	17000
No epoch	10
Batch size	250
Learning rate	5e-4 and 1e-4
1st conv layer	Filters: 5 x 5, filter size: 20, Padding: [2,2], Stride:[2,2]
Fully conncted layers (FC)	25
Drop-out	0.25
FC before classification	17

Table3.1

We use this model on two different learning rate:

a) Learning rate: 5e-4 Table 3.1 A

Epoch	Iteration	Time Elapsed (seconds)	Mini-batch Loss	Mini-batch Accuracy	Base Learning Rate
1	1	0.42	3.0780	4.40%	0.0005
1	50	14.00	2.0580	20.00%	0.0005
2	100	28.11	1.6167	34.40%	0.0005
3	150	42.16	1.2581	47.60%	0.0005
4	200	56.23	1.0022	56.80%	0.0005
5	250	70.35	0.9284	59.20%	0.0005
5	300	84.20	0.8855	59.60%	0.0005
6	350	98.32	0.9208	62.00%	0.0005
7	400	112.42	0.6599	69.60%	0.0005
8	450	126.46	0.7805	67.20%	0.0005
9	500	140.53	0.6181	74.80%	0.0005
10	550	154.57	0.6921	73.60%	0.0005
10	600	168.41	0.6150	73.20%	0.0005
10	610	171.17	0.7236	67.20%	0.0005

Trained in in 183.09 seconds

train_acc = 0.7561, val_acc = 0.7082, test_acc = 0.7068

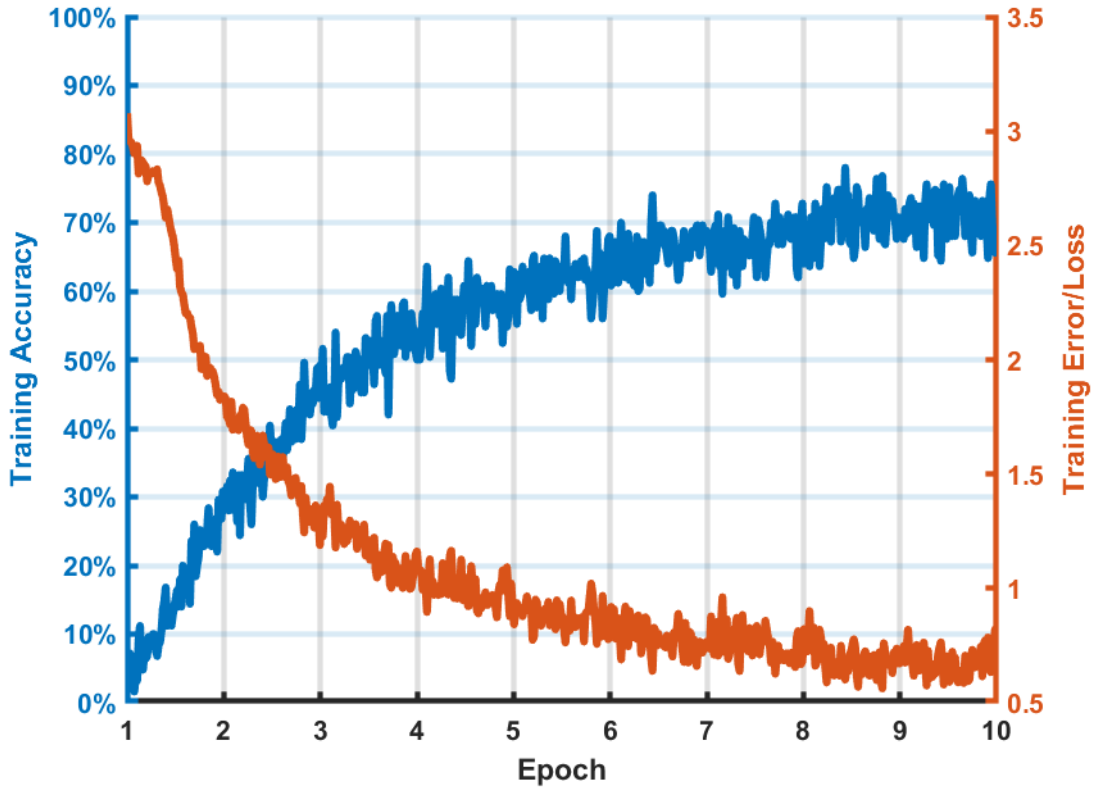


fig 3.1

Train Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	900	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	227	86	345	242	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	465	435	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	462	438	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	466	434	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	900	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	98	802	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	900	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	528	372	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	531	369	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	10	890	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	18	882	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	900	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	52	848	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	58	842	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	900	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	899

Table 3.1.2

Train Classification Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0.252222	0.095556	0.383333	0.268889	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0.516667	0.483333	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0.513333	0.486667	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0.517778	0.482222	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0.108889	0.891111	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0.586667	0.413333	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0.59	0.41	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0.011111	0.988889	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0.02	0.98	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0.057778	0.942222	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0.064444	0.935556	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.001111	0.998889

Table 3.1.3

Validation Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	48	10	34	8	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	74	26	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	68	32	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	66	34	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	20	80	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	52	48	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	56	43	1	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	8	90	2	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	7	93	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	11	88	1	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	99

Table 3.1.4

Validation Classification Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0.48	0.1	0.34	0.08	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0.74	0.26	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0.68	0.32	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0.66	0.34	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0.2	0.8	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0.52	0.48	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0.56	0.43	0.01	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0.08	0.9	0.02	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0.07	0.93	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0.11	0.88	0.01	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	0.99

Table 3.1.5

Test Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	223	87	324	366	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	484	516	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	452	548	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	477	523	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	1000	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	69	931	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	1000	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	633	367	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	653	347	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	33	967	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	69	919	12	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	1000	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	115	885	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	134	860	6	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1000	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	991

Table 3.1.6

Test Classification Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0.223	0.087	0.324	0.366	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0.484	0.516	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0.452	0.548	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0.477	0.523	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0.069	0.931	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0.633	0.367	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0.653	0.347	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0.033	0.967	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0.069	0.919	0.012	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0.115	0.885	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0.134	0.86	0.006	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.009	0.991

Table 3.1.7

b) Learning rate: 1 e-4

Epoch	Iteration	Time Elapsed (seconds)	Mini-batch Loss	Mini-batch Accuracy	Base Learning Rate
1	1	0.41	0.7698	67.20%	1.00e-04
1	50	14.17	0.4892	81.60%	1.00e-04
2	100	28.38	0.5455	77.60%	1.00e-04
3	150	43.05	0.5212	77.60%	1.00e-04
4	200	57.22	0.5764	75.60%	1.00e-04
5	250	71.33	0.5502	78.00%	1.00e-04
5	300	85.24	0.6205	73.60%	1.00e-04
6	350	99.59	0.5160	76.80%	1.00e-04
7	400	113.67	0.5472	75.60%	1.00e-04
8	450	127.86	0.5241	75.60%	1.00e-04
9	500	141.94	0.4687	82.80%	1.00e-04
10	550	156.45	0.5919	74.40%	1.00e-04
10	600	170.29	0.5336	78.80%	1.00e-04
10	610	173.14	0.4906	81.20%	1.00e-04

Trained in in 182.98 seconds

Table 3.1 B

train_acc = 0.7751, val_acc = 0.7162, test_acc = 0.7148

We got slight better accuracy in this case because of learning rate is less compare to first case. But it took more time due to lower learning rate.

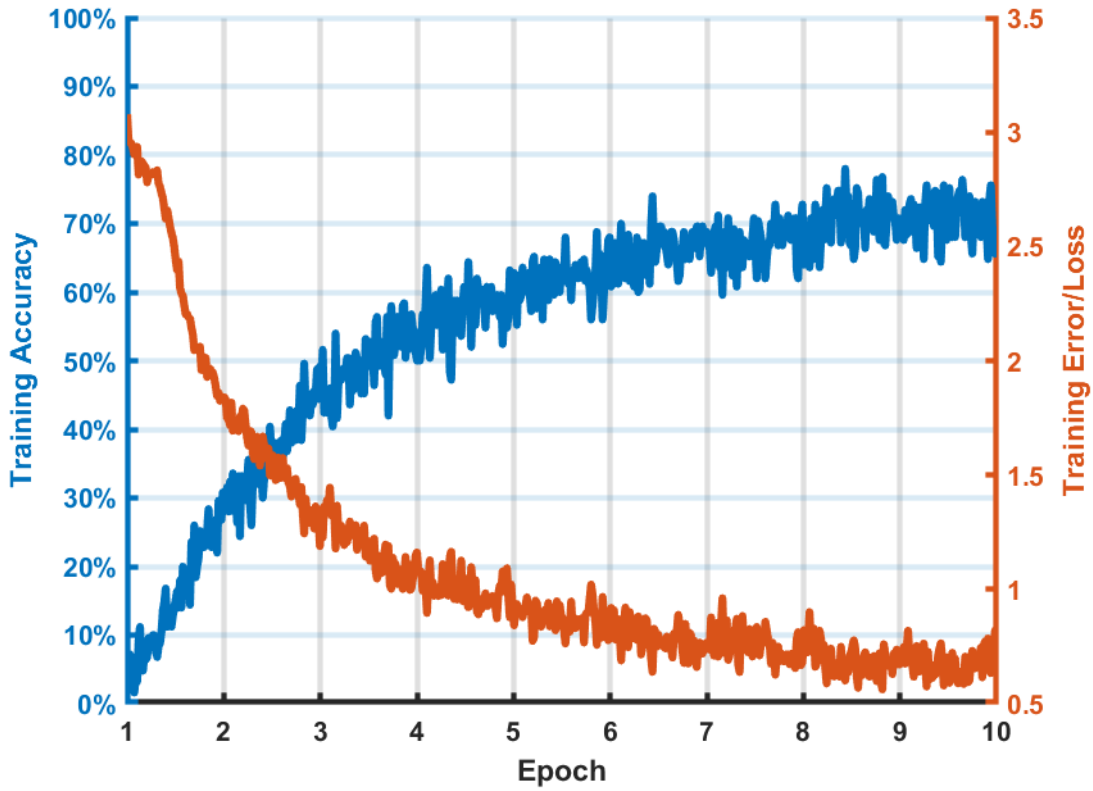


fig 3.2

Train Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	900	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	227	86	345	242	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	465	435	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	462	438	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	466	434	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	900	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	98	802	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	900	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	528	372	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	531	369	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	10	890	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	18	882	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	900	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	52	848	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	58	842	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	900	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	899

Table 3.1.8

Train Classification Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0.252222	0.095556	0.383333	0.268889	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0.516667	0.483333	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0.513333	0.486667	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0.517778	0.482222	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0.108889	0.891111	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0.586667	0.413333	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0.59	0.41	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0.011111	0.988889	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0.02	0.98	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0.057778	0.942222	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0.064444	0.935556	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.001111	0.998889

Table 3.1.9

Val Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	48	10	34	8	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	74	26	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	68	32	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	66	34	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	20	80	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	52	48	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	56	43	1	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	8	90	2	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	7	93	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	11	88	1	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	99

Table 3.1.10

Val Classification matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0.48	0.1	0.34	0.08	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0.74	0.26	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0.68	0.32	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0.66	0.34	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0.2	0.8	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0.52	0.48	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0.56	0.43	0.01	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0.08	0.9	0.02	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0.07	0.93	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0.11	0.88	0.01	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	0.99

Table 3.1.11

Test Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	223	87	324	366	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	484	516	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	452	548	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	477	523	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	1000	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	69	931	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	1000	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	633	367	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	653	347	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	33	967	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	69	919	12	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	1000	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	115	885	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	134	860	6	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1000	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	991

Table 3.1.12

Test Classification Matrix:

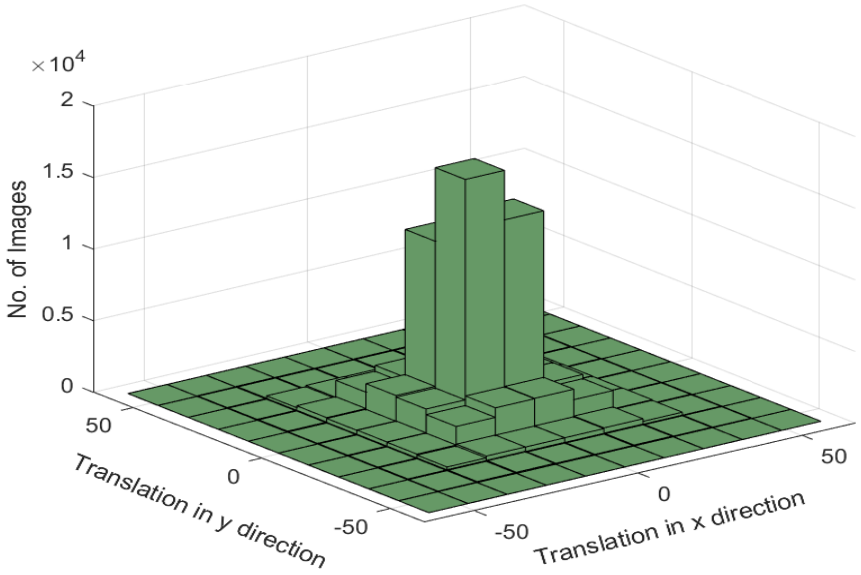
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0.223	0.087	0.324	0.366	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0.484	0.516	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0.452	0.548	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0.477	0.523	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0.069	0.931	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0.633	0.367	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0.653	0.347	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0.033	0.967	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0.069	0.919	0.012	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0.115	0.885	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0.134	0.86	0.006	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.009	0.991

Table 3.1.13

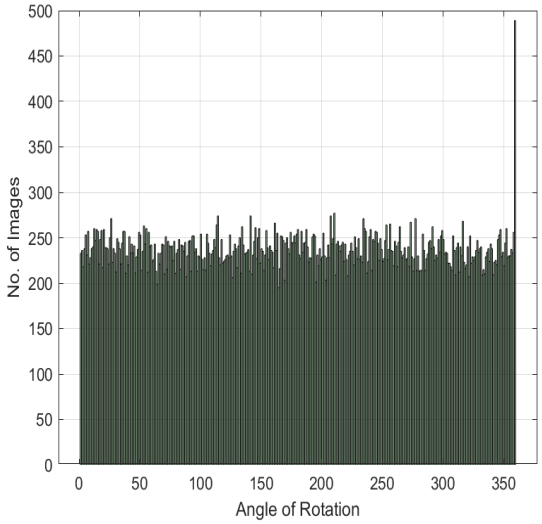
2) Data Augmentation:

We augmented original dataset 5 times to get augmented dataset. Every image has some random scaling, rotation and cropping.

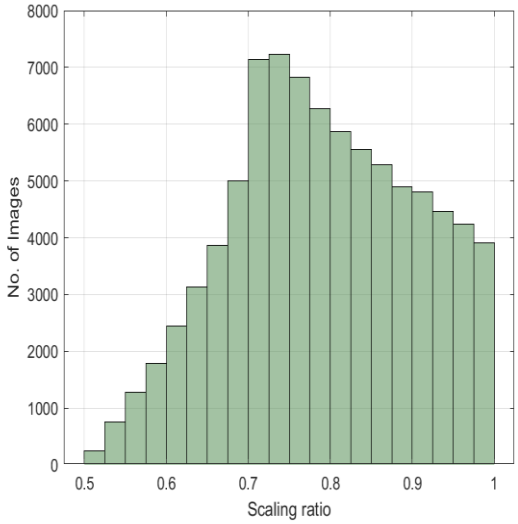
Translation Histogram:



Rotation Histogram



Scaling Histogram



3) Skinny ConvNet on Augmented dataset:

CONV -> ReLU -> POOL -> CONV -> ReLU -> POOL -> CONV -> ReLU -> POOL -> FC -> DROPOUT -> FC -> SOFTMAX

In this case we added more convnet layers keeping number of filters same.

Parameters	Values
Dataset size	85000
No epoch	10
Input Image Layer	128 x 128
Batch size	32
Learning rate	1e-3
1st conv layer	Filters: 5 x 5, filter size: 40, Padding: [2,2], Stride:[1,1]
2nd conv layer	Filters: 5 x 5, filter size: 40, Padding: [2,2], Stride:[1,1]
3rd conv layer	Filters: 3 x 3, filter size: 80, Padding: [2,2], Stride:[1,1]
Fully conncted layers (FC)	100
Drop-out	0.25
FC before classification	17

Result:

Epoch	Iteration	Time Elapsed (seconds)	Mini-batch Loss	Mini-batch Accuracy	Base Learning Rate
1	1	0.75	2.8311	9.38%	0.0010
1	50	8.01	2.8299	9.38%	0.0010
1	100	15.36	2.8289	6.25%	0.0010
1	150	22.75	2.8326	3.13%	0.0010
.....					
2	2400	360.19	2.3066	25.00%	0.0010
2	2450	367.69	2.4523	12.50%	0.0010
2	2500	375.18	2.3098	6.25%	0.0010
2	2550	382.66	2.5065	12.50%	0.0010
2	2600	390.10	2.2829	25.00%	0.0010
.....					
10	23800	3550.04	1.3111	59.38%	0.0010
10	23850	3557.49	1.0790	59.38%	0.0010
10	23900	3564.95	1.5053	50.00%	0.0010

Trained in in 3609.25 seconds

train_acc = 0.5356, val_acc =0.4232, test_acc =0.3678

This network tool lot of time to train. With augmentation, increase in training accuracy was expected. But in this case it is lesser, may be better hyper-parameter tuning is required. Sometimes it is also depends on distribution of augmented data. We followed some other technique to augment data, we got train accuracy of 80% , val accuracy as 75%. But Test accuracy was 15%. It shows that test set and train set distributions were different.

Train Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	2626	1874	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	280	4220	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	1339	2558	603	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	3834	666	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	4500	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	2872	1628	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	4500	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	2279	2221	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	1575	2925	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	4289	211	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	4135	365	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	3132	1368	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	4322	178	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	2700	1800	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	2046	2454	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2839	1661	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	949	3551

Table 3.2.1

Train Classification Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	0.583556	0.416444	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0.062222	0.937778	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0.297556	0.568444	0.134	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0.852	0.148	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0.638222	0.361778	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0.506444	0.493556	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0.35	0.65	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0.953111	0.046889	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0.918889	0.081111	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0.696	0.304	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0.960444	0.039556	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0.6	0.4	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0.454667	0.545333	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.630889	0.369111	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.210889	0.789111

Table 3.2.2

From classification matrix we can say that even though accuracy is less, most of labels are misclassified in adjacent classes.

Val Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	287	213	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	18	482	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	185	286	29	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	446	54	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	500	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	358	142	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	500	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	301	199	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	184	316	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	500	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	7	451	42	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	333	167	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	500	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	14	341	145	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	283	217	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	378	122	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	132	368

Table 3.2.3

Validation classification Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	0.574	0.426	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0.036	0.964	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0.37	0.572	0.058	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0.892	0.108	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0.716	0.284	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0.602	0.398	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0.368	0.632	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0.014	0.902	0.084	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0.666	0.334	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0.028	0.682	0.29	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0.566	0.434	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.756	0.244	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.264	0.736

Table 3.2.4

Test Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	537	445	18	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	1000	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	414	519	67	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	831	169	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	1000	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1000	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	8	992	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1000	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	293	687	20	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	1000	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	566	434	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	587	413	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	366	634	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	731	269	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	242	758	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	38	922	40	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	400	600

Table 3.2.5

Test Classification Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	0.537	0.445	0.018	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0.414	0.519	0.067	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0.831	0.169	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0.008	0.992	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0.293	0.687	0.02	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0.566	0.434	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0.587	0.413	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0.366	0.634	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0.731	0.269	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0.242	0.758	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0.038	0.922	0.04	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.4	0.6

Table 3.2.6

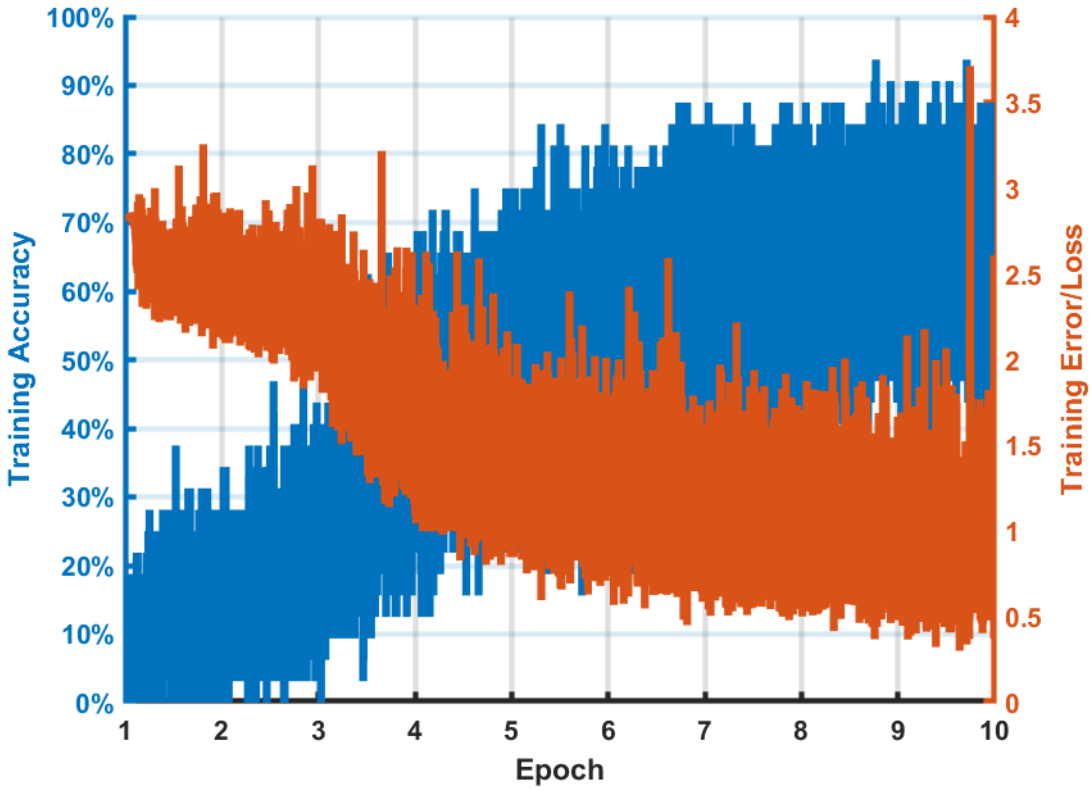


Fig 3.2

3) Wide ConvNet on Augmented dataset:

In this section we kept number of convolution layers same but increased number of filters.

Parameters	Values
Dataset size	85000
No epoch	10
Batch size	100
Input Image Layer	128 x 128
Learning rate	1e-3
1st conv layer	Filters: 5 x 5, filter size: 80, Padding: [2,2], Stride:[1,1]
2nd conv layer	Filters: 3 x 3 , filter size: 120, Padding: [2,2], Stride:[1,1]
Fully conncted layers (FC)	50
Drop-out %	0.25
FC before classification	17

Table 3.3

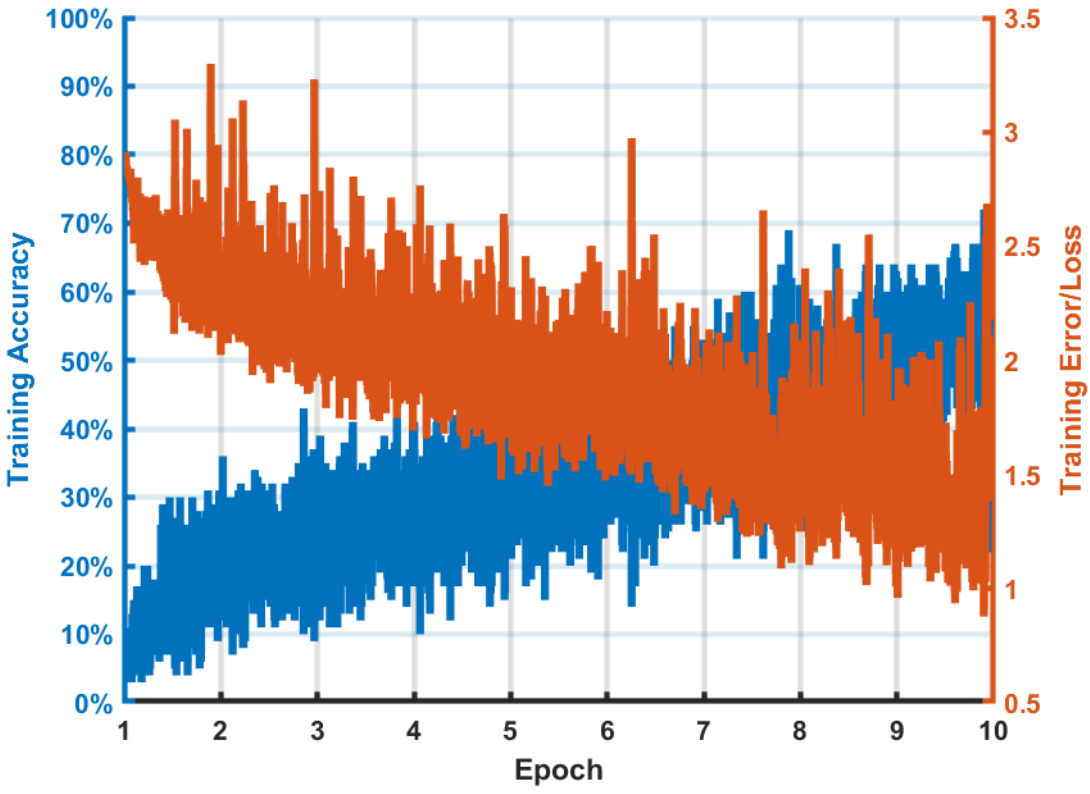


fig 3.3

Figure 3.3 shows that as number of epoch increases, accuracy increases and training error tends to decrease.

Result:

Epoch	Iteration	Time Elapsed (seconds)	Mini-batch Loss	Mini-batch Accuracy	Base Learning Rate
1	1	0.99	2.8931	5.00%	0.0010
1	50	23.81	2.8066	7.00%	0.0010
1	100	47.28	2.5741	8.00%	0.0010
1	150	70.63	2.5709	13.00%	0.0010
1	200	93.90	2.5543	10.00%	0.0010
.....					
2	800	374.04	2.3356	12.00%	0.0010
2	850	397.29	2.3585	21.00%	0.0010
2	900	420.53	2.1666	28.00%	0.0010
2	950	443.83	2.2411	26.00%	0.0010
.....					
10	7500	3488.15	1.3734	46.00%	0.0010
10	7550	3511.29	1.3940	48.00%	0.0010
10	7600	3534.43	1.6715	37.00%	0.0010
10	7650	3557.57	1.8535	40.00%	0.0010

Trained in in 3603.91 seconds

train_acc =0.3996, val_acc = 0.2338, test_acc =0.2331

By comparing the results of skinny network and wide network, we can conclude that Skinny network has more accuracy and low over-fitting. But training loss of wide network is lower than that of skinny network.

Training Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	4500	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	232	4268	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	534	1113	2581	272	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	3881	619	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	99	2924	1477	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	4500	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	4500	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	1330	3170	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	356	4144	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	3052	1448	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	1583	2917	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	289	1103	747	2361	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4500	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1162	3338	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4500	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4369	131
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4500

Table 3.3.1

Training Classification Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0.051556	0.948444	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0.118667	0.247333	0.573556	0.060444	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0.862444	0.137556	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0.022	0.649778	0.328222	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0.295556	0.704444	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0.079111	0.920889	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0.678222	0.321778	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0.351778	0.648222	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0.064222	0.245111	0.166	0.524667	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.258222	0.741778	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.970889	0.029111
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 3.3.2

Validation Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	500	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	19	481	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	115	103	282	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	34	431	35	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	27	346	127	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	500	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	500	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	151	349	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	9	491	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	311	189	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	91	335	74	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	48	53	399	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	500	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	65	435	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	500	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	500	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	34	466

Table 3.3.3

Validation Classification Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0.038	0.962	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0.23	0.206	0.564	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0.068	0.862	0.07	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0.054	0.692	0.254	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0.302	0.698	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0.018	0.982	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0.622	0.378	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0.182	0.67	0.148	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0.096	0.106	0.798	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.13	0.87	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.068	0.932

Table 3.3.4

Testing Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	465	535	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	680	279	41	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	539	461	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	463	151	386	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	504	496	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1000	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	935	65	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	642	358	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	1000	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	456	544	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	96	749	155	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	72	99	829	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	795	205	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1000	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1000	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	319	681

Table 3.3.5

Test Classification Matrix:

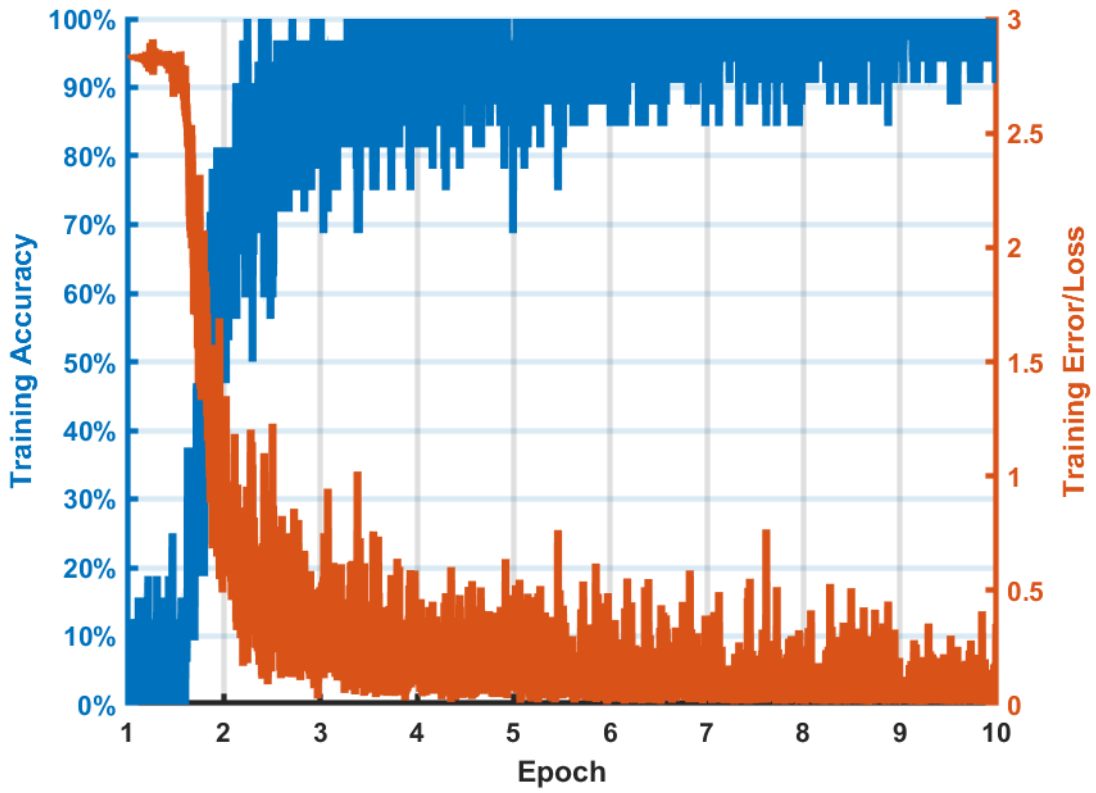
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0.465	0.535	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0.68	0.279	0.041	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0.539	0.461	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0.463	0.151	0.386	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0.504	0.496	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0.935	0.065	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0.642	0.358	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0.456	0.544	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0.096	0.749	0.155	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0.072	0.099	0.829	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.795	0.205	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.319	0.681

Table 3.3.6

4) Skinny Network on Original(non-Augmented) Dataset:

In this section we tried to find out effect of increased conv layers on original dataset.

Parameters	Values
Dataset size	17000
No epoch	10
Input Image Layer	128 x 128
Batch size	32
Learning rate	1e-3
1st conv layer	Filters: 5 x 5, filter size: 40, Padding: [2,2], Stride:[1,1]
2nd conv layer	Filters: 5 x 5, filter size: 40, Padding: [2,2], Stride:[1,1]
3rd conv layer	Filters: 3 x 3, filter size: 80, Padding: [2,2], Stride:[1,1]
Fully conncted layers (FC)	100
Drop-out	0.25
FC before classification	17



Result:

Epoch	Iteration	Time Elapsed (seconds)	Mini-batch Loss	Mini-batch Accuracy	Base Learning Rate
1	1	0.61	2.8311	3.13%	0.0010
1	50	8.08	2.8452	0.00%	0.0010
1	100	15.52	2.8115	9.38%	0.0010
1	150	22.98	2.8438	3.13%	0.0010
.....					
2	500	75.91	1.3788	40.63%	0.0010
2	550	83.42	0.9795	62.50%	0.0010
2	600	90.90	0.3626	90.63%	0.0010
2	650	98.42	0.5434	81.25%	0.0010
.....					
10	4700	710.06	0.4088	93.75%	0.0010
10	4750	717.57	0.0126	100.00%	0.0010
10	4780	722.08	0.2618	90.63%	0.0010

Trained in in 732.15 seconds

train_acc = 0.7871, val_acc =0.7700, test_acc =0.7654

Accuracy is much improved in skinny network as compare to single conv network on original dataset.

Train Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	900	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	7	879	14	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	882	18	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	889	11	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	887	13	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	881	19	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	882	18	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	900	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	73	827	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	75	825	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	85	815	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	91	809	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	891	9	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	891	9	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	900	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	900	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	896

table 3.4.1

Train Classification Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0.007778	0.976667	0.015556	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0.98	0.02	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0.987778	0.012222	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0.985556	0.014444	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0.978889	0.021111	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0.98	0.02	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0.081111	0.918889	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0.083333	0.916667	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0.094444	0.905556	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0.101111	0.898889	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0.99	0.01	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0.99	0.01	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.004444	0.995556

Table 3.4.2

Validation Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	94	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	92	8	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	95	5	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	95	5	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	91	9	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	92	8	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	2	98	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	2	98	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	6	94	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	10	90	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	98	2	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	98	2	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	99

Table 3.4.3

Validation Classification Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0.94	0.06	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0.92	0.08	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0.95	0.05	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0.95	0.05	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0.91	0.09	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0.92	0.08	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0.02	0.98	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0.02	0.98	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0.06	0.94	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0.1	0.9	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0.98	0.02	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0.98	0.02	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	0.99

Table 3.4.4

Test Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	7	973	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	974	26	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	984	16	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	982	18	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	972	28	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	974	26	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	1000	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	75	925	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	77	923	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	91	909	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	101	899	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	989	11	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	989	11	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1000	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1000	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	5	995

Table 3.4.5

Test Classification Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0.007	0.973	0.02	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0.974	0.026	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0.984	0.016	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0.982	0.018	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0.972	0.028	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0.974	0.026	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0.075	0.925	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0.077	0.923	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0.091	0.909	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0.101	0.899	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0.989	0.011	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0.989	0.011	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.005	0.995

Table 3.4.6

5) Wide Network on Original(non-Augmented) Dataset:

In this section we tried to find out effect of increased filters on original dataset.

Parameters	Values
Dataset size	17000
No epoch	10
Batch size	100
Input Image Layer	128 x 128
Learning rate	1e-3
1st conv layer	Filters: 5 x 5, filter size: 80, Padding: [2,2], Stride:[1,1]
2nd conv layer	Filters: 3 x 3 , filter size: 120, Padding: [2,2], Stride:[1,1]
Fully conncted layers (FC)	50
Drop-out %	0.25
FC before classification	17

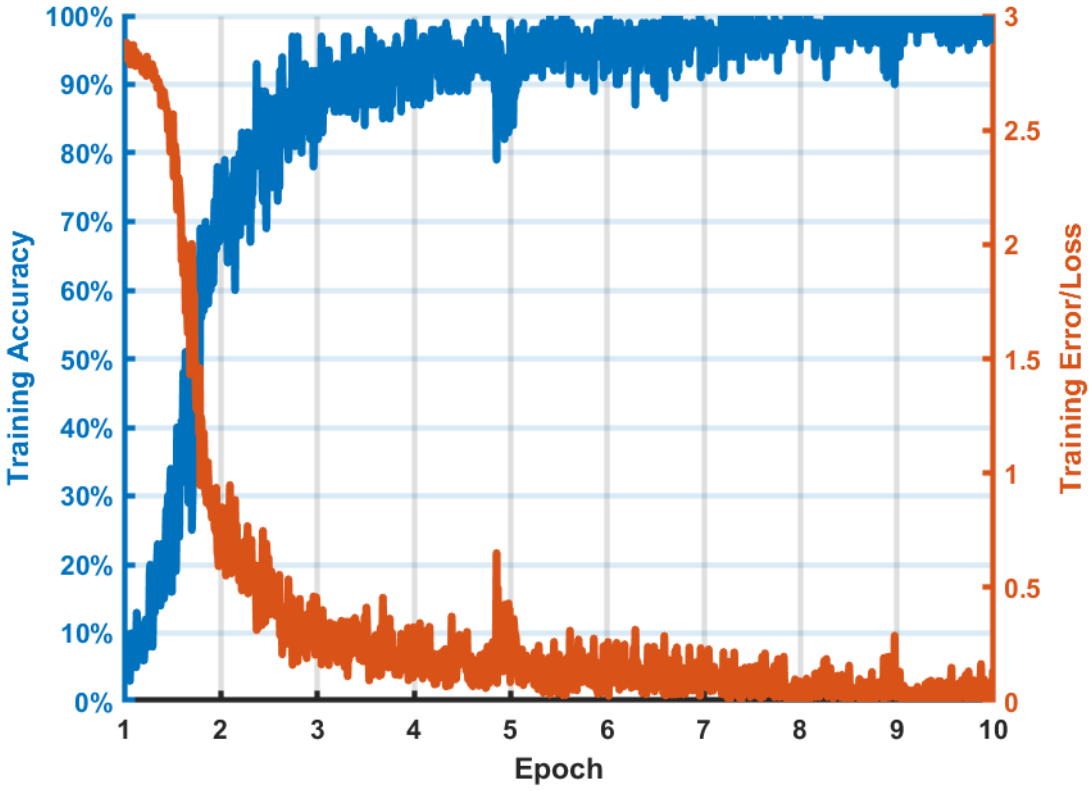


Fig 3.5

Result:

Epoch	Iteration	Time Elapsed (seconds)	Mini-batch Loss	Mini-batch Accuracy	Base Learning Rate
1	1	2.58	2.8177	6.00%	0.0010
1	50	27.09	2.7893	11.00%	0.0010
1	100	50.72	2.1676	35.00%	0.0010
1	150	74.46	0.8994	67.00%	0.0010
2	200	99.48	0.6065	75.00%	0.0010
2	250	123.29	0.6617	74.00%	0.0010
2	300	146.99	0.3864	81.00%	0.0010
3	350	172.23	0.4013	83.00%	0.0010
3	400	196.12	0.2432	93.00%	0.0010
3	450	219.85	0.1121	95.00%	0.0010
4	500	245.22	0.3242	88.00%	0.0010
4	550	268.92	0.0771	97.00%	0.0010
4	600	292.70	0.2300	95.00%	0.0010
5	650	317.95	0.1915	92.00%	0.0010
5	700	341.52	0.0950	96.00%	0.0010
5	750	365.18	0.0311	100.00%	0.0010
6	800	390.00	0.2726	92.00%	0.0010
6	850	413.60	0.0417	100.00%	0.0010
6	900	437.23	0.3156	87.00%	0.0010
7	950	461.85	0.1110	93.00%	0.0010
7	1000	485.28	0.1105	97.00%	0.0010
7	1050	508.84	0.1234	96.00%	0.0010
8	1100	533.75	0.0985	97.00%	0.0010
8	1150	557.24	0.1628	95.00%	0.0010
8	1200	580.69	0.0519	98.00%	0.0010
9	1250	605.29	0.0854	97.00%	0.0010
9	1300	628.77	0.0746	96.00%	0.0010
9	1350	652.27	0.0536	100.00%	0.0010
10	1400	676.84	0.0401	99.00%	0.0010
10	1450	700.29	0.0140	100.00%	0.0010
10	1500	723.77	0.0462	97.00%	0.0010
10	1530	737.97	0.1195	96.00%	0.0010

Trained in in 742.95 seconds

train_acc = 0.7927, val_acc = 0.7694, test_acc = 0.7904

On original dataset, wide network has slightly better accuracy as compare to skinny network because skinny network requires large data to perform well.

Train Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	900	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	891	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	889	11	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	889	11	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	889	11	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	900	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	66	834	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	68	832	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	63	837	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	63	837	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	68	827	5	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	900	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	900	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	1	899	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	1	899	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	893	7
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	900

Table 3.5.1

Train Classification Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0.99	0.01	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0.987778	0.012222	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0.987778	0.012222	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0.987778	0.012222	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0.073333	0.926667	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0.075556	0.924444	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0.07	0.93	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0.07	0.93	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0.075556	0.918889	0.005556	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0.001111	0.998889	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0.001111	0.998889	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.992222	0.007778
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 3.5.2

Validation Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	4	90	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	94	6	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	95	5	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	95	5	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	100	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	6	94	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	4	95	1	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	99	1	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	6	91	3	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	1	97	2	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	98	2	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	94	6
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	100

Table 3.5.3

Validation Classification Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0.04	0.9	0.06	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0.94	0.06	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0.95	0.05	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0.95	0.05	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0.06	0.94	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0.04	0.95	0.01	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0.99	0.01	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0.06	0.91	0.03	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0.01	0.97	0.02	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0.98	0.02	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.94	0.06
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

Table 3.5.6

Test confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	4	981	15	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	983	17	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	984	16	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	984	16	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1000	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	72	928	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	72	928	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	62	938	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	62	938	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	74	918	8	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	1000	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	1	998	1	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	999	1	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1000	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	987	13
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1000

Test Classification Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0.004	0.981	0.015	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0.983	0.017	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0.984	0.016	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0.984	0.016	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0.072	0.928	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0.072	0.928	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0.062	0.938	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0.062	0.938	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0.074	0.918	0.008	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0.001	0.998	0.001	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0.999	0.001	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.987	0.013
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1

6) Skinny model from augmented dataset transfer learning on original (non-Augmented)dataset:

Since our skinny model workspace on augmented dataset is ready, we can use transfer learning and see the results of that model on non-augmented dataset.

Epoch	Iteration	Time Elapsed (seconds)	Mini-batch Loss	Mini-batch Accuracy	Base Learning Rate
1	1	0.35	4.6534	18.75%	0.0010
1	50	7.63	2.9378	9.38%	0.0010
1	100	15.04	2.7158	6.25%	0.0010
1	150	22.41	2.8145	6.25%	0.0010
1	200	29.77	2.3552	37.50%	0.0010
1	250	37.14	1.7074	40.63%	0.0010
1	300	44.51	1.5819	53.13%	0.0010
1	350	51.94	1.3863	56.25%	0.0010
1	400	59.34	1.0234	62.50%	0.0010
1	450	66.75	1.0247	56.25%	0.0010
2	500	74.51	0.7591	81.25%	0.0010
2	550	81.94	0.6270	68.75%	0.0010
2	600	89.34	1.4823	59.38%	0.0010
2	650	96.74	0.7774	65.63%	0.0010
2	700	104.15	0.6153	84.38%	0.0010
2	750	111.55	0.4725	75.00%	0.0010
2	800	118.96	0.4661	87.50%	0.0010
2	850	126.36	0.5089	78.13%	0.0010
2	900	133.77	0.1644	93.75%	0.0010
2	950	141.18	0.4175	87.50%	0.0010

.....

4	1450	215.86	0.3492	84.38%	0.0010
4	1500	223.26	0.1964	90.63%	0.0010
4	1550	230.67	0.2859	87.50%	0.0010
4	1600	238.08	0.0563	96.88%	0.0010
4	1650	245.48	0.3625	90.63%	0.0010
4	1700	252.92	0.0176	100.00%	0.0010
4	1750	260.34	0.1719	96.88%	0.0010
4	1800	267.75	0.1888	84.38%	0.0010
4	1850	275.17	0.1489	93.75%	0.0010
4	1900	282.57	0.1964	96.88%	0.0010
5	1950	290.50	0.0820	93.75%	0.0010
5	2000	297.91	0.0744	96.88%	0.0010
5	2050	305.32	0.1725	93.75%	0.0010
5	2100	312.72	0.1723	90.63%	0.0010
5	2150	320.13	0.0994	96.88%	0.0010
5	2200	327.54	0.1304	93.75%	0.0010
5	2250	334.96	0.1279	96.88%	0.0010
5	2300	342.38	0.3402	87.50%	0.0010
5	2350	349.79	0.2248	90.63%	0.0010
5	2390	355.72	0.2562	93.75%	0.0010

Trained in in 361.44 seconds

train_acc =0.9371, val_acc =0.9076, test_acc = 0.9341

We can clearly see that test accuracy on non augmented dataset increased by 20% because of transfer learning.

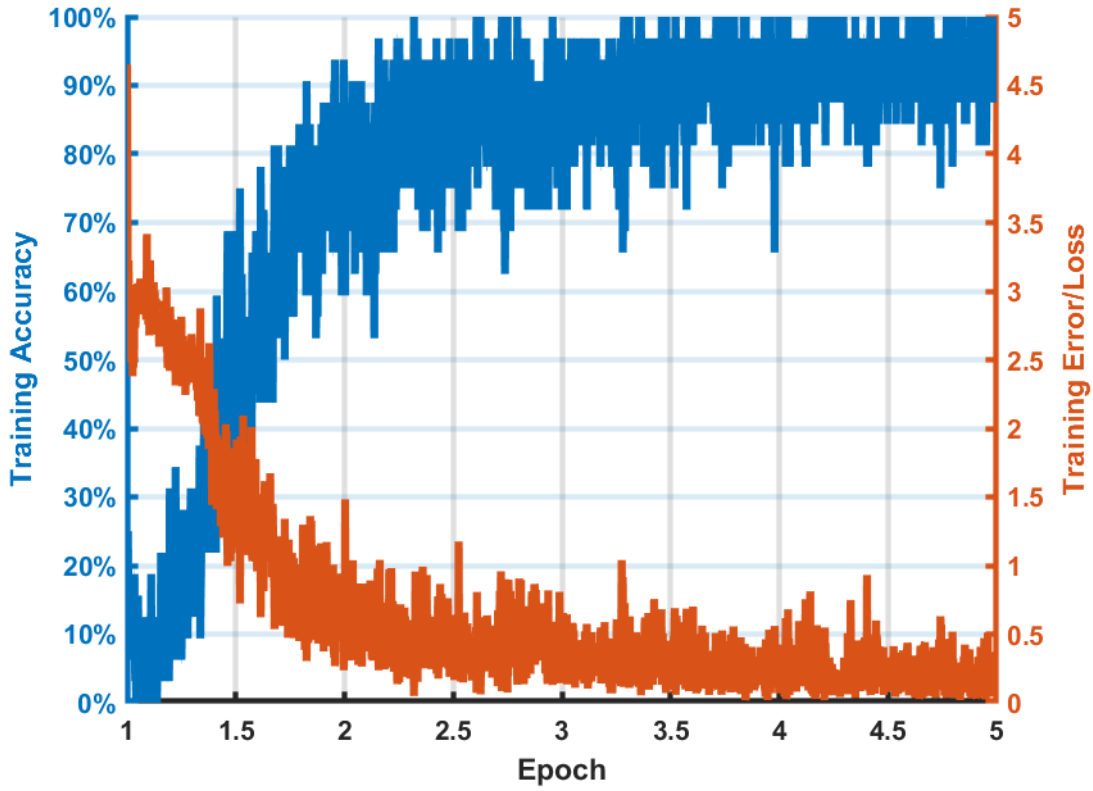


Fig 3.6

Training Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	899	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	900	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	10	822	68	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	843	57	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	786	114	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	711	189	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	730	170	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	651	249	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	707	193	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	704	196	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	900	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	39	860	1	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	886	14	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	883	17	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	898	2	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	900	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	892

Table 3.6.1

Training Classification Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	0.998889	0.001111	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0.011111	0.913333	0.075556	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0.936667	0.063333	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0.873333	0.126667	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0.79	0.21	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0.811111	0.188889	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0.723333	0.276667	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0.785556	0.214444	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0.782222	0.217778	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0.043333	0.955556	0.001111	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0.984444	0.015556	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0.981111	0.018889	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.997778	0.002222	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.008889	0.991111

Table 3.6.2

Val Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	100	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	2	93	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	81	19	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	84	16	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	81	19	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	71	29	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	72	28	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	55	45	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	58	42	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	66	34	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	98	2	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	100	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	1	99	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	4	96	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	4	96	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	99	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	99

Table 3.6.3

Val Classification:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0.02	0.93	0.05	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0.81	0.19	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0.84	0.16	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0.81	0.19	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0.71	0.29	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0.72	0.28	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0.55	0.45	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0.58	0.42	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0.66	0.34	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0.98	0.02	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0.01	0.99	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0.04	0.96	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0.04	0.96	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	0.99	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.01	0.99

Table 3.6.4

Test Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	1	999	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	5	908	87	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	927	73	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	867	133	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	782	218	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	802	198	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	706	294	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	765	235	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	770	230	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	1000	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	37	963	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	990	10	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	987	13	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	999	1	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1000	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	9	991

Test Classification Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0.001	0.999	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0.005	0.908	0.087	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0.927	0.073	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0.867	0.133	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0.782	0.218	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0.802	0.198	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0.706	0.294	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0.765	0.235	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0.77	0.23	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0.037	0.963	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0.99	0.01	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0.987	0.013	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.999	0.001	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.009	0.991

7) Wide model from augmented dataset transfer learning on original (non-Augmented)dataset:

Since our wide model workspace on augmented dataset is ready, we can use transfer learning and see the results of that model on non-augmented dataset.

Epoch	Iteration	Time Elapsed (seconds)	Mini-batch Loss	Mini-batch Accuracy	Base Learning Rate
1	1	1.00	1.7026	43.00%	0.0010
1	50	23.67	1.4188	55.00%	0.0010
1	100	46.84	1.4350	41.00%	0.0010
1	150	69.97	1.4360	54.00%	0.0010
1	200	93.17	1.3571	53.00%	0.0010
1	250	116.41	1.5503	52.00%	0.0010
1	300	139.64	1.3765	52.00%	0.0010
1	350	162.86	1.3367	53.00%	0.0010

.....

2	800	373.70	1.3344	57.00%	0.0010
2	850	396.97	1.0633	61.00%	0.0010
2	900	420.26	1.0748	62.00%	0.0010
2	950	443.54	1.3498	47.00%	0.0010
2	1000	466.79	1.1151	64.00%	0.0010
2	1050	490.03	1.0638	62.00%	0.0010
2	1100	513.30	1.2679	47.00%	0.0010
2	1150	536.56	1.3848	51.00%	0.0010

.....

5	3100	1448.89	0.8680	71.00%	0.0010
5	3150	1472.30	0.8520	66.00%	0.0010
5	3200	1495.73	0.7744	79.00%	0.0010
5	3250	1519.25	1.0553	62.00%	0.0010
5	3300	1542.54	1.2633	58.00%	0.0010
5	3350	1565.92	1.1618	62.00%	0.0010
5	3400	1589.29	1.2138	62.00%	0.0010
5	3450	1612.70	0.9761	67.00%	0.0010
5	3500	1635.94	1.3422	60.00%	0.0010
5	3550	1659.13	1.2662	61.00%	0.0010
5	3600	1682.32	1.1553	57.00%	0.0010
5	3650	1705.54	0.9942	64.00%	0.0010
5	3700	1728.71	1.0561	70.00%	0.0010
5	3750	1751.93	1.0199	67.00%	0.0010
5	3800	1775.13	1.1977	59.00%	0.0010
5	3825	1786.74	1.0009	65.00%	0.0010

Trained in in 1834.13 seconds

train_acc = 0.7553, val_acc = 0.2641, test_acc = 0.2203

Result clearly shows that from wide network transfer learning we got more variance as compare to skinny model results and reason is same that wider network has already lower accuracy as compare to skinny network. But accuracy of this network is still better than that of wide network accuracy on augmented dataset.

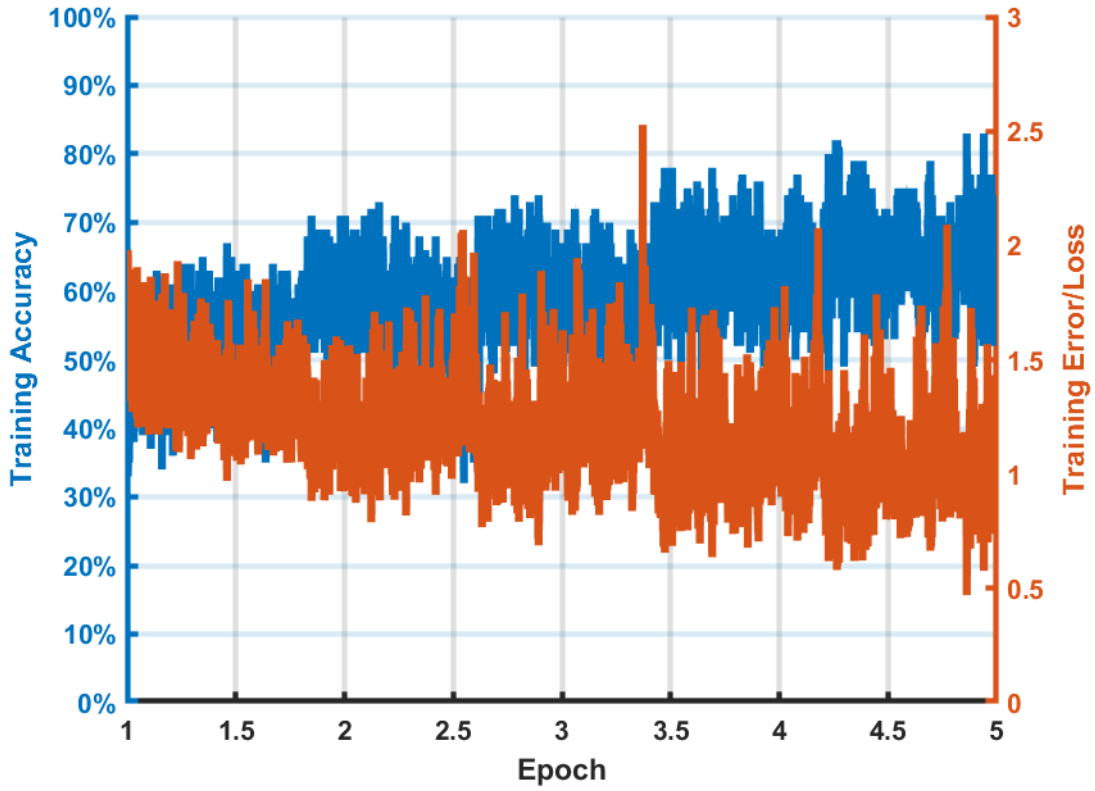


fig 3.7

Training Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	4500	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	844	3656	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	2855	1645	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	2766	1734	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	3899	601	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	3820	680	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	3203	1297	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	3222	1278	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	2799	1701	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	1322	2430	748	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	4500	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	51	3788	661	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	4500	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	186	4314	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	52	3791	657	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4500	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	88	4412

Table 3.7.1

Train Classification Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0.187556	0.812444	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0.634444	0.365556	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0.614667	0.385333	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0.866444	0.133556	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0.848889	0.151111	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0.711778	0.288222	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0.716	0.284	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0.622	0.378	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0.293778	0.54	0.166222	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0.011333	0.841778	0.146889	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0.041333	0.958667	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0.011556	0.842444	0.146	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.019556	0.980444

Table 3.7.2

Val Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	500	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	158	342	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	492	8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	500	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	18	482	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	183	317	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	124	376	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	56	444	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	53	407	40	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	132	200	168	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	465	35	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	352	148	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	500	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	49	451	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	8	335	157	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	500	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	58	442

Table 3.7.3

Validation Classification matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0.316	0.684	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0.984	0.016	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0.036	0.964	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0.366	0.634	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0.248	0.752	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0.112	0.888	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0.106	0.814	0.08	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0.264	0.4	0.336	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0.93	0.07	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0.704	0.296	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0.098	0.902	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0.016	0.67	0.314	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.116	0.884

Table 3.7.4

Test Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	831	169	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	1000	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	731	269	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	1000	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	78	922	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	665	335	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	450	550	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	519	481	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	740	260	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	411	260	329	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	83	917	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	494	506	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	259	741	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	335	633	32	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	472	528	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	502	498

Table 3.7.5

Test Classification Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0.831	0.169	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0.731	0.269	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0.078	0.922	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0.665	0.335	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0.45	0.55	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0.519	0.481	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0.74	0.26	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0.411	0.26	0.329	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0.083	0.917	0	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0.494	0.506	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0.259	0.741	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0.335	0.633	0.032	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.472	0.528	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.502	0.498

Table 3.7.6

3.2 Comparison between models and dataset

Model	Train Accuracy	Val Accuracy	Test Accuracy
1. Basic network with α = 5e-4	0.7561	0.7082	0.7068
2. Basic network with α = 1e-4	0.7751	0.7162	0.7148
3. Skinny network on augmented dataset	0.5356	0.4232	0.3678
4. wide network on augmented dataset	0.3996	0.2338	0.2331
5. Skinny network on non-augmented(original) dataset	0.7871	0.7700	0.7654
6. Wide network on non-augmented(original) dataset	0.7927	0.7694	0.7904
7. Skinny network transfer learning	0.9371	0.9076	0.9341
8. Wide network transfer learning	0.7553	0.2641	0.2203

1. Consider first two, rows, as learning rate decreased, accuracy slightly increased.
2. Consider row 3 and 4, skinny network on augmented data outperformed over wide network. In fact time required for skinny network on augmented data is less that wide network. Tough if we consider last epoch of both models, training loss of wider network is less than skinny network. It means that rate of increase in accuracy is not always proportional to rate of decrease in training loss.
3. From row 5 and 6, we can observe that wider network performed well on original dataset. It's test accuracy is impressive.
4. Both skinny and wider networks achieved far better training accuracy on transfer learning. As dataset is large which is healthy for skinny network, it gave high accuracy on test data too. Overall, due to transfer learning, accuracy of our model increases. But wider network got some over-fitting issue.
5. For skinny and wide network, we increased learning rate, still it took approx 23000 sec for convergence. Reduce learning rate might have given better accuracy.

3.3 Extra Credit

1)Alexnet (Transfer learning):

Rather than learning whole model, we used pre-trained model alexnet which is widely proven to be one of the best transfer learning model. We used alexnet inbuilt model from neural network kit of Matlab.

Layers: We have already covered layer structure of alexnet in section 2.4. We will not use last 3 layers of alexnet as our classification layer requires 17 inputs.

Dataset = 85000

No of Epoch =10

Learning rate = 1e-3

Result:

Epoch	Iteration	Time Elapsed (seconds)	Mini-batch Loss	Mini-batch Accuracy	Base Learning Rate
1	1	2.48	3.5114	5.00%	0.0005
1	50	52.87	2.3469	8.00%	0.0005
1	100	97.26	2.4787	10.00%	0.0005
1	150	141.74	2.1399	19.00%	0.0005
1	200	185.58	1.8773	32.00%	0.0005
1	250	229.90	1.9201	30.00%	0.0005
.....					
2	800	635.67	1.0509	58.00%	0.0005
2	850	669.09	1.2098	46.00%	0.0005
2	900	701.88	1.1531	47.00%	0.0005
2	950	735.17	1.1440	52.00%	0.0005
.....					
10	7450	7242.35	0.3169	88.00%	0.0005
10	7500	7275.73	0.1950	90.00%	0.0005
10	7550	7309.41	0.3947	82.00%	0.0005
10	7600	7343.16	0.2281	91.00%	0.0005
10	7650	7375.59	0.3454	90.00%	0.0005

Trained in in 7516.57 seconds

Train Accuracy= 0.9063 Test Accuracy= 0.8742 Validation Accuracy=0.8721

Performance of alex model is best as compare to other models we saw in this project. It took more time than wider network because it has five convolution layer which takes time to run the network.

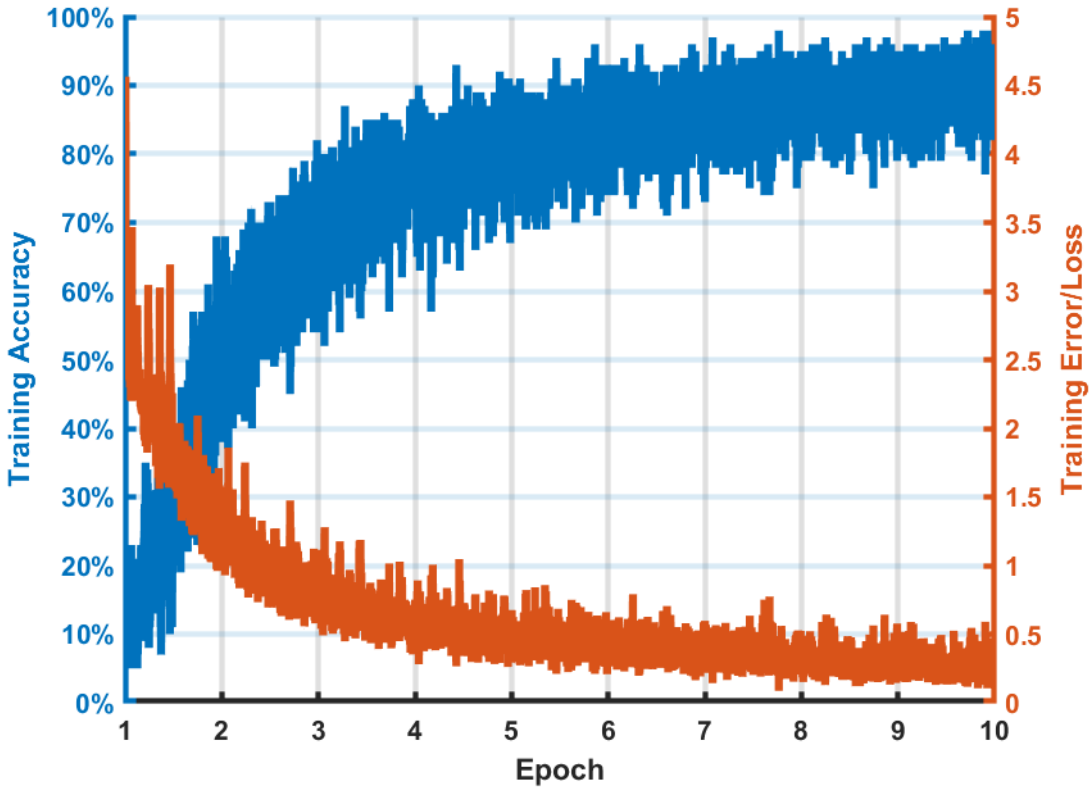


fig 3.8

Train Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	4327	173	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	3680	820	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	3624	876	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	3623	877	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	3819	681	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	3621	879	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	3637	863	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	4466	34	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	3979	521	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	4500	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	154	4346	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	443	3738	319	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	4500	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	144	4356	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	315	4185	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1319	3181	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	269	4231

Table 3.8.1

Train Classification Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	0.961556	0.038444	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0.817778	0.182222	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0.805333	0.194667	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0.805111	0.194889	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0.848667	0.151333	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0.804667	0.195333	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0.808222	0.191778	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0.992444	0.007556	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0.884222	0.115778	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0.034222	0.965778	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0.098444	0.830667	0.070889	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0.032	0.968	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0.07	0.93	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.293111	0.706889	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.059778	0.940222

Table 3.8.2

Val Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	448	52	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	394	106	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	388	112	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	377	123	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	389	111	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	374	126	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	391	109	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	500	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	16	438	46	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	500	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	42	458	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	67	380	53	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	488	12	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	500	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	24	476	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	176	324	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	56	444

Table 3.8.3

Val Classification Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	0.896	0.104	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0.788	0.212	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0.776	0.224	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0.754	0.246	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0.778	0.222	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0.748	0.252	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0.782	0.218	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0.032	0.876	0.092	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0.084	0.916	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0.134	0.76	0.106	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0.976	0.024	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0.048	0.952	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.352	0.648	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.112	0.888

Table 3.8.4

Test Classification Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	922	78	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	755	245	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	796	204	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	768	232	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	785	215	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	736	264	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	779	221	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	704	296	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	638	362	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	704	296	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	858	142	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	938	62	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	1000	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	103	897	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	261	739	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	425	575	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	179	821

Table 3.8.5

Test Confusion Matrix:

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q
1	0.922	0.078	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0.755	0.245	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0.796	0.204	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0.768	0.232	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0.785	0.215	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0.736	0.264	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0.779	0.221	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0.704	0.296	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0.638	0.362	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0.704	0.296	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0.858	0.142	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0.938	0.062	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0.103	0.897	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0.261	0.739	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.425	0.575	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.179	0.821

Table 3.8.6

Comparison with Alexnet:

Model	**Train Accuracy**	**Val Accuracy**	**Test Accuracy**
Skinny Network	0.5356	0.4232	0.3678
Wide Network	0.3996	0.2338	0.2331
Alexnet	0.9063	0.8721	0.8742

2) Visualisation of 1st Layer of every model:

a) Basic (single convolution layer) model:

Basic model first layer visualisation

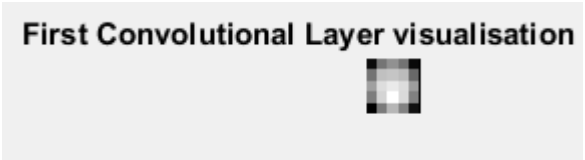
Pyramid Level	Iteration	Activation Strength
1	1	1.44
1	2	1.70
1	3	1.95
1	4	2.21
1	5	2.46
1	6	2.72
1	7	2.98
1	8	3.23
1	9	3.49
1	10	3.74

b) Skinny network on augmented dataset

First Convolutional Layer visualisation for Skinny Aug

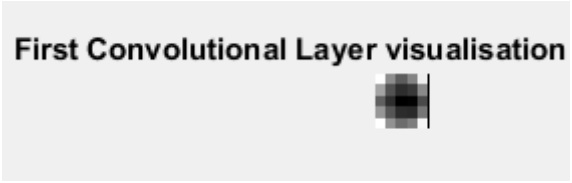
Pyramid Level	Iteration	Activation Strength
1	1	16.25
1	2	14.68
1	3	13.11
1	4	11.54
1	5	9.97
1	6	8.40
1	7	6.82
1	8	5.25
1	9	3.68
1	10	2.11

c) wide network on augmented dataset



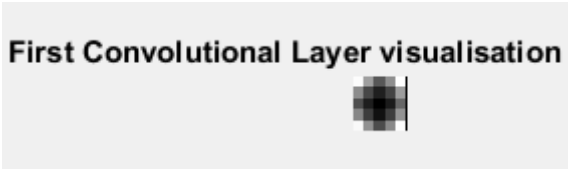
Pyramid Level	Iteration	Activation Strength
1	1	31.27
1	2	34.33
1	3	37.38
1	4	40.43
1	5	43.48
1	6	46.54
1	7	49.59
1	8	52.64
1	9	55.69
1	10	58.75

d) Skinny network on non-augmented(original) dataset



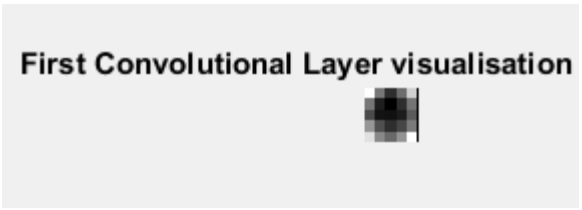
Pyramid Level	Iteration	Activation Strength
1	1	34.62
1	2	31.26
1	3	27.90
1	4	24.54
1	5	21.17
1	6	17.81
1	7	14.45
1	8	11.09
1	9	7.73
1	10	4.36

e) Wide network on non-augmented(original) dataset



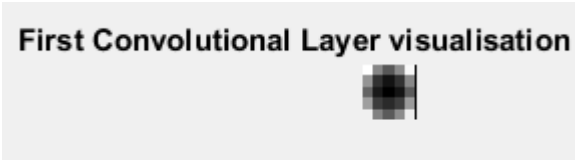
Pyramid Level	Iteration	Activation Strength
1	1	55.04
1	2	49.74
1	3	44.43
1	4	39.12
1	5	33.82
1	6	28.51
1	7	23.21
1	8	17.90
1	9	12.59
1	10	7.29

f) Skinny network transfer learning:



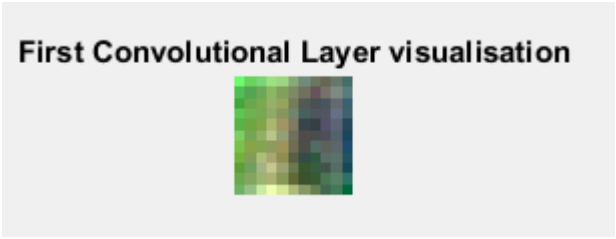
Pyramid Level	Iteration	Activation Strength
1	1	18.35
1	2	16.57
1	3	14.79
1	4	13.01
1	5	11.23
1	6	9.45
1	7	7.67
1	8	5.88
1	9	4.10
1	10	2.32

g) Wide network transfer learning



Pyramid Level	Iteration	Activation Strength
1	1	26.58
1	2	24.03
1	3	21.47
1	4	18.91
1	5	16.35
1	6	13.79
1	7	11.23
1	8	8.68
1	9	6.12
1	10	3.56

h) Alexnet :



Pyramid Level	Iteration	Activation Strength
1	1	116.39
1	2	104.19
1	3	92.00
1	4	79.81
1	5	67.62
1	6	55.43
1	7	43.23
1	8	31.04
1	9	18.85
1	10	6.66

3) t-SNE Visualisation:

a) Skinny Network Visualisation:

Train data:

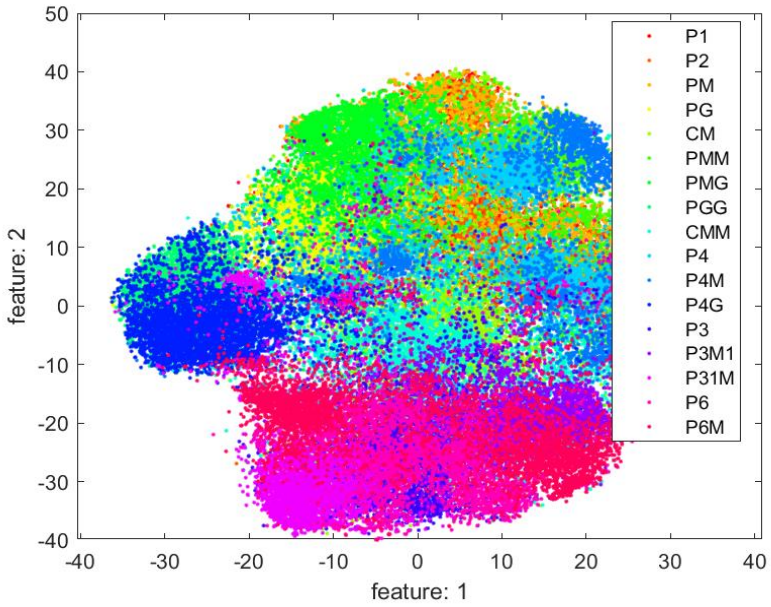


fig 3.9.1

Val data:

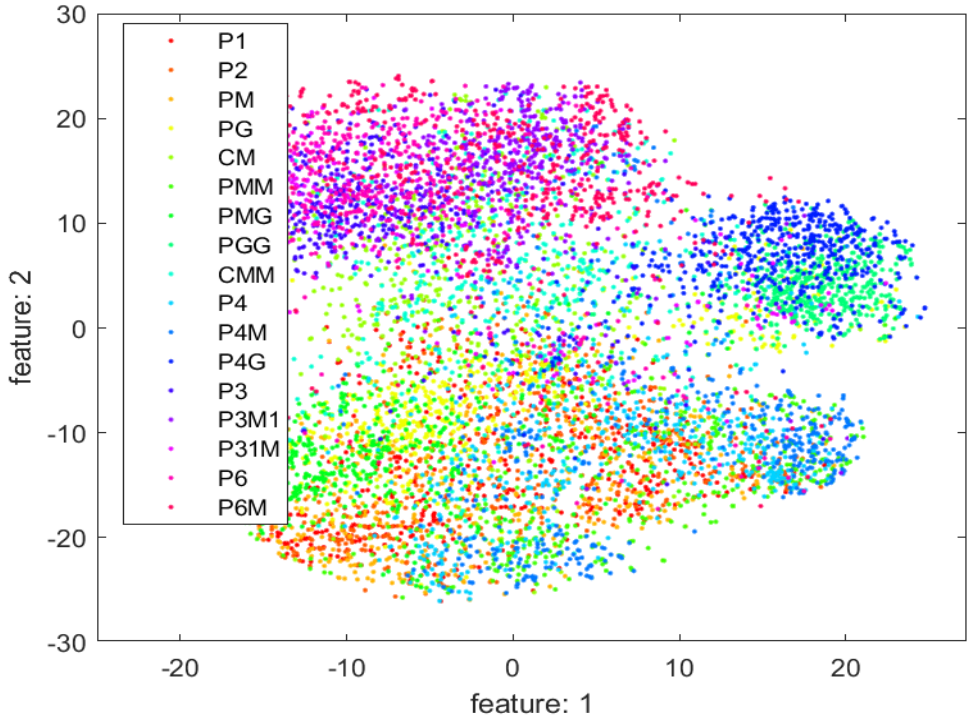


fig 3.9.2

Test Data:

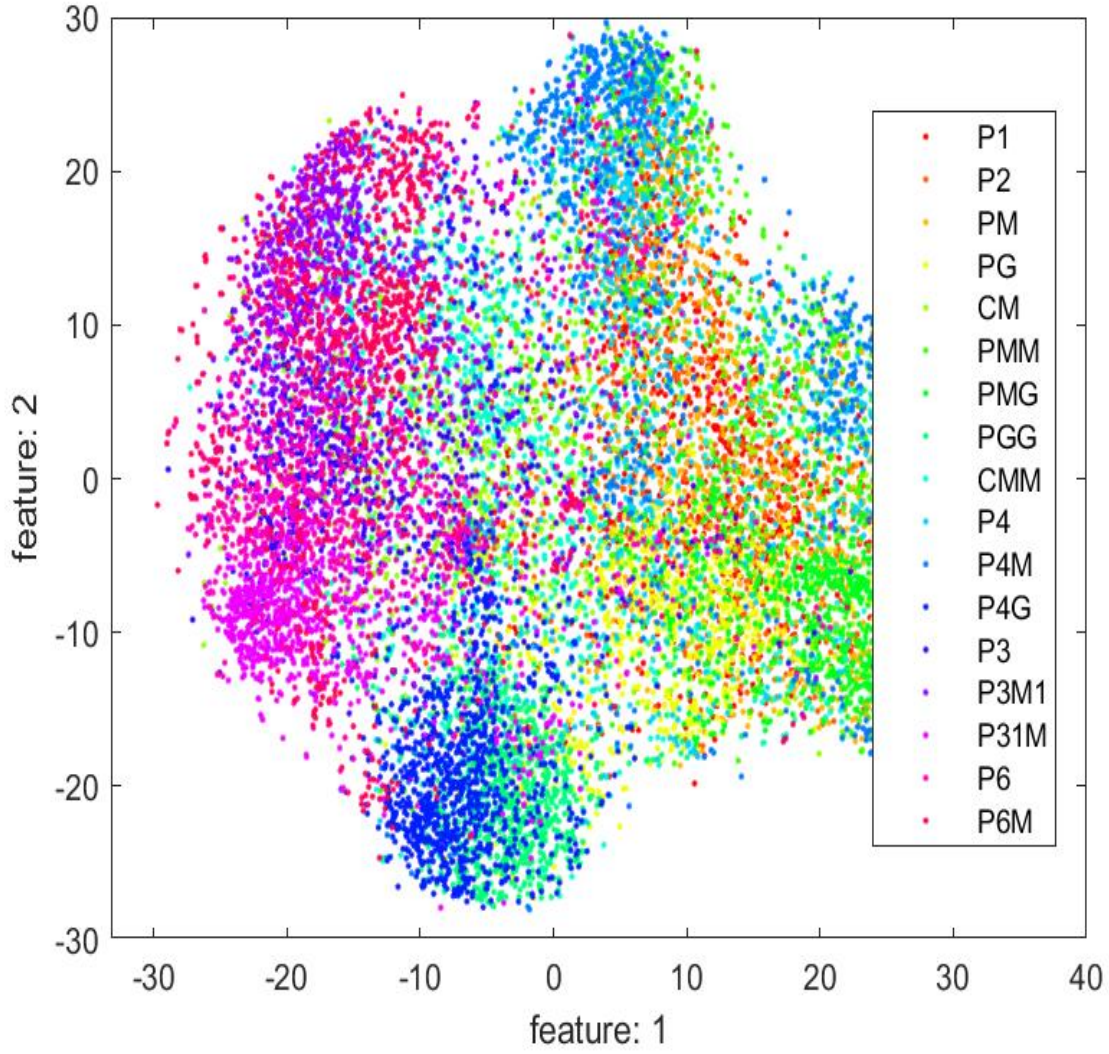


fig 3.9.3

b) Wide Network:

Train Data:

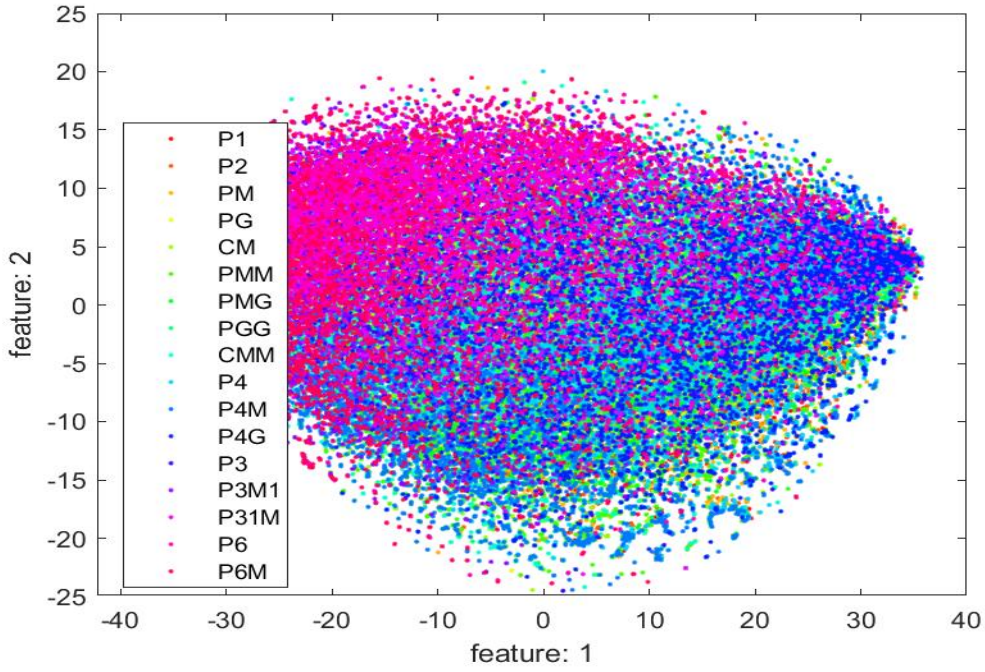


fig 3.9.4

Val data:

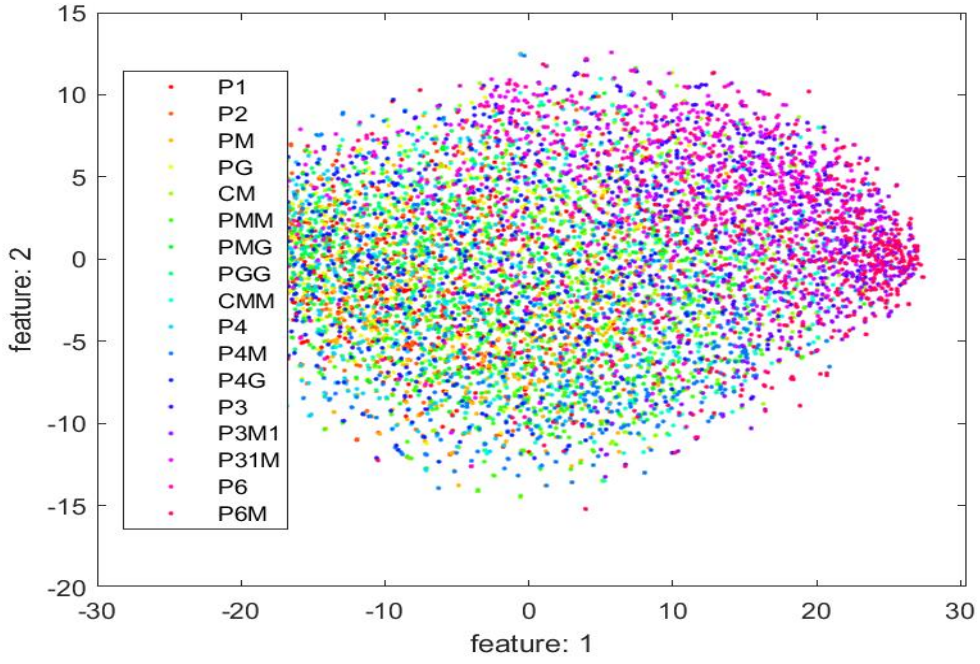


fig 3.9.5

Test data:

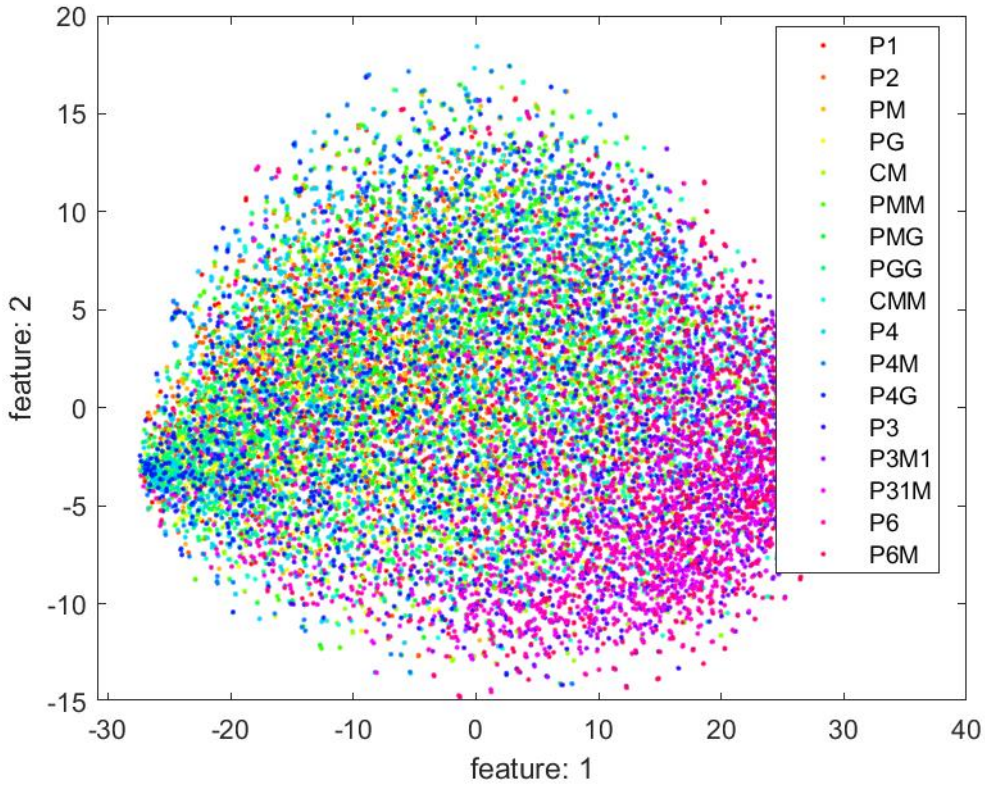


fig 3.9.6

4. Conclusion

1. Because of parameter sharing and sparsity of connection, convolution neural net is efficient rather than only multi-layer neural network. Number of parameters and their computation is far less when we use convolution neural network.

2. Data augmentation come out to be best way to multiply your dataset. Applications where huge dataset can't be obtained, are best suitable for data augmentation. But while augmenting data, care must be taken that it should not change the distribution of data. It must be aligned to distribution of test set data.

3. Low learning rate makes model slow, but it gives more accuracy. Reason is that in stochastic gradient descent process, it takes low steps to reach global minima. Being said that, it should not be too much less that gradient may vanish. Sometime we need to trade-off between accuracy and time for convergence of gradient. In our case lesser learning rate has little higher accuracy than higher learning rate and also it takes more time. If time is constrain, we should go for higher learning rate. In case of convolution neural net, hyper-parameter tuning should be done on validation data rather than test data and maximum portion of dataset should be given to training data as it requires more information to tune the parameters of model.

4. When we have hug data, we should go for skinny network rather than wide network because wide network has more filters which can give nuance from image. But it has lesser layers, so training of this information not properly processed. In other hand skinny network has multiple convolution layers, which can learn nuance details using filters and it also filters on previous layer output for better understanding of insight of data. If dataset is not big enough rather than going with skinny net, wide network could be useful.

5. Dealing with convolution neural net requires hours of run time, resources like GPU, high end devices. In such case, transfer learning could be more helpful as pre-processed networks like Alexnet, VGG, ResNet has high accuracy and it saves end to end run computation time and resources.

5. References

[1] Wikipedia
[2] Cs 321 Stanford course notes
[3] Cs deep learning Stanford course
[4] Coursera-deep learning- Convolutional Neural Network
[5]ImageNet Classification with Deep ConvolutionalNeural Networks by Alex, Sutskever