CSE 572: Lab 14

In this lab, you will practice implementing techniques for dimensionality reduction using features extracted by pre-trained neural networks and classification with transfer learning.

To execute and make changes to this notebook, click File > Save a copy to save your own version in your Google Drive or Github. Read the step-by-step instructions below carefully. To execute the code, click on each cell below and press the SHIFT-ENTER keys simultaneously or by clicking the Play button.

When you finish executing all code/exercises, save your notebook then download a copy (.ipynb file). Submit the following **three** things:

1. a link to your Colab notebook,
2. the .ipynb file, and
3. a pdf of the executed notebook on Canvas.

To generate a pdf of the notebook, click File > Print > Save as PDF.

Load the dataset

We will use the same “Labeled Faces in the Wild” (LFW) dataset that we used in Labs 12 and 13 for this lab. Since we are going to use neural networks pre-trained using the ImageNet database which has RGB color images, we will load in the images in color rather than grayscale format (`color=True`) and keep the original image size (`resize=1.0`).

```
from sklearn.datasets import fetch_lfw_people

lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.8, color=True)

# data attribute gives the data matrix with the image dimension flattened
X = lfw_people.data
print('Num samples: {}'.format(X.shape[0]))
print('Num features: {}'.format(X.shape[1]))

# images attribute gives the unflattened image dimension
print('Image dimensions: {} x {}'.format(lfw_people.images.shape[1], lfw_people.images.shape[2]))

# the label to predict is the id of the person
y = lfw_people.target
# target_names attribute tells us the name of the person associated with each id
target_names = lfw_people.target_names
n_classes = target_names.shape[0]
print('Num classes: {}'.format(n_classes))
print('Class names:', target_names)
```

```
Num samples: 1288
Num features: 22500
Image dimensions: 100 x 75
Num classes: 7
Class names: ['Ariel Sharon' 'Colin Powell' 'Donald Rumsfeld' 'George W Bush'
 'Gerhard Schroeder' 'Hugo Chavez' 'Tony Blair']
```

The dataset has 1,288 images of the faces of 7 different people. The code below prints the number of samples from each class.

```
for i in range(n_classes):
    print('Number of instances in class {} ({}): {}'.format(i,
                                                          target_names[i],
                                                          y[y==i].shape[0]
                                                         ))
```

```
Number of instances in class 0 (Ariel Sharon): 77
Number of instances in class 1 (Colin Powell): 236
Number of instances in class 2 (Donald Rumsfeld): 121
Number of instances in class 3 (George W Bush): 530
Number of instances in class 4 (Gerhard Schroeder): 109
Number of instances in class 5 (Hugo Chavez): 71
Number of instances in class 6 (Tony Blair): 144
```

To get an idea of what is in our dataset, we visualize a random face from each class below. (Note: You can run this cell many times to see different random examples.)

```
import numpy as np
import matplotlib.pyplot as plt

fig, axes = plt.subplots(ncols=n_classes, figsize=(13,6))

for i in range(n_classes):
    rand_ind = np.random.randint(0, y[y==i].shape[0])
    axes[i].imshow(lfw_people.images[y==i][rand_ind])
    axes[i].axis('off')
    axes[i].set_title(target_names[i])
```

Next, we will split the data into training and test subsets, using 30% of the data for testing.

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)
```

Next we standardize the data so that the mean of all attributes is 0 using the StandardScaler() object in scikit-learn to standardize the data. We fit the scaler to the training data and apply it to both the training and test data.

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Feature extraction with pre-trained network

One approach for non-linear dimensionality reduction is to extract features using a pre-trained neural network. The network is typically a large architecture that has shown state of the art performance for the pre-training task using a large dataset, in this case ImageNet. We will use the InceptionV3 model pre-trained with ImageNet for our feature extractor.

Note that we pass the `include_top=False` argument to get the InceptionV3 architecture *without* the final classification layer so that the output will be the last hidden layer of the model.

```
import tensorflow as tf

from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.applications.inception_v3 import preprocess_input

# Set the random seed for reproducibility
seed = 0
tf.keras.utils.set_random_seed(seed)


base_model = InceptionV3(weights='imagenet', include_top=False)

base_model.summary()
```

```
                              2048)                                'mixed9_0[0][0]',
                                                                   'concatenate_2[0][0]',
                                                                   'activation_178[0][0]']

 conv2d_183 (Conv2D)          (None, None, None,   917504      ['mixed9[0][0]']
                              448)

 batch_normalization_183 (Batch  (None, None, None,   1344       ['conv2d_183[0][0]']
 Normalization)               448)

 activation_183 (Activation)  (None, None, None,   0           ['batch_normalization_183[0][0]']
                              448)

 conv2d_180 (Conv2D)          (None, None, None,   786432      ['mixed9[0][0]']
                              384)

 conv2d_184 (Conv2D)          (None, None, None,   1548288     ['activation_183[0][0]']
                              384)

 batch_normalization_180 (Batch  (None, None, None,   1152       ['conv2d_180[0][0]']
 Normalization)               384)

 batch_normalization_184 (Batch  (None, None, None,   1152       ['conv2d_184[0][0]']
 Normalization)               384)

 activation_180 (Activation)  (None, None, None,   0           ['batch_normalization_180[0][0]']
                              384)

 activation_184 (Activation)  (None, None, None,   0           ['batch_normalization_184[0][0]']
                              384)

 conv2d_181 (Conv2D)          (None, None, None,   442368      ['activation_180[0][0]']
                              384)

 conv2d_182 (Conv2D)          (None, None, None,   442368      ['activation_180[0][0]']
                              384)

 conv2d_185 (Conv2D)          (None, None, None,   442368      ['activation_184[0][0]']
                              384)

 conv2d_186 (Conv2D)          (None, None, None,   442368      ['activation_184[0][0]']
                              384)

 average_pooling2d_17 (AverageP  (None, None, None,   0          ['mixed9[0][0]']
 ooling2D)                    2048)

 conv2d_179 (Conv2D)          (None, None, None,   655360      ['mixed9[0][0]']
```

Now that we've loaded the pre-trained InceptionV3 model, we can use it to process the new feature vectors for our LFW dataset. First, we need to reshape the flattend image vector to back into the original image dimensions (125 x 94 x 3).

```
# Reshape to be 2D images
X_train_rshp = X_train.reshape([X_train.shape[0],
                                lfw_people.images.shape[1],
                                lfw_people.images.shape[2],
                                lfw_people.images.shape[3]])

X_train_rshp.shape
```

```
(901, 100, 75, 3)
```

Now we input our images through `base_model` to get the feature vector output. Note that we also apply the `preprocess_input()` function to apply the same preprocessing that was used for pre-training the InceptionV3 model.

```
X_train_features = base_model.predict(preprocess_input(X_train_rshp))
```

```
29/29 [==============================] - 13s 352ms/step
```

The dimensions of each sample output by the model are 2 x 1 x 2048 (2048 features), so we will reshape these feature vectors to flatten the 2x1x2048 dimension. The result is a dataset of 4096-dimensional feature vectors, one for each of our 901 training samples.

```
X_train_features.shape
```

```
(901, 1, 1, 2048)
```

```
X_train_features = X_train_features.reshape([X_train_features.shape[0],
                                        X_train_features.shape[1]*X_train_features.shape[2]*X_train_features.shape[3]])
```

```
X_train_features.shape
```

```
(901, 2048)
```

SVM classifier with InceptionV3 features

Now that we've extracted new features using InceptionV3, we can use these new features as our inputs for classification. In this example, we'll use a Support Vector Machine (SVM) classifier. In the cell below, train an SVM with `C=10000` and `kernel='rbf'` as we did in Labs 11-12.

```
# YOUR CODE HERE
from sklearn.svm import SVC

clf = SVC(C=10000, kernel='rbf')
clf = clf.fit(X_train_features, y_train)
```

In the cell(s) below, use the trained model to extract the InceptionV3 feature vectors for the test set. Note that you will have to reshape the test data and the output features in the same way we did for the training data.

```
# YOUR CODE HERE
X_test_reshape = X_test.reshape([X_test.shape[0],
                                lfw_people.images.shape[1],
                                lfw_people.images.shape[2],
                                lfw_people.images.shape[3]])
```

```
X_test_features = base_model.predict(preprocess_input(X_test_reshape))
X_test_features = X_test_features[:,0,0]
print("Shape of X_test_reshape: {}".format(X_test_reshape.shape))
print("Shape of X_test_features: {}".format(X_test_features.shape))
```

```
13/13 [==============================] - 4s 300ms/step
Shape of X_test_reshape: (387, 100, 75, 3)
Shape of X_test_features: (387, 2048)
```

Now that you've extracted the InceptionV3 features for the test dataset, you can use the SVM classifier to make predictions for the test set based on these feature vectors. Do this in the cell below and store your predictions in a variable named `y_pred`.

```
# YOUR CODE HERE
y_pred = clf.predict(X_test_features)
```

The next cell shows how we can use the `classification_report()` function in sklearn to print several metrics computed for the test set.

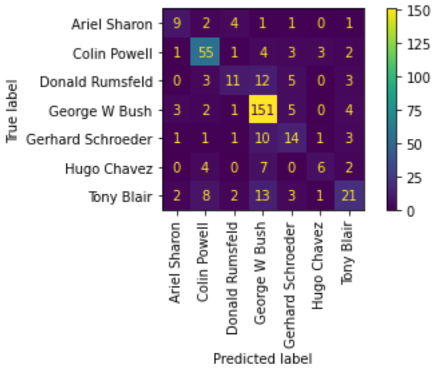
```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred, target_names=target_names))
```

```
                   precision    recall  f1-score   support

     Ariel Sharon       0.56      0.50      0.53        18
     Colin Powell       0.73      0.80      0.76        69
  Donald Rumsfeld       0.55      0.32      0.41        34
    George W Bush       0.76      0.91      0.83       166
Gerhard Schroeder       0.45      0.45      0.45        31
      Hugo Chavez       0.55      0.32      0.40        19
       Tony Blair       0.58      0.42      0.49        50

         accuracy                           0.69       387
        macro avg       0.60      0.53      0.55       387
     weighted avg       0.67      0.69      0.67       387
```

We can also use the `ConfusionMatrixDisplay` object to visualize the confusion matrix for the test set predictions.

```
from sklearn.metrics import ConfusionMatrixDisplay
```

```
ConfusionMatrixDisplay.from_estimator(
    clf, X_test_features, y_test, display_labels=target_names, xticks_rotation="vertical"
)

plt.tight_layout()
plt.show()
```



Question 1: Compare the results from the SVM with InceptionV3 features to the results from KernelPCA in Lab 13 and linear PCA in Lab 12. How do the results differ?

Answer:

YOUR ANSWER HERE Better results obtained with SVM with InceptionV3

Fine-tune pre-trained network for face classification

One way to increase performance even further is to "fine-tune" the InceptionV3 model for our new classification task by adding a new classification layer to the end of the network and optionally unfreezing some of the later layers in the network.

In the cells below, we load our InceptionV3 pre-trained base model in the same way we did in the first exercise.

```
import tensorflow as tf
tf.config.run_functions_eagerly(True)

from tensorflow.keras import layers
from tensorflow.keras import Model
```

```
pre_trained_model = InceptionV3(
    input_shape=(lfw_people.images.shape[1], lfw_people.images.shape[2], 3), include_top=False, weights='imagenet')
```

For this exercise, we'll freeze all of the layers in the original InceptionV3 model by setting the `trainable` attribute of each layer to `False`.

```
for layer in pre_trained_model.layers:
    layer.trainable = False
```

We then truncate the pre-trained model at one of the deeper hidden layers and append new layers, including our new classification layer for our LFW face classes, to the end.

```
last_layer = pre_trained_model.get_layer('mixed7')

print('last layer output shape:', last_layer.output_shape)

last_output = last_layer.output
```

```
     last layer output shape: (None, 4, 3, 768)
```

```
# Flatten the output layer to 1 dimension
x = layers.Flatten()(last_output)

# Add a fully connected layer with 1,024 hidden units and ReLU activation
x = layers.Dense(1024, activation='relu')(x)

# Add a dropout rate of 0.2
x = layers.Dropout(0.2)(x)
```

```
# Add a final sigmoid layer for classification
x = layers.Dense(len(target_names), activation='sigmoid')(x)
```

We now compile the new model.

```
# Configure and compile the model
model = Model(pre_trained_model.input, x)
model.compile(loss='categorical_crossentropy',
              optimizer='sgd',
              metrics=['acc'])
```

Next, we can use our LFW training data and labels to fine-tune the model. Note that first we need to convert our labels to a 1-hot encoding.

```
# Convert labels to one-hot encoding
y_train_cat = tf.keras.utils.to_categorical(y_train, num_classes=len(target_names))
```

```
model.fit(x=X_train_rshp, y=y_train_cat,
                epochs=10,
                batch_size=32,
                shuffle=True)
```

```
Epoch 1/10
/usr/local/lib/python3.8/dist-packages/tensorflow/python/data/ops/structured_function.py:256: UserWarning: Even though the `tf.config.ex
  warnings.warn(
29/29 [==============================] - 68s 2s/step - loss: 2.9507 - acc: 0.3818
Epoch 2/10
29/29 [==============================] - 74s 3s/step - loss: 1.6614 - acc: 0.4373
Epoch 3/10
29/29 [==============================] - 64s 2s/step - loss: 1.5993 - acc: 0.4295
Epoch 4/10
29/29 [==============================] - 66s 2s/step - loss: 1.5049 - acc: 0.4806
Epoch 5/10
29/29 [==============================] - 64s 2s/step - loss: 1.5120 - acc: 0.4906
Epoch 6/10
29/29 [==============================] - 65s 2s/step - loss: 1.3971 - acc: 0.5294
Epoch 7/10
29/29 [==============================] - 65s 2s/step - loss: 1.4142 - acc: 0.5105
Epoch 8/10
29/29 [==============================] - 65s 2s/step - loss: 1.3419 - acc: 0.5283
Epoch 9/10
29/29 [==============================] - 64s 2s/step - loss: 1.3594 - acc: 0.5216
Epoch 10/10
29/29 [==============================] - 64s 2s/step - loss: 1.3318 - acc: 0.5383
<keras.callbacks.History at 0x7fbf952bbe80>
```

Now that we've fine-tuned our model, we can use it to make predictions about our test set. Use the trained model to make predictions for the test dataset in the cell(s) below.

```
# YOUR CODE HERE
y_pred = model.predict(X_test_reshape)

y_pred.shape
```

```
13/13 [==============================] - 7s 574ms/step
(387, 7)
```

The `predict()` function returns the class probabilities output by the model for each of the 7 classes. To get the index of the predicted class, we can use the `np.argmax()` function to get the index with the highest class probability.

```
y_pred_class = np.argmax(y_pred, axis=1)
```

Now that we have an array of the predicted classes for the test set, we can compute the performance metrics as in the first exercise. In the cell below, print the classification report for the test set.

```
# YOUR CODE HERE
print(classification_report(y_test, y_pred_class, target_names=target_names))
```

```
                   precision    recall  f1-score   support

     Ariel Sharon       0.23      0.56      0.33        18
     Colin Powell       0.54      0.88      0.67        69
  Donald Rumsfeld       1.00      0.06      0.11        34
    George W Bush       0.71      0.93      0.80       166
Gerhard Schroeder       0.00      0.00      0.00        31
      Hugo Chavez       0.17      0.05      0.08        19
       Tony Blair       0.80      0.08      0.15        50

         accuracy                           0.60       387
        macro avg       0.49      0.37      0.31       387
     weighted avg       0.61      0.60      0.51       387

/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-de
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-de
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.8/dist-packages/sklearn/metrics/_classification.py:1344: UndefinedMetricWarning: Precision and F-score are ill-de
  _warn_prf(average, modifier, msg_start, len(result))
```

Question 2: Compare the results from the fine-tuned network to the SVM with InceptionV3 features from the first exercise. How do the results differ?

Answer:

YOUR ANSWER HERE First Result:

```
                 precision    recall  f1-score   support

Ariel Sharon       0.56      0.50      0.53        18
Colin Powell       0.73      0.80      0.76        69
Donald Rumsfeld    0.55      0.32      0.41        34
George W Bush      0.76      0.91      0.83       166
Gerhard Schroeder  0.45      0.45      0.45        31
Hugo Chavez        0.55      0.32      0.40        19
Tony Blair         0.58      0.42      0.49        50

    accuracy                           0.69       387
   macro avg       0.60      0.53      0.55       387
weighted avg       0.67      0.69      0.67       387
```

Second Result:

```
              precision    recall  f1-score   support

Ariel Sharon       0.23      0.56      0.33        18
Colin Powell       0.54      0.88      0.67        69
Donald Rumsfeld    1.00      0.06      0.11        34
George W Bush      0.71      0.93      0.80       166
Gerhard Schroeder  0.00      0.00      0.00        31
Hugo Chavez        0.17      0.05      0.08        19
Tony Blair         0.80      0.08      0.15        50

    accuracy                           0.60       387
   macro avg       0.49      0.37      0.31       387
weighted avg       0.61      0.60      0.51       387
```

If we compare both models first and second, we can see that, for some classes it is comparable, for precision values SVM classifier outperfoermed well than this one. Recall values are comparables while for f1-score are better in SVM

```
**Question 3: What are some reasons that might explain the difference
performance between the fine-tuned network and the SVM with InceptionV
features?**

**Answer:**
```

Question 3: What are some reasons that might explain the difference in performance between the fine-tuned network and the SVM with InceptionV3 features?

```
YOUR ANSWER HERE

It is probable that the model is overfitting to the dominant classes d
fine-tuning since the classes with the lowest accuracy, recall, and F1
have less sample sizes than the classes with greater performance.

It might also be underfit because the model hasn't encountered enough
of the uncommon classes to develop useful features for recognizing the
trained the model for more epochs, this issue might be resolved.
```

Answer:

YOUR ANSWER HERE

It is probable that the model is overfitting to the dominant classes during fine-tuning since the classes with the lowest accuracy, recall, and F1 scores have less sample sizes than the classes with greater performance

It might also be underfit because the model hasn't encountered enough instances of the uncommon classes to develop useful features for

0s completed at 6:52 AM