Capstone Projects

1) Customer Segmentation
Grouping customers into sections based on their common characteristics is called Customer Segmentation. These clusters allow the companies to target the customers with the correct marketing message and tailor their offers for a specific group. This not only helps them boost their sales, but also helps them build customer relations and understand them in a better way.

In this project, our aim will be to perform customer segmentation on Online Retail Dataset (https://archive.ics.uci.edu/ml/datasets/Online+Retail#) to understand the customers. Given this dataset, our task is to:

a) Load the dataset and perform a descriptive analysis on it (Total number of entries, the column types, unique/non-null entries for each attribute, unique stock items, visualizing various attributes using bar charts/pie-charts and so on).
b) Perform data cleaning. Specifically, given the dataset, handle the entries that either have missing information or have attribute values that are not feasible such as negative quantity.
c) Perform data pre-processing for the required attribute fields.
d) Since this database has no additional attribute information for the customer, we will use RFM model (refer: https://clevertap.com/blog/rfm-analysis/) for segmentation. Modify the database to include RFM model attributes.
e) Now once you have your database ready, perform data clustering on this dataset by assuming a fixed number of clusters.
f) Find the optimal number of clusters that the customers can be divided into.

2) Music Genre Classification
Sound/Audio signals can be represented in the form of various parameters such as frequency, bandwidth, roll-off and so on. Using various python libraries, we can perform feature extraction for these audio signals. These features can then be processed and further used to perform classification.
In this project, we will use GTZAN dataset (https://www.kaggle.com/andradaolteanu/gtzan-dataset-music-genre-classification) which consists of 10 genre with 100 songs each, all having a length of 30 seconds. Given this dataset, our task is to:

a) Take two songs from each of the genre and visualize them and also find their spectrogram.
b) Create a dataset by extracting feature for each of the songs in GTZAN dataset. For our task, we will specifically use the following features: Mel-Frequency Cepstral Coefficients, Spectral Centroid, Zero Crossing Rate, Chroma Frequencies and Spectral Roll-off.
c) Given total 1000 examples, perform K-Means-Clustering on the dataset to cross verify that the optimal number of clusters are 10 (one for each genre).
d) Divide the dataset into two parts: 90% train and 10% test i.e. for each genre use 90% of the dataset as train and the remaining as test dataset.
e) Perform classification using any of the four classification algorithms and compare the accuracy obtained. Study the architecture of the model used and describe the reason for the model with best accuracy.

3) Face Detection and Recognition

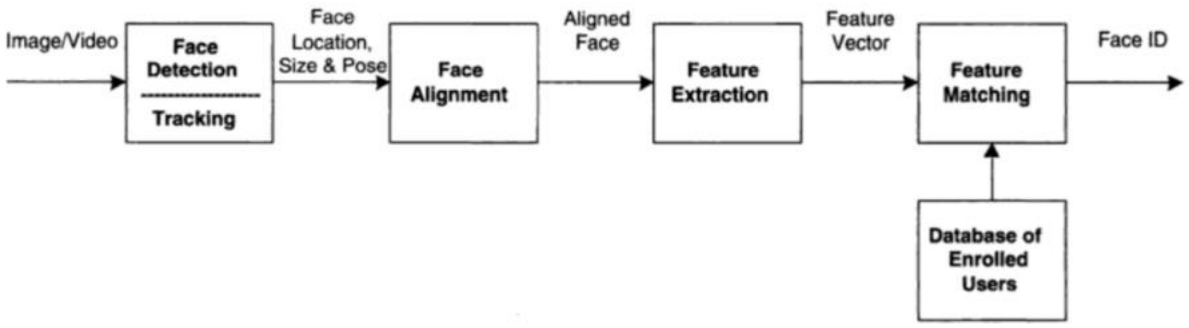
In this project we will tackle the task of face detection followed by face recognition. As shown in the image below, face recognition pipeline involves four main steps:

a) Finding the face in the given image: Implement a face detection algorithm for detection multiple faces in a given image. You can either use Deep Learning methods or machine learning methods such as Histogram of Oriented Gradients or Harr Cascade Classifiers for this.

b) Face Alignment: Once the coordinate of the faces are obtained, crop the face and make them translational invariant i.e. scale and rotate the image to ensure similar nature of all facial images.

c) Feature Extraction: Perform Feature extraction for the given faces. Particularly, use an encoder network to encode the image into a fixed dimensional vector (eg. 64/128). You can use a pre trained MLP or train your own MLP particularly for facial encoding using triplet loss function.

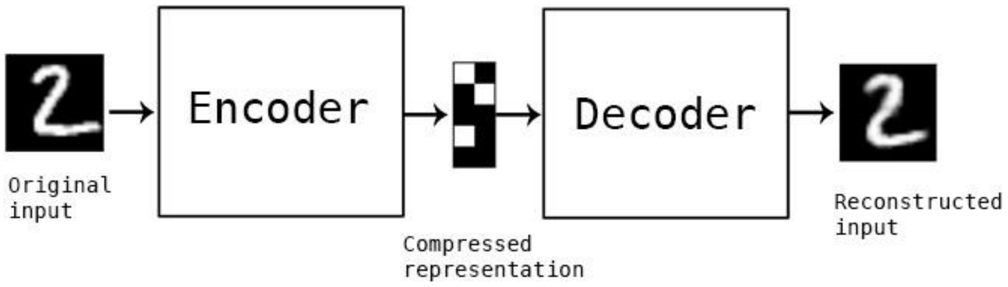
d) Prediction: Now once the model is trained, given any test image, find if it belongs to a specific person or not (use encodings for comparison).



4) AE: MNIST Dataset

The MNIST database (http://yann.lecun.com/exdb/mnist/) of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. The dataset consists of 28 x 28 pixel images comprising of gray scale images of handwritten digits between 0-9. The task of this project is to

a) Build an auto-encoder (AE) from scratch i.e. a MLP architecture (encoder) that takes the flattened image (28x28 = 784 dimension) as input and encodes it into a lower dimensional latent vector (can be anything less than 32 dimension. Eg: 32, 16 or even 4 or 2 dimensional encodings). Using this latent vector, implement another MLP architecture (decoder) that decodes this encoded vector to give back the encoded image.



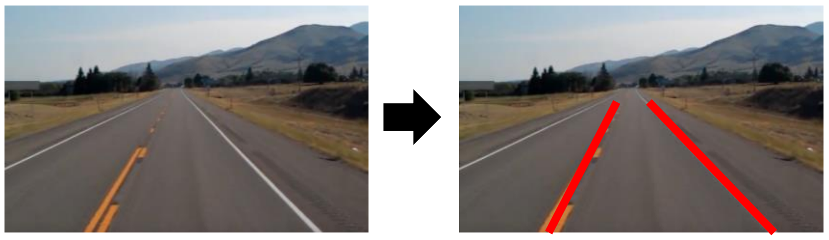
b) Use the encoding vectors to visualize the encoding. Specifically, if encoding vector is 2D, plot the encodings of 50 test examples from each class or in case of higher dimensional encoding, use PCA or TSNE plot to visualize the clusters formed due to each digits encoding.

c) Optional: You may additionally try to perform k-means on these encodings to further justify the information stored in each of these encodings.

5) Self-Driving Vehicle: Lane Detection

Implementation of a completely autonomous self-driving car such as a Tesla involves the use of many external sensors and a lot of dependence on Deep Neural Networks for tasks such as object detection for understanding the nearby environment, traffic lights and safety signs detection and so on. However, one of the most crucial task is to make the car drive in one particular lane. Whenever we drive, these lanes guide us where a road takes us and act as reference for where to steer. Thus, this fundamental knowledge is required by self-driving vehicles as well. In this project we aim to tackle the problem of lane detection given a set of images from a car dashboard or videos (e.g. dataset: https://xingangpan.github.io/projects/CULane.html, you can choose any such video or dataset for this purpose).

a) Implement an auto-encoder model that takes the image as input and outputs the images with the lanes marked. You may instead use python along with OpenCV to implement this as well. Your final goal is to detect the two adjacent lanes (one on left and one on right) given an image. Sample input and output:



b) Once the lanes have been detected, use these lanes as guidelines to decide the steering angle (Steering angle will depend on the center of the captured image and the center of the lanes detected).