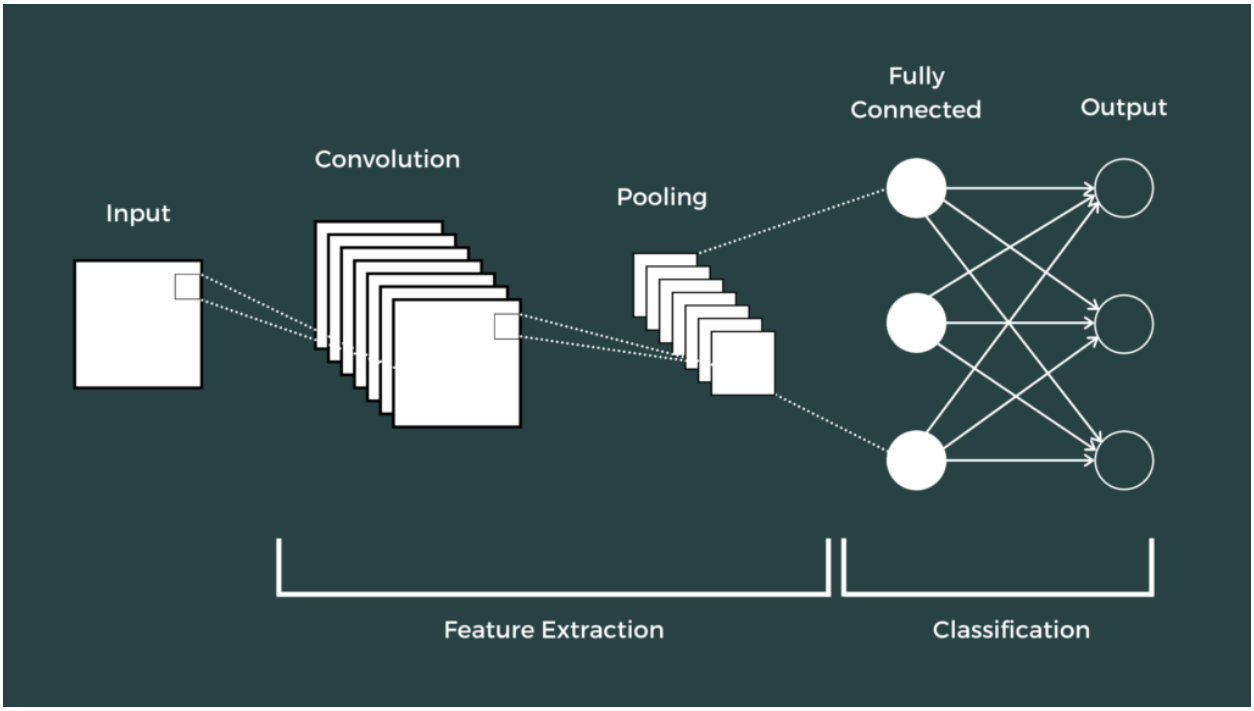
Timeline:

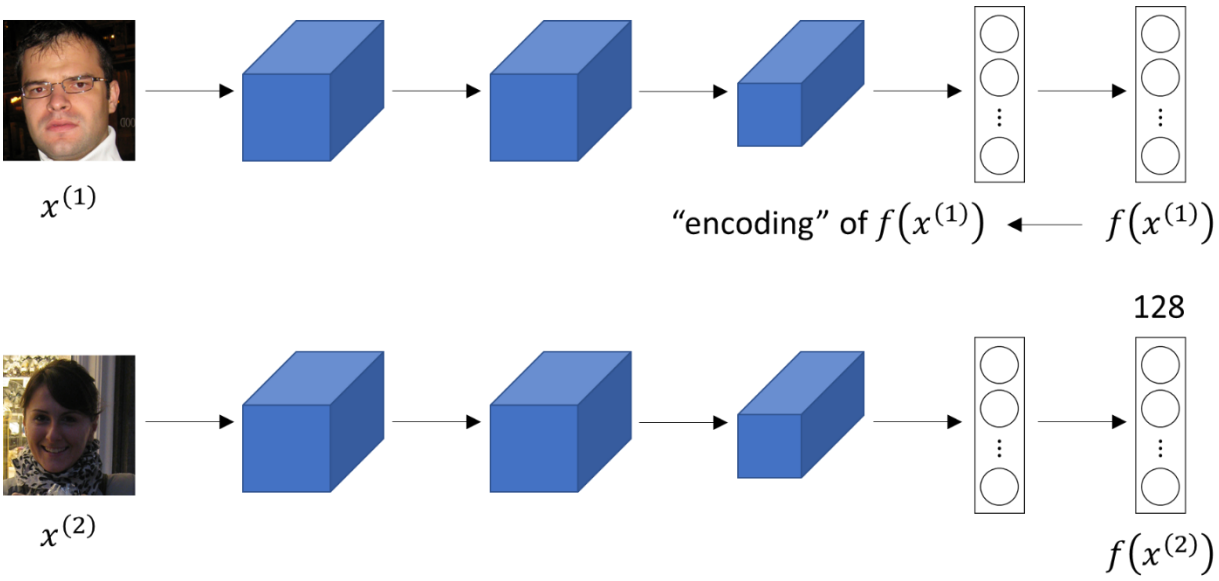
First, we found that before starting the project we have to learn about neural networks and CNN.

So, we started by a course for neural networks for Andrew ng, to understand deeply how the face recognition models work as the face recognition models like Siamese uses neural networks mainly to produce a high accuracy and efficient output.

Then, we have taken a CNN course by Andrew ng. CNN are a type of neural network that have shown to be particularly effective in face recognition tasks.



After that we used Siamese neural networks which is type of neural network architecture that is used for tasks such as face recognition, signature verification, and similarity detection.
It consists of two identical neural networks that share the same set of weights and are trained to learn a similarity metric between pairs of inputs.



Then, we started building the application and used a pre-trained model which is `face-recognition` that have high accuracy and easy to use.

FACE RECOGNITION PROJECT TIMELINE

Face Detection:

Face detection is a technology that allows computer systems to identify and locate human faces within an image or video. The process involves analysing the pixels of an image or video frame to detect and locate areas that contain human faces.

There are various techniques and algorithms used for face detection, including:

1. Viola-Jones Algorithm (Which we used in our model): This algorithm uses a Haar cascade classifier to detect faces in an image or video. The algorithm is a popular technique in computer vision, works by scanning the image with a sliding window and analysing the pixel values of each window to determine if it contains a face.

2. Deep Learning-Based Algorithms: These algorithms use neural networks to detect faces in an image or video. They work by training the neural network on a large dataset of images with labelled faces, which allows the network to learn how to identify facial features and patterns.

3. Skin Colour-Based Detection: This technique involves identifying areas of an image or video frame that contain skin-coloured pixels and then analysing the shape and structure of those areas to detect and locate faces.

The following steps describe how to perform face detection using Cascade Classifier:

1. Load the pre-trained classifier: A pre-trained classifier for face detection is provided with OpenCV. This classifier is stored as an XML file and can be loaded using the CascadeClassifier class.

   face cascade = cv2.CascadeClassifier('haarcascade_frontalface_default.xml')

2. Load the image: The image to be processed is loaded using the imread function.

   img = cv2.imread('image.jpg')

3. Convert the image to grayscale: The CascadeClassifier algorithm requires the input image to be in grayscale format. The cvtColor function is used to convert the image to grayscale.

   gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

4. Detect faces: The detectMultiScale method of the CascadeClassifier class is used to detect faces in the grayscale image. This method returns a list of rectangles that contain the detected faces.

   faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5)

- scaleFactor: will decrease the detection time but may miss smaller faces if increased.
- minNeighbors: will reduce false positives but may miss some faces.

5. Draw rectangles around detected faces: The rectangle function of the OpenCV library is used to draw a rectangle around each detected face.

   ```
   for (x, y, w, h) in faces:
       cv2.rectangle(img, (x, y), (x+w, y+h), (0, 255, 0), 2)
   ```

6. Display the image: The imshow function of the OpenCV library is used to display the image with the detected faces.

   ```
   cv2.imshow('img', img)
   cv2.waitKey()
   ```

The waitKey function is used to wait for a key press before closing the window.

Conclusion:

face detection using CascadeClassifier is a simple and effective technique for detecting faces in an image. By adjusting the parameters, the sensitivity and accuracy of the algorithm can be optimized for different use cases.

Face Recognition

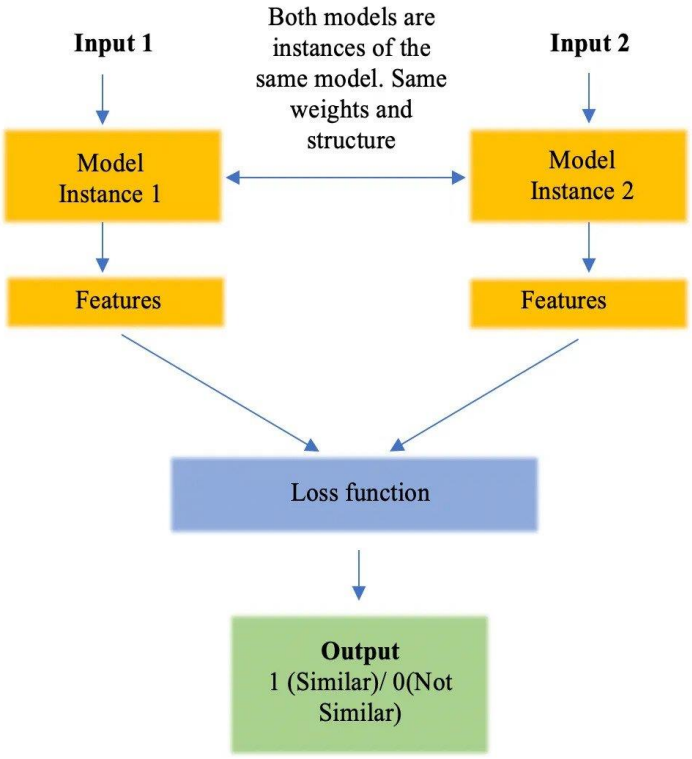
In this project we trained a Siamese Neural Network on this data. The project uses in the deployed application a pre-trained model called `face-recognition`.

Siamese model:

Siamese is a deep learning model that is used for image recognition and image segmentation.

How it works?

We feed a pair of input images to Siamese networks. Each network computes the features of one input, then compute the similarity of features using their



difference or the dot product. If they are similar class input pairs, output 1 and for different classes input pairs, output 0.

Pros:

1. Fewer data samples required
2. Can work with highly imbalanced data

Cons:

1. A large amount of training data because of so many pairs of classes.
2. Not generalizable. A model trained for one task cannot be used for another task.
3. Sensitive to some variations in the input.

Face-Recognition model:

It is the world's simplest face recognition library which built using dlib. The model has accuracy of 99.38% on the lab Labeled Faces in the Wild benchmark.

Steps:

- Find faces in pictures.
- Find and manipulate facial features in pictures.
- Identify faces in pictures.

Installation:

Check if python is installed. You can install it from here if you don't have python.

>>> python --version

Check if pip is installed. You can install it from here if it wasn't.

>>> pip --version

To install the required libraries, use the following commands in the command line:

1. Face-Recognition library:
   >>> pip install face-recognition
2. OpenCV library:
   >>> pip install opencv-python
3. NumPy library:
   >>> pip install numpy
4. Streamlit library:
   >>> pip install streamlit