Step 0 Import Required Packages

In [27]:

```
import numpy as np
import scipy.io
import sklearn.metrics
import sklearn
import os
import random
import pandas as pd
import time
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
import pickle
from sklearn.preprocessing import StandardScaler
from keras import Sequential
from keras.layers import Dense, Activation, Flatten, Input, Dropout, BatchNormalization
from keras.models import Model
from keras import initializers
from keras.optimizers import Adam
import matplotlib.pyplot as plt
import tensorflow as tf
```

Step 1 Read The Files

In [2]:

```
# When using Colab, you can upload train_set.zip in the content folder and run this ker
nel.
# !unzip -qq /content/train_set.zip
```

In [28]:

```
# Set your directory to read the data, default is the directory in colab.
unzipped_folder_path = 'C:/Users/wannian/Desktop/face/train_set'
```

In [29]:

```
def read_data(unzipped_folder_path):

  # read labels
  labels = pd.read_csv(unzipped_folder_path+'/label.csv')
  y= labels['label'].to_numpy()

  # read points
  n = 3000
  for i in range(1,n+1):
    p_path = str(i).zfill(4)+'.mat'
    mat = scipy.io.loadmat(unzipped_folder_path+'/points/'+p_path)
    if 'faceCoordinatesUnwarped' in mat:
      cords = mat['faceCoordinatesUnwarped']
    else:
      cords = mat['faceCoordinates2']

    distance = sklearn.metrics.pairwise_distances(cords)
          # compute the pairwise distances in each mat
    flatten_distance = distance[np.triu_indices(len(cords[:,0]), k = 1)]
          # stretch the upper triangle of the symmetric matrix
          # to a long array with dimension 3003
          # 3003 = (1+77)*78/2
    if i==1:
      distances = np.mat([flatten_distance])
    else:
      distances = np.append(distances, np.mat([flatten_distance]), axis = 0)
  return (distances, y)
```

In [30]:

```
read_time_start=time.time()
Ori_X, Ori_Y = read_data(unzipped_folder_path)
print("Read the original dataset takes %s seconds" % round((time.time() - read_time_sta
rt),3))
```

```
Read the original dataset takes 78.222 seconds
```

In [31]:

```
Ori_X.shape, Ori_Y.shape
# should be (3000,3003) and (3000,)
# which means 3000 number of cases
# and 3003 numbers of pairwise distances
# of 78 fiducial points.
# 3003 = (1+77)*78/2
```

Out[31]:

```
((3000, 3003), (3000,))
```

Step 2 Data Preprocessing For the Imbalanced Dataset & Generate New Data to Improve Learning Accuracy

From the following analysis, we found that the Original Dataset is unbalanced. So we decided to generate new data for the class with smaller number of original samples. By generating new data, we not only balanced the data with equal number of samples in different class, but also create new data to help improve the learning accuracy.

- Because the number of Class 1 samples is less than the number of Class 0 samples, we decided to add more data in Class 1.
- The way we generate more data is that we randomly select two original cordinates of fiducial points in Class 1 and average them to generate new data of fiducial points and then calculate its pairwise distances and give it the label of 1.
- It would make sense cause our models believe that the fiducial points in the same class will generate similar distribution in pairwise distances.

In [32]:

```
# Analyzing the data
n = Ori_Y.shape[0]
print('The number of class 0 is ' + str(n-sum(Ori_Y)))
print('The number of class 1 is ' + str(sum(Ori_Y)))
print('Only %.2f'% (sum(Ori_Y)/n*100) + '% of total dataset are class 1. ')
print('So, it is an unbalanced dataset, we need to do some data preprocessing.')
print('Here, we are using oversampling to generate more class 1 datasets.')
```

```
The number of class 0 is 2402
The number of class 1 is 598
Only 19.93% of total dataset are class 1.
So, it is an unbalanced dataset, we need to do some data preprocessing.
Here, we are using oversampling to generate more class 1 datasets.
```

In [33]:

```
def data_preprocessing(Ori_X, Ori_Y, unzipped_folder_path):

  # data preprocessing

  distances = Ori_X
  y = Ori_Y

  n = y.shape[0]
  mat_1 = np.add(np.where(y == 1),1)
  n_oversample = (n-sum(y))-sum(y)
    # how many samples do we need to generate

  for i in range(n_oversample):
    samples_index = random.sample(list(list(mat_1)[0]), 2)
      # pick two random index of class 1 samples.

    p_path = str(samples_index[0]).zfill(4)+'.mat'
    mat = scipy.io.loadmat(unzipped_folder_path+'/points/'+p_path)
    if 'faceCoordinatesUnwarped' in mat:
      cords_0 = mat['faceCoordinatesUnwarped']
    else:
      cords_0 = mat['faceCoordinates2']

    p_path = str(samples_index[1]).zfill(4)+'.mat'
    mat = scipy.io.loadmat(unzipped_folder_path+'/points/'+p_path)
    if 'faceCoordinatesUnwarped' in mat:
      cords_1 = mat['faceCoordinatesUnwarped']
    else:
      cords_1 = mat['faceCoordinates2']

    cords_new = (cords_0 + cords_1) / 2
        # averaging two sets of cordinates to generate new set of cordinates
    distance = sklearn.metrics.pairwise_distances(cords_new)
        # compute the pairwise distances in each mat
    flatten_distance = distance[np.triu_indices(len(cords_new[:,0]), k = 1)]
        # stretch the upper triangle of the symmetric matrix
        # to a long array with dimension 3003
        # 3003 = (1+77)*78/2

    distances = np.append(distances, np.mat([flatten_distance]), axis = 0)
    y = np.append(y,np.array(1))
        # Append new data to the original dataset

  return (distances, y)
```

In [34]:

```
Balanced_X, Blanced_Y = data_preprocessing(Ori_X, Ori_Y, unzipped_folder_path)
```

In [35]:

```
Balanced_X.shape, Blanced_Y.shape
```

Out[35]:

```
((4804, 3003), (4804,))
```

Step 3 Baseline Model: GBM on Original Dataset

1. Create train and test features and labels

In [9]:

```
#Create train and test features and labels from Balanced Data set
train_features, test_features, train_labels, test_labels = train_test_split(Ori_X,Ori_Y
,test_size=0.2,random_state=42)
print(train_features.shape,test_features.shape,train_labels.shape,test_labels.shape)
```

```
(2400, 3003) (600, 3003) (2400,) (600,)
```

2. Train a GBM model using random parameters on original data set

In [10]:

```
gbm = GradientBoostingClassifier(learning_rate=0.1,max_depth=2,n_estimators=100)
start_time=time.time()
gbm.fit(train_features, train_labels)
print("Training  model takes %s seconds" % round((time.time() - start_time),3))
```

```
Training  model takes 209.637 seconds
```

In [11]:

```
print('Accuracy of the GBM on test set: {:.3f}'.format(gbm.score(test_features,test_lab
els)))

start = time.time()
prediction = gbm.predict(test_features)
end = time.time()

predprob = gbm.predict_proba(test_features)[:,1]

print("Predicting test data takes %s seconds" % round((end - start),3))
print('Classification error rate:', np.mean(np.array(test_labels)!= prediction))
print('Classification report \n', classification_report(test_labels, prediction))

#Since the class distribution is imbalanced/ skewed, we should look at the confusion ma
trix and AUC
print('Confusion Matrix \n', confusion_matrix(test_labels, prediction))
print('AUC is: {:.4f}'.format(roc_auc_score(test_labels, predprob)))
```

```
Accuracy of the GBM on test set: 0.797
Predicting test data takes 0.014 seconds
Classification error rate: 0.20333333333333334
Classification report
               precision    recall  f1-score   support

           0       0.80      0.98      0.88       461
           1       0.74      0.19      0.30       139

    accuracy                           0.80       600
   macro avg       0.77      0.58      0.59       600
weighted avg       0.79      0.80      0.75       600

Confusion Matrix
 [[452   9]
 [113  26]]
AUC is: 0.7992
```

3.GBM Cross Validation and Parameter tuning

3.1 Cross Validation on GBM learning rate and max_depth

In [18]:

```
# param_grid = {'learning_rate':[0.05,0.1], 'max_depth': [1,2,3]}
# grid = GridSearchCV(GradientBoostingClassifier(),param_grid,refit=True,verbose=3)
# grid.fit(train_features,train_labels)
```

In [ ]:

```
# print(grid.best_params_)
# print(grid.best_estimator_)
```

best_params: {'learning_rate': 0.1, 'max_depth': 2}

3.2 CrossValidation on GBM with n_estimators

In [ ]:

```
# param_grid2 = {'n_estimators':[50,100,250,500]}
# grid2 = GridSearchCV(GradientBoostingClassifier(learning_rate = 0.1, max_depth = 2),p
aram_grid= param_grid2,refit=True,verbose=3)
# grid2.fit(train_features,train_labels)
```

In [ ]:

```
# print(grid2.best_params_)
# print(grid2.best_estimator_)
# grid2_predictions = grid2.predict(test_features)
# print(confusion_matrix(test_labels,grid2_predictions))
# print(classification_report(test_labels,grid2_predictions))
```

best_params: {'n_estimators': 500}

3.3 Best GBM Model

- Final Parameter for baseline GBM set at: learning_rate=0.1, n_estimators=500, max_depth=2

In [12]:

```
#Training baseline: GBM using best parameters found above through CV

gbm_best = GradientBoostingClassifier(learning_rate=0.1,max_depth=2,n_estimators=500)
start_time=time.time()
gbm_best.fit(train_features, train_labels)
print("Training  model takes %s seconds" % round((time.time() - start_time),3))
```

```
Training  model takes 1058.803 seconds
```

3.4 Evaluate BGM Model

In [13]:

```
print('Accuracy of the GBM on test set: {:.3f}'.format(gbm_best.score(test_features,tes
t_labels)))

start = time.time()
baseline_pred = gbm_best.predict(test_features)
end = time.time()

baseline_predprob = gbm_best.predict_proba(test_features)[:,1]

print("Predicting test data takes %s seconds" % round((end - start),3))
print('Classification error rate:', np.mean(np.array(test_labels)!= baseline_pred))
print('Classification report \n', classification_report(test_labels, baseline_pred))

#Since the class distribution is imbalanced/ skewed, we should look at the confusion ma
trix and AUC
print('Confusion Matrix \n', confusion_matrix(test_labels, baseline_pred))
print('AUC is: {:.4f}'.format(roc_auc_score(test_labels, baseline_predprob)))
```

```
Accuracy of the GBM on test set: 0.817
Predicting test data takes 0.031 seconds
Classification error rate: 0.18333333333333332
Classification report
               precision    recall  f1-score   support

           0       0.83      0.96      0.89       461
           1       0.71      0.35      0.47       139

    accuracy                           0.82       600
   macro avg       0.77      0.65      0.68       600
weighted avg       0.80      0.82      0.79       600

Confusion Matrix
 [[441  20]
 [ 90  49]]
AUC is: 0.8091
```

Cross validation improved accuracy from 0.797 to 0.82, and AUC from 0.797 to 0.81

3.5 Save The Model

In [16]:

```
# Save best gbm model
save_weights_path = '../output/baseline_gbm.p'
pickle.dump(gbm_best, open(save_weights_path,'wb'))
```

Step 4 Advanced Model -- Densely Connected Neural Network

- Based on the paper Densely Connected Convolutional Networks (https://arxiv.org/abs/1608.06993) , Desely Connected Convolutional Neural Networks is a good model for image classification. With the improved data -- fiducial points, we will get a better accuracy and auc.

4.1 Data Scaling On Balanced Dataset And Train Test Split

In [36]:

```
scaler = StandardScaler()
scaler.fit(Balanced_X)
distances_scale = scaler.transform(Balanced_X)
```

In [37]:

```
X_train, X_test, y_train, y_test = sklearn.model_selection.train_test_split(distances_s
cale, Blanced_Y, random_state=123)
```

In [38]:

```
one_hot_test=tf.one_hot(y_test,depth=2)
one_hot_train=tf.one_hot(y_train,depth=2)
```

4.2 Build The Architecture Of The Model

In [39]:

```
model = tf.keras.Sequential([
        Input([3003]),
        BatchNormalization(),
        Dense(600,activation='relu',kernel_initializer=initializers.glorot_normal(seed=
4)),
        Dropout(0.25),
        BatchNormalization(),
        Dense(300,activation='relu',kernel_initializer=initializers.glorot_normal(seed=
4)),
        Dropout(0.25),
        Dense(150,activation='relu',kernel_initializer=initializers.glorot_normal(seed=
4)),
        Dropout(0.25),
        Dense(50,activation='relu',kernel_initializer=initializers.glorot_normal(seed=4
)),
        Dense(2,activation='softmax',kernel_initializer=initializers.glorot_normal(seed
=4))
])
```

In [40]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
batch_normalization (BatchNo (None, 3003)              12012     
_________________________________________________________________
dense (Dense)                (None, 600)               1802400   
_________________________________________________________________
dropout (Dropout)            (None, 600)               0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 600)               2400      
_________________________________________________________________
dense_1 (Dense)              (None, 300)               180300    
_________________________________________________________________
dropout_1 (Dropout)          (None, 300)               0         
_________________________________________________________________
dense_2 (Dense)              (None, 150)               45150     
_________________________________________________________________
dropout_2 (Dropout)          (None, 150)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 50)                7550      
_________________________________________________________________
dense_4 (Dense)              (None, 2)                 102       
=================================================================
Total params: 2,049,914
Trainable params: 2,042,708
Non-trainable params: 7,206
_________________________________________________________________
```

In [41]:

```
start_time = time.time()
model.compile(loss='binary_crossentropy',optimizer = Adam(lr=0.001),metrics=['accuracy'
])
model_history = model.fit(X_train,one_hot_train,epochs = 80)
print("training  model takes %s seconds" % round((time.time() - start_time),3))
```

```
Epoch 1/80
113/113 [==============================] - 3s 26ms/step - loss: 0.6487 - a
ccuracy: 0.6531
Epoch 2/80
113/113 [==============================] - 3s 26ms/step - loss: 0.5531 - a
ccuracy: 0.7238
Epoch 3/80
113/113 [==============================] - 3s 27ms/step - loss: 0.5005 - a
ccuracy: 0.7560
Epoch 4/80
113/113 [==============================] - 3s 27ms/step - loss: 0.4647 - a
ccuracy: 0.7785
Epoch 5/80
113/113 [==============================] - 3s 26ms/step - loss: 0.4408 - a
ccuracy: 0.7893
Epoch 6/80
113/113 [==============================] - 3s 27ms/step - loss: 0.4197 - a
ccuracy: 0.8007
Epoch 7/80
113/113 [==============================] - 3s 26ms/step - loss: 0.3887 - a
ccuracy: 0.8213 2s - los
Epoch 8/80
113/113 [==============================] - 3s 27ms/step - loss: 0.3916 - a
ccuracy: 0.8224
Epoch 9/80
113/113 [==============================] - 3s 28ms/step - loss: 0.3587 - a
ccuracy: 0.8349
Epoch 10/80
113/113 [==============================] - 3s 26ms/step - loss: 0.3591 - a
ccuracy: 0.8368
Epoch 11/80
113/113 [==============================] - 3s 26ms/step - loss: 0.3475 - a
ccuracy: 0.8429 0s - loss: 0
Epoch 12/80
113/113 [==============================] - 3s 24ms/step - loss: 0.3316 - a
ccuracy: 0.8551
Epoch 13/80
113/113 [==============================] - 3s 24ms/step - loss: 0.3117 - a
ccuracy: 0.8609
Epoch 14/80
113/113 [==============================] - 3s 25ms/step - loss: 0.3058 - a
ccuracy: 0.8615
Epoch 15/80
113/113 [==============================] - 3s 25ms/step - loss: 0.2925 - a
ccuracy: 0.8757
Epoch 16/80
113/113 [==============================] - 3s 24ms/step - loss: 0.3041 - a
ccuracy: 0.8648
Epoch 17/80
113/113 [==============================] - 2s 22ms/step - loss: 0.2830 - a
ccuracy: 0.8737
Epoch 18/80
113/113 [==============================] - 2s 22ms/step - loss: 0.2698 - a
ccuracy: 0.8823
Epoch 19/80
113/113 [==============================] - 2s 22ms/step - loss: 0.2830 - a
ccuracy: 0.8712
Epoch 20/80
113/113 [==============================] - 3s 23ms/step - loss: 0.2709 - a
ccuracy: 0.8804 0s - loss: 0.275
Epoch 21/80
```

```
113/113 [==============================] - 2s 21ms/step - loss: 0.2721 - a
ccuracy: 0.8740
Epoch 22/80
113/113 [==============================] - 2s 20ms/step - loss: 0.2544 - a
ccuracy: 0.8879
Epoch 23/80
113/113 [==============================] - 2s 20ms/step - loss: 0.2493 - a
ccuracy: 0.8893
Epoch 24/80
113/113 [==============================] - 2s 20ms/step - loss: 0.2467 - a
ccuracy: 0.8926
Epoch 25/80
113/113 [==============================] - 2s 20ms/step - loss: 0.2405 - a
ccuracy: 0.8890 0s - loss: 0.2407 - accura
Epoch 26/80
113/113 [==============================] - 2s 20ms/step - loss: 0.2584 - a
ccuracy: 0.8859
Epoch 27/80
113/113 [==============================] - 2s 21ms/step - loss: 0.2333 - a
ccuracy: 0.8959
Epoch 28/80
113/113 [==============================] - 2s 20ms/step - loss: 0.2369 - a
ccuracy: 0.8945
Epoch 29/80
113/113 [==============================] - 2s 22ms/step - loss: 0.2307 - a
ccuracy: 0.9009
Epoch 30/80
113/113 [==============================] - 2s 20ms/step - loss: 0.2197 - a
ccuracy: 0.9009
Epoch 31/80
113/113 [==============================] - 2s 20ms/step - loss: 0.2302 - a
ccuracy: 0.8954
Epoch 32/80
113/113 [==============================] - 2s 20ms/step - loss: 0.2222 - a
ccuracy: 0.9045
Epoch 33/80
113/113 [==============================] - 2s 20ms/step - loss: 0.2142 - a
ccuracy: 0.9059 0s - loss: 0.2231 - accura - ETA: 0s - loss: 0.2179 - accu
racy
Epoch 34/80
113/113 [==============================] - 2s 21ms/step - loss: 0.2246 - a
ccuracy: 0.9076
Epoch 35/80
113/113 [==============================] - 2s 20ms/step - loss: 0.2015 - a
ccuracy: 0.9178
Epoch 36/80
113/113 [==============================] - 2s 20ms/step - loss: 0.2024 - a
ccuracy: 0.9159
Epoch 37/80
113/113 [==============================] - 2s 20ms/step - loss: 0.1912 - a
ccuracy: 0.9153
Epoch 38/80
113/113 [==============================] - 3s 25ms/step - loss: 0.1925 - a
ccuracy: 0.9187
Epoch 39/80
113/113 [==============================] - 2s 21ms/step - loss: 0.1911 - a
ccuracy: 0.9192
Epoch 40/80
113/113 [==============================] - 2s 20ms/step - loss: 0.1885 - a
ccuracy: 0.9170
Epoch 41/80
```

```
113/113 [==============================] - 2s 21ms/step - loss: 0.1892 - a
ccuracy: 0.9203
Epoch 42/80
113/113 [==============================] - 2s 20ms/step - loss: 0.1896 - a
ccuracy: 0.9278
Epoch 43/80
113/113 [==============================] - 2s 21ms/step - loss: 0.1729 - a
ccuracy: 0.9301
Epoch 44/80
113/113 [==============================] - 2s 20ms/step - loss: 0.1888 - a
ccuracy: 0.9234
Epoch 45/80
113/113 [==============================] - 2s 21ms/step - loss: 0.1679 - a
ccuracy: 0.9292 0s
Epoch 46/80
113/113 [==============================] - 3s 23ms/step - loss: 0.1581 - a
ccuracy: 0.9326
Epoch 47/80
113/113 [==============================] - 3s 23ms/step - loss: 0.1698 - a
ccuracy: 0.9262
Epoch 48/80
113/113 [==============================] - 3s 23ms/step - loss: 0.1726 - a
ccuracy: 0.9265
Epoch 49/80
113/113 [==============================] - 3s 22ms/step - loss: 0.1598 - a
ccuracy: 0.9339
Epoch 50/80
113/113 [==============================] - 3s 23ms/step - loss: 0.1619 - a
ccuracy: 0.9320
Epoch 51/80
113/113 [==============================] - 3s 23ms/step - loss: 0.1660 - a
ccuracy: 0.9348
Epoch 52/80
113/113 [==============================] - 3s 23ms/step - loss: 0.1562 - a
ccuracy: 0.9381
Epoch 53/80
113/113 [==============================] - 2s 22ms/step - loss: 0.1597 - a
ccuracy: 0.9345 0s - loss: 0.1588 - accuracy: 0.
Epoch 54/80
113/113 [==============================] - 2s 20ms/step - loss: 0.1389 - a
ccuracy: 0.9414
Epoch 55/80
113/113 [==============================] - 2s 19ms/step - loss: 0.1617 - a
ccuracy: 0.9348
Epoch 56/80
113/113 [==============================] - 2s 19ms/step - loss: 0.1462 - a
ccuracy: 0.9392
Epoch 57/80
113/113 [==============================] - 2s 19ms/step - loss: 0.1649 - a
ccuracy: 0.9331
Epoch 58/80
113/113 [==============================] - 2s 19ms/step - loss: 0.1509 - a
ccuracy: 0.9417
Epoch 59/80
113/113 [==============================] - 2s 20ms/step - loss: 0.1350 - a
ccuracy: 0.9498
Epoch 60/80
113/113 [==============================] - 2s 21ms/step - loss: 0.1467 - a
ccuracy: 0.9412
Epoch 61/80
113/113 [==============================] - 3s 24ms/step - loss: 0.1421 - a
```

```
ccuracy: 0.9414
Epoch 62/80
113/113 [==============================] - 3s 25ms/step - loss: 0.1285 - a
ccuracy: 0.9495
Epoch 63/80
113/113 [==============================] - 3s 26ms/step - loss: 0.1372 - a
ccuracy: 0.9462
Epoch 64/80
113/113 [==============================] - 2s 21ms/step - loss: 0.1284 - a
ccuracy: 0.9462
Epoch 65/80
113/113 [==============================] - 3s 23ms/step - loss: 0.1388 - a
ccuracy: 0.9467
Epoch 66/80
113/113 [==============================] - 2s 21ms/step - loss: 0.1421 - a
ccuracy: 0.9459
Epoch 67/80
113/113 [==============================] - 2s 21ms/step - loss: 0.1350 - a
ccuracy: 0.9478
Epoch 68/80
113/113 [==============================] - 2s 20ms/step - loss: 0.1258 - a
ccuracy: 0.9512
Epoch 69/80
113/113 [==============================] - 2s 21ms/step - loss: 0.1215 - a
ccuracy: 0.9517
Epoch 70/80
113/113 [==============================] - 2s 21ms/step - loss: 0.1204 - a
ccuracy: 0.9509 0s - loss: 0.1198 - accuracy:
Epoch 71/80
113/113 [==============================] - 2s 22ms/step - loss: 0.1229 - a
ccuracy: 0.9517
Epoch 72/80
113/113 [==============================] - 2s 20ms/step - loss: 0.1107 - a
ccuracy: 0.9534
Epoch 73/80
113/113 [==============================] - 2s 20ms/step - loss: 0.1124 - a
ccuracy: 0.9548
Epoch 74/80
113/113 [==============================] - 2s 20ms/step - loss: 0.1138 - a
ccuracy: 0.9520
Epoch 75/80
113/113 [==============================] - 2s 21ms/step - loss: 0.1153 - a
ccuracy: 0.9520
Epoch 76/80
113/113 [==============================] - 2s 21ms/step - loss: 0.1077 - a
ccuracy: 0.9567
Epoch 77/80
113/113 [==============================] - 2s 21ms/step - loss: 0.1061 - a
ccuracy: 0.9567
Epoch 78/80
113/113 [==============================] - 2s 21ms/step - loss: 0.1191 - a
ccuracy: 0.9520
Epoch 79/80
113/113 [==============================] - 2s 22ms/step - loss: 0.1079 - a
ccuracy: 0.9589
Epoch 80/80
113/113 [==============================] - 2s 22ms/step - loss: 0.1211 - a
ccuracy: 0.9536
training  model takes 204.233 seconds
```

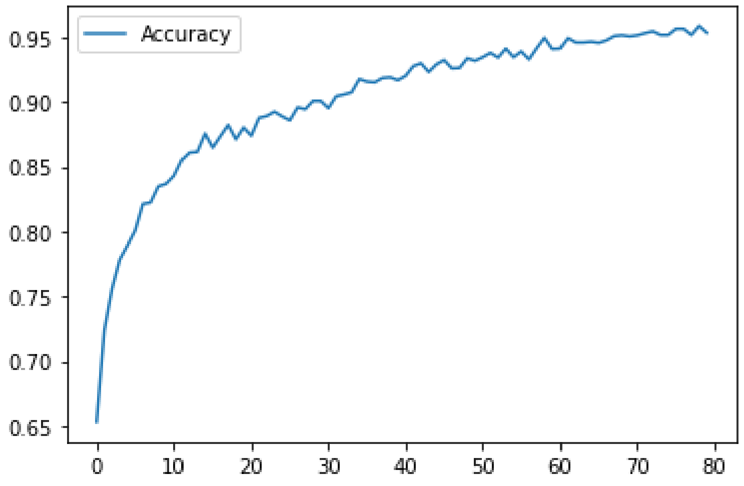
4.3 Visualize The Training Process

In [42]:

```
his_plot = pd.DataFrame(model_history.history)
plt.plot(his_plot['accuracy'],label = 'Accuracy')
plt.legend()
```

Out[42]:

```
<matplotlib.legend.Legend at 0x2292f4c4850>
```

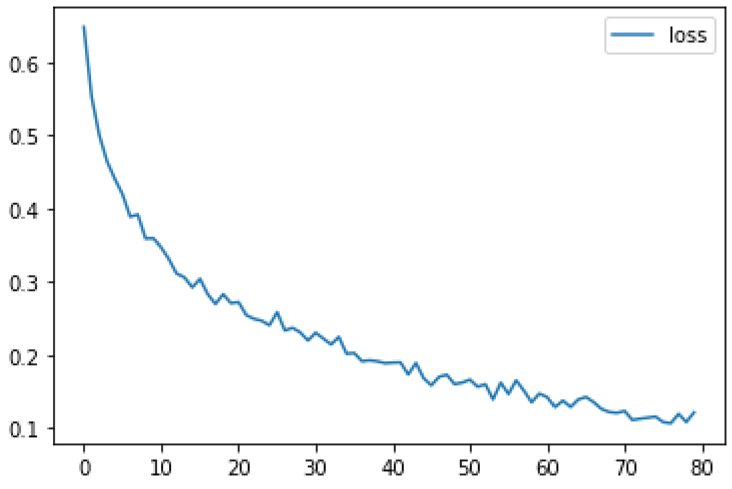


In [43]:

```
plt.plot(his_plot['loss'],label = 'loss')
plt.legend()
```

Out[43]:

```
<matplotlib.legend.Legend at 0x2293a80fee0>
```



4.4 Evaluate The Model On Test Accuracy and AUC

In [44]:

```
# Test on the balanced dataset
start_time = time.time()
y_fitprob = model.predict(X_train)
y_fit = np.argmax(y_fitprob, axis=-1)
print("Testing model on train_dataset takes %s seconds" % round((time.time() - start_ti
me),3))

start_time = time.time()
y_predprob = model.predict(X_test)
y_pred = np.argmax(y_predprob, axis=-1)
print("Testing model on test_dataset takes %s seconds" % round((time.time() - start_tim
e),3))

print("Train dataset -- Accuracy:  %.2f" % sklearn.metrics.accuracy_score(y_train, y_fi
t))
print("Train dataset -- AUC:  %.2f" % sklearn.metrics.roc_auc_score(one_hot_train, y_fi
tprob))
print("Test dataset -- Accuracy:  %.2f" % sklearn.metrics.accuracy_score(y_test,y_pred
))
print("Test dataset -- AUC:  %.2f" % sklearn.metrics.roc_auc_score(one_hot_test, y_pred
prob))
```

```
Testing model on train_dataset takes 1.308 seconds
Testing model on test_dataset takes 0.231 seconds
Train dataset -- Accuracy:  0.99
Train dataset -- AUC:  1.00
Test dataset -- Accuracy:  0.90
Test dataset -- AUC:  0.98
```

In [45]:

```
# Test on the original dataset

scaler = StandardScaler()
scaler.fit(Ori_X)
ori_scale = scaler.transform(Ori_X)
one_hot_o = tf.one_hot(Ori_Y,depth=2)

start_time = time.time()
y_fitprob_o = model.predict(ori_scale)
y_fit = np.argmax(y_fitprob_o, axis=-1)
print("Testing model on original dataset takes %s seconds" % round((time.time() - start
_time),3))

print("Train dataset -- Accuracy:  %.2f" % sklearn.metrics.accuracy_score(Ori_Y, y_fit
))
print("Train dataset -- AUC:  %.2f" % sklearn.metrics.roc_auc_score(one_hot_o, y_fitpro
b_o))
```

```
Testing model on original dataset takes 0.458 seconds
Train dataset -- Accuracy:  0.85
Train dataset -- AUC:  0.98
```

We can see that the model have 0.9-0.91 Test Auccuracy and 0.97-0.98 AUC. It can generalize well.

4.5 Save The Model

In [46]:

```
# Save the model
model.save("../output/DNN")
```

```
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\tensorf
low\python\training\tracking\tracking.py:111: Model.state_updates (from te
nsorflow.python.keras.engine.training) is deprecated and will be removed i
n a future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied
automatically.
WARNING:tensorflow:From C:\ProgramData\Anaconda3\lib\site-packages\tensorf
low\python\training\tracking\tracking.py:111: Layer.updates (from tensorfl
ow.python.keras.engine.base_layer) is deprecated and will be removed in a
future version.
Instructions for updating:
This property should not be used in TensorFlow 2.0, as updates are applied
automatically.
INFO:tensorflow:Assets written to: ../output/DNN\assets
```

Step 5 Predict Test Data

- If you skip the previous steps and want to predict test data immediately, please import the required dataset.

In [1]:

```
import numpy as np
import scipy.io
import sklearn.metrics
import sklearn
import os
import random
import pandas as pd
import time
import pickle
from sklearn.preprocessing import StandardScaler
import keras.models
import tensorflow as tf
```

- Run the function to read the test dataset

In [2]:

```
def read_test_data(unzipped_folder_path,n):

  # read points
  for i in range(1,n+1):
    p_path = str(i).zfill(4)+'.mat'
    mat = scipy.io.loadmat(unzipped_folder_path+'/points/'+p_path)
    if 'faceCoordinatesUnwarped' in mat:
      cords = mat['faceCoordinatesUnwarped']
    else:
      cords = mat['faceCoordinates2']

    distance = sklearn.metrics.pairwise_distances(cords)
          # compute the pairwise distances in each mat
    flatten_distance = distance[np.triu_indices(len(cords[:,0]), k = 1)]
          # stretch the upper triangle of the symmetric matrix
          # to a long array with dimension 3003
          # 3003 = (1+77)*78/2
    if i==1:
      distances = np.mat([flatten_distance])
    else:
      distances = np.append(distances, np.mat([flatten_distance]), axis = 0)
  return (distances)
```

- Set path to the unfolded test dataset and path to where you want to store the output csv file.

In [8]:

```
test_data_path = 'C:/Users/wannian/Desktop/face/train_set'
output_labels_path = 'C:/Users/wannian/Desktop/face/train_set/label_prediction.csv'
save_weights_path = '../output/baseline_gbm.p'
```

- Read the data

In [4]:

```
start_time = time.time()
n=2000
test_distances = read_test_data(test_data_path,n)
print("Read the training dataset takes %s seconds" % round((time.time() - start_time),3
))
```

```
Read the training dataset takes 36.417 seconds
```

In [5]:

```
test_distances.shape
```

Out[5]:

```
(2000, 3003)
```

- Scale the data

In [6]:

```
# Scale the input distances
scaler = StandardScaler()
scaler.fit(test_distances)
test_distances_scale = scaler.transform(test_distances)
```

- Load the model

In [47]:

```
# Load DNN model
predict_model = keras.models.load_model("../output/DNN")
# Load gbm model
predict_model_baseline=pickle.load(open(save_weights_path,'rb'))
```

- Predict the labels

In [16]:

```
start_time = time.time()
test_predprob = predict_model.predict(test_distances_scale)
test_classes = np.argmax(test_predprob, axis=-1)
test_classes_baseline = predict_model_baseline.predict(test_distances)
print("Testing model on test dataset takes %s seconds" % round((time.time() - start_tim
e),3))
```

```
Testing model on test dataset takes 0.587 seconds
```

- Write the labels to csv

In [25]:

```
df=pd.DataFrame(np.array(range(1,n+1)),columns=['Index'])
df['Baseline']=pd.DataFrame(test_classes_baseline)
df['Advanced']=pd.DataFrame(test_classes)
```

In [26]:

```
df.to_csv(output_labels_path)
```

In [ ]:

```
 
```