Doc32 Internship Report

Mentor: Sourav Sarkar
Submitted by: Anwesha Samaddar & Shivangi Srivastava

Tasks

- Head pose estimation for static images
- Realtime Head pose estimation
- Demo for head pose detection
- Smile detection using deep learning to detect smiles from a live video feed
- Multi Scale Template matching
- Feature Detection
- Extraction of Instagram post data using Instaloader
- Using AWS Server
- Sentiment analysis of comments from instagram posts
- Learned GIT fundamentals for performing basic operations

Head pose estimation

- Detection and extraction of facial landmarks from an image using **Dlib, OpenCV, and Python**: The **pre-trained facial landmark detector** inside the dlib library is used to estimate the location of **68 (x, y)-coordinates** that map to facial structures on the face.

- The dlib package works upto **python 3.6,** not later versions so we needed to create a virtual environment of python 3.6 and then install dlib.

- The **"shape_predictor_68_face_landmarks.py"** file needs to be downloaded and kept in the same path.

- 2D coordinates of a few points : the 2D (x,y) locations of a few points of a face like the **corners of the eyes, the tip of the nose, corners of the mouth were chosen out of the 68 coordinates detected by Dlib's facial landmark detector.**

- 3D locations of the same points : the 3D location of the 2D feature points are also required. The 3D points are in some arbitrary **reference frame / coordinate system** and are called the **World Coordinates.**

Head pose estimation

- Defined **camera internals** and **projected a 3D point (0, 0, 1000.0)** onto the image plane to draw a **line sticking out of the nose tip**. We can approximate the optical center by the center of the image, approximate the focal length by the width of the image in pixels and assume that **radial distortion** does not exist.
- **Calculated the angle** by which the face is **tilted from** the desired posture. From the angle obtained we get to know the direction in which one **must rotate or tilt one's head.**
- Displayed the image with the **message** and the **six facial landmarks.**

Realtime Head pose estimation

- We used the the **VideoCapture function of Opencv** to capture live video feed from the webcam. Image is captured **frame-by-frame** and our operations on the frame begin by **grayscaling** the image and then by detection of faces in the webcam's image.

- The facial landmarks (coordinates) are predicted using **Dlib's facial landmark detector** as above and transformed into numpy array. The required six facial coordinates (2D) are chosen and the 3D model points are selected.

- Similarly as above mentioned we defined camera internals and projected a 3D point onto the image plane to **draw a line sticking** out of the nose tip.

- Calculated the **tilt angle** and displayed the **relevant message** for getting the correct head pose.

- Press 'Q' to exit and turning off the webcam

Demo for head pose detection

Objective:

- To create a *local webpage* for demonstrating real time head pose estimation

Tools Used:

- We used opencv.js and tensorflow.js' posenet to create this Cycle.js demo local webpage
- *Opencv.js* was used because it has the javascript implementation of opencv functions like *solvePnP and projectPoints* which are required for head pose estimation
- *Tensorflow.js' posenet* is used as it ***gives us the coordinates of keypoints like left eye, right eye and nose*** in the image which are required to get the *rotation and translation vectors*

Results:

- When the person was looking towards the camera, the webpage displayed the message: *"Looking towards the Camera"* and when the person was looking away from the camera, the webpage displayed the message: *"You are looking away from the Camera"*
- Thus we can say that *the demo ran successfully*

Smile detection using deep learning to detect smiles from a live video feed

- Used a **pre-trained keras model** for detecting if a person is **‘smiling’ or ‘not smiling’** by taking **live video feed** through webcam.

- First we **grab the current frame** and then **resize and grayscale** our frame. Then we **detect faces** by first making a clone of the input frame and then using the **CascadeClassifier function** of Opencv. **Haar Cascade** is a machine learning **object detection algorithm** used to identify objects in an image or video. We need to include the entire path of the **'haarcascade_frontalface_default.xml'** file for face detection.

- Extracted the **Region of Interest (ROI) of the face** from the grayscaled image and resized it to a fixed 28x28 pixels, and then prepared the ROI for classification via the **pre-trained CNN model.**

- Determined the **probabilities of both 'smiling' and 'not smiling'** and then set the label accordingly

- Displayed the **label and bounding box** on the output frame

Feature Detection

Use and Algorithm:

- Instead of matching two images based on there pixel values this methods tries to find the ***common features and keypoints*** that are present in both the images and then tries to match the two images
- The tools we have used mostly depend on ***image gradients***
- We compute the ***cosine distance*** between the feature vector of our search image and feature vectors database obtained from our image database, and then just output Top N results

Tools Used:

- *KAZE* : It is shipped in the base OpenCV library, which simplifies installation
- ORB : builds on the well-known FAST keypoint detector and the BRIEF descriptor

Results:

- The result obtained via this method was ***a lot more accurate*** than our previous approaches i.e. normal template matching and multi-scale template matching
- For every 10 queries made, the results were accurate for almost 6-7 of them, hence the ***accuracy*** can be said to be about ***65%*** for this method.

Multi Scale Template matching

Use:

- Useful when the dimensions of the template doesn’t match the dimensions of the region in the image where the matching needs to be done

Algorithm:

- It involves looping over the image at multiple scales and applying template matching using the *cv2.matchTemplate* function of the openCV library
- The resizing of the image is done using the ***imutils.resize()*** method and both the template and the image are converted to grayscale
- Before matching, *Canny function* of the openCV library is applied on both the template and the resized image in order to increase the efficiency
- After looping over all scales, we take the region with the *largest correlation coefficient* and use that as our “matched” region

Result:

- In our case, this method ***didn’t show significant accuracy*** but it is better than implementing just normal template matching

Extraction of Instagram post data using Instaloader

- **Instaloader** exposes its internally used methods and structures as a Python module, making it a powerful and intuitive **Python API for Instagram,** allowing to further customize obtaining **media and metadata.**

- By first getting an **instance** of **Instaloader,** we got **post shortcodes** for all the recent posts (posts in the last **365 days**), which we used to make post instances for each post through which we extracted the post metadata like **media id, owner id, date and time of post, likes, post captions,number of comments, comments, user ids of commenters, usernames of commenters and date and time of comments**.

- Also, we got a **profile engagement summary** containing the **number of followers, number of recent posts, engagement, recent post frequency**.

Using the AWS Server

Use:

- Running the extraction process *locally on our laptops was not feasible always* hence we used AWS to utilise its server for our computations and for running our code

Tools Used:

- ***AWS Lambda*** : used to write functions known as lambda functions in python as well as node.js format and this function can be *invoked* by several types of requests. In our case we *scheduled* our function to run *every 2 days* to be up-to date with the data
- ***AWS Lambda Layer*** : *Instaloader* is *not a native library* and can't be used directly in lambda functions. Hence we *pack this library in a lambda layer* and then use this layer to access instaloader inside our lambda function
- ***AWS DynamoDB*** : This is a *NoSQL* Database provided by AWS. We used the *single table design pattern* in order to reduce the time taken in joins for making queries. This gave us an edge over using a SQL database
- ***Serverless*** : used to *deploy* our project from our local system to AWS. It also creates the required resources like the layer and DynamoDB table and hence *automates the process*

Using the AWS Server

Results:

- The instaloader project was ***successfully deployed*** to AWS through the help of serverless
- The ***posts and comments of a user*** can be easily obtained by the property of ***partition key*** and ***sort key*** of DynamoDB tables. Hence the database can be queried easily and data analysis can be done easily, for ex. we ran a sentiment analysis of the comments of the users

Sentiment analysis of comments for instagram posts

- Downloaded 'Yelp reviews polarity' dataset consisting of **'positive' and 'negative'** comments. Training of a **Recurrent Neural Networks** model was done using the 'Yelp' dataset. Performed **pre-processing on the comments** for removal of special **characters and extra spaces, removed hyperlinks and expanded the shortened words (performed contraction mapping)**.

- Created a **word2id** dictionary **(label encoded the words)** for training into the RNN model. Used **bidirectional LSTM and added a dropout layer** for better model training and more accuracy and then used **Attention mechanism** for predicting the polarity of a comment.

- The RNN model gives a **validation accuracy of 93%** which is **higher** in comparison to the other machine learning models like **Random Forests Classifier or Multinomial Naïve Bayes.**

Sentiment analysis of comments for instagram posts

- **Using the RNN model** the polarity of each comment of each post was predicted and then an average of the ‘negative’ and ‘positive’ was done to get the **overall polarity of the comments of a post.** For Hindi and English mixed comments we translated fragments of the **Hindi parts into English using Google Translate** (this translation was done as no training datasets with **multilingual comments** were found) .
- This would give us an idea about the overall sentiment of a post (positive, negative or neutral).

THANK YOU!