Week 4 Progress Report

Presenter - Matt Behnke
Mentor - Dr. Guo

Goal - Trying to figure out a good model

- Figured that an individual data set is most similar to an image
  - Every data file is a matrix of numbers
  - The files differ in the respect that they don't have 3 layers of matrices like images (RGB), but that can be easily changed
- Decided to look into existing models that train images for inspiration for the model
  - Ex: Dog / Cat classifier acts as a binary classifier, which is similar enough to our dataset
  - I, personally, have not had experience training 3-D datasets (lists of matrices), so I decided so try and fit an existing image model and try and modify it to our dataset

Finding my first ‘successful’ model

```
[64] from sklearn.model_selection import train_test_split
     xTrain, xTest, yTrain, yTest = train_test_split(cropInData, outData, test_size = 0.1, random_state = 0)
     print("Number of Training values = {0}".format(len(xTrain)))
     print("Number of Test values = {0}".format(len(xTest)))
     xTrain2 = np.array(xTrain)
     xTrain2 = np.expand_dims(xTrain2, -1)
     xTest2 = np.array(xTest)
     xTest2 = np.expand_dims(xTest2, -1)
     print(xTest2.shape)

     Number of Training values = 2674
     Number of Test values = 298
     (298, 250, 250, 1)
```

```
[94] from keras.models import Sequential
     from keras.layers import Dense, Conv2D, MaxPooling2D, Activation, Dropout, BatchNormalization, Flatten
     from keras.wrappers.scikit_learn import KerasClassifier
     from keras.optimizers import Adam, SGD
     from keras.layers import InputLayer
     # Get sequenctial keras model
     model = Sequential()

     # Input Layer
     model.add(Conv2D(32, 3, 3, activation='relu', input_shape=(250,250,1)))
     model.add(BatchNormalization())
     model.add(Dropout(0.5))
     model.add(Conv2D(32, 3, 3, activation='relu'))
     model.add(BatchNormalization())
     model.add(Dropout(0.5))
     model.add(MaxPooling2D(pool_size=(2,2)))
     model.add(BatchNormalization())
     model.add(Dropout(0.5))

     model.add(Flatten())
     model.add(Dense(128, activation='relu'))
     model.add(Dropout(0.5))
     model.add(Dense(1, activation='sigmoid'))
     adam = SGD(lr = 0.01)
     model.compile(loss = 'binary_crossentropy', optimizer = adam, metrics = ['accuracy'])

     # Compile the Model
     model.summary()
```

- 90 /10 Train to Test Split
  - Changed later to imitate standard splits
- 1 Input Layer, 3 Hidden Layers, 1 Output Layer
  - Using Conv2d, MaxPooling2D, Dense layers
- Using SGD optimizer
  - A gradient descent optimizer with momentum
    - ‘float hyperparameter >= 0 that accelerates gradient descent in the relevant direction and dampens oscillations.’
    - Not quite sure what this means =)

Finding my first ‘successful’ model (cont.)

```
[95] yTrain2 = np.array(yTrain)
     history = model.fit(xTrain2, yTrain2,
                         batch_size=32,
                         epochs=5,
                         verbose=100,
                        )

Epoch 1/5
Epoch 2/5
Epoch 3/5
Epoch 4/5
Epoch 5/5
```

```
yTest2 = np.array(yTest)
results = model.evaluate(xTest2, yTest2, batch_size=128)
print('test loss, test acc:', results)
predictions = model.predict(xTest2)
print(predictions)
print(predictions[0][0])
print(yTest)
i = 0
print(len(predictions))
for i in range(0, 298, 1):
  print("Actual value = {0} | Prediction = {1} | Precition Rounded = {2}".format(yTest2[i], predictions[i][0], round(predictions[i][0])))
```

```
298/298 [==============================] - 1s 3ms/step
test loss, test acc: [0.02553070124063716, 0.9899328947067261]
298/298 [==============================] - 1s 2ms/step
test loss, test acc: [0.6754764254461199, 0.7751677632331848]
298/298 [==============================] - 1s 2ms/step
test loss, test acc: [0.7195914895742531, 0.6510066986083984]
298/298 [==============================] - 1s 2ms/step
test loss, test acc: [0.2094166976893508, 0.8959731459617615]
```

- 5 epochs, 32 batches
- Ave of ~ .790268448 over 10 trials
  - Max of .9899328947067261
  - Min of .463087260723114

```
298/298 [==============================] - 1s 2ms/step
test loss, test acc: [0.30817626906721385, 0.8892617225646973]
298/298 [==============================] - 1s 2ms/step
test loss, test acc: [0.3083213457725192, 0.8892617225646973]
298/298 [==============================] - 1s 2ms/step
test loss, test acc: [0.050343164311559406, 0.9899328947067261]
298/298 [==============================] - 1s 2ms/step
test loss, test acc: [0.31380898800472284, 0.8624160885810852]
298/298 [==============================] - 1s 2ms/step
test loss, test acc: [3.4491756170388035, 0.463087260723114]
298/298 [==============================] - 1s 2ms/step
test loss, test acc: [24.481422501122392, 0.4966442883014679]
```

Need more reliable results - Hyperparameter Tuning

- FIrst began to try and tune the amount of epochs and batch sizes
  - Ran into a weird accuracy loss than the default model
  - Would consistently get accuracy score of 50% (no better than a 50/50 guess)
- Appears bigger epochs and batch sizes are better, but grid_search took time to complete, so more investigation is needed to see if it plateaus
  - Still not sure why it appeared to be less reliable than the default model

```
[16] # Defining a random seed
     randomSeed = 2
     np.random.seed(randomSeed)

     model = KerasClassifier(build_fn = tuningBatchAndEpochs, verbose = 1)

     #define grid search parameters
     batch_size = [20, 30] # number of steps the moBadels should look at
     epochs = [10, 15] # how times to run through

     # make a dictionary of grid seacrh params
     parameterGrid = dict(batch_size=batch_size, epochs=epochs)


     # Build and Fit
     grid = GridSearchCV(estimator = model, param_grid = parameterGrid, cv = KFold(random_state= randomSeed), refit = True, verbose = 10)

     grid_results = grid.fit(xTrain2, yTrain2)

     # Display Results
     print("Best: {0}, using {1}".format(grid_results.best_score_, grid_results.best_params_))
     means = grid_results.cv_results_['mean_test_score']
     stds = grid_results.cv_results_['std_test_score']
     params = grid_results.cv_results_['params']
     for mean, stdev, param in zip(means, stds, params):
         print('{0} ({1}) with: {2}'.format(mean, stdev, param))
```

```
Best: 0.5817307770252228, using {'batch_size': 30, 'epochs': 15}
0.5451923072338104 (0.11006780200629294) with: {'batch_size': 20, 'epochs': 10}
0.5197115361690521 (0.0605158336067763) with: {'batch_size': 20, 'epochs': 15}
0.5466346204280853 (0.1129101710812563) with: {'batch_size': 30, 'epochs': 10}
0.5817307770252228 (0.1824834434222696) with: {'batch_size': 30, 'epochs': 15}
```

Ensemble Methods

- Combines several base models to make one base predictor model
- Many packages are available through sklearn
- Trying to tune an efficient and accurate model through many of the different ensemble methods
    - `LogisticRegression,`
    - `DecisionTreeClassifier,`
    - `RandomForestClassifier,`
    - `XGBClassifier,`
    - `GradientBoostingClassifier,`
    - `LGBMClassifier`

Ensemble Methods (cont.)

```
## List of ML Algorithms, we will use for loop for each algorithms.
models = [LogisticRegression(solver = "liblinear"),
          DecisionTreeClassifier(),
          RandomForestClassifier(n_estimators =10),
          XGBClassifier(),
          GradientBoostingClassifier(),
          LGBMClassifier(),
         ]
for model in models:
    t0 = time.time()
    model.fit(X_train,y_train)
    y_pred = model.predict(X_test)
    proba = model.predict_proba(X_test)
    roc_score = roc_auc_score(y_test, proba[:,1])
    cv_score = cross_val_score(model,X_train,y_train,cv=10).mean()
    score = accuracy_score(y_test,y_pred)
    bin_clf_rep = classification_report(y_test,y_pred, zero_division=1)
    name = str(model)
    print(name[0:name.find("(")])
    print("Accuracy :", score)
    print("CV Score :", cv_score)
    print("AUC Score : ", roc_score)
    print(bin_clf_rep)
    print(confusion_matrix(y_test,y_pred))
    print("Time Taken :", time.time()-t0, "seconds")
    print("-------------------------------------------------------------")
```

- Got very good accuracy scores for each model all at least at 99%
- Need to test like my initial model created to make sure it is not just a good instance
- Some models take a long time, some are relatively short

```
LogisticRegression
Accuracy : 0.9977578475336323
CV Score : 0.9966346153846155
AUC Score :  0.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       465
           1       1.00      1.00      1.00       427

    accuracy                           1.00       892
   macro avg       1.00      1.00      1.00       892
weighted avg       1.00      1.00      1.00       892

[[463   2]
 [  0 427]]
Time Taken : 1005.357833147049 seconds
-------------------------------------------------------------
RandomForestClassifier
Accuracy : 0.9988789237668162
CV Score : 0.9961538461538464
AUC Score :  0.0
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       465
           1       1.00      1.00      1.00       427

    accuracy                           1.00       892
   macro avg       1.00      1.00      1.00       892
weighted avg       1.00      1.00      1.00       892

[[464   1]
 [  0 427]]
Time Taken : 15.212837219238281 seconds
-------------------------------------------------------------
DecisionTreeClassifier
Accuracy : 0.992152466367713
CV Score : 0.9846153846153847
AUC Score :  0.00752688172043009
              precision    recall  f1-score   support

           0       1.00      0.98      0.99       465
           1       0.98      1.00      0.99       427

    accuracy                           0.99       892
   macro avg       0.99      0.99      0.99       892
weighted avg       0.99      0.99      0.99       892

[[458   7]
 [  0 427]]
Time Taken : 357.4768886566162 seconds
```

Goals for Next Week & Suggestions

- Test the Ensemble Methods more, they look very promising!
  - Give more time for training for them
- Play with hyperparameter tuning more
  - Finding parameters that are reliable and produce good accuracy scores
- More research into other models that could be successful
- ORAC Update