CSE583/EE552 Pattern Recognition and Machine Learning: Project #3

Due on March 14th, 2022 at 11:59pm

PROFESSOR Yanxi Liu Spring 22

Anish Phule **asp5607@psu.edu**

This report consists of:

Problem 1 - Deep Learning:

1. Data Augmentation process methods and parameters

2. Comparison of Original Data and Augmented Data

3. Visualization of Results

4. Accuracy and Loss

5. Describing each network

6. Answers to questions

Problem 1

Deep Learning

1. Data Augmentation process methods and parameters

We are required to create an augmented dataset for training and testing, so that the network can better understand the patterns. We Rotate, Scale, Translate, and finally Crop the image for creating the new dataset.

The augmentations are performed 5 times on each image in the training dataset, and once on each image of testing dataset.

We use a random number generator(RNG) function to generate values between a specified range.

- Rotation: We first rotate the input image I. The rotation angle should be between 0 and 360 degrees, so the RNG gives a random value of the angle. We then use the 'imrotate' function in Matlab to rotate our image.
- Scaling: We take the rotated image and uniformly scale it. Since the required scaling range is between 100 and 200 percent, the input to the RNG is 1 and 2. The value is then given to the 'imresize' function in matlab.
- Translation: We intend to move our image by specific amounts in the x or the y direction. In order to not exceed our limits, that is, image boundaries, we input a smaller translation number. So we only generate a random number between [-10,10] in both x and y directions. The 'imtranslate' function helps us move the image.
- Cropping: The final task is cropping the image to a smaller dimension of [128, 128] size. This is done using the 'centerCropWindow2d' and 'imcrop' function in matlab.

These give us an effective 17,000*5 that is 85,000 training images, and 17,000*1 that is 17000 testing images. Note that I also resize the images back to [256 256], but that doesn't have any effects on the networks training.

We now intend to visualize how the augmentations are distributed, for both training and testing data.

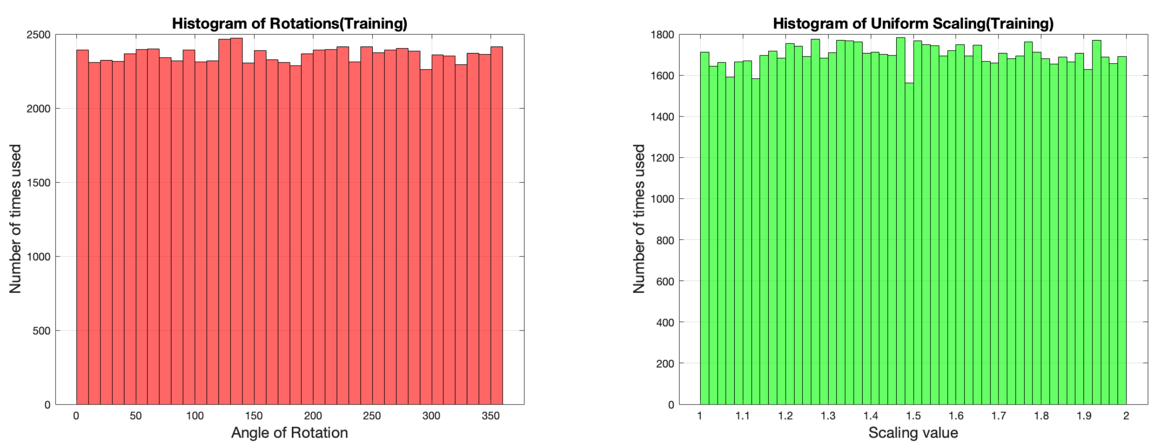


Figure 1: Histogram of Training Rotation and Scaling augmentations

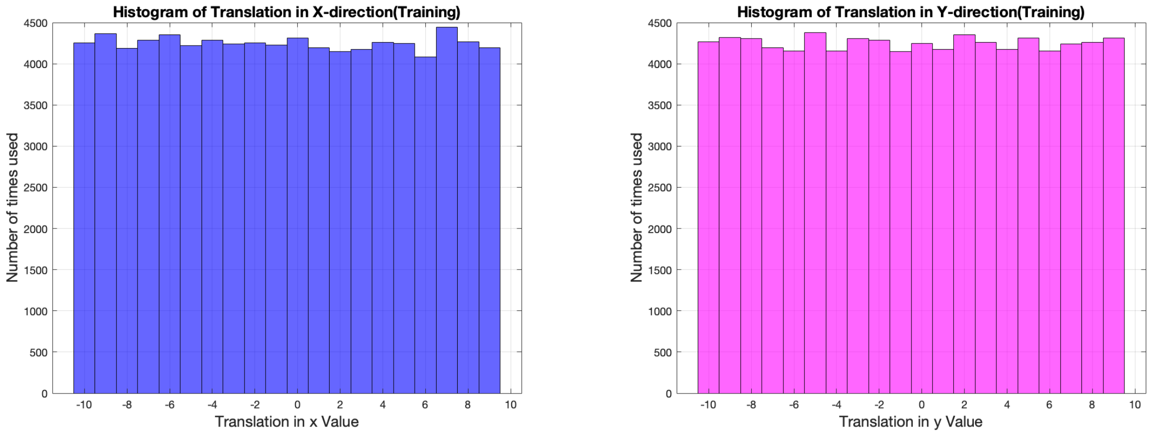


Figure 2: Histogram of Training Translation augmentations

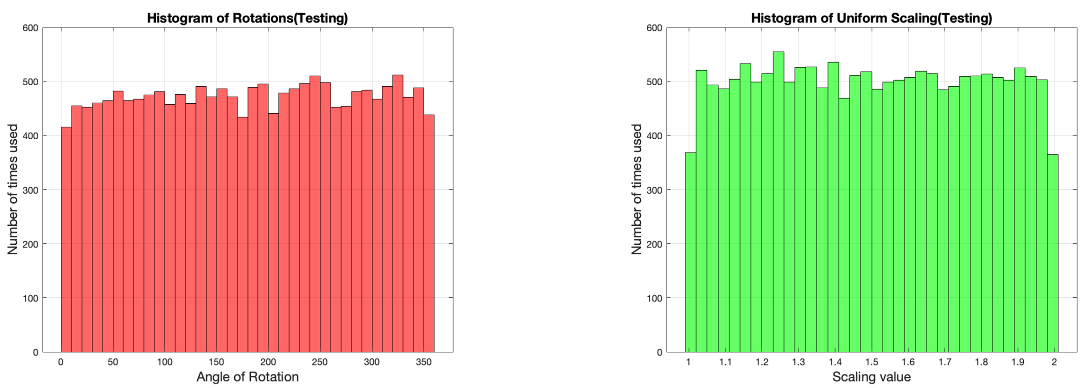


Figure 3: Histogram of Testing Rotation and Scaling augmentations

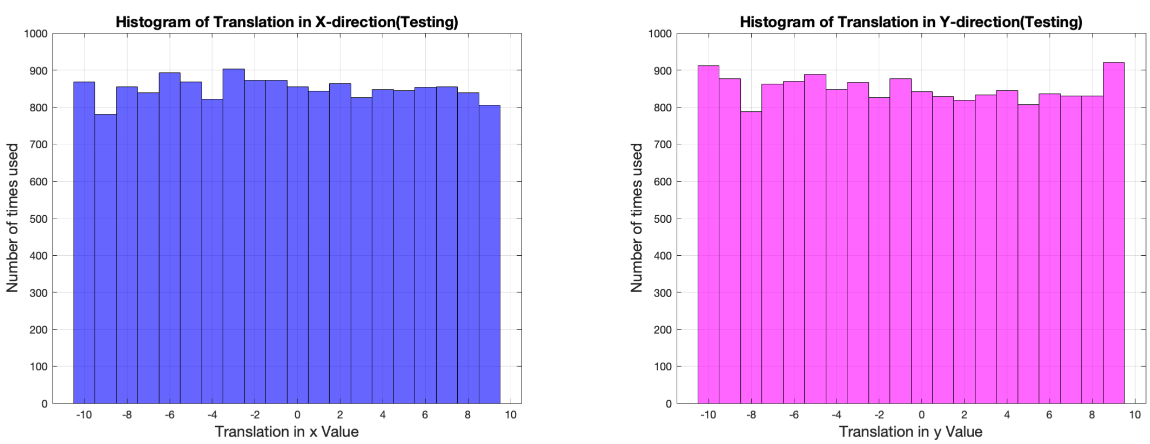


Figure 4: Histogram of Testing Translation augmentations

Problem 1 continued on next page. . .

2. Comparison of Original Data and Augmented Data

We now compare our original wallpaper image dataset with Augmented images.

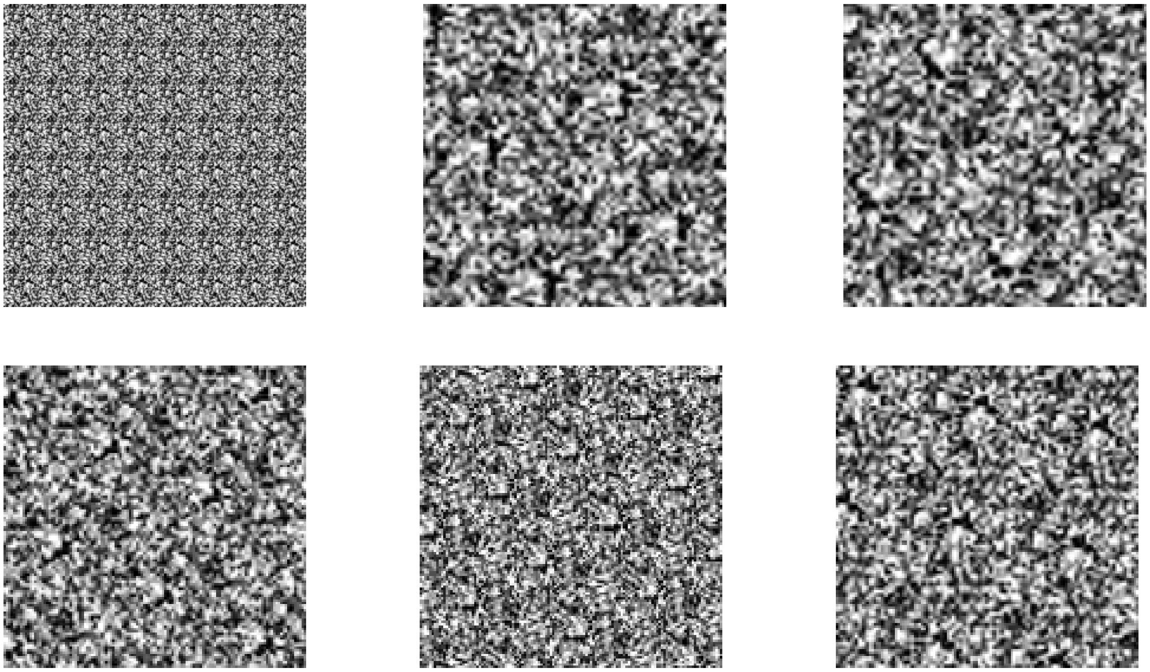


Figure 5: Training datasets: Original Image, Augment 1, Augment 2, Augment 3, Augment 4, Augment 5

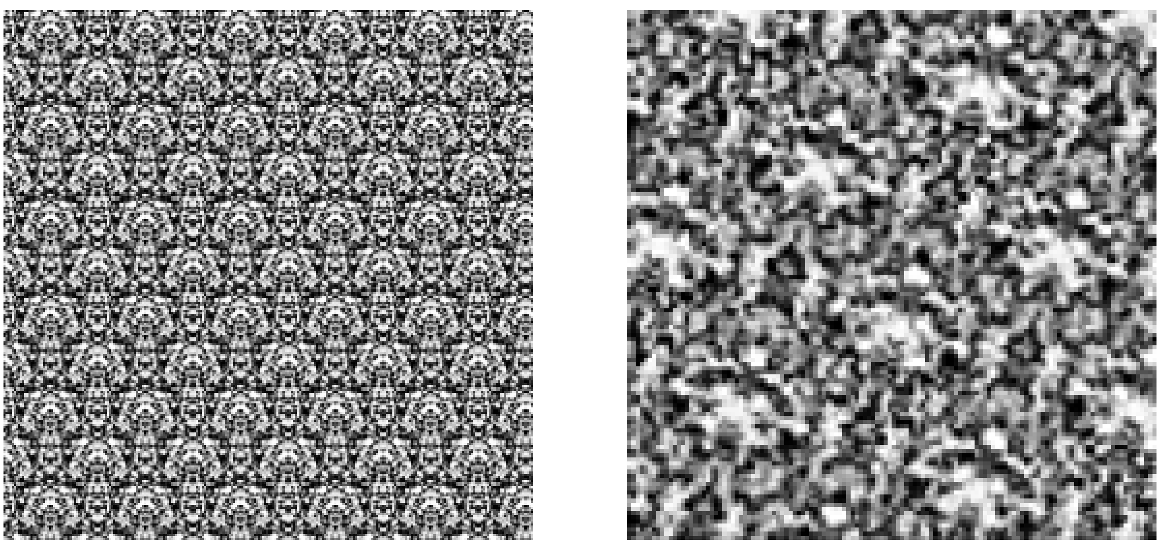


Figure 6: Testing datasets: Original Image, Augmented Image

3. Visualization of Results((baseline, designed, pretrained(AlexNet based))

We take a look at the results for each of the models((baseline, designed, pretrained) through confusion matrices, filters and tsne. **Please note:** The baseline and self-designed models were trained for 10 and 11 epochs respectively, and AlexNet for 5 epochs only, due to time and computation constraints.

Filters:

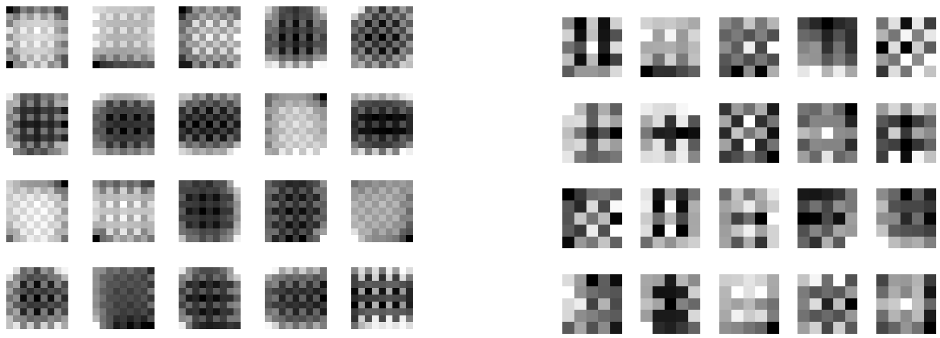


Figure 7: Filters visualized for a. Baseline model, b. Designed model(Pyramids-1, Verbose-0)

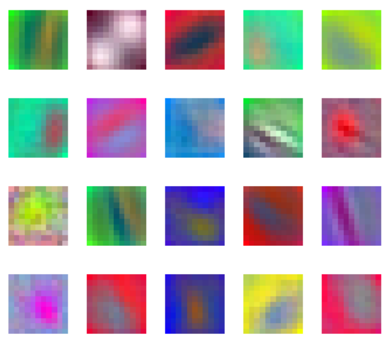


Figure 8: Filters visualized for Pre-trained(AlexNet Based) model

Training performance:

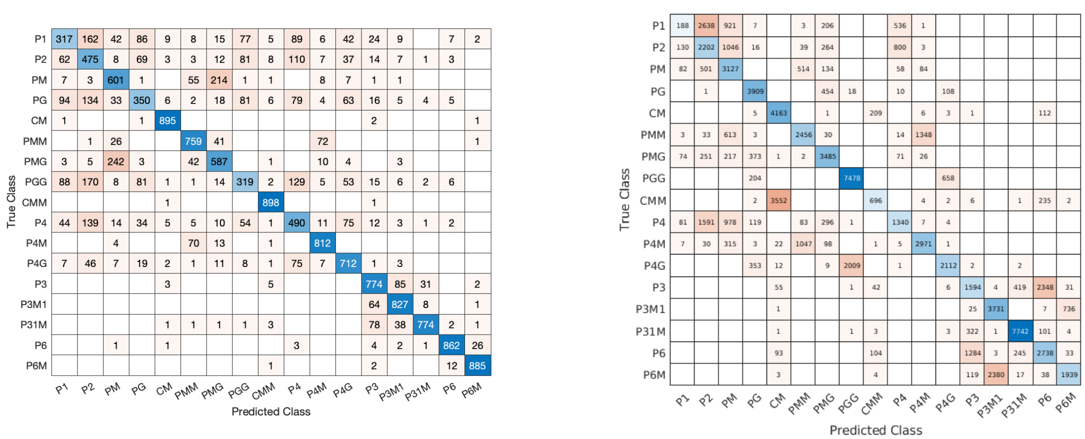


Figure 9: Training Confusion matrix for a. Baseline model, b. Designed model

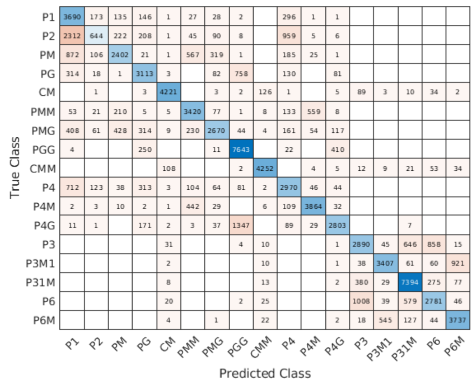
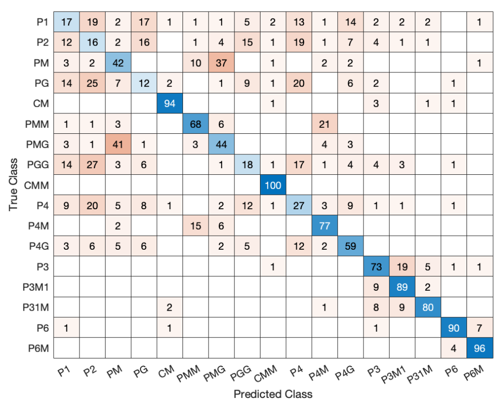


Figure 10: Training confusion matrix for Pre-trained(AlexNet Based) model

Validation Performance:



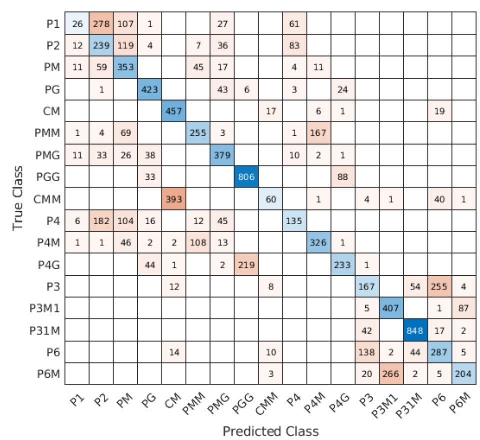


Figure 11: Validation Confusion matrix for a. Baseline model, b. Designed model

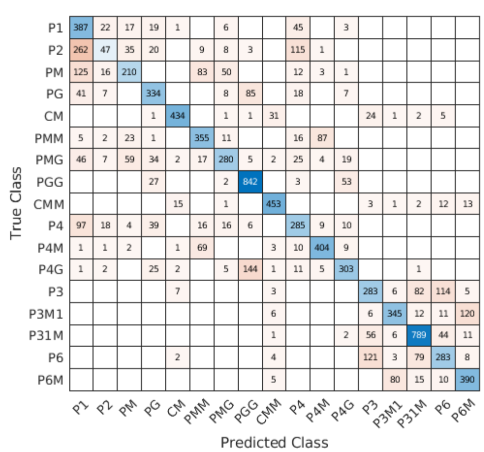
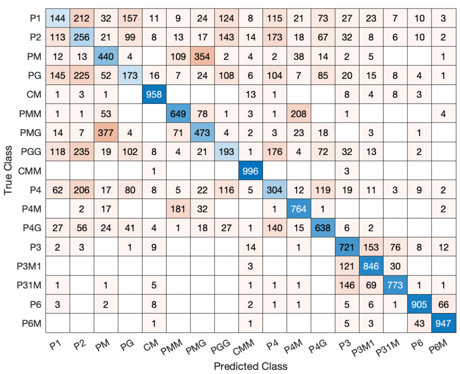


Figure 12: Validation confusion matrix for Pre-trained(AlexNet Based) model

Testing Performance:



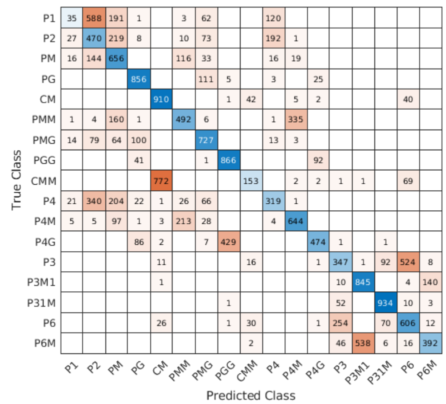


Figure 13: Testing Confusion matrix for a. Baseline model, b. Designed model

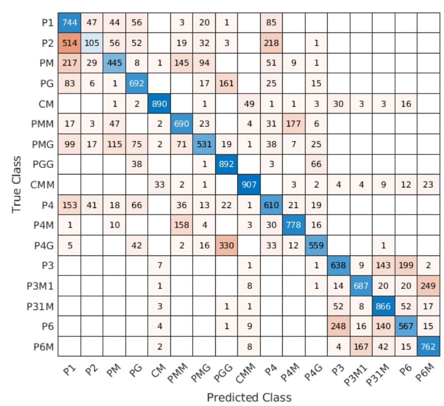
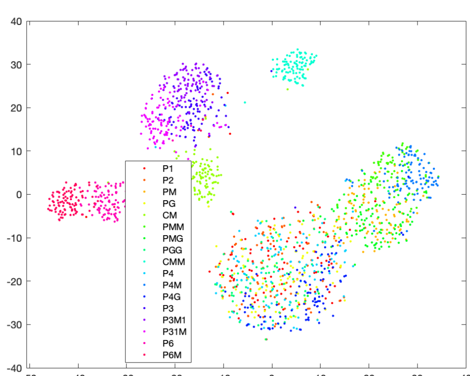


Figure 14: Testing confusion matrix for Pre-trained(AlexNet Based) model

TSNE:



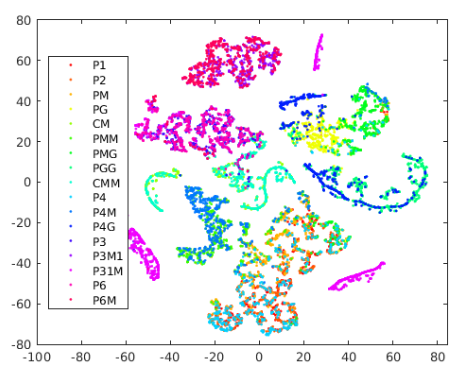


Figure 15: TSNE for a. Baseline model, b. Designed model

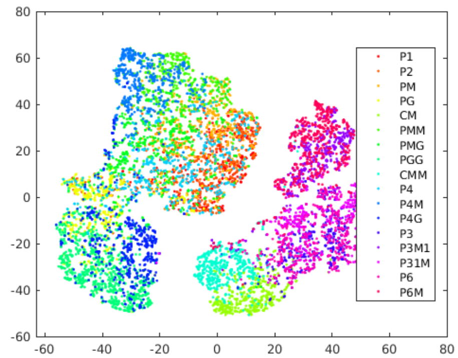


Figure 16: TSNE for Pre-trained(AlexNet Based) model

4. Accuracy and Loss

- Baseline Network: Time taken to train(10 epochs): 36.67 minutes
  Training accuracy: 74.50
  Validation accuracy: 59.06
  Testing accuracy: 60.47
  Loss: 2.3206
- Self - Designed Network: Time taken to train(11 epochs): 7.1 hours
  Training accuracy: 61.74
  Validation accuracy: 60.04
  Testing accuracy: 57.21
  Loss: 0.8474
- Pre-trained Network(AlexNet based): Time taken to train(5 epochs): 2 hours
  Training accuracy: 73.68
  Validation accuracy: 68.81
  Testing accuracy: 66.84
  Loss = 0.8748

5. Describing each network

1. Baseline Network:

	Name	Type	Activations	Learnables
1	imageinput 256×256×1 i…	Image Input	256×256×1	–
2	conv 20 5×5×1 con…	Convolution	128×128×20	Weights 5×5×1×20 Bias 1×1×20
3	relu ReLU	ReLU	128×128×20	–
4	maxpool 2×2 max pooli…	Max Pooling	64×64×20	–
5	fc_1 25 fully conne…	Fully Connected	1×1×25	Weights 25×81920 Bias 25×1
6	dropout 25% dropout	Dropout	1×1×25	–
7	fc_2 17 fully conne…	Fully Connected	1×1×17	Weights 17×25 Bias 17×1
8	softmax softmax	Softmax	1×1×17	–
9	classoutput crossentropye…	Classification Output	1×1×17	–

Figure 17: Baseline network

The baseline network has 9 layers - One convolutional layer.
Solver: 'sgdm', Number of epochs = 10(5+5), minibatch size = 250 and learning rate = 1e-5. The size and parameters of each layer are given in fig. 17.
This is a basic network with single conv layer, and while it may perform well for basic dataset, it won't be able to perform well in the presence of strong, challenging datasets. It is because of this simplicity that it also executes very quickly.

2. Self-Designed Network:

	Name	Type	Activations	Learnables
1	imageinput 256x256x1 images with 'zerocenter' normalization	Image Input	256×256×1	-
2	conv_1 20 5x5x1 convolutions with stride [2 2] and padding [2 2 2 2]	Convolution	128×128×20	Weights 5×5×1×20 Bias 1×1×20
3	batchnorm_1 Batch normalization with 20 channels	Batch Normalization	128×128×20	Offset 1×1×20 Scale 1×1×20
4	relu_1 ReLU	ReLU	128×128×20	-
5	conv_2 15 5x5x20 convolutions with stride [2 2] and padding [2 2 2 2]	Convolution	64×64×15	Weights 5×5×20×15 Bias 1×1×15
6	batchnorm_2 Batch normalization with 15 channels	Batch Normalization	64×64×15	Offset 1×1×15 Scale 1×1×15
7	relu_2 ReLU	ReLU	64×64×15	-
8	maxpool_1 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	32×32×15	-
9	conv_3 10 5x5x15 convolutions with stride [2 2] and padding [2 2 2 2]	Convolution	16×16×10	Weights 5×5×15×10 Bias 1×1×10
10	batchnorm_3 Batch normalization with 10 channels	Batch Normalization	16×16×10	Offset 1×1×10 Scale 1×1×10
11	relu_3 ReLU	ReLU	16×16×10	-
12	conv_4 20 3x3x10 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	16×16×20	Weights 3×3×10×20 Bias 1×1×20
13	batchnorm_4 Batch normalization with 20 channels	Batch Normalization	16×16×20	Offset 1×1×20 Scale 1×1×20
14	relu_4 ReLU	ReLU	16×16×20	-
15	maxpool_2 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	8×8×20	-
16	conv_5 40 3x3x20 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	8×8×40	Weights 3×3×20×40 Bias 1×1×40
17	relu_5 ReLU	ReLU	8×8×40	-
18	batchnorm_5 Batch normalization with 40 channels	Batch Normalization	8×8×40	Offset 1×1×40 Scale 1×1×40
19	conv_6 40 3x3x40 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	8×8×40	Weights 3×3×40×40 Bias 1×1×40
20	batchnorm_6 Batch normalization with 40 channels	Batch Normalization	8×8×40	Offset 1×1×40 Scale 1×1×40
21	relu_6 ReLU	ReLU	8×8×40	-
22	maxpool_3 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	4×4×40	-
23	fc_1 50 fully connected layer	Fully Connected	1×1×50	Weights 50×640 Bias 50×1
24	relu_7 ReLU	ReLU	1×1×50	-
25	fc_2 50 fully connected layer	Fully Connected	1×1×50	Weights 50×50 Bias 50×1
26	relu_8 ReLU	ReLU	1×1×50	-
27	dropout 40% dropout	Dropout	1×1×50	-
28	fc_3 17 fully connected layer	Fully Connected	1×1×17	Weights 17×50 Bias 17×1
29	softmax softmax	Softmax	1×1×17	-
30	classoutput crossentropyex with 'P1' and 16 other classes	Classification Output	-	-

Figure 18: Self-designed network

The Self-designed network has 30 layers - 6 convolutional layer.
Solver: 'sgdm', Number of epochs = 11(5+3+3), minibatch size = 120 and learning rate = 1e-3 for first 2 and 1e-4 for third training phase. The size and parameters of each layer are given in fig. 18. The filter size

and number of filters per conv layer goes from 5,20 to 5,10 to 3,40.
This is a more complex network with 6 conv layers. It is because of this complexity and number of layers that the batch-size is brought down, as it fills the GPU memory quickly. It performs satisfactorily well on the augmented dataset where the baseline fails. It is also because of the complexity and amount of computations that the network takes a long time. With better hardware, it will perform faster and maybe better as well.

2. Pre-trained(AlexNet) Network:

1	data 227x227x3 images with 'zerocenter' normalization	Image Input	227×227×3	-
2	conv1 96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]	Convolution	55×55×96	Weights 11×11×3×96 Bias 1×1×96
3	relu1 ReLU	ReLU	55×55×96	-
4	norm1 cross channel normalization with 5 channels per element	Cross Channel Nor...	55×55×96	-
5	pool1 3x3 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	27×27×96	-
6	conv2 2 groups of 128 5x5x48 convolutions with stride [1 1] and padding [2 2 2 2]	Grouped Convolution	27×27×256	Weigh... 5×5×48×128... Bias 1×1×128×2
7	relu2 ReLU	ReLU	27×27×256	-
8	norm2 cross channel normalization with 5 channels per element	Cross Channel Nor...	27×27×256	-
9	pool2 3x3 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	13×13×256	-
10	conv3 384 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	13×13×384	Weight.. 3×3×256×3... Bias 1×1×384
11	relu3 ReLU	ReLU	13×13×384	-
12	conv4 2 groups of 192 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]	Grouped Convolution	13×13×384	Weigh... 3×3×192×19... Bias 1×1×192×2
13	relu4 ReLU	ReLU	13×13×384	-
14	conv5 2 groups of 128 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]	Grouped Convolution	13×13×256	Weigh... 3×3×192×12... Bias 1×1×128×2
15	relu5 ReLU	ReLU	13×13×256	-
16	pool5 3x3 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	6×6×256	-
17	fc6 4096 fully connected layer	Fully Connected	1×1×4096	Weights 4096×9216 Bias 4096×1
18	relu6 ReLU	ReLU	1×1×4096	-
19	drop6 50% dropout	Dropout	1×1×4096	-
20	fc7 4096 fully connected layer	Fully Connected	1×1×4096	Weights 4096×4096 Bias 4096×1
21	relu7 ReLU	ReLU	1×1×4096	-
22	drop7 50% dropout	Dropout	1×1×4096	-
23	fc 17 fully connected layer	Fully Connected	1×1×17	Weights 17×4096 Bias 17×1
24	softmax softmax	Softmax	1×1×17	-
25	classoutput crossentropyex	Classification Output	-	-

Figure 19: Pre-trained(AlexNet) Network

The Pre-trained(AlexNet) network has 25 layers - 5 convolutional layers.
Solver: 'adam', Number of epochs = 5), minibatch size = 200 and learning rate = 1e-5. The size and parameters of each layer are given in fig. 19.
This is a more complex network with 5 conv layers. The AlexNet based network keeps most of the layers the same, except changing the last 3 layers to have the last fully connected layer with 27 connections. It performs satisfactorily well on the augmented dataset where the baseline fails. The minibatch size had to be brought down a bit(250 to 200), and the solver was changed to 'adam' that showed better performance. But the main reason for the network performing so well, despite having larger batch size and taking less time, is due to using grouped convolutions. This enables it to have more convolutional functions in lesser number of layers.

6. Answers to questions

1. What did you learn about creating convolutional neural networks?
Ans: Creating Neural networks requires an understanding of how layers need to change w.r.t the structure, so that the network learns to the maximum, as well as making it efficient enough to execute in acceptable time. Randomly adding layers will not make it a network.
Also, parameters like solver, learning rate and mini-batch size play an important role on how the network trains. Especially learning rate, since that can potentially lead to overfitting.

2. Does a decrease in loss/error directly translate into an increase in accuracy? Why or why not?
Ans: The answer is No. The reason lies in the definition of loss/error and accuracy.
Loss/error can be defined as the difference or distance between a calculated value for a parameter, and the ground truth. The closer you are to the ground truth value, the lesser the loss will be.
Whereas, accuracy in terms of learning can be defined as the degree to which a prediction is correct, or how well the network is able to learn.
While a program might be able to gradually decrease loss by getting closer to the GT values through some method, whether it learns from the data is a completely different thing altogether. The program could very well come close to training values, but when given a new datapoint, if it doesn't learn, the accuracy will be very low.
An example of this is the baseline network in this project. If we feed the augmented data to it, the loss gradually goes down, but the accuracy barely crosses 10 percent, since it doesn't have the ability to learn from it.
Hence, a decrease in loss/error doesn't directly translate into an increase in accuracy.