Kyungdong University (KDU)
Bachelor of Smart Computing (BSC)
Research Methodology
Final Project

Student ID:	**1924106**
Student Name:	**Wafula Celine Nerima**
Student Signature:	**NCW**
Submission Deadline:	**2022.06.13 (11.59 PM)**

Full Marks: 100

Write a complete research article based on your corresponding projects that is assigned in the Digital Forensics and Investigations course.

Please follow the below instructions:

1. Individual report needs to submit.
2. Maximum page limit is 10 and minimum page limit is 4.
3. Please follow the writing style of the IEEE Transactions Journals. For your convenience, I already attached it with the email.
4. Please submit the document file using following file naming order:
   Student ID_ Student Name_ Project Name.
5. While sending the report in email please use subject as follow:
   Research Methodology_ Group Number_ Project Name.

Facial Landmarks Detection Using Mediapipe Library

Celine N. Wafula, Christina J. Mjema, Rodrigue Ntagashobotse, George Garang, Matthew A. Oguru, Jonathan M. Kalibala, Bachelor of Smart Computing, Kyungdong University Global Campus, Gosung, 24764, Korea

***Abstract*—Facial landmarks detection has come to be applied in various fields, ranging from the generation of facial filters in social media applications like snap chat to security and identity verification i.e. iris detection. In this project, a facial landmarks detection model is implemented using Mediapipe solution packages. The facial landmarks solution offers multiple implementations including image-based and real-time detection with allowance for a defined number of faces per instance. The model successfully detects facial landmarks even on faces positioned at an angle.**

***Index Terms*—Computer Vision, Facial Landmarks, Face Mesh, Mediapipe, Python**

I. Introduction

This project is an implementation of the combined Mediapipe modules in order to build an application in any python IDE (Integrated Development Environment).

Previous models have worked with face detection using Mediapipe library only but it became difficult to detect the landmarks points since they were not that clear when other elements of the face were being visualized i.e. the major facial key points in different angles. To cope up with this problem, an alternative method was suggested; which is to detect all the landmarks points, which are 468. [2] There was also a problem of estimating positions of the 3D mesh vertices with a neural network, and treating each vertex as an independent landmark. [1] The face mesh topology makes sure the points are arranged in fixed quads. The points were manually selected in accordance with the supposed applications, such as expressive AR effects, virtual accessory and apparel try-on and makeup. The areas that were expected to have higher variability and higher importance in human perception were allocated with higher point density. [1] In Figure 1, the face mesh prediction of how all the facial landmarks should be represented are shown including faces positioned even at an angle.

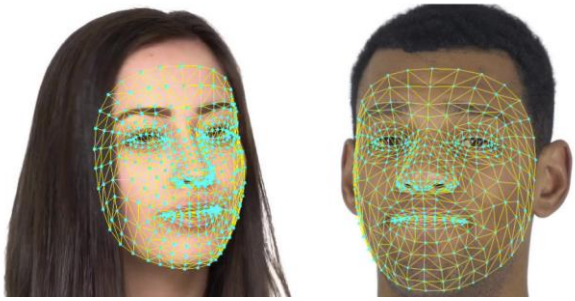

Figure 1: Face mesh prediction examples[1]

II. ML Pipeline

The model employs machine learning (ML) to infer the 3D facial surface, requiring only a single camera input without the need for a dedicated depth sensor. It makes use of the lightweight model architectures together with GPU acceleration throughout the pipeline, the solution is intended to deliver real-time performance critical for live experiences. [3] In addition, the solution is bundled with the Face Transform module that bridges the gap between the face landmark estimation and useful real-time augmented reality (AR) applications. It establishes a metric 3D space and uses the face landmark screen positions to estimate a face transform within that space. The face transform data consists of common 3D primitives, including a face pose transformation matrix and a triangular face mesh. [3]

This ML pipeline consists of two real-time deep neural network models that work together: A detector that operates on the full image and computes face locations and a 3D face landmark model that operates on those locations and predicts the approximate 3D surface via regression. The face should accurately be cropped drastically to reduce the need for common data augmentations like affine transformations consisting of rotations, translation and scale changes. This allowed the network to dedicate most of its capacity towards coordinate prediction accuracy. In addition, in this pipeline the crops can also be generated based on the face landmarks identified in the previous frame, and only when the landmark model can no longer identify a face presence in the face detector invoked to re-localize the face. [3]

III. Face Detection Model

The face detector used was the BlazeFace model used in MediaPipe Face Detection. [3] The BlazeFace model architecture was built around four important design considerations that will be slightly discussed in this paper below: [4]

Enlarging the receptive field sizes. This is where the essence of a double BlazeBlock that was used as the bottleneck of choice for the higher abstraction level layers of BlazeFace were formed. [4]

Feature extractor. In this specific model, the focus was put on the feature extractor for the front-facing camera model. [4]

Anchor scheme. Here, SSD-like object detection models rely on priors or anchors which were pre-defined bounding boxes that had a fixed size. [4]

Post-processing. For this model, the suppression algorithm was replaced with a blending strategy in order to avoid fluctuation between different anchors including the exhibit of temporal jitter (human perceptible noise). [4]

IV. Face Landmark Model

For 3D face landmarks, transfer learning was employed and the network was trained with several objectives: to simultaneously predict 3D landmark coordinates on synthetic rendered data and 2D semantic contours (see Figure 2) on annotated real-world data. The resulting network provided reasonable 3D landmark predictions not just on synthetic but also on real-world data. [3]

Figure 2: The 2D semantic contours used during the initial bootstrap process[1]

During training, further augmentation of the dataset with standard cropping and image processing of primitives was carried out, and in addition; a few specialized primitives i.e. modelling camera which censored noise and application of a randomized non-linear parametric transformation to the image intensity histogram (the latter is supposed to help in simulation of marginal lighting conditions). [1]

V. Building The Facial Landmarks Detection Application

The building of the face mesh application using Mediapipe involved four major steps:

A. Import the Libraries

This step involved initializing the model because that was the mandatory step. For that reason, the initialization of the Facial landmarks detection model was done by using the *face_mesh* method of Mediapipe's solutions class. In this method, the main function which performed the landmarks detection was *FaceMesh()*. [2]

B. Read an Image

First, the image was read from the particular path and ensured that the complete path is mentioned only if the image is not present in the same folder. Then, the figure function of the *matplotlib* library was used to set the figure size i.e. image size. Lastly, the image was displayed on which testing the facial landmarks detection model happened. [2]

C. Perform Face Landmarks Detection

After the face mesh model was initialized using the Mediapipe library. Then came the landmarks detection based on the previous pre-processing and with the help of *FaceMesh*'s process function where the 468 facial landmarks points in the image were gotten. [2]

As discussed, the detection was done using the function but before that, the image has to be converted from the color format of BGR to RGB. [2]

Next, there was storing of the left eye and right eye coordinate values in separate variables using the *iter tools* chain method. But before the main functionality was tackled, some validation was done to check whether faces had been found or not. [2]

If the above statement was correct then only iterations over all the faces depending on the arguments was carried out. Then the face number was displayed upon successfully detecting all the faces. [2]

At this point, the landmarks points of the left eye and right eye were displayed plus the name of them too. First, the iteration over two landmarks points of the left eye. Then the same was done for the right eye using *for* loop. [2]

Then finally the displaying of the landmarks points but those indexes were in normalized form so that the ML pipeline behind this detection could give equal weightage to each point. [2]

D. Drawing the Results

Firstly, a copy of the sample image was made and converted into the RGB format because the result had to be visualized. Then, the process from here resembled the previous step to determine if faces had been found. The results or landmarks points on the image were then drawn using the default setting i.e. *face mesh tesselation style* in the connection argument. As for the displaying of the face mesh contours the same default setting was applied. At last, setting the figure/image size was done along with displaying all the results on the image. [2]

VI. Results and Discussion

A. Resultant Image

Figure 3 shows an example of output from the implementation of how the facial landmarks in an image can be detected by the help of the Mediapipe library in python. In the final model implementation, the face mesh annotations on the image were drawn by the help of the *facemesh_tesselation* style setting.

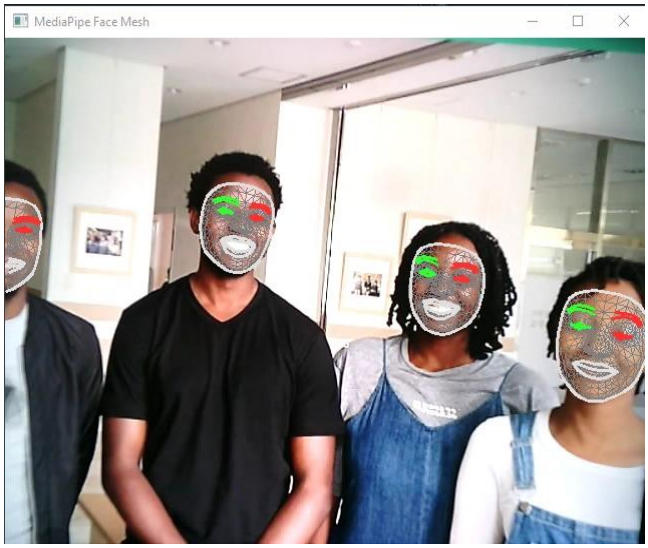

Figure 3: Example of output of the face mesh landmark model

B. Discussion

The facial landmarks can be detected and therefore enable the building of things like Snapchat-like filters i.e. in facial texture painting and AR object rendering (see Figure 4) which are booming in the entertainment industry. [2]

In this model application, it is established that printing the results and how they're drawn are through the *draw_landmarks()* function during the setting up of the facial landmarks detection application. [2]

The main takeaway from this paper is that segregating all the predefined functions can be done so that we can learn the in-depth intuition behind the ML Mediapipe algorithms for facial landmarks detection [2]

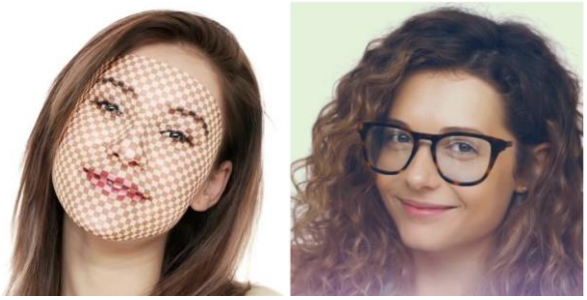

Figure 4: Other application examples: Facial texture painting and AR object rendering (glasses)[1]

VII. Conclusion

From this paper we have seen that the detection of facial landmarks as opposed to the previous face detection model using Mediapipe can be very much handy in real-world whether it is applied in the medicine or entertainment industry.

VIII. Acknowledgment

I would like to extend my thanks to Professor Zubaer for guiding my colleagues and I through the basic steps of preparing this report, and as well as teaching us the standard ways of carrying out this research project.

IX. References

[1] Bazarevsky, V., Kartynnik, Y., Vakunov, A., Raveendran, K., & Grundmann, M. (2019, July 14). *Blazeface: Sub-millisecond neural face detection on mobile gpus*. arXiv.org. [Online]. Available: https://arxiv.org/abs/1907.05047

[2] Analytics Vidhya. (2022, March 22). *Facial landmarks detection using Mediapipe Library*. [Online]. Available: https://www.analyticsvidhya.com/blog/2022/03/facial-landmarks-detection-using-mediapipe-library/

[3] mediapipe. (n.d.). *Face mesh*. [Online]. Available: https://google.github.io/mediapipe/solutions/face_mesh.html#ml-pipeline

[4] Kartynnik, Y., Ablavatski, A., Grishchenko, I., & Grundmann, M. (2019, July 15). *Real-time facial surface geometry from monocular video on mobile gpus*. arXiv.org. [Online]. Available: https://arxiv.org/abs/1907.06724