Name – Deepesh Kumar Singh CSULB ID - 016149963
Teammate Name – Sumeet Sunil Gadkari CSULB ID - 016115942

DL Assignment 3

We followed below steps to complete the assignment -

MATLAB 2017

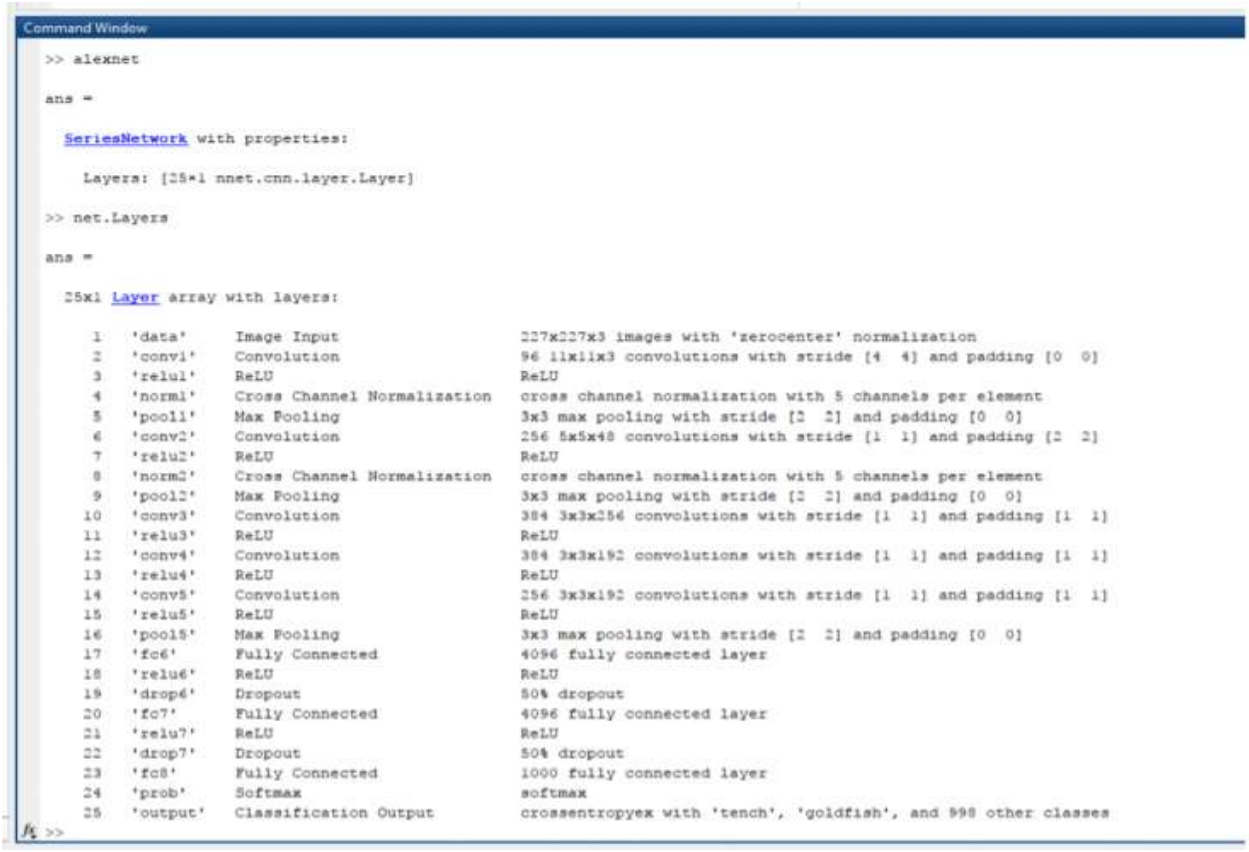
We downloaded matlab 2017 for this assignment

Download AlexNet Network

AlexNet is a pre-trained neural network model used to classify images. AlexNet is trained on ImageNet dataset. In this assignment, we are expected to use this pre-trained model to classify images as well as train model on custom dataset.

Install AlexNet

Before using AlexNet we made sure that we have its package installed in our environment. For this, we ran the command 'AlexNet' and then the package was installed. We ran net.Layers in the MATLAB to check if we can see below output for verifying correct installation of AlexNet package -



Name – Deepesh Kumar Singh CSULB ID - 016149963
Teammate Name – Sumeet Sunil Gadkari CSULB ID - 016115942

1) Classifying an image

In this section, we are suppose to use AlexNet to classify images.

```
net = AlexNet;
```

In Order to classify the images we first loaded AlexNet using above command. The network consists of 25 layers. The output object 'net' is of type SeriesNetwork.

```
I = imread('peppers.png');
% Read the image using imread
figure
imshow(I)
% output the read image
```

Next, we need to read image from test data and display the image to make sure that loaded image is correct. In this case, we are loading bell pepper image.

```
sz = net.Layers(1).InputSize
```

Next, we determine the input size of the network. The input layer has dimensions equal to 27*27*3. We need to make sure the input image is of same dimension. The input size of first layer is obtained by using its attribute `InputSize` attribute

```
I = I(1:sz(1),1:sz(2),1:sz(3));
% imresize (Image Processing Toolbox™). %
```

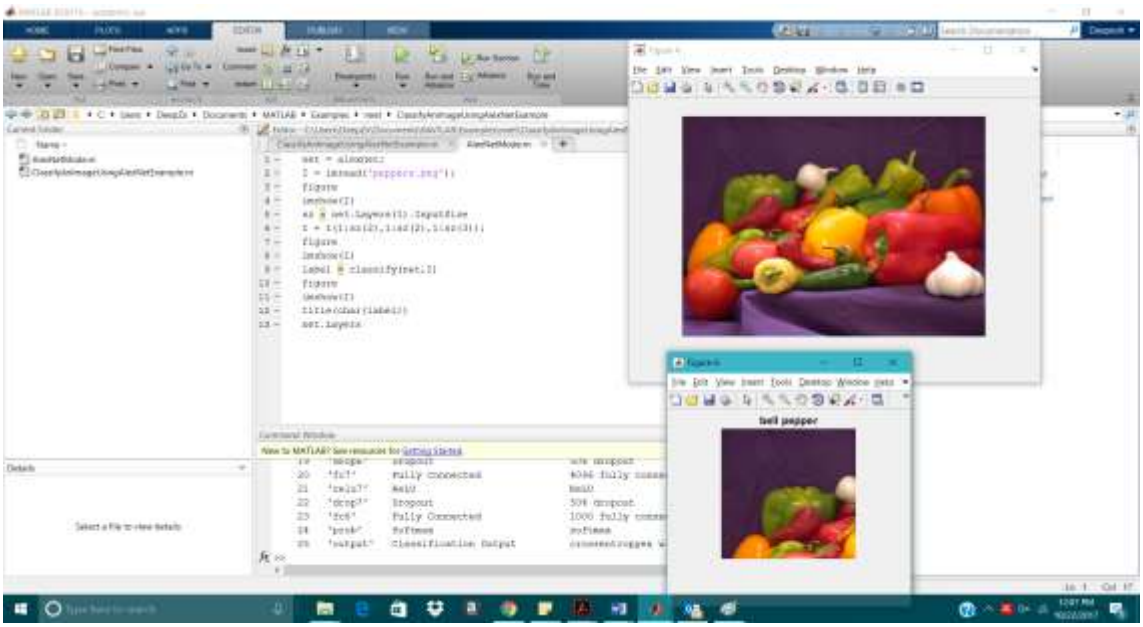
If image dimensions do not match then we need to resize or crop image. In this case we are doing slicing using the dimensions obtained from InputSize

Now, input data has valid dimensions and can be supplied to the model for classification. In the next step, classify function takes our input image and pretrained model and returns the label for the image. Printing the obtained label using title function

```
label = classify(net,I)
title(char(label))
```

Below figure illustrates that we supplied bell pepper image as an input and our model indeed predicted the image as bell pepper

.



Sample output from classification program

2) Perform feature extraction

In this step, we extract the features of new dataset using AlexNet

```
unzip('MerchData.zip');
images = imageDatastore('MerchData',...
    'IncludeSubfolders',true,...
    'LabelSource','foldernames');
```

We unzip images for which features need to be extracted. The unzipped images are then divided into test and train data using 70:30 split. The dimension of images is same as that of input layer on model and hence no resizing is required

```
[trainingImages,testImages] = splitEachLabel(images,0.7,'randomized');
```

Now, we load AlexNet model. The model constructs hierarchical represntation of images. Deep layers have high-level features. To get features of training and test data we use activations from fully connected layer ‘fc7’. Also lables of train and test data are extracted

```
layer = 'fc7';
trainingFeatures = activations(net,trainingImages,layer);
testFeatures = activations(net,testImages,layer);
trainingLabels = trainingImages.Labels;
testLabels = testImages.Labels;
```

3) Use the features extracted from the training images as predictor variable

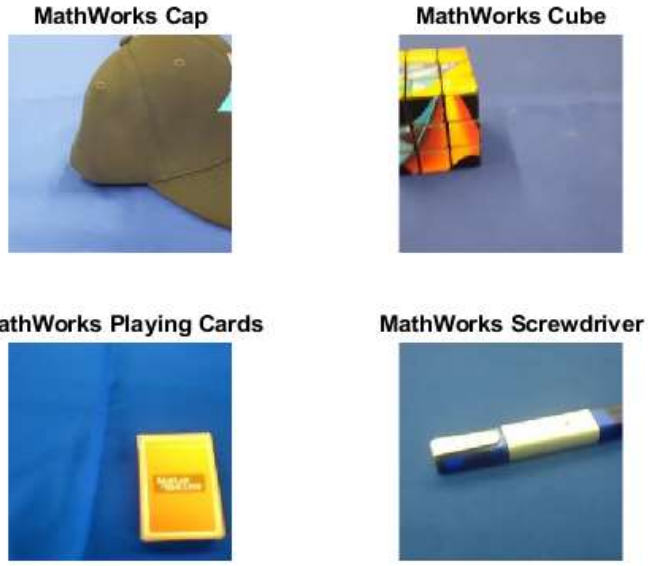
After extracting the features of new dataset, we train SVM using fitcecoc function. This function fits our model with extracted features from AlexNet acting as training data.

```
classifier = fitcecoc(trainingFeatures,trainingLabels);
```

4) Classify test images

Final step is to use fitted SVM model to test new images. This is done using predict function which takes SVM model and test data as parameters. We can plot test image with its predicted label using for loop

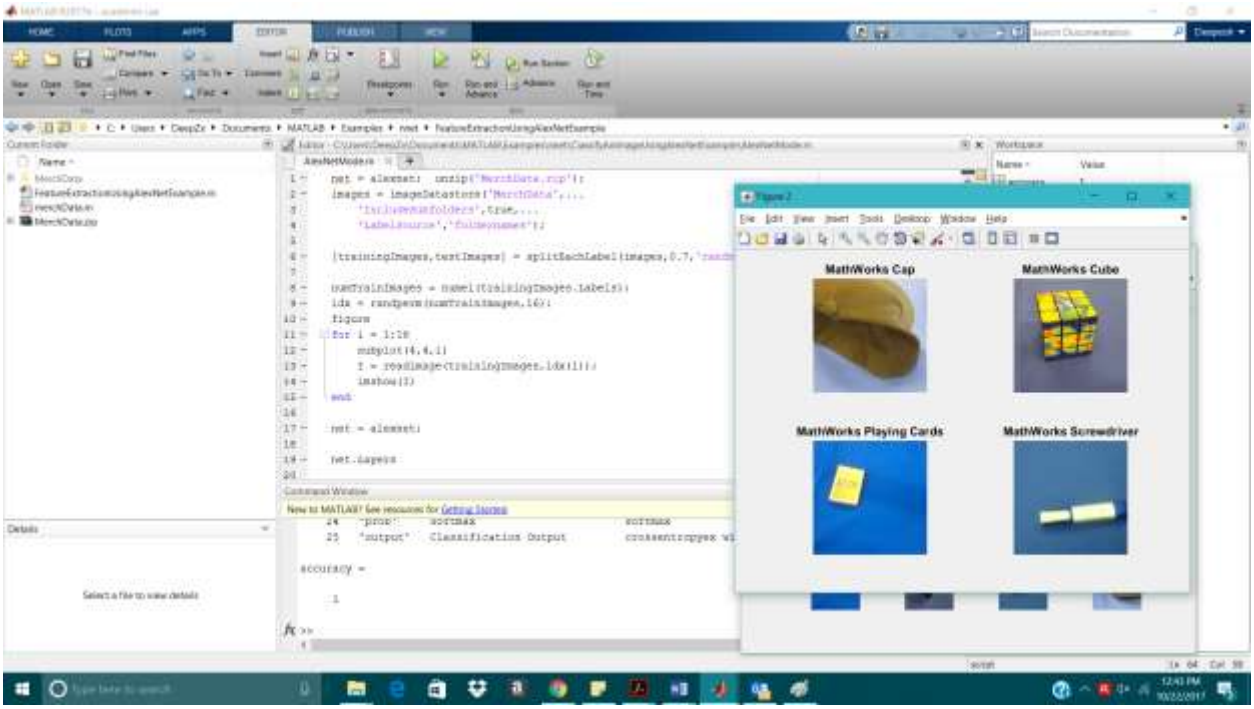
```
predictedLabels = predict(classifier,testFeatures);
```




```
accuracy = mean(predictedLabels == testLabels)
```

We also calculated the classification accuracy on the test set. Accuracy is the fraction of labels that the network predicts correctly. This SVM has high accuracy.

accuracy comes to be 1



Program written in MATLAB for feature extraction and classifying images.

5) Transfer Learning Using AlexNet

Load Data as done in previous step. Now we will transfer Layers to New Network

layersTransfer = net.Layers(1:end-3);

In AlexNet, last three layers of the neural network are configured for 1000 classes.
We need to fine tune these layers to train with our own new dataset. So, we would first extract all layers, except the last three in AlexNet.

numClasses = numel(categories(trainingImages.Labels))

This will get the number of classes to be classified into in the unizipped dataset.

9) Train the new network

We would replace the last three layers with a fully connected layer, a softmax layer, and a classification output layer. WeightLearnRateFactor and BiasLearnRateFactor is specified in parameter to increase the learning rate. Then we will do below fine tuning -

```
miniBatchSize = 10;
numIterationsPerEpoch = floor(numel(trainingImages.Labels)/miniBatchSize);
options = trainingOptions('sgdm',...
   'MiniBatchSize',miniBatchSize,...
   'MaxEpochs',4,...
   'InitialLearnRate',1e-4,...
   'Verbose',false,...
   'Plots','training-progress',...
   'ValidationData',validationImages,...
   'ValidationFrequency',numIterationsPerEpoch);
```

We are providing the training options parameter here. We would want the initial learning rate to be small to slow down the learning in the transferred layers. Also, batch size is specified to be used in training. We would also specify the max number of epochs on which training would be done. We would plot the training progress graph. Code will do the software validate every ValidationFrequency and number of iteration per epoch is specified.

```
netTransfer = trainNetwork(trainingImages,layers,options);
```

Train the network that consists of the transferred and new layers. Above function trainNetwork will use GPU by default. If GPU is not available then CPU will be used for training. Above method is ran on transferred layers and the new layers. Training options are provided in the trainNetwork function parameter defined above..

```
predictedLabels = classify(netTransfer,validationImages);
```

After training of training dataset we would now classify the validation images using the fine tuned neural network.

After this we will print any four validation image with predicted label. Also, the accuracy is calculated by comparing the validation predicted label with the actual label.

6) Create a new dataset

```
camera = webcam; % Connect to the camera
picture = camera.snapshot; % Take a picture
```

We first acquired few images from webcam and saved it into a directory name - MathWorks label_name. We created directory name of this format because imageDatastore will automatically labels the images based on folder names and store the data as imageDatastore object. To get more clear pictures we also used good camera from our mobile phones and included these to our dataset folder. We assumed that taking a good quality photos from better

camera would help AlexNet in getting more features. We had to reduce the pixels to 227*227 because the AlexNet input layer accept the size of 227*227 only. So we created below folders in our machines -

Each folder has 16 images related to each label. Now we created a zip file containing all the above folder . We will provide the path of the above zip file to unzip as below in program -

url = 'C:\Users\DeepZx\Desktop\STUDY\Selected Topic\Assignment\dataTrain.zip';
unzip(url);

This will unzip our newly created dataset into our workspace which we will use in next step to train the AlexNet using learning transfer .

7) Collect analytical data for training progress and do complete analysis -

Now we will use dataset we created above to train the pretrained AlexNet neural network.

```
images = imageDatastore('dataTrain',...
   'IncludeSubfolders',true,...
   'LabelSource','foldernames');
```

In above code we will use the name of the zip file imported i.e. dataTrain here. ImageDatastore will automatically labels the images based on sub folder names and store the data as imageDatastore object. We will use this object to divide the dataset into training and validation later in below step.

```
[trainingImages,validationImages] = splitEachLabel(images,0.7,'randomized');
```

In above code 70% of data is used to train the model and other 30% of data is used for validation after training. Randomized parameter is used above to split the dataset randomly.

```
numTrainImages = numel(trainingImages.Labels);
idx = randperm(numTrainImages,16);
```

```
figure
for i = 1:16
   subplot(4,4,i)
   I = readimage(trainingImages,idx(i));
   imshow(I)
end
```

In above code, we are displaying 16 images in 4x4 format. Numel function will get number of array elements in trainingImages.Labels array. **Randperm** will return random permutation from 1 to 16.

```
net = AlexNet;
```

It will load the pretrained AlexNet as explained above in doc.

```
layersTransfer = net.Layers(1:end-3);
```

In AlexNet last three layers of the neural network are configured for 1000 classes.
We need to fine tune these layers to train with our own new dataset. So, we would first extract all layers, except the last three in AlexNet.

```
numClasses = numel(categories(trainingImages.Labels))
```

This will get the number of classes to be classified into in the unizipped dataset.

```
layers = [
   layersTransfer

fullyConnectedLayer(numClasses,'WeightLearnRateFactor',20,'BiasLearnRateFactor',20)
   softmaxLayer
   classificationLayer];
```

We would replace the last three layers with a fully connected layer, a softmax layer, and a classification output layer. WeightLearnRateFactor and BiasLearnRateFactor is specified in parameter to increase the learning rate.

```
miniBatchSize = 10;
numIterationsPerEpoch = floor(numel(trainingImages.Labels)/miniBatchSize);
options = trainingOptions('sgdm',...
   'MiniBatchSize',miniBatchSize,...
   'MaxEpochs',4,...
   'InitialLearnRate',1e-4,...
```

```
    'Verbose',false,...
    'Plots','training-progress',...
    'ValidationData',validationImages,...
    'ValidationFrequency',numIterationsPerEpoch);
```

We are providing the training options paramter here. We would want the initial learning rate to be small to slow down the learning in the transferred layers. Also, batch size is specified to be used in training. We would also specify the max number of epochs on which training would be done. We would plot the training progress graph. Code will do the software validate every ValidationFrequency and number of iteration per epoch is specified.

```
netTransfer = trainNetwork(trainingImages,layers,options);
```

Train the network that consists of the transferred and new layers. Above function trainNetwork will use GPU by default. If GPU is not available then CPU will be used for training. Above method is ran on transferred layers and the new layers. Training options are provided in the trainNetwork function parameter defined above..

```
predictedLabels = classify(netTransfer,validationImages);
```

After training of training dataset we would now classify the validation images using the fine tuned neural network.

```
idx = [1 5 10 15];
figure
for i = 1:numel(idx)
    subplot(2,2,i)
    I = readimage(validationImages,idx(i));
    label = predictedLabels(idx(i));
    imshow(I)
    title(char(label))
end
```

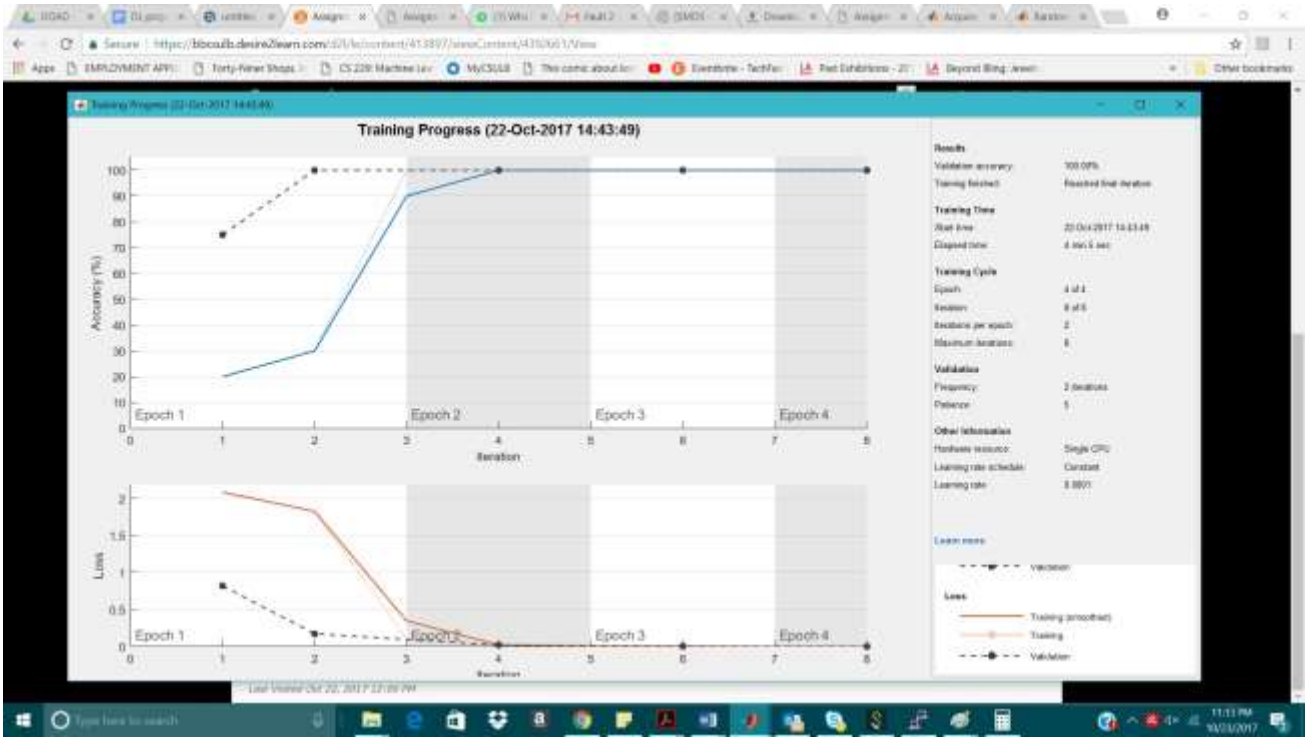
To check the results of validation we would now display four sample images with their predicted labels.

```
valLabels = validationImages.Labels; % This will get the actual labels of the dataset.
accuracy = mean(predictedLabels == valLabels)
```

We now match the predicted labels with the actual label to calculate the classification accuracy on the validation set.

This trained network has high accuracy of ~100

Below final graph is obtained during 8 iterations and 4 epochs (**Training Progress**)-



Above graph shows accuracy and loss at each iterations and epochs. In the results section is shows validation accuracy and training finished or in progress. Training start time and elapsed time is displayed as well in the right side. Information related to training cycle like epochs and number of iterations is displayed in above plot. Other informations like hardware resources used (GPU or CPU) and learning rate etc is also displayed.

Name – Deepesh Kumar Singh CSULB ID - 016149963
Teammate Name – Sumeet Sunil Gadkari CSULB ID - 016115942

In below image we show that the validation image is correctly classified into respective classes -