

# **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  $17,000 \times 5$  that is 85,000 training images, and  $17,000 \times 1$  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.

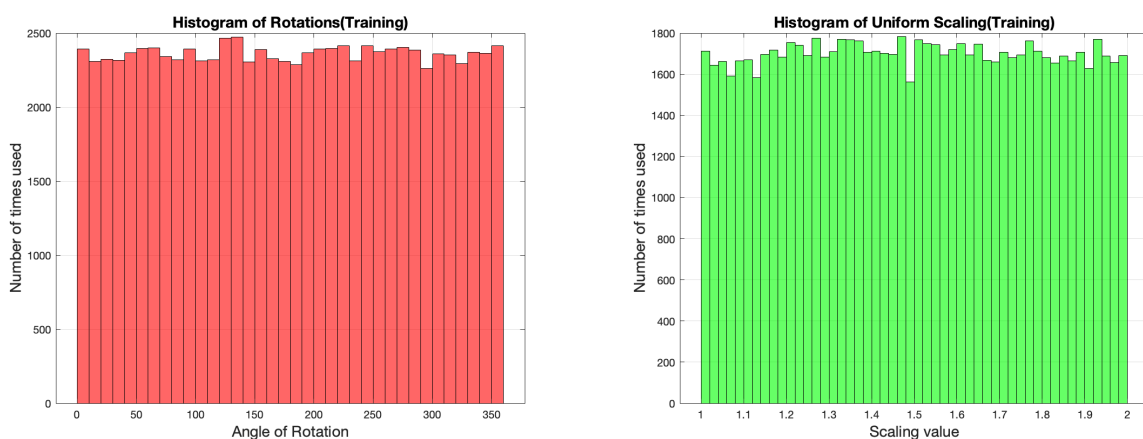


Figure 1: Histogram of Training Rotation and Scaling augmentations

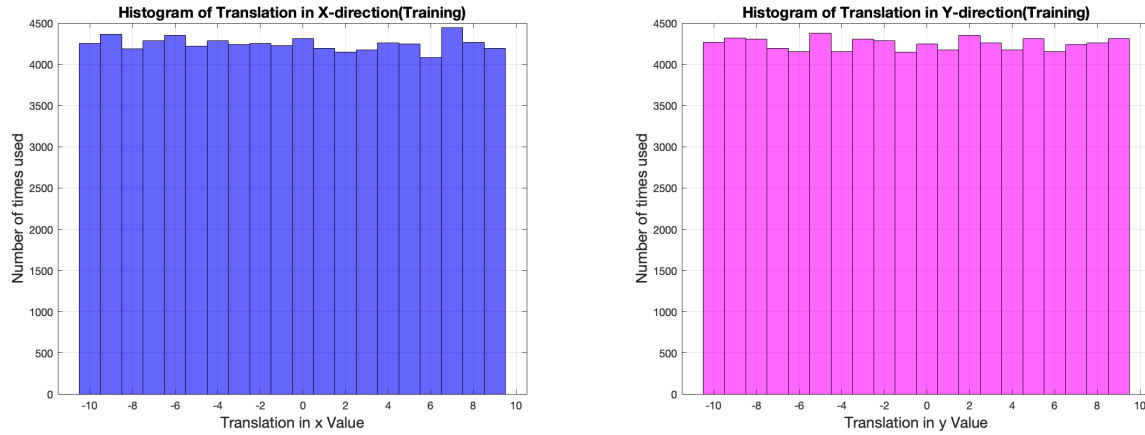


Figure 2: Histogram of Training Translation augmentations

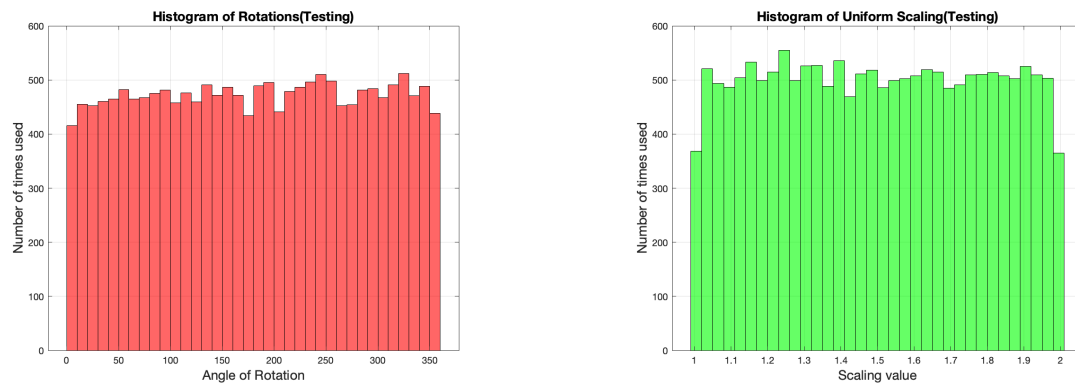


Figure 3: Histogram of Testing Rotation and Scaling augmentations

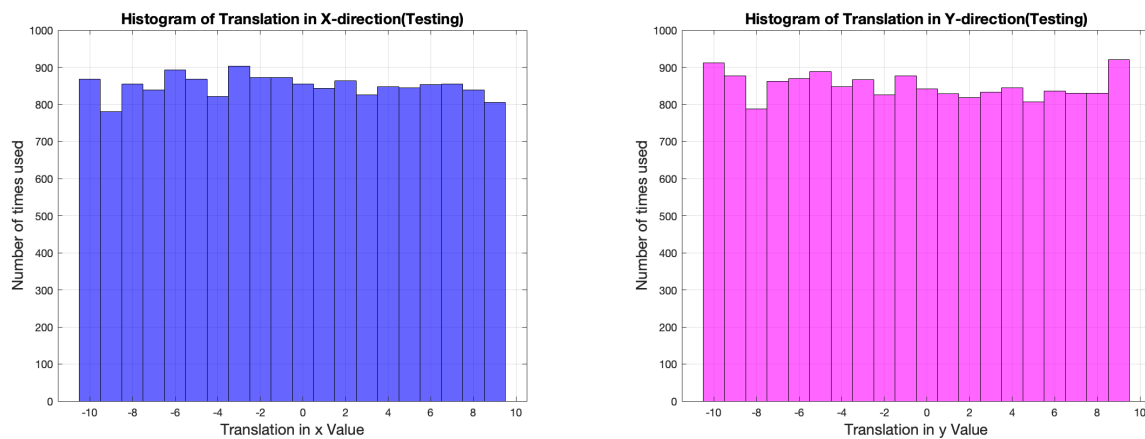


Figure 4: Histogram of Testing Translation augmentations

## 2. Comparison of Original Data and Augmented Data

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

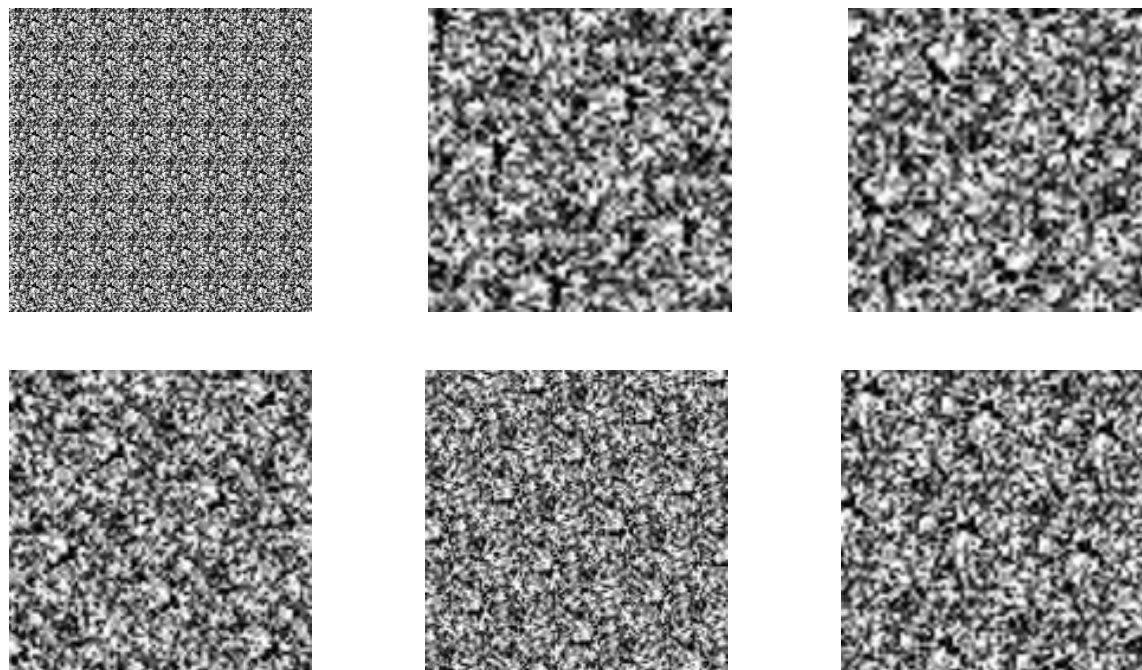


Figure 5: Training datasets: Original Image, Augment 1, Augment 2, Augment 3, Augment 4, Augment 5

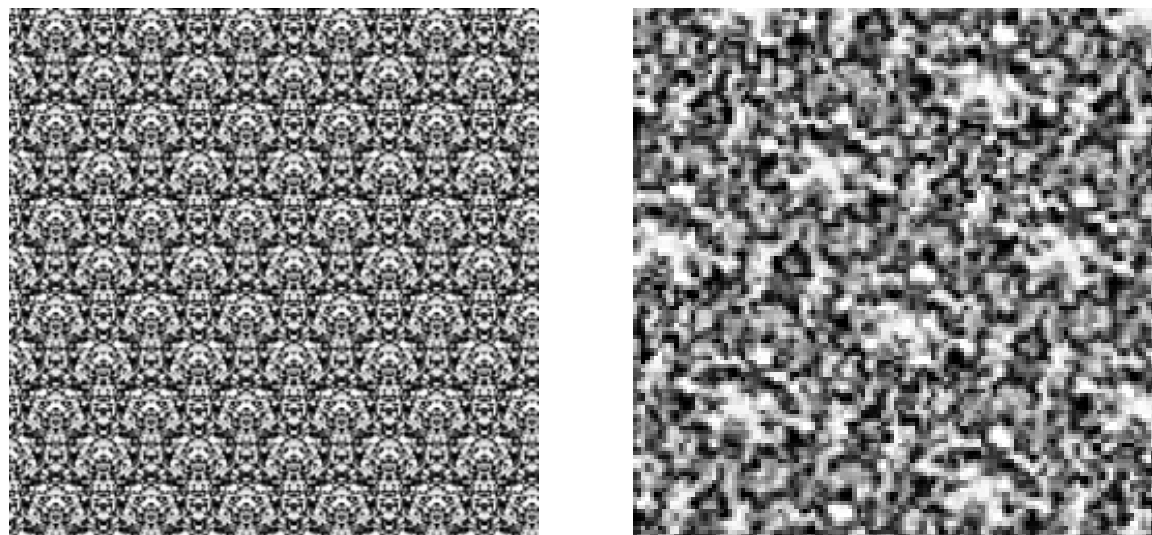


Figure 6: Testing datasets: Original Image, Augmented Image

## 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:

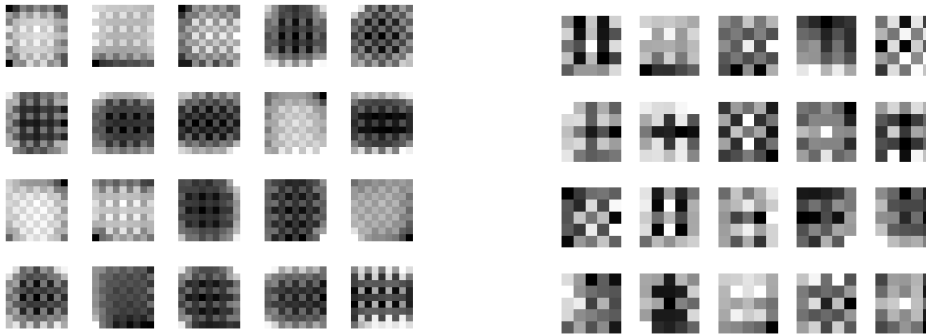


Figure 7: Filters visualized for a. Baseline model, b. Designed model(Pyramids-1, Verbose-0)

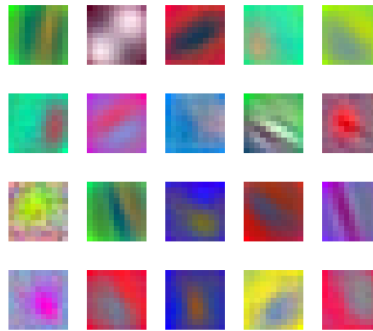


Figure 8: Filters visualized for Pre-trained(AlexNet Based) model

## Training performance:

True Class	P1	317	162	42	86	9	8	15	77	5	89	6	42	24	9	7	2	
	P2	62	475	8	69	3	3	12	81	8	110	7	37	14	7	1	3	
	PM	7	3	601	1			55	214	1	1	8	7	1	1			
	PG	94	134	33	350	6	2	18	81	6	79	4	63	16	5	4	5	
	CM	1			1	895									2		1	
	PMM		1	26			759	41				72					1	
	PMG	3	5	242	3		42	587		1		10	4		3			
	PGG	88	170	8	81	1	1	14	319	2	129	5	53	15	6	2	6	
	CMM					1				896					1			
	P4	44	139	14	34	5	5	10	54	1	490	11	75	12	3	1	2	
	P4M			4			70	13			812							
	P4G	7	46	7	19	2	1	11	8	1	75	7	712	1	3			
	P3					3				5				774	85	31		2
	P3M1											64	827	8				1
	P31M					1	1	1	1	3			78	38	774	2	1	
	P6			1	1	1					3		4	2	1	862	26	
	P6M									1			2		12	885		
	P1	P2	PM	PG	CM	PMM	PMG	PGG	CMM	P4	P4M	P4G	P3	P3M1	P6	P6M		

True Class	P1	188	2638	921	7			3	206			536	1					
	P2	130	2202	1046	16			39	264			800	3					
	PM	82	501	3127				514	134			58	84					
	PG		1		3909			454	18			10	108					
	CM				5	4163		1		209		6	3	1			112	
	PMM	3	33	613	3			2456	30			14	1348					
	PMG	74	251	217	373	1	2	3485				71	26					
	PGG				204				7475					658				
	CMM			2	3552					696		4	2	6	1	235	2	
	P4	81	1591	978	119			83	296	1		1340	7	4				
	P4M	7	30	315	3	22	1047	98		1	5	2971	1					
	P4G			353	12		9	2009			1		2112	2		2		
	P3				55			1	42				6	1594	4	419	2348	31
	P3M1				1								25	3731		7	736	
	P31M				1			1	3		3	322	1		7742	101	4	
	P6				83				104				1284	3	245	2738	33	
	P6M				3				4				119	2380	17	38	1939	
	P1	P2	PM	PG	CM	PMM	PMG	PGG	CMM	P4	P4M	P4G	P3	P3M1	P6	P6M		
	Predicted Class																	

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

P1	9690	179	135	146	1	27	28	2	296	1	1								
P2	2312	644	222	208	1	45	90	8	959	5	6								
PM	872	106	2402	21	1	567	319	1	185	25	1								
PG	314	18	1	3113	3		82	758	130	81									
CM	1		3	4221		3	2	126	1	5	89	3	10	34	2				
PMM	53	21	210	5	5	3429	77	1	8	133	559	8							
PMG	408	61	428	314	9	230	2670	44	4	161	54	117							
PGG	4			250			11	7641	22	410									
CMM				108			2	4252		4	5	12	9	21	53	34			
P4	712	123	38	313	3	104	64	81	2	2970	46	44							
P4M	2	3	10	2	1	442	29	6	109	3864	32								
P4G	11	1		171	2	3	37	1347		89	29	2893			7				
P3						31		4	10		1	2890	43	646	858	15			
P3M1						2				1	38	3407	61	60	921				
P31M						8			13	2	380	29	7194	275	77				
P6						20		2	25		1008	39	578	2781	46				
P6M						4		1	22		2	18	545	127	44	9737			

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

## Validation Performance:

P1	17	19	2	17	1	1	1	5	2	13	1	14	2	2	2	1			
P2	12	16	2	16		1	4	15	1	19	1	7	4	1	1				
PM	3	2	42			10	37	1		2	2					1			
PG	14	25	7	12	2		1	9	1	20		6	2				1		
CM					94							3		1	1				
PMM	1	1	3			68	6			21									
PMG	3	1	41	1		3	44			4	3								
PGG	14	27	3	6		1	18	1	17	1	4	4	3		1				
CMM							100												
P4	9	20	5	8	1		2	12	1	27	3	9	1	1		1			
P4M						15	6			77									
P4G	3	6	5	6		2	5		12	2	59								
P3								1			73	19	5	1	1				
P3M1										9	89	2							
P31M						2				1	8	9	80						
P6					1								90	7					
P6M													4	96					

P1	26	278	107	1			27			61									
P2	12	239	119	4			7	36		83									
PM	11	59	353				45	17		4	11								
PG		1		423			43	6		3		24							
CM					457				17	6	1					19			
PMM	1	4	69			255	3			1	167								
PMG	11	33	26	38			379			10	2	1							
PGG				33				806				88							
CMM					393			60		1	4	1	40	1					
P4	6	182	104	16			12	45		135									
P4M	1	1	46	2	2	108	13			326	1								
P4G				44	1	2	219			233	1								
P3					12			8			167		54	255	4				
P3M1											5	407		1	87				
P31M											42		848	17	2				
P6						14				10			138	2	44	287	5		
P6M									3				20	266	2	5	204		

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

P1	387	22	17	19	1		6			45	3								
P2	262	47	35	20			9	8	3	115	1								
PM	125	16	210				83	50		12	3	1							
PG	41	7		334			8	85		18	7								
CM				1	434		1	1	31			24	1	2	5				
PMM	5	2	23	1		355	11			16	87								
PMG	46	7	59	34	2	17	280	5	2	25	4	19							
PGG						27		2	842		3								
CMM						15			453			3	1	2	12	13			
P4	97	18	4	39			16	16	6	285	9	10							
P4M	1	1	2		1	69			3	10	404	9							
P4G	1	2		25	2		5	144	1	11	5	303			1				
P3						7			3			283	6	82	114	5			
P3M1									6			6	345	12	11	120			
P31M									1	2	56	6	709	44	11				
P6					2				4			121	3	79	283	8			
P6M									5			80	15	10		390			

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

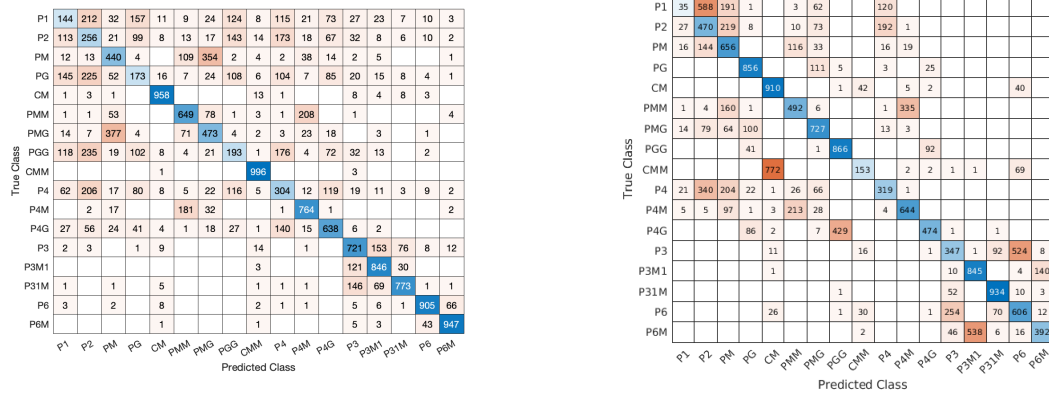
**Testing Performance:**

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

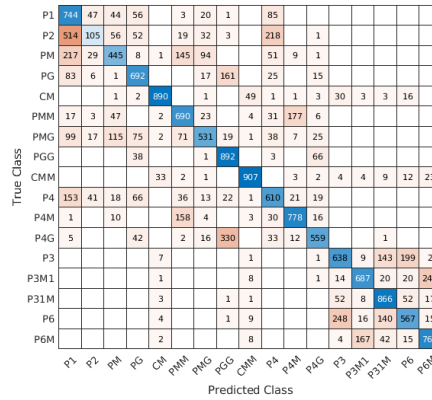


Figure 14: Testing confusion matrix for Pre-trained(AlexNet Based) model

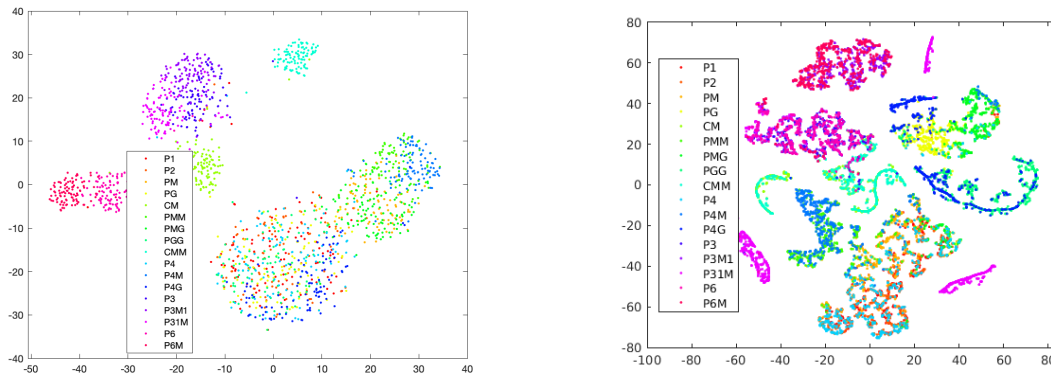
**TSNE:**

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



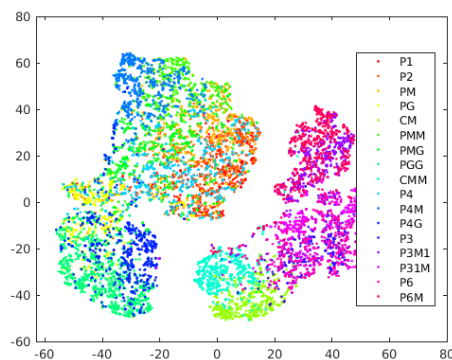


Figure 16: TSNE for Pre-trained(AlexNet Based) model

#### 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 pool...	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 crossentropy...	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	<b>imageinput</b> 256x256x1 images with 'zerocenter' normalization	Image Input	256x256x1	-
2	<b>conv_1</b> 20 5x5x1 convolutions with stride [2 2] and padding [2 2 2 2]	Convolution	128x128x20	Weights 5x5x1x20 Bias 1x1x20
3	<b>batchnorm_1</b> Batch normalization with 20 channels	Batch Normalization	128x128x20	Offset 1x1x20 Scale 1x1x20
4	<b>relu_1</b> ReLU	ReLU	128x128x20	-
5	<b>conv_2</b> 15 5x5x20 convolutions with stride [2 2] and padding [2 2 2 2]	Convolution	64x64x15	Weights 5x5x20x15 Bias 1x1x15
6	<b>batchnorm_2</b> Batch normalization with 15 channels	Batch Normalization	64x64x15	Offset 1x1x15 Scale 1x1x15
7	<b>relu_2</b> ReLU	ReLU	64x64x15	-
8	<b>maxpool_1</b> 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	32x32x15	-
9	<b>conv_3</b> 10 5x5x15 convolutions with stride [2 2] and padding [2 2 2 2]	Convolution	16x16x10	Weights 5x5x15x10 Bias 1x1x10
10	<b>batchnorm_3</b> Batch normalization with 10 channels	Batch Normalization	16x16x10	Offset 1x1x10 Scale 1x1x10
11	<b>relu_3</b> ReLU	ReLU	16x16x10	-
12	<b>conv_4</b> 20 3x3x10 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	16x16x20	Weights 3x3x10x20 Bias 1x1x20
13	<b>batchnorm_4</b> Batch normalization with 20 channels	Batch Normalization	16x16x20	Offset 1x1x20 Scale 1x1x20
14	<b>relu_4</b> ReLU	ReLU	16x16x20	-
15	<b>maxpool_2</b> 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	8x8x20	-

16	<b>conv_5</b> 40 3x3x20 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	8x8x40	Weights 3x3x20x40 Bias 1x1x40
17	<b>relu_5</b> ReLU	ReLU	8x8x40	-
18	<b>batchnorm_5</b> Batch normalization with 40 channels	Batch Normalization	8x8x40	Offset 1x1x40 Scale 1x1x40
19	<b>conv_6</b> 40 3x3x40 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	8x8x40	Weights 3x3x40x40 Bias 1x1x40
20	<b>batchnorm_6</b> Batch normalization with 40 channels	Batch Normalization	8x8x40	Offset 1x1x40 Scale 1x1x40
21	<b>relu_6</b> ReLU	ReLU	8x8x40	-
22	<b>maxpool_3</b> 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	4x4x40	-
23	<b>fc_1</b> 50 fully connected layer	Fully Connected	1x1x50	Weights 50x640 Bias 50x1
24	<b>relu_7</b> ReLU	ReLU	1x1x50	-
25	<b>fc_2</b> 50 fully connected layer	Fully Connected	1x1x50	Weights 50x50 Bias 50x1
26	<b>relu_8</b> ReLU	ReLU	1x1x50	-
27	<b>dropout</b> 40% dropout	Dropout	1x1x50	-
28	<b>fc_3</b> 17 fully connected layer	Fully Connected	1x1x17	Weights 17x50 Bias 17x1
29	<b>softmax</b> softmax	Softmax	1x1x17	-
30	<b>classoutput</b> crossentropy 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	227x227x3	-
2	conv1 96 11x11x3 convolutions with stride [4 4] and padding [0 0 0 0]	Convolution	55x55x96	Weights 11x11x3x96 Bias 1x1x96
3	relu1 ReLU	ReLU	55x55x96	-
4	norm1 cross channel normalization with 5 channels per element	Cross Channel Nor...	55x55x96	-
5	pool1 3x3 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	27x27x96	-
6	conv2 2 groups of 128 5x5x48 convolutions with stride [1 1] and padding [2 2 2 2]	Grouped Convolution	27x27x256	Weights 5x5x48x128... Bias 1x1x128x2
7	relu2 ReLU	ReLU	27x27x256	-
8	norm2 cross channel normalization with 5 channels per element	Cross Channel Nor...	27x27x256	-
9	pool2 3x3 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	13x13x256	-
10	conv3 384 3x3x256 convolutions with stride [1 1] and padding [1 1 1 1]	Convolution	13x13x384	Weights 3x3x256x3... Bias 1x1x384
11	relu3 ReLU	ReLU	13x13x384	-
12	conv4 2 groups of 192 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]	Grouped Convolution	13x13x384	Weights 3x3x192x19... Bias 1x1x192x2
13	relu4 ReLU	ReLU	13x13x384	-
14	conv5 2 groups of 128 3x3x192 convolutions with stride [1 1] and padding [1 1 1 1]	Grouped Convolution	13x13x256	Weights 3x3x192x12... Bias 1x1x128x2
15	relu5 ReLU	ReLU	13x13x256	-
16	pool5 3x3 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	6x6x256	-
17	fc6 4096 fully connected layer	Fully Connected	1x1x4096	Weights 4096x9216 Bias 4096x1
18	relu6 ReLU	ReLU	1x1x4096	-
19	drop6 50% dropout	Dropout	1x1x4096	-
20	fc7 4096 fully connected layer	Fully Connected	1x1x4096	Weights 4096x4096 Bias 4096x1
21	relu7 ReLU	ReLU	1x1x4096	-
22	drop7 50% dropout	Dropout	1x1x4096	-
23	fc 17 fully connected layer	Fully Connected	1x1x17	Weights 17x4096 Bias 17x1
24	softmax softmax	Softmax	1x1x17	-
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.