Documentation

F1: Complete
F2: Cross validation: Complete
ROC Curve: Not Complete
Confusion Matrix: Not Complete

How to run the program and software dependencies
Software dependencies:

- Python 3.5 or above
- Spyder
- Libraries
  - Keras
  - Tensorflow
  - Numpy
  - Pickle
  - Sklearn

To run the program, open spyder, ensure the modules above are imported, and run the file. You will be presented with a menu asking for an input. Read the menu and choose an option. This option will then execute and you will be presented with the output in the console window.

How the functionalities and additional requirements are implemented
To run f1 directly, i.e. see the results of functionality f1 by loading the saved models, without training, enter the number 1 or 2 to design and build the two different deep neural networks. They will not be trained here. To train the models, open the source code and go to the end of the code for both convolutional() and nonConv() methods and uncomment the last 2 lines. This will allow the models to be trained and saved.

For f2, only the cross validation has been implemented for the knn classifier, convolutional and non convolutional neural networks. To run this code, input 3 at the menu option. This will perform cross validation on the saved models. This does take a little while as it is performing 10 epochs at batch size 10 for each fold for each model (3). I have not done it for my implementation of knn from the last assignment as I had not correctly saved the model, therefore it would not work for cross validation. I otherwise would have easily implemented this.

Details of implementation (variables, description of model evaluation)
I have started by creating a menu for the user to choose their options, depending upon the option chosen, a different method and functionality will be run. I did this as it makes the program a lot more user friendly and simpler to run each separate functionality.

F1. If 1 is chosen, then the convolutional neural network is designed and built, but not trained or saved. The digits variable is assigned to the dataset containing the data for the digits. The dataset is then split with 0.2 as the testing data and assigned to XTrain, XTest, yTrain, yTest respectively. The X data is reshaped to the size of the data followed by 8, 8, 1, then normalised with an axis of 1, making the data between 1 and 0. The y data is one hot

Documentation

encoded to make the targets into an array. The array will have a single 1 in the position of the target label, the rest will be filled with zeros. The model is then defined as Sequential, and the layers are added. First the single dense layer with activation relu, followed by 2 convolutional layers. The model is then flattened to 1D and softmax is applied. There are comments within the source code to help explanation. The model is then compiled using the adam optimizer and the loss function is categorical_crossentropy as that is used for single label categorization. To train and save the model, uncomment the bottom 2 lines of the method.

F1. If 2 is chosen at the menu, then the non convolutional neural network is designed and built. This is almost exactly the same as the convolutional neural network but does not have a convolutional layer. The variables are the same as in the convolutional layer. XTrain/XTest holds the images, while yTrain/yTest holds the labels for the images. yTrainHot/yTestHot holds the one hot encoded version of the labels to be used when fitting. The model is once again called classif and the layers are added to the model using .add.

F2. If 3 is chosen at the menu, then cross validation is performed on the two deep neural networks and the knn classifier from the first assignment. My knn implementation from assignment 1 could not be loaded to perform cross validation as it was not quite correct. Otherwise, I would have implemented it easily.
The digits dataset is first split into 5 folds, reshaped/one hot encoded for the deep neural networks and kept original for the knn classification. Y1, y2, y3, y4, y5 and x1, x2, x3, x4, x5 both contain the labels/data for each fold. The models are then loaded under the names convModel, nonConv, and knnModel. A for loop is then used to iterate 5 times (as 5 folds required). For each fold, a copy of the original image and label data is kept, to prevent the changes affecting future folds. The test data (XTest, yTest, knnXTest, knnyTest), is the index i of the copies. The training data (XTrain, yTrain, knnXtrain, knnyTrain), are the remaining data from the dataset. To do this, I deleted the testing data from the copy, concatenated the remaining, and set it to corresponding variable. Using these variables, I was able to fit the models with the data for each fold. At the same time, the accuracy was calculated for each fold, where the accuracy variable for the respected model is updated by adding the new value (after it has been divided by 10 as 10 epochs). Finally, given the sum of the accuracies for each fold for each model, I was able to output to the user the overall input by dividing by 5 (as 5 folds in total).
REMEMBER: This will take a little while to run, roughly a minute and a half.

Parameters
.reshape(a, b) - reshapes data to fit neural network from a to shape b
normalize(a, b) - normalizes the data(a) from between 0 to 255, to between 0 and 1. With b being the axis, in this case axis = 1.
to_categorical(a) - one hot encoded a
Dense(a, activation = *) - where a is an integer, dimensionality of the output space and activation is the activation to use
Conv2D(a, (b,b), input_shape = (*)) - where a is an integer for output space, b is the matrix and input_shape is the 4D tensor with shape (samples, rows, cols, channels)

Documentation

MaxPooling2D(pool_size = *) - where pool_size is a tuple of 2 integers by which to downscale (vertical, horizontal)
.compile(optimizer = a, loss = b, metrics = c) - where a is “adam” optimizer that is an adaptive learning rate method, meaning it computes individual learning rates for different parameters. b is categorical crossentropy where the targets are in categorical format, and metrics show the accuracy of the model.
np.array_split(a, b) - where a is the array to split and b is the number to split it by
pickle.load(open(a, b)) - where a is the file name and b is the type of read
np.delete(a, b) - deletes index b from list a
Np.concatenate(a) - concatenates all of a
.fit(a, b, batch_size = *, epochs = *, validation_data = (c, d)) - where a is the X training data, b is the label for the training data, batch_size defines the number of samples that are propagated through the network, epochs is a measure of the number of times all the training vectors are used once to update the weights. C and d are the testing data and labels respectively.