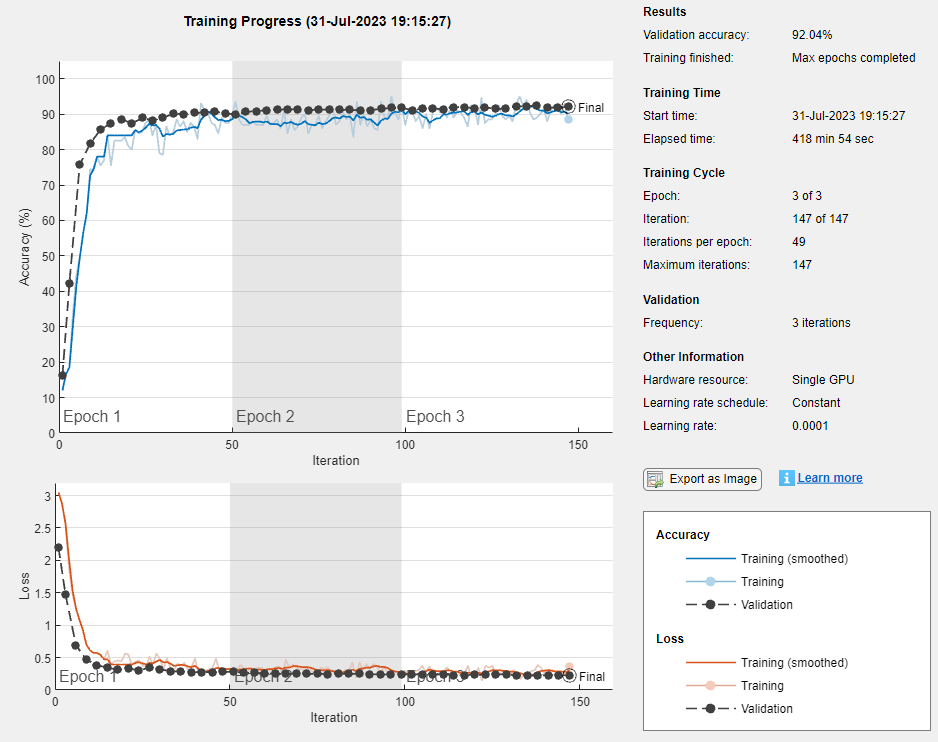
Task 1

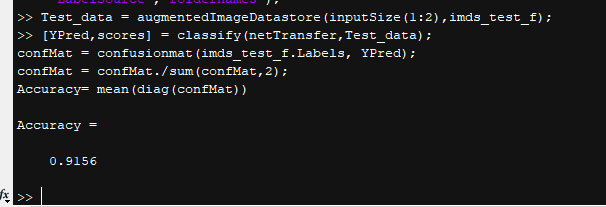
• Change the last layers so that it can classify the desired number of classes as needed for provided dataset

|  |
| --- |
| inputSize = net.Layers(1).InputSize  layersTransfer = net.Layers(1:end-3);  numClasses = numel(categories(imdsTrain.Labels))  layers = [  layersTransfer  fullyConnectedLayer(numClasses,'WeightLearnRateFactor',20,'BiasLearnRateFactor',20)  softmaxLayer  classificationLayer]; |

• Train the VGG16 with your training data (from provided dataset)



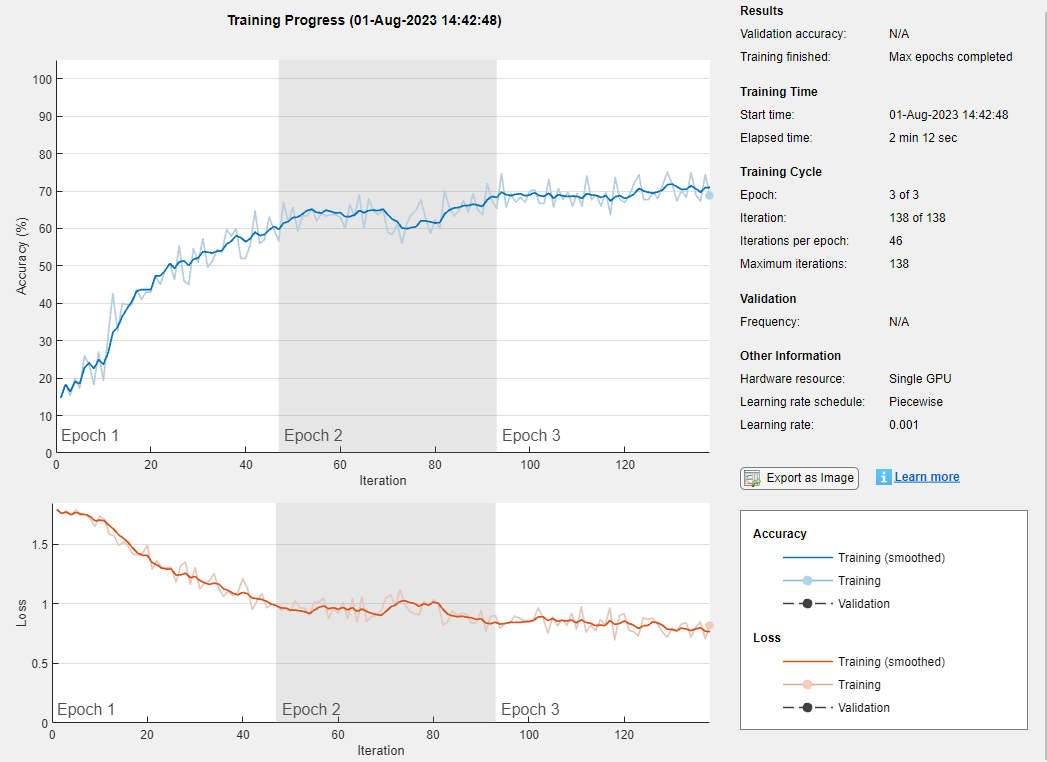
• Test the accuracy

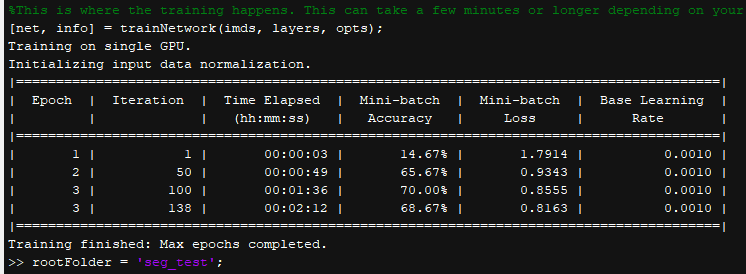


Task 2

• Train your network with the dataset provided

This time, not like the HW3 I did, I set epoch 3 with and I used the same design as HW3. To apply the data seg\_train and seg\_test, I added the new line for resize the data.





• Test the accuracy

A computer screen with white text

Description automatically generated

Comparisons:

• Which network is giving you best accuracy? And why ?

As the result, using network VGG16 gaves the better accuracy (91.56%) over the CNN built from scratch in HW3 (71.14%). This is because the network has the standard designed layers that is well studied and has a lot of trials. It will be very difficult to design the best layers for CNN from scratch in short amount of time.

• Can you think of anyother comparison matrices except accuracy? If so, in real world where we can consider those matrices while deciding which network to use?

Other thing I’ve noticed is that applying convolution in the layers of size 3\*3 with more layers has less computing cost over 5\*5.

Source Code: (Network)

|  |
| --- |
| %transfer learning with VGG16 with Seg\_train Dataset  % Load Training data  imds = imageDatastore('seg\_train', ...  'IncludeSubfolders',true, ...  'LabelSource','foldernames');  %Divide the data into training and validation data sets. Use 70% of the images for training and 30% for validation  [imdsTrain,imdsValidation] = splitEachLabel(imds,0.7,'randomized');  %Calculate the number of training class  numClasses = numel(categories(imdsTrain.Labels))  %load the pretrained VGG16  net = vgg16;  % Use analyzeNetwork to display an interactive visualization of the network architecture and detailed information about the network layers  analyzeNetwork(net)  %get the input size of the first layer  inputSize = net.Layers(1).InputSize  % Replace Final Layers  % The last three layers of the pretrained network net are configured for 1000 classes.  % These three layers must be fine-tuned for the new classification problem.  % Extract all layers, except the last three, from the pretrained network.  layersTransfer = net.Layers(1:end-3);  % Transfer the layers to the new classification task by replacing  % the last three layers with a fully connected layer, a softmax layer, and a classification output layer.  % Specify the options of the new fully connected layer according to the new data.  % Set the fully connected layer to have the same size as the number of classes in the new data.  % To learn faster in the new layers than in the transferred layers, increase the WeightLearnRateFactor and BiasLearnRateFactor values of the fully connected laye  numClasses = numel(categories(imdsTrain.Labels))  layers = [  layersTransfer  fullyConnectedLayer(numClasses,'WeightLearnRateFactor',20,'BiasLearnRateFactor',20)  softmaxLayer  classificationLayer];  % Train Network  % The network requires input images of size 227-by-227-by-3, but the images in the image datastores have different sizes.  % Use an augmented image datastore to automatically resize the training images.  % Specify additional augmentation operations to perform on the training images: randomly flip the training images along the vertical axis, and randomly translate them up to 30 pixels horizontally and vertically.  % Data augmentation helps prevent the network from overfitting and memorizing the exact details of the training images.  pixelRange = [-30 30];  imageAugmenter = imageDataAugmenter( ...  'RandXReflection',true, ...  'RandXTranslation',pixelRange, ...  'RandYTranslation',pixelRange);  augimdsTrain = augmentedImageDatastore(inputSize(1:2),imdsTrain, ...  'DataAugmentation',imageAugmenter);  %To automatically resize the validation images without performing further data augmentation, use an augmented image datastore without specifying any additional preprocessing operations.  augimdsValidation = augmentedImageDatastore(inputSize(1:2),imdsValidation);  %Specify the training options  options = trainingOptions('sgdm', ...  'MiniBatchSize',200, ...  'MaxEpochs',3, ...  'InitialLearnRate',1e-4, ...  'Shuffle','every-epoch', ...  'ValidationData',augimdsValidation, ...  'ValidationFrequency',3, ...  'Verbose',false, ...  'Plots','training-progress');  %Train the network that consists of the transferred and new layers. By default, trainNetwork uses a GPU if one is available, otherwise, it uses a CPU.  % Training on a GPU requires Parallel Computing Toolbox� and a supported GPU device  netTransfer = trainNetwork(augimdsTrain,layers,options);  %Load the Test data(this is a little different than previous code) Can you  %tell the diffence inoutput that it makes?  imds\_test\_f = imageDatastore('seg\_test', ...  'IncludeSubfolders',true, ...  'LabelSource','foldernames');  %remember, your test data size is also 32x32x3, But alexnet uses 227x227x3.  %So like the validation data , use agmented image datastore  Test\_data = augmentedImageDatastore(inputSize(1:2),imds\_test\_f);  %classify the test data. Note that you can also generate scores for each  %classes  [YPred,scores] = classify(netTransfer,Test\_data);  %create the Confusion matrix to calculate the accuracy  confMat = confusionmat(imds\_test\_f.Labels, YPred);  confMat = confMat./sum(confMat,2);  Accuracy= mean(diag(confMat)) |

Source Code: (Scratch CNN)

|  |
| --- |
| %Deep Learning Example: Traning from scratch using CIFAR-10 Dataset  %Copyright 2017 The MathWorks, Inc.  %This example explores creating a convolutional neural network (CNN) from scratch. You will need to download images in order to run this example.  %Please see the file in this directory: DownloadCIFAR10.m Running this file will help you download CIFAR10 if you choose to use those images.  % Running this file will download CIFAR10 and place the images into a training folder and test folder in the current directory  % These will be used for the three demos in this folder. % Please note this will take a few minutes to run, but only needs to be run  % once.  % Copyright 2017 The MathWorks, Inc.  %% Download the CIFAR-10 dataset  % if ~exist('cifar-10-batches-mat','dir')  % cifar10Dataset = 'cifar-10-matlab';  % disp('Downloading 174MB CIFAR-10 dataset...');  % websave([cifar10Dataset,'.tar.gz'],...  % ['https://www.cs.toronto.edu/~kriz/',cifar10Dataset,'.tar.gz']);  % gunzip([cifar10Dataset,'.tar.gz'])  % delete([cifar10Dataset,'.tar.gz'])  % untar([cifar10Dataset,'.tar'])  % delete([cifar10Dataset,'.tar'])  % end    %% Prepare the CIFAR-10 dataset  % if ~exist('cifar10Train','dir')  % disp('Saving the Images in folders. This might take some time...');  % saveCIFAR10AsFolderOfImages('cifar-10-batches-mat', pwd, true);  % end  %Load training data  % Please note: these are 4 of the 10 categories available  % Feel free to choose which ever you like best!  categories = {'buildings','forest','glacier','mountain','sea','street'};  %% ,'horse','bird','airplane','automobile','ship','truck'  rootFolder = 'seg\_train';  imds = imageDatastore(fullfile(rootFolder, categories), ...  'LabelSource', 'foldernames');  % img = imresize(imds,[32 32])  imageSize = [32, 32, 3];  imds.ReadFcn = @(filename)imresize(imread(filename), imageSize(1:2));  %Define Layers  %Training from scratch gives you a lot of freedom to explore the architecture. Take a look at this architecture and see how you might want to alter it: for example, how would you add another convolutional layer?  varSize = 32;  conv1 = convolution2dLayer(5,32,'Padding',2,'BiasLearnRateFactor',2);  conv1.Weights = gpuArray(single(randn([5 5 3 32])\*0.0001));  % conv2 = convolution2dLayer(5,32,'Padding',2, 'BiasLearnRateFactor',2);  % conv2.Weights = single(randn([5 5 32 32])\*0.0001);  %  % conv3 = convolution2dLayer(3,128,'Padding',2, 'BiasLearnRateFactor',2);  % conv3.Weights = single(randn([3 3 128 128])\*0.0001);  fc1 = fullyConnectedLayer(128,'BiasLearnRateFactor',2);  fc1.Weights = gpuArray(single(randn([128 512])\*0.1));  fc2 = fullyConnectedLayer(6,'BiasLearnRateFactor',2);  fc2.Weights = gpuArray(single(randn([6 128])\*0.1));  layers = [  imageInputLayer([varSize varSize 3]);  conv1;  maxPooling2dLayer(3,'Stride',1);  reluLayer();    convolution2dLayer(3,64,'Padding',"same",'BiasLearnRateFactor',2);  reluLayer();  averagePooling2dLayer(3,'Stride',2);  convolution2dLayer(5,128,'Padding',"same",'BiasLearnRateFactor',2);  reluLayer();  maxPooling2dLayer(3,'Stride',2);    convolution2dLayer(3,256,'Padding',"same",'BiasLearnRateFactor',2);  reluLayer();  averagePooling2dLayer(3,'Stride',2);  convolution2dLayer(5,512,'Padding',3,'BiasLearnRateFactor',2);  reluLayer();  maxPooling2dLayer(3,'Stride',2);  % convolution2dLayer(3,1024,'Padding',3,'BiasLearnRateFactor',2);  % reluLayer();  % maxPooling2dLayer(3,'Stride',2);    fc1;  reluLayer();  fc2;  softmaxLayer()  classificationLayer()];  %Define training options  %The training options is another parameter that can significantly increase or decrease the accuracy of the network. Try altering some of these values and see what happens to the overall accuracy of the network.  opts = trainingOptions('sgdm', ...  'InitialLearnRate', 0.001, ...  'LearnRateSchedule', 'piecewise', ...  'LearnRateDropFactor', 0.1, ...  'LearnRateDropPeriod', 8, ...  'L2Regularization', 0.004, ...  'MaxEpochs', 3, ...  'MiniBatchSize', 300, ...  'Verbose', true,...  'plots','training-progress');  %Train!  %This is where the training happens. This can take a few minutes or longer depending on your hardware. Training on a GPU is recommended.  [net, info] = trainNetwork(imds, layers, opts);  % Load test data  rootFolder = 'seg\_test';  imds\_test = imageDatastore(fullfile(rootFolder, categories), ...  'LabelSource', 'foldernames');  imageSize = [32, 32, 3];  imds\_test.ReadFcn = @(filename)imresize(imread(filename), imageSize(1:2));  %Do it all at once  %This section of code will run through all the test data and compare the predicted labels with the actual labels. This will give a feel for how the network is doing overall with one average prediction value.  % This could take a while if you are not using a GPU  labels = classify(net, imds\_test);  confMat = confusionmat(imds\_test.Labels, labels);  confMat = confMat./sum(confMat,2);  mean(diag(confMat)) |

Finally, I would like to inform part 1 is done by me and one of my classmate Shefiu due to the amount of the time it took for the process and his environment problem.