CPE-440 Final Project

By Michael Ryan

Goal

- Develop two different classifiers to classify whether a spectrogram image belonged to a 16QAM, 32QAM, or 64QAM wireless signal
- First Classifier would be a Convolutional Neural Network
- Second Classifier would be a Support Vector Machine

Process

- First, we will pass the given data into MatLab and split it into training and testing sets
- Next, we will train our own Neural Network through transfer learning and the use of a pre-trained neural network called Alexnet
- Once our CNN is trained, we classify the test data and output statistics
- After that, we take the features gathered by our CNN and pass them into our SVM for training
- Once our SVM is trained, we classify the test data and output statistics

Why use the features from our CNN?

- Training a multi-class SVM from scratch using built in MatLab feature extraction is very slow and computationally expensive
- Assuming the accuracy of our CNN is high, there is almost no chance of having low accuracy using this method

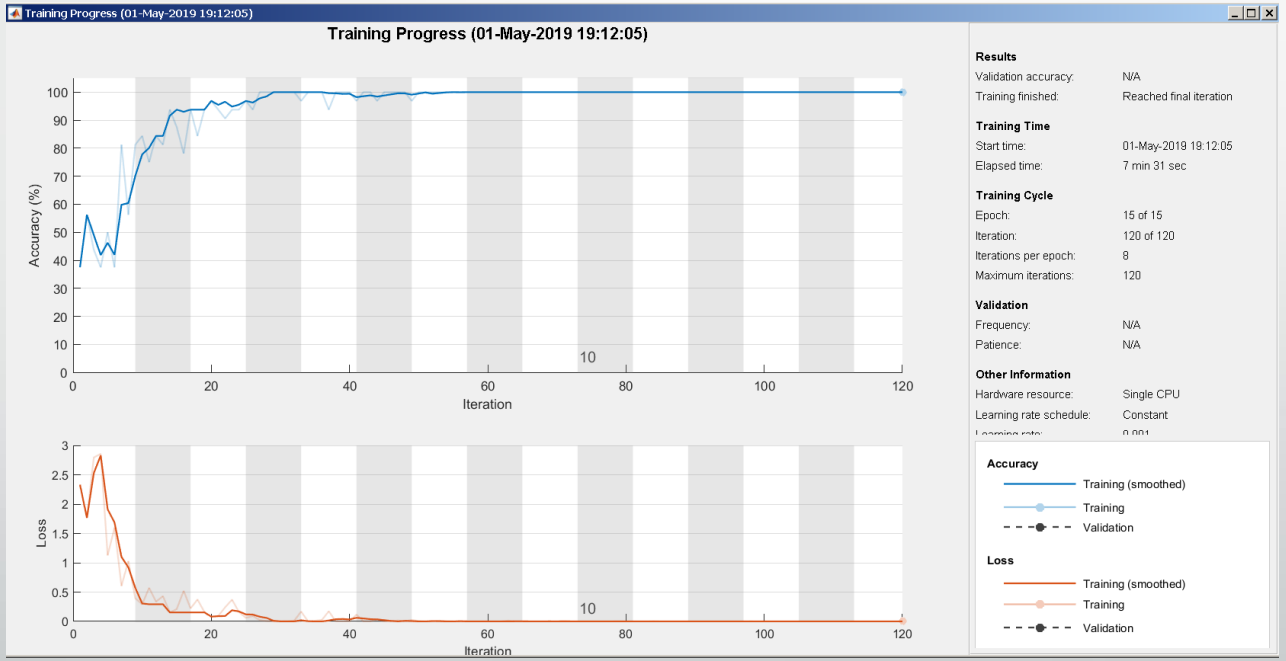
Setting up the data

```
%Michael Ryan
%CPE 440 Final Project
%First block of code will set up our data
%path will be used to pull the project data
path = 'C:\Users\Test\Documents\AI Final\Project Data';
%We will use imageDatastore to convert the raw project data into format
%matlab can use
data = imageDatastore(path, ...
    'IncludeSubfolders',true, ...
    'LabelSource','foldernames');
%split data into testing and training sets
[trainData,testData] = splitEachLabel(data,0.3,'randomize');
```

CNN Classifier

```
%%
%CNN Classifier
%We will use alexnet to train our own neural network using transfer
%learning
net = alexnet;
options = trainingOptions('sgdm', 'InitialLearnRate', 0.001, 'MaxEpochs', 15, 'MiniBatchSize', 32
    'Plots', 'training-progress');
%alter alexnet slightly to accomodate our classification of three classes
%only
layers = net.Layers;
layers(23) = fullyConnectedLayer(3);
layers(25) = classificationLayer;
%resize both sets of data according to the imageSize required by alexnet
imageSize = net.Layers(1).InputSize;
resizedTrainingData = augmentedImageDatastore(imageSize, trainData);
resizedTestData = augmentedImageDatastore(imageSize, testData);
%train new neural network through transfer learning
newNet = trainNetwork(resizedTrainingData, layers, options);
%classify using new CNN by inputting testData
cnnPredictedLabels = classify(newNet, resizedTestData);
%output statistics
cnnAccuracy = mean(cnnPredictedLabels == testData.Labels);
confMat = confusionmat(testData.Labels, cnnPredictedLabels);
```

Training


Training Progress (01-May-2019 19:12:05)
Training Progress (01-May-2019 19:12:05)
Accuracy (%)
Iteration
Loss
Iteration
Results
Validation accuracy: N/A
Training finished: Reached final iteration
Training Time
Start time: 01-May-2019 19:12:05
Elapsed time: 7 min 31 sec
Training Cycle
Epoch: 15 of 15
Iteration: 120 of 120
Iterations per epoch: 8
Maximum iterations: 120
Validation
Frequency: N/A
Patience: N/A
Other Information
Hardware resource: Single CPU
Learning rate schedule: Constant
Accuracy
Training (smoothed)
Training
Validation
Loss
Training (smoothed)
Training
Validation

Statistics

- Accuracy: 99.84% correct
- Confusion Matrix:

		Predicted		
		16QAM	32QAM	64QAM
TRUE	16QAM	210	0	0
	32QAM	1	209	0
	64QAM	0	0	210

SVM Classifier

```
%%
%SVM Classifier
%Rather than train a new SVM classifier from scratch which would take a lot of resources, we wi
%feautres extracted by our CNN which we trained above.
%We can choose whichever layer we want, but for best accuracy, the layer
%directly before classification would have extracted the "best" features
featureLayer = 'prob';
%pull features from featureLayer
trainingFeatures = activations(newNet, resizedTrainingData, featureLayer, ...
    'MiniBatchSize', 32, 'OutputAs', 'columns');
%train multi-class svm using extracted trainingFeatures
classifier = fitcecoc(trainingFeatures, trainData.Labels, ...
    'Learners', 'Linear', 'Coding', 'onevsall', 'ObservationsIn', 'columns');
%pull features from testing data
testFeatures = activations(newNet, resizedTestData, featureLayer, ...
    'MiniBatchSize', 32, 'OutputAs', 'columns');
%classify using the testFeatures
svmPredictedLabels = predict(classifier, testFeatures, 'ObservationsIn', 'columns');
%output statistics
svmConfMat = confusionmat(testData.Labels, svmPredictedLabels);
svmAccuracy = mean(svmPredictedLabels == testData.Labels);
%It is also possible to train an SVM model from scratch using matlab's
%built in feature extraction. This can be more accurate, but is also
%significantly more computationally expensive.
```

Statistics

- We expect the results to be similar since we are using features from our CNN
- Accuracy: also 99.84% correct
- Confusion Matrix:

		Predicted		
		16QAM	32QAM	64QAM
TRUE	16QAM	210	0	0
	32QAM	1	209	0
	64QAM	0	0	210