Intro to ML - CS 419

Topic: Attendance Marking using Face Recognition

Spring 2023

Members: Aryan, Abhishek, Harsheet, Tanmay

Project Report

1 Introduction

In today's fast-paced world, managing attendance has become an essential aspect for many organizations, schools, and universities. The traditional method of marking attendance manually can be time-consuming and prone to errors. In recent years, face recognition technology has emerged as a promising solution for automating attendance marking, and it has the potential to revolutionize the way we manage attendance. In this project report, we present our work on attendance marking using face recognition. We have used the concept of Siamese networks and deep learning to build a robust attendance system that can accurately and quickly mark attendance.

2 What are Siamese Networks?

The Siamese network approach for face recognition involves training a deep neural network on pairs of images to learn a similarity metric. The network architecture consists of two identical subnetworks, also called twins or branches, that share the same weights and are trained simultaneously. Each subnetwork takes an input image and passes it through several convolutional layers followed by pooling layers and fully connected layers to extract high-level features. The output of each subnetwork is a fixed-length feature vector that represents the input image.

3 What is YOLOv5?

YOLOv5 is a deep learning algorithm for object detection that can be used to detect objects in images, videos, and live streams. It is developed and maintained by Ultralytics and released under the GPL-3.0 license. The YOLOv5 algorithm is an improvement over the previous versions of the YOLO algorithm, and it achieves state-of-the-art accuracy while being faster and more efficient. The YOLOv5 algorithm is available in PyTorch, TorchCode, ONNX, OpenVINO, TensorRT, CoreML, TensorFlow SavedModel, TensorFlow GraphDef, TensorFlow Lite, TensorFlow Edge TPU, and PaddlePaddle formats. It can be used for a variety of object detection tasks, such as detecting people, cars, animals, and other objects in real-world scenes. The YOLOv5 algorithm uses a single neural network to perform the task of object detection, and it is trained on large datasets such as COCO, Pascal VOC, and Open Images. The YOLOv5

algorithm is highly configurable and can be used with a wide range of parameters to achieve different levels of accuracy and speed.

4 Data

The Olivetti dataset is a well-known face image dataset widely used in the field of computer vision and machine learning. It was created by the Olivetti Research Laboratory in Cambridge, UK. The dataset consists of grayscale images of faces taken from a set of 40 distinct subjects. For each subject, there are 10 different images captured under different lighting conditions, facial expressions, and facial details such as glasses or facial hair.

Each image in the Olivetti dataset is of size 64x64 pixels, resulting in a total of 400 images (40 subjects * 10 images per subject). The images are preprocessed and aligned so that the faces are approximately centered and have a consistent scale.

5 Image Uploading

We are using the Flask framework to create a web application that allows a user to upload an image and then detect faces in the image using the YOLOv5 object detection model. It then crops the detected faces and compares them with the faces in the database.

The code starts by importing the necessary libraries including Flask, Pandas, shutil, cv, and os. It then defines the upload folder path and allowed file types, which are txt, pdf, png, jpg, jpeg, and gif. It also sets the template and static folders for Flask. The code defines several routes. The index route simply renders the index.html template. The uploadFile route is called when a user uploads a file. It first empties the "uploads" folder and deletes the "exp" folder, if they exist. It then saves the uploaded file to the "uploads" folder and stores its path in a Flask session. It then renders the index2.html template.

The displayImage route imports the detect.py code and calls its run() function to detect faces in the uploaded image. It then moves the cropped images to the "static/cropped" folder and imports the compare.py code to compare the detected faces with the faces in the database.

Finally, the code defines a takepath() function in the compare.py code, which is called to compare the detected faces with the faces in the database. The path to the detected face and the path to the face in the database are passed to this function, and it returns the similarity score between the two faces.

6 Detection

We start by importing necessary modules such as argparse, os, and torch, and some utility functions. The code defines a run function that takes several arguments for configuration such as the model weights, input source (an image, video, directory, webcam, etc.), dataset information, inference size, confidence threshold, and more. This function is decorated with smart_inference_mode, which is a decorator that intelligently selects the appropriate inference mode based on the model and input source.

The source parameter is used to define the input source. It can be a webcam, a local file, a directory containing images or videos, or a URL for streaming videos.

The code creates a DetectMultiBackend object using the weights specified, which is the YOLOv5 model, loads the input source using LoadImages, LoadScreenshots, or LoadStreams depending on the input source, and runs the detection inference. The detected objects are then filtered using non-maximum suppression (NMS) to remove overlapping bounding boxes and output is saved to the specified output directory.

7 Face Recognition

This code creates a Siamese neural network using Keras to perform face recognition. It is trained on the Olivetti Faces dataset, which contains grayscale images of faces. The code imports several libraries, including TensorFlow, Keras, NumPy, and Scikit-learn.

The Olivetti Faces dataset is loaded from Scikit-learn, which consists of 400 images of faces (10 images per person) in grayscale with a size of 64x64 pixels. The data and target variables are assigned to the variables images_dataset and labels_dataset, respectively

The create_model() function defines the architecture of the neural network. The model uses a series of convolutional layers followed by max-pooling and dropout layers to extract features from the input images. The output of the convolutional layers is then passed through a global average pooling layer and a dense layer to produce a feature vector. The feature vectors for two input images are then compared using the Euclidean distance between them. The model is trained using binary cross-entropy loss and the Adam optimizer.

The generate_train_image_pairs() function creates pairs of images for training the Siamese network. For each image in the dataset, the function randomly selects another image from the same person (positive pair) and another image from a different person (negative pair). The function returns an array of image pairs and their corresponding labels.

The generate_test_image_pairs() function creates pairs of images for testing the Siamese network. For each image in the dataset, the function selects another image from the same person (positive pair) and images from all other people (negative pairs). The function returns an array of image pairs and their corresponding labels.

Finally, the code trains the Siamese network on the training pairs, saves the weights of the trained model, and uses the trained model to predict the similarity between the images in the test pairs.

8 Comparison

This code defines a function called takepath() that takes two image paths as input, loads the images, pre-processes them, and feeds them into a pre-trained deep learning model to generate feature vectors. It then compares the feature vectors of the two images to determine if they are similar or not, based on a given threshold. If the feature vectors are within the threshold, the function returns 1 indicating similarity, otherwise, it returns 0 indicating dissimilarity.

The deep learning model used in this code is defined in the create_model() function. It is a convolutional neural network (CNN) with multiple convolutional and pooling layers followed by two dense layers at the end. The model takes 64x64 grayscale images as input and generates a 128-dimensional feature vector as output. The weights of the pre-trained model are loaded from a file called model_weights.h5.

Overall, this code can be used to compare two images and determine if they are similar or not based on their feature vectors generated by a pre-trained deep learning model.

9 Conclusion

The basic objective of the project was to build an attendance portal for easy attendance-marking in educational institutes and offices.

We trained the model on the **"olivetti"** dataset and got the weights so that we can test the model on different images of our choice.

We had to keep the threshold quite low since when 2 images of same person are provided, the prediction arrays do not differ by a large value and vice-versa for images of 2 different persons.

We tried to test our model on blur images but came out with wrong results. Thus, we concluded that the user should give clear images otherwise wrong outputs would be displayed.

To conclude, we hope to improve our attendance system so that it can be used in various fields with high precision and accurate results.

10 References

- **https://medium.com/wicds/face-recognition-using-siamese-networks-84d6f2e54ea4**
- **https://github.com/ultralytics/yolov5**

- **Chatgpt for debugging the code at various stages**