# Lab Demonstration 2
# Pattern Recognition

Shekhar "Shakes" Chandra

Version 1.41

In this lab, we will study dimensionality reduction and classification as a way of learning the basics of Tensorflow/PyTorch. The foundation of these frameworks is based on linear algebra in a similar manner to Numpy. As a consequence, a number of functions from Numpy are also available.

The lab is partitioned into three main parts. Firstly, we will look to introduce PCA of human faces and classification using the random forest, where you will be asked to create a TF/PyTorch equivalent program. Then build a traditional convolutional neural network (CNN) approach to do the same. Finally, you will have a chance to implement a CNN solution of your choice from a list of problems and data in order to solve an example recognition problem.

**[This is the lab sheet for the Pattern Recognition Demonstration. You must attempt as many tasks (ideally all tasks) AND demonstrate it to the instructor in one of your practical sessions BEFORE the due date in order to be awarded marks. Please check the ECP for the correct due date. Note that tasks are 'complete' and marks are awarded by attempting each task AND correctly answering related questions to the satisfaction of the instructor.]**

## 1 Part 1 - Eigenfaces (4 Marks)

We will compute Eigenfaces - the PCA of human faces using Numpy and the funnelled "Labeled Faces in the Wild" (LFW). Scikit learn Eigenfaces example is modified below to show how the PCA model of faces is constructed using Numpy arrays.

First load the relevant data and functions

```
from sklearn.datasets import fetch_lfw_people
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import numpy as np

# Download the data, if not already on disk and load it as numpy arrays
lfw_people = fetch_lfw_people(min_faces_per_person=70, resize=0.4)
```

Extract the meaningful parameters of the faces dataset

```
# introspect the images arrays to find the shapes (for plotting)
n_samples, h, w = lfw_people.images.shape

# for machine learning we use the 2 data directly (as relative pixel
# positions info is ignored by this model)
X = lfw_people.data
n_features = X.shape[1]

# the label to predict is the id of the person
y = lfw_people.target
target_names = lfw_people.target_names
n_classes = target_names.shape[0]

print("Total dataset size:")
print("n_samples: %d" % n_samples)
print("n_features: %d" % n_features)
print("n_classes: %d" % n_classes)
```

It is important in machine learning to split the data accordingly into training and testing sets to avoid contamination of the model. Ideally, you should also have a validation set.

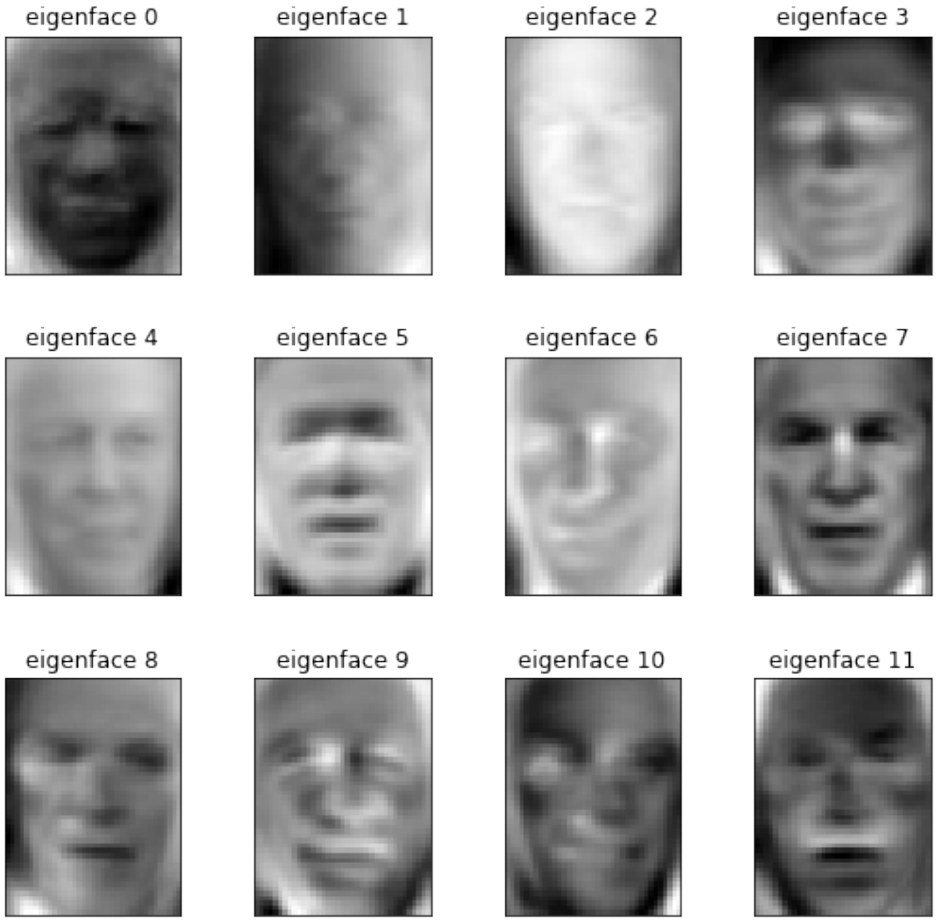
```
# Split into a training set and a test set using a stratified k fold
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state=42)

# Compute a PCA (eigenfaces) on the face dataset (treated as unlabeled
# dataset): unsupervised feature extraction / dimensionality reduction
n_components = 150
```

Compute the PCA via eigen-decomposition of the data matrix $X$ after the mean of the training set is removed. This results in a model with variations from the mean. We also transform the training and testing data into 'face space', i.e. the learned sub space of the eigen-faces.

```
# Center data
mean = np.mean(X_train, axis=0)
X_train -= mean
X_test -= mean

#Eigen-decomposition
U, S, V = np.linalg.svd(X_train, full_matrices=False)
components = V[:n_components]
eigenfaces = components.reshape((n_components, h, w))

#project into PCA subspace
X_transformed = np.dot(X_train, components.T)
print(X_transformed.shape)
X_test_transformed = np.dot(X_test, components.T)
print(X_test_transformed.shape)
```

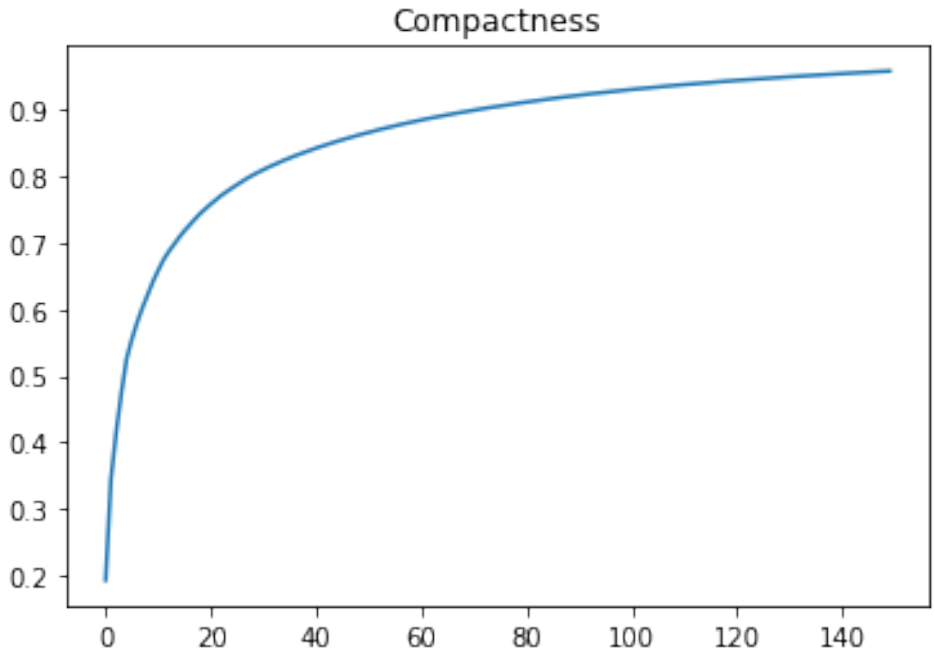
Finally, plot the resulting eigen-vectors of the face PCA model, AKA the eigenfaces

```
import matplotlib.pyplot as plt

# Qualitative evaluation of the predictions using matplotlib
def plot_gallery(images, titles, h, w, n_row=3, n_col=4):
    """Helper function to plot a gallery of portraits"""
    plt.figure(figsize=(1.8 * n_col, 2.4 * n_row))
    plt.subplots_adjust(bottom=0, left=.01, right=.99, top=.90, hspace=.35)
    for i in range(n_row * n_col):
    plt.subplot(n_row, n_col, i + 1)
    plt.imshow(images[i].reshape((h, w)), cmap=plt.cm.gray)
    plt.title(titles[i], size=12)
    plt.xticks(())
    plt.yticks(())

    eigenface_titles = ["eigenface %d" % i for i in range(eigenfaces.shape[0])]
    plot_gallery(eigenfaces, eigenface_titles, h, w)

    plt.show()
```



We should always evaluate the performance of the dimensionality reduction via a compactness plot

```
explained_variance = (S ** 2) / (n_samples - 1)
total_var = explained_variance.sum()
explained_variance_ratio = explained_variance / total_var
ratio_cumsum = np.cumsum(explained_variance_ratio)
print(ratio_cumsum.shape)
eigenvalueCount = np.arange(n_components)

plt.plot(eigenvalueCount, ratio_cumsum[:n_components])
plt.title('Compactness')
plt.show()
```




Use the PCA 'face space' as features and build a random forest classifier to classify the faces according to the labels. We then view its classification performance.

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

#build random forest
estimator = RandomForestClassifier(n_estimators=150, max_depth=15, max_features=150)
estimator.fit(X_transformed, y_train) #expects X as [n_samples, n_features]
```

```
predictions = estimator.predict(X_test_transformed)
correct = predictions==y_test
total_test = len(X_test_transformed)
#print("Gnd Truth:", y_test)
print("Total Testing", total_test)
print("Predictions", predictions)
print("Which Correct:",correct)
print("Total Correct:",np.sum(correct))
print("Accuracy:",np.sum(correct)/total_test)

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

This should show you the performance of the RF/PCA based face recognition. (1 Mark)

Your task for this part is to re-implement the above PCA algorithm of the Eigenfaces problem using TF or PyTorch functions. (3 Marks)

# 2 Part 2 - CNNs (6 Marks)

In this part, you will create a CNN based classifier that should hopefully out perform the above Eigenfaces algorithm from the previous part. Multiple layers will be strung together and tied to a fully connected (or dense) layers to result in a classified output. Using either TF/Keras/PyTorch/JAX, implement a CNN based classifier for the same LFW dataset from Part 1 with two convolution layers of 3x3 with 32 filters each and dense layers for classification. (1 Mark)

You can reuse the initial elements of your code in Part 1 that loads and creates the training and the testing sets. You may use the Adam optimiser and (sparse) categorical cross entropy loss. Note that convolution layers expect 4D tensors so that you will need normalise and resize your arrays, see for example

```
#normalise
X_train = X_train / 255.0
X_test = X_test / 255.0
X_train = X_train[:, :, :, np.newaxis]
X_test = X_test[:, :, :, np.newaxis]
print("X_train shape:", X_train.shape)
```

Finally, construct a Fast CIFAR10 dataset classification network using one of TF/Keras/PyTorch/JAX to meet the standards of the DAWNBench challenge as the following:

1. The model must achieve more than 93% accuracy and that is trainable in the fastest time possible (usually under 30 mins on a cluster).
2. Usage of pre-built models will generally not be allowed unless approved by the demonstrator.
3. The model must run inference or a single epoch of training on the Ranpur compute cluster (see appendix A for details) during the demonstration.

Using a ResNet-18 and mixed precision, it is possible to achieve 94% on a model trained for only approximately 360 seconds on a NVIDIA V100 GPU on the cluster. Can you achieve a time that is equivalent or faster? See appendix B for more resources on the DAWNBench challenge. (5 Marks)

# 3 Part 3 - Recognition (10 Marks)

For this part, you will need to develop your own solution to one of the recognition problems provided. There are a number of problems available, each having different levels of difficulty and will ultimately affect the total maximum mark attainable for this section. Partial marks can be awarded (with respect to the maximum) for having a go at crafting a solution to any one of the problems below and answering some of the questions.

Solve **one** of the following problems in the subsequent sections using TF/Keras/PyTorch, noting that the marks are allocated according to the level of difficulty. You must obtain reasonable results and be able to explain all results, network layers and code to the demonstrator in order to obtain the marks available. Usage of pre-built/pre-trained models will generally not be allowed unless approved by the demonstrator.

**[You must create a GitHub project in your own account with relevant commit logs in addition to your demonstration in order to receive ANY marks for this part of the lab. The demonstrator may request evidence that this is your own account by logging into the account during the demo. Note that it is okay to refer to internet sources, but only for learning purposes and it is NOT okay to copy code. Please cite the sources appropriately and the instructors will check your understanding thoroughly with questions they feel is appropriate. You must convince and justify your code to the satisfaction of your instructor.]**

Marks will be awarded according to the following breakdown:

1. Code functions as required and completes task. (1-6 Marks based on task selected)
2. GitHub project is valid and hosts code correctly (2 Marks)
3. Code is commented and structured correctly for use by others (1 Marks)
4. Commit messages are reasonable and meaningful (1 Marks)

## 3.1 Variational Autoencoder

Construct a Variational Autoencoder (VAE) of the magnetic resonance (MR) images of the brain via the Preprocessed OASIS dataset - **Medium Difficulty (Maximum 5 Marks out of 10)**

To obtain full marks, you might train the model and visualise the resulting manifold created by the VAE. For the visualisation, you may use sampling methods to create a 2D image of the manifold or use a dimensionality reduction technique such as UMAP.

## 3.2 UNet

UNet based magnetic resonance (MR) image segmentation of the brain via the Preprocessed OASIS dataset - **Medium Difficulty (Maximum 8 Marks out of 10)**

The segmentation accuracy of your model will need to be validated and achieve > 0.9 DSC for all labels. You may use categorical (one-hot) or without, but accuracy as mentioned must be obtained. You must also visualise some of the segmentation results to justify the DSC scores obtained. You must run inference at demonstration on a test set and show the model is working correctly.

**A note of warning, this requires knowledge of tuning and mixed precision implementations of networks and some patience as you'll be waiting a lot! You must explain each element of your training, the methods used to obtain your 'fast' model and the model itself to justify the training times obtained. You must use your own implementation of models if that model is already posted on the challenge site. You do not have to submit your model to the challenge. You will have to demonstrate the training in realtime!**

### 3.3 Generative Adversarial Networks

Realistic face generation using generative adversarial networks (GANs) of the CelebA dataset - **Very Hard Difficulty (Maximum 10 Marks out of 10)**

Faces must be suitably realistic and evidence of training (generated images, training loss plots etc.) of the models must be provided. The 'realism' of the faces will be judged by your instructor, so check with them if the results are appropriate for full or partial marks.

**A note of warning, GANs have very chaotic convergence and therefore difficult to train. Only attempt it if you feel confident with deep learning and if you do, you do so at your own risk. You might want to start out using the MNIST digit dataset first for GANs before attempting it with the CelebA dataset. Results obtained must look reasonably like faces for full marks. Marks will only be awarded for CelebA results for GANs.**

# 4 Appendix

# A Rangpur High Performance Computing Cluster

Information on connecting and accessing the Ranpur computing cluster can be found here. You can see an introduction to using such computing clusters and the SLURM queuing system in the HPC Summer of AI 2022 video.

# B DAWNBench Resources

There a number of resources required that you can use to accomplish the DAWNBench challenge task:

- You can find the Crash Course in Deep Learning Summer of AI 2022 video that walks you through a DAWNBench solution.
- See also Shakes' JAX vision library code that solves the challenge here.