Folder 1: Age and Gender Detection

This Python script uses OpenCV to perform age and gender detection on faces in real-time video. It imports the necessary libraries, defines a function to detect faces and draw bounding boxes around them, and then loads pre-trained models for face detection, age detection, and gender detection. It then opens a video capture device and processes frames from the video, using the face detection function to find faces and the age and gender models to make predictions about each face. Finally, the script displays the video with the predicted ages and genders overlaid on the faces.

Folder 2: Emotion Based Filter

First, we focused on detecting a face in the frame and applying a dog filter to the detected face. We were able to do this by using a custom-made function called ‘apply mask’ and already existing code for face detection.

Then, we identified motion in the input face via least squares emotion classifier using vectorized images and one-hot encoded labels. The training and testing accuracy were 0.4749 and 0.4528 respectively.

After featurizing and regularizing our inputs to remove overfitting, we got the training and testing accuracy as 0.6511 and 0.6221 respectively. To increase the accuracy, we used a least square classifier and deep learning library called PyTorch.

We applied PyTorch abstractions on one-hot encoded labels, and then set up neural network layers to optimize hyperparameters and to run stochastic gradient descent. We then run a prediction using the model.

After training it for 20epochs with a learning rate of 0.01 and momentum of 0.9, our model attains a training accuracy of 87.9% and validation accuracy of 75.5%. Hence, running this code along with real time face detection code previously used, we were able to build an emotion-based dog filter.

All python files in detail:

**step\_2\_face\_detec.py** - This code detects faces in an image using the Haar Cascade classifier and outputs an image with the detected faces enclosed in a green rectangle.

**step\_3\_camera\_face\_detect.py** - This code detects faces in real-time video using the Haar Cascade classifier and displays the video with the detected faces enclosed in a green rectangle.

**step\_4\_dog\_mask.py** - This code is an implementation of a real-time video filter that adds a mask to faces detected in a video stream using OpenCV.

**step\_4\_dog\_mask\_simple.py** - This code applies a mask image to a face image, and saves the result as a new image.

**step\_5\_ls\_simple.py** - This script is performing linear regression on the FER2013 dataset using Ordinary Least Squares (OLS) method. It loads the training and testing data from the "fer2013\_train.npz" and "fer2013\_test.npz" files, respectively. It then converts the training and testing labels to one-hot encoded vectors, selects the first 100 dimensions of the input data, and trains the OLS model using these inputs and one-hot encoded labels. Finally, it evaluates the accuracy of the trained model on both the training and testing data and prints the results.

**step\_6\_ls\_simple.py** - The code implements ridge regression on the FER2013 dataset, using the first 1000 dimensions of the input features. It randomly initializes a weight matrix **W** with dimensions corresponding to the input data, and then projects the input data into the new space defined by **W** using matrix multiplication. It then applies ridge regression to the projected training data, using a regularization parameter of 1e10, and evaluates the resulting model on both the training and test sets using the **evaluate** function which calculates the accuracy of the predicted labels relative to the true labels.The output displays the train and test accuracy of the model using ridge regression.

**step\_7\_fer.py** - This code defines a neural network model (called Net) with convolutional layers and fully connected layers for facial emotion recognition. It also defines a dataset class called Fer2013Dataset for loading and processing the FER2013 dataset, which consists of grayscale images of faces with labels indicating the emotion (happy, sad, angry). The code also includes functions for evaluating the model's accuracy on the training and test sets and for creating a predictor that maps images of faces to emotion labels using the trained model. In the main function, the model is loaded from a pre-trained file, and its accuracy is evaluated on the training and test sets.

**step\_7\_fer\_simple.py** - This code defines a convolutional neural network to perform facial emotion recognition on the FER2013 dataset. It loads the data using a custom PyTorch Dataset class, splits it into a training set and a test set, and trains the neural network using stochastic gradient descent.

The main difference between step\_7\_fer.py and step\_7\_fer\_simple.py files is that the first one includes the definition of a neural network model (**Net** class) and a function that loads a pre-trained model and uses it to make predictions on new data (**get\_image\_to\_emotion\_predictor**), whereas the second one only includes the definition of the **Fer2013Dataset** class. The **Fer2013Dataset** class is common to both code snippets and represents a PyTorch dataset that can be used to load and preprocess the FER (Facial Emotion Recognition) dataset into PyTorch tensors.

**step\_8\_dog\_emotion\_mask.py** - This code is a Python script that applies an emotion-based filter to a live video stream from a webcam. It uses OpenCV (cv2) for image processing and the FER model trained in the previous step to predict the emotion of a person's face in real-time. It then applies a mask to the person's face based on their predicted emotion.

The masks used are images of dogs, and there are three different masks to choose from. The masks are resized to fit the face and are centered over the person's face using the coordinates of the detected face. The script uses the Haar Cascade classifier to detect faces in the video stream.

The **apply\_mask** function takes two NumPy arrays as input: **face** and **mask**. It resizes the mask to fit the face and then adds the mask to the face. It centers the mask over the face and applies it only to non-white pixels of the resized mask.

The **main** function sets up the video capture from the webcam, loads the dog masks, loads the FER model, and initializes the Haar Cascade classifier. It then loops over each frame of the video stream, detects faces in the frame using the Haar Cascade classifier, predicts the emotion of each detected face using the FER model, and applies the corresponding dog mask to each face. Finally, it displays the resulting video stream and exits when the user presses 'q'.

Overall, the script uses computer vision and machine learning techniques to create a fun and interactive application that applies an emotion-based filter to a live video stream.

Folder 3: Real Time Face Recognition

This project is a real-time face recognition system implemented in Python using the OpenCV and face\_recognition libraries. It consists of two Python files: "function.py" and "main.py".

The "function.py" file defines a class called "SimpleFacerec" that contains methods for loading image encodings of known faces and detecting known faces in a frame. The class initializes by creating empty lists to store the known face encodings and names. It also defines the frame\_resizing attribute, which is used to resize the frame for faster processing.

The load\_encoding\_images() method loads image encodings of known faces from a specified path using the glob module. It then iterates over the images and stores their encodings and names in the corresponding lists.

The detect\_known\_faces() method takes a frame as input and resizes it. It then uses the face\_recognition library to detect face locations and encodings in the resized frame. The method compares the encodings of the detected faces with the known face encodings and determines the best match. It returns the locations and names of the detected faces.

The "main.py" file imports the SimpleFacerec class from "function.py" and creates an instance of it. It loads image encodings of known faces from the "images/" folder using the load\_encoding\_images() method. It then starts the video capture from the default camera using OpenCV's VideoCapture() function.

In the main loop, the program reads frames from the camera and detects known faces using the detect\_known\_faces() method. It then draws a rectangle around the detected faces and displays their names using OpenCV's putText() function. The program exits when the 'q' key is pressed.

Overall, this project demonstrates how to implement a real-time face recognition system using Python and the OpenCV and face\_recognition libraries.

Thank You!