3D- GUIDED FACE SHAPE CLASSIFICATION

A Project

Presented to the

Faculty of

California State Polytechnic University, Pomona

In Partial Fulfillment

Of the Requirements for the Degree

Master of Science

In

Computer Science

By

Hinal N. Shah

2019

SIGNATURE PAGE

PROJECT:	3D- GUIDED FACE SHAPE CLASSIFICATION
AUTHOR:	Hinal N. Shah
DATE SUBMITTED:	Fall 2019
	Department of Computer Science

Dr. Hao Ji
Project Committee Chair
Computer Science

Dr. Yu Sun
Project Committee Member
Computer Science

ACKNOWLEDGEMENTS

I would like to express my deep and sincere gratitude to my project supervisor Dr. Hao Ji for the patient guidance, and support throughout the course. I am extremely lucky to have him as my advisor who has always encouraged me and motivated me. His timely and scholarly advice has helped me to great extent for accomplishing this work. I would also like to thank Dr. Yu Sun for his willingness to serve as committee member and to review my work.

I am also grateful to all those whom have I worked with in other academic projects. Finally, the sincerest gratitude to my parents for an unconditional love and care and for always being there to give me moral support throughout my ups and downs. I would also like to thank my husband for always providing me an unfailing support throughout my academic years. Above all, thanks to God for his blessings and for being my strength.

Thank you,

Hinal N. Shah

ABSTRACT

Face shape classification plays an important role in many facial analysis and beautification applications. Most existing approaches for determining face shape are in the image space. In this project, we investigate the approaches for face shape classification in 3D space. 3D face reconstruction from a single 2D image is used to measure the forehead, cheekbone, jawline, and face length. Using the facial features measured in 3D space, we evaluate machine learning algorithms such as decision trees, random forest, and SVM methods for face shape classification.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1: INTRODUCTION

Face shape recognition has become very useful in many computer vision applications. So, an algorithm to classify the face shape correctly is needed. Many existing algorithms detects face shape from 2D data but very few have been studied based on 3D data. There can problems if the images are not of good quality and has pose variability but there are not such issues in 3D data and hence can be useful to get properly classify the face shape. However, in 3D data, it is difficult to get the analyze the face shape from .In this paper, I aim to classify the five types of face shapes: heart, oval, round, oblong and square. The face shape is to be analyzed from the frontal face side.

In this paper, we analyze the 3D data obtained from 2D image and evaluate the machine learning classification algorithms to determine the face shape. Many times, 2D data have faces that are not properly aligned which makes it difficult to detect and get the facial landmarks using library like dlib. Hence, in this experiment, FaceAlignment library is used which properly detect such faces and gives facial landmarks in 2D and 3D space. For reconstruction of 3D data from single 2D images, EOS fitting model is used in the experiment. EOS fitting framework is a lightweight framework built in C++ which provides many functions to generate and operate on 3D mesh.

From 3D mesh generated using EOS library, the next step is to extract facial features from 3D mesh. And this is done by selecting 3D vertices to get the basic facial features such as face length, width of cheekbones, jawline, chin and forehead. Using these features as input to machine learning algorithms for face shape classification. In this experiment, Decision Tree, Random Forest and SVM algorithms are used. The project is done using python as programming language.

CHAPTER 2: LITERATURE REVIEW

This section gives brief explanation of the two techniques that are used to get 3D mesh data from 2D image.

2.1 Face Alignment Network (FAN)

This method uses CNN for face alignment in 2D and 3D space by regressing facial landmark locations. It is based on two components: the state-of-the-art Hour-Glass (HG) network, and the hierarchical, parallel & multi-scale block. It used a stack of four HG and replace the bottleneck block with hierarchical, parallel and multi-scale block. Implementing this type of block and by keeping all the same number of network parameter resulted in better performance (Figure 1). Its 2D-FAN is trained using 300W-LP-2D training dataset, 300-W test dataset, 300-VW (both training and test datasets), and Menpo frontal subset. On more than 220,000 images, 2D-FAN model is evaluated and for 3DFAN dataset used: 300W-LP-3D.

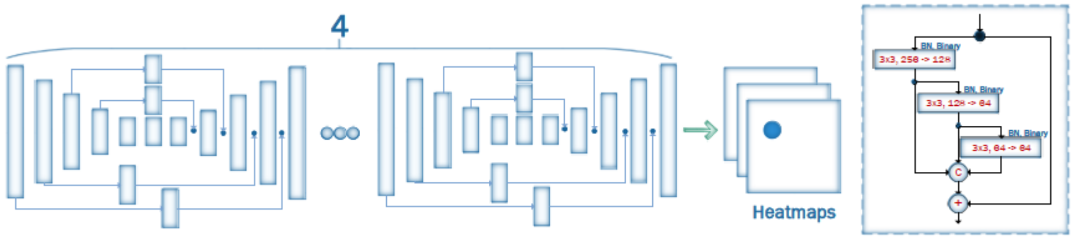


Figure 1. Face Alignment Network Architecture Overview

As the 2D landmarks of 300-W-LP-2D are slightly different from the 2D landmarks of the 300-W test set, 300-VW and Menpo, the 2D-FAN was further tuned on the 300-W training dataset. The 2D-FAN and 3D-FAN are trained separately by setting initial learning rate to 10^{-4} with minibatch of 10. During the training process, the learning rate decreased to 10^{-5} after 15 epochs and then to 10^{-6} for another 15 epochs. Total 40 epochs taken for training. (2D landmark samples are shown in Figure 2) .

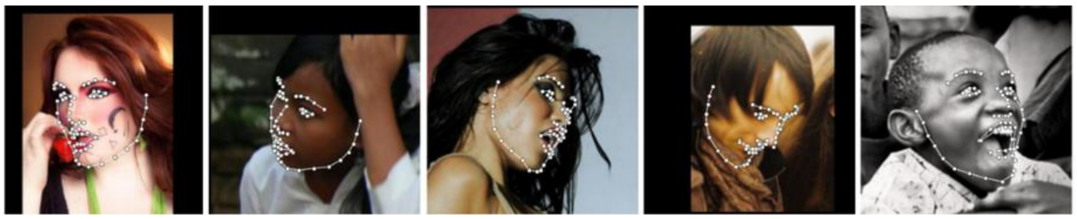


Figure 2. 2D Landmarks Using 2D-FAN

Usually the metric used for face alignment is point-to-point Euclidean distance normalized by interocular distance, but this is biased when interocular distance is small for profile faces. So, this algorithm normalize the bounding box and used normalized mean error.

2.2 EOS Model Fitting Framework

This framework presents Surrey Face Model which is multi resolution 3D Morphable Face Model that has three level of resolution of shape and color, pose invariant image texture representation and landmark point information as metadata. The model built using number of 3D scans. These scans are then registered using Iterative Multi-resolution Dense 3D Registration (IMDR) algorithm. Gaussian and median filtering denoising algorithm is applied to the target scan. It uses the generic reference face model that has 845 vertices and 1610 triangles and performs global mapping on it using facial landmarks to get the target scan. Based on distances between vertices of reference and target, local matching is done on current resolution. After energy minimization process, the generic face model is subdivided until the highest resolution of mesh is achieved.

The low-level resolution model consists of 3448 vertices whereas the high-resolution model consists of 29587 vertices. As the higher resolution meshes are built upon the lower resolution, each vertex of a lower resolution mesh is present in higher resolution meshes. It consists of two PCA models each for shape and color information. Both the models are built using 169 registered scans. Figure 3 shows the PCA shape and color coefficients and the mean model. The high frequency information gets lost if only PCA color model is used to represent the face. So, the texture information of the face is considered and remapped it onto the model and saved as 2D representation using isomap algorithm.

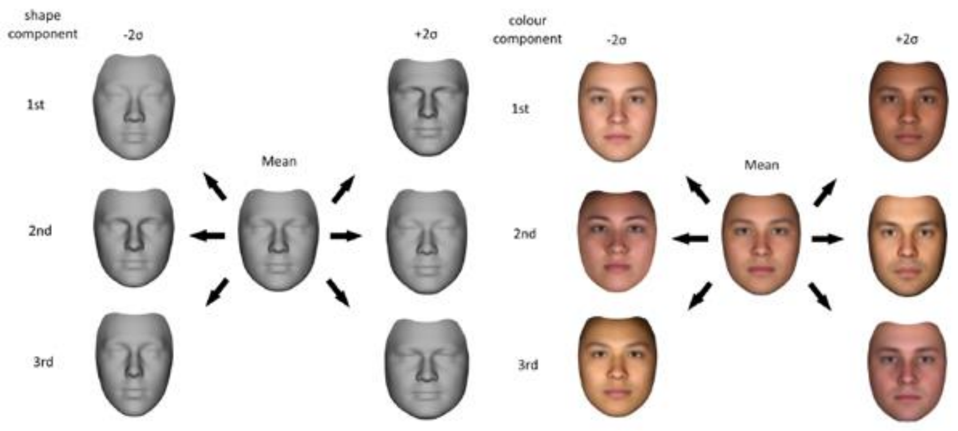


Figure 3.EOS PCA models for shape and color

The software framework of this model is lightweight cross-platform library which is built using C++ that provides various classes and functions to use model. The core functionality has a PcaModel class which contains a mean, the PCA basis vectors and eigenvalues, and a MorphableModel class which contains a shape and color. It provides LandmarkMapper class which stores landmark information. It includes supporting files for low resolution model. It also includes functions to fit shape and pose of the model. For pose estimation , it assumes affine camera model and implements Gold Standard Algorithm .

2.3 Regressing Robust and Discriminative 3DMM

There are other methods to get 3D mesh and one of them is Regressing Robust and Discriminative 3D Morphable Models technique which regresses 3D Morphable

Model (3DMM) face shape parameters from the input image by implementing deep neural network. Multi image 3DMM reconstruction estimation method is used on 500k unconstrained faces in CASIA WebFace dataset to generate training data. For 3DMM representation, it uses Basel Face Model (BFM). To fit 3DMM to each training image, it applies CLNF facial landmark detector which is used to obtain initial estimate for the input image and then performs optimization over shape, pose, texture. Once the optimization converges, the shape and texture parameters are used as single image 3DMM estimates. For multi-image 3D face shape estimation, pooling is performed for shape and texture 3DMM parameters across all the N single view estimates of the same subject. Using this data, training of CNN is performed. The CNN model uses deep ResNet architecture with 101 layers with modified last fully connected layer to give 198D 3DMM feature vector.

To evaluate the accuracy of the estimated 3D shapes, it tested on videos and images and their corresponding ground truth values from MICC Florence Faces dataset. The datasets LFW, YTF and the new IARPA JANUS Benchmark-A (IJB-A) are used for single and multi-image face recognition in order to test the model robustness of shapes. However, in this experiment, EOS Model Fitting Framework is used.

CHAPTER 3: METHODOLOGY

This section describes the whole process of generating 3D mesh from images and applying various machine learning algorithms for classification

3.1 Data Collection

The data is collected for basic face shapes namely heart, diamond, oval, oblong, round and square. Below is the brief description about each face shape characteristics:

- Heart Shape: These faces typical have wider forehead and a narrow chin. There is sloping sides that starts from the forehead and meets at the chin. Celebrities like Chloe Grace Moretz and Jennifer Love Hewitt are examples of this kind of face shape.
- Diamond Shape: These faces are characterized by wide cheekbones, pointed chins and hairline will be narrower. Examples: the face of celebrity Christian Bale, Jennifer Lopez
- Oblong Shape: This kind of faces has the largest face length and has similar forehead, cheekbones and jawline length. Celebrities Kiefer Sutherland, Jessica Parker are well known to have this kind of face.
- Oval Shape: These faces have face length longer than cheekbone width and forehead width larger than the jawline such that the height to width ratio is of about 3:2. Celebrity with this kind of face are Emma Watson and Uma Thurman.
- Round Shape: These faces have similar face length and cheekbone width and have similar sized forehead and jawline. The round shape has softer angles. Example: Selena Gomez, Leonardo DiCaprio

- Square Shape: These faces usually have strong angular jaws and have face length and width are in equal proportions. Celebrities that has prominent square face are Tom Cruise and Olivia Wilde

For this experiment, a face shape dataset from Adonis Tio work is used. In this dataset, there are 17 or 18 celebrities, each with 5 or 10 images in each face shape category to get total of 100 images in each category which makes the total up to 500 (Figure 4). The dataset contains images that are widely agreed upon face shape to minimize human bias and help to improve the classification process.

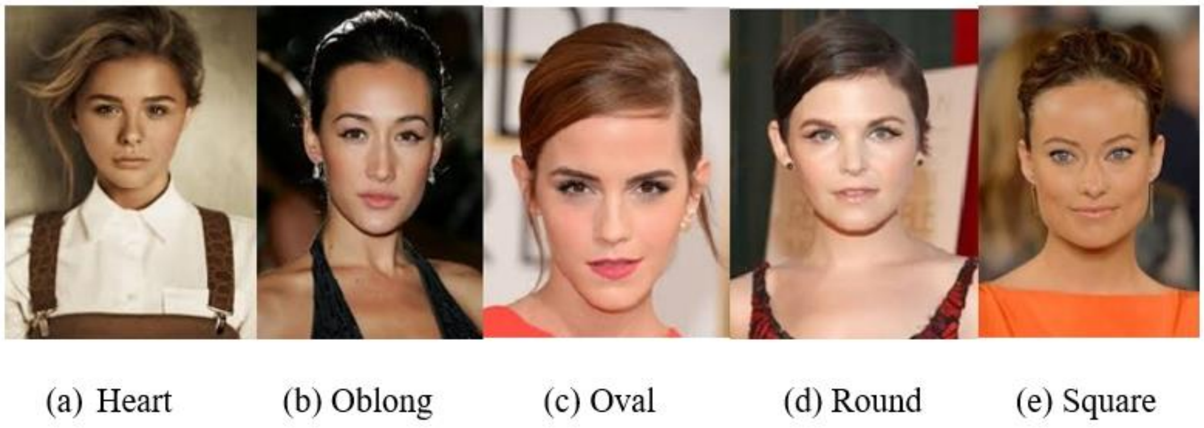


Figure 4. Sample Images from female dataset

3.2 Face Alignment Landmarks

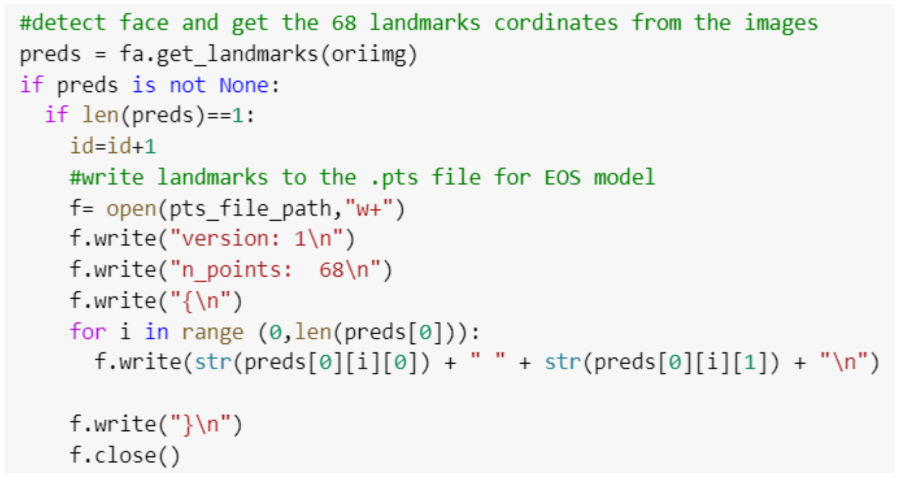
For generation of 3D mesh using EOS model fitting library, .pts file is need as input for each 2D image. The .pts file consist of coordinates of 68 facial landmarks. The face alignment module can be installed using following command:

```
pip install face-alignment
```

The very first step to generate .pts is to load face alignment model for 2D images

```
#load faceAlignment model for 2D images
fa = face_alignment.FaceAlignment(face_alignment.LandmarksType._2D, flip_input=False)
```

To get the 68 landmarks, the module has get_landmarks() function that take the input image path and returns the 68 landmarks which are then saved as .pts file (Figure 5)



```
#detect face and get the 68 landmarks cordinates from the images
preds = fa.get_landmarks(oriimg)
if preds is not None:
  if len(preds)==1:
    id=id+1
    #write landmarks to the .pts file for EOS model
    f= open(pts_file_path,"w+")
    f.write("version: 1\n")
    f.write("n_points:  68\n")
    f.write("{\n")
    for i in range (0,len(preds[0])):
      f.write(str(preds[0][i][0]) + " " + str(preds[0][i][1]) + "\n")

    f.write("}\n")
    f.close()
```

Figure 5.Generate .pts file

3.3 3D Data Generation

The 3D mesh is generated from 2D images from the dataset using EOS- lightweight 3D Morphable Face Model (3DMM) fitting library (Figure 6).

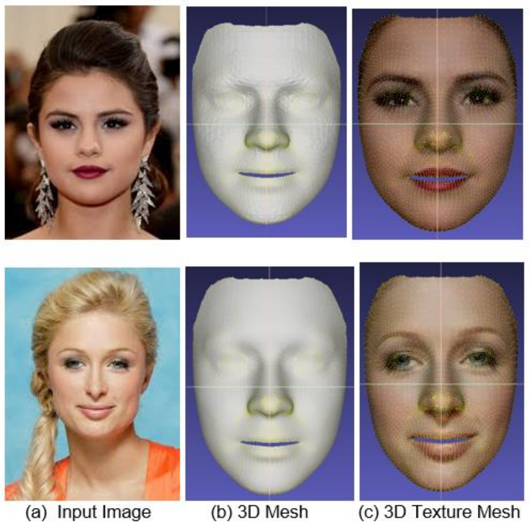
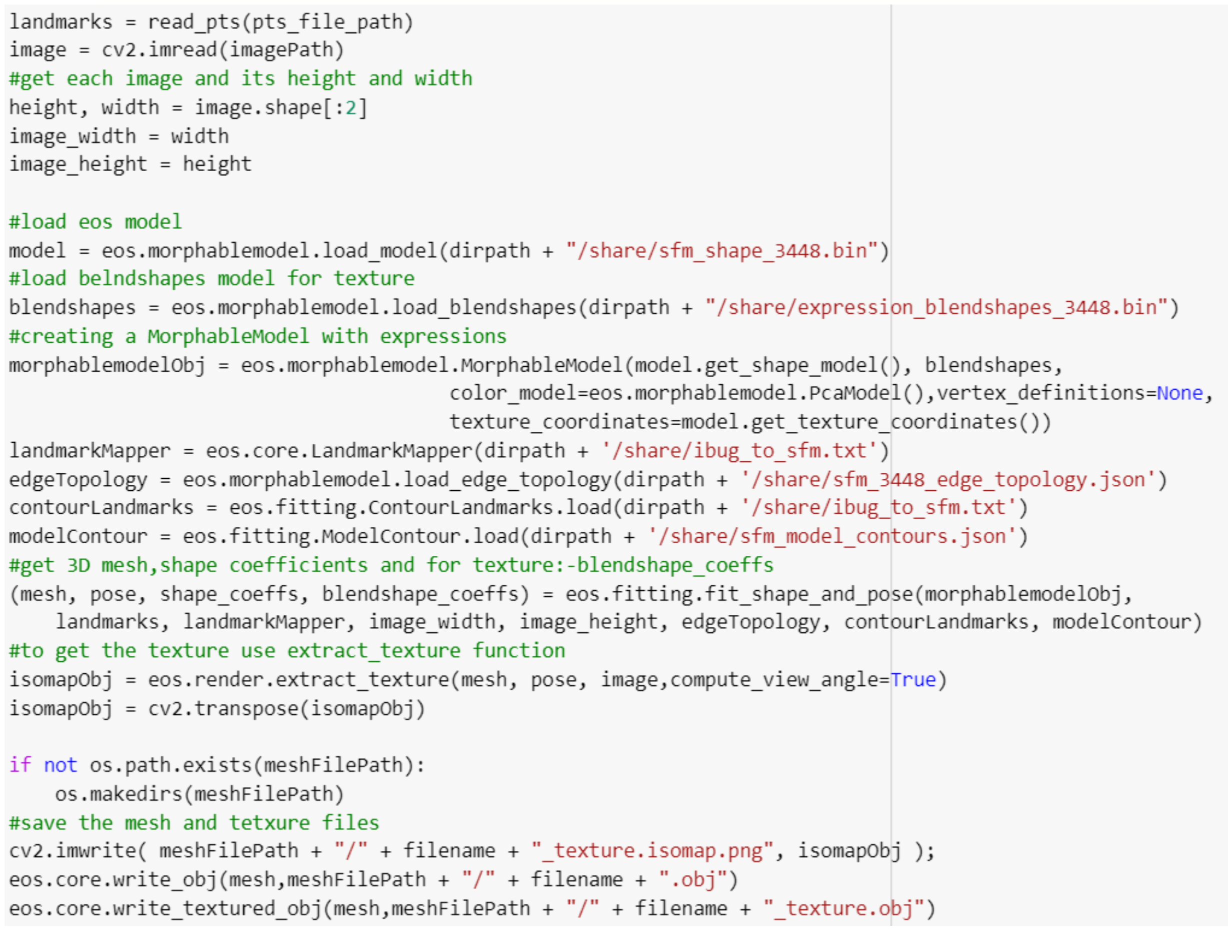


Figure 6. 3D Mesh from 2D image

The library provides basic classes and functions to generate and operate on 3DMM. From this library following functions are used in this project to get 3D mesh:

- load_model(): This function of eos.morphablemodel namespace loads the morphable model
- load_blendshapes(): This function of eos.morphablemodel namespace loads the morphable model for blendshapes
- get_shape_model(): It gives PCA shape model.
- PcaModel(): Gives a PCA model from the mean, normalised PCA basis, eigenvalues and triangle list.
- MorphableModel(): It creates a Morphable Model from a shape and a color PCA model
- LandmarkMapper: It constructs a new landmark mapper from a file that has landmark mappings to another.
- load_edge_topology: It loads a 3DMM edge topology file from a json file.
- ContourLandmarks: This class holds definitions for the contour (outline) on the right and left side of the reference 3D face model These can be found in the file share/model_contours.json
- fit_shape_and_pose():fits pose (camera), the shape model, and expression blendshapes to landmarks, in an iterative (alternating) way. It fits both sides of the face contour as well.
- extract_texture: Extracts the texture of the face from the given image and stores it as isomap.
- write_obj: Writes the given Mesh to an obj file that for example can be read by MeshLab. If the mesh contains vertex color information, it will be written to the obj as well.

- write_textured_obj: Writes an obj file of the given Mesh, including texture coordinates, and an mtl file containing a reference to the isomap. The obj will contain texture coordinates for the mesh, and the mtl file will link to a file named <filename>.isomap.png



```
landmarks = read_pts(pts_file_path)
image = cv2.imread(imagePath)
#get each image and its height and width
height, width = image.shape[:2]
image_width = width
image_height = height

#load eos model
model = eos.morphablemodel.load_model(dirpath + "/share/sfm_shape_3448.bin")
#load belndshapes model for texture
blendshapes = eos.morphablemodel.load_blendshapes(dirpath + "/share/expression_blendshapes_3448.bin")
#creating a MorphableModel with expressions
morphablemodelObj = eos.morphablemodel.MorphableModel(model.get_shape_model(), blendshapes,
                                      color_model=eos.morphablemodel.PcaModel(),vertex_definitions=None,
                                      texture_coordinates=model.get_texture_coordinates())
landmarkMapper = eos.core.LandmarkMapper(dirpath + '/share/ibug_to_sfm.txt')
edgeTopology = eos.morphablemodel.load_edge_topology(dirpath + '/share/sfm_3448_edge_topology.json')
contourLandmarks = eos.fitting.ContourLandmarks.load(dirpath + '/share/ibug_to_sfm.txt')
modelContour = eos.fitting.ModelContour.load(dirpath + '/share/sfm_model_contours.json')
#get 3D mesh,shape coefficients and for texture:-blendshape_coeffs
(mesh, pose, shape_coeffs, blendshape_coeffs) = eos.fitting.fit_shape_and_pose(morphablemodelObj,
    landmarks, landmarkMapper, image_width, image_height, edgeTopology, contourLandmarks, modelContour)
#to get the texture use extract_texture function
isomapObj = eos.render.extract_texture(mesh, pose, image,compute_view_angle=True)
isomapObj = cv2.transpose(isomapObj)

if not os.path.exists(meshFilePath):
    os.makedirs(meshFilePath)
#save the mesh and tetxure files
cv2.imwrite( meshFilePath + "/" + filename + "_texture.isomap.png", isomapObj );
eos.core.write_obj(mesh,meshFilePath + "/" + filename + ".obj")
eos.core.write_textured_obj(mesh,meshFilePath + "/" + filename + "_texture.obj")
```

Figure 7. 3D Mesh Generation using EOS Model framework

3.4 Facial Features Extraction

From the generated 3D mesh, landmarks are selected to get facial features such as cheekbone width, face length, jawline width and chin width. Below figure shows the red dots are the 3D vertices that are selected as landmarks to get facial features. For example, to get the face length, the top and bottom landmarks are used to measure the distance between them. Similarly, the distances are measured for other facial features like forehead, cheekbone, jawline and chin.

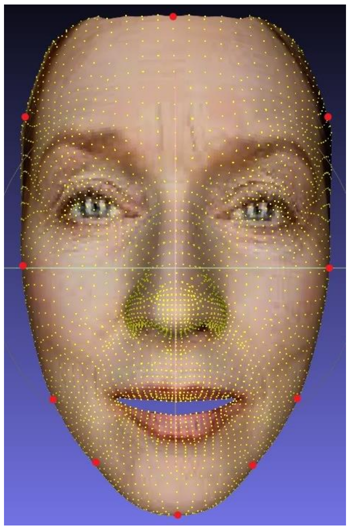
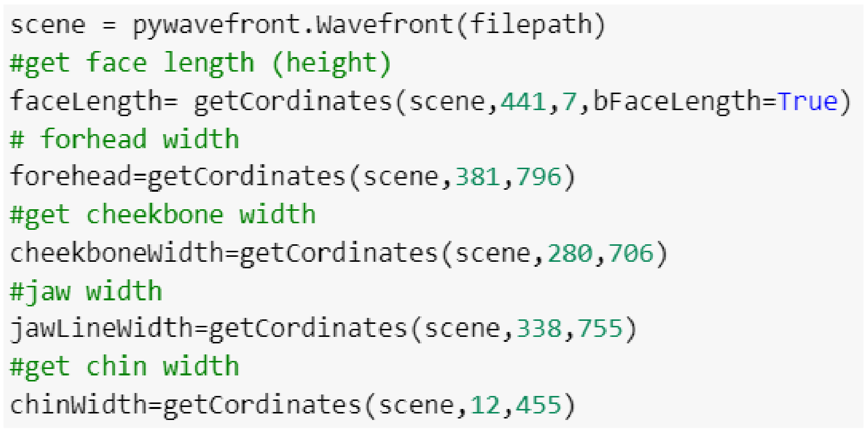


Figure 8. 3D Landmarks used for facial features

In order to get the vertices of 3D mesh pywavefront library is used and it can be installed using below command:

```
pip install pywavefront
```

Pywavefront provides various functions to read and visualizations of 3D objects. The Wavefront function loads the .obj file and scene.vertices is used to get landmarks coordinates which are then used to measure the distance for each facial feature

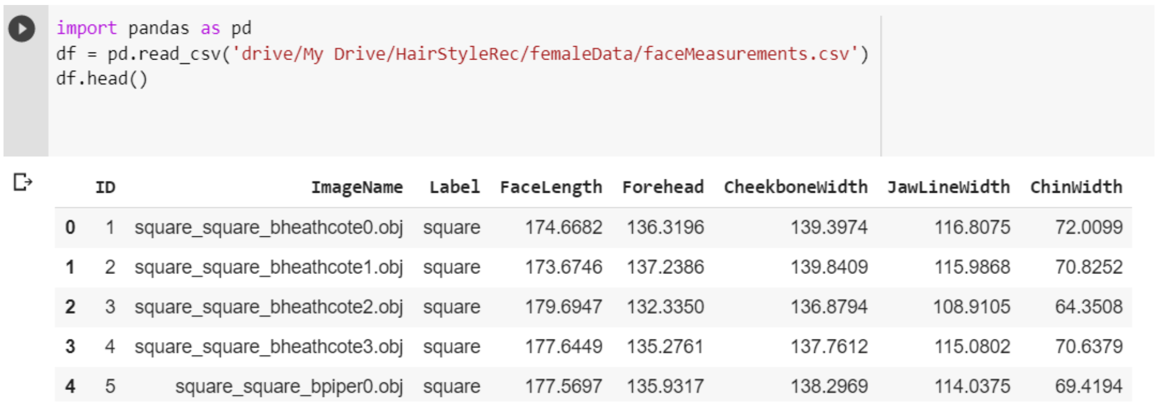


```
scene = pywavefront.Wavefront(filepath)
#get face length (height)
faceLength= getCordinates(scene,441,7,bFaceLength=True)
# forhead width
forehead=getCordinates(scene,381,796)
#get cheekbone width
cheekboneWidth=getCordinates(scene,280,706)
#jaw width
jawLineWidth=getCordinates(scene,338,755)
#get chin width
chinWidth=getCordinates(scene,12,455)
```

Figure 9. Using pywavefront

CHAPTER 4: CLASSIFICATION RESULTS

All the features extracted are saved in a .csv file format. Using pandas library, the system reads the file and then the data is split into training and testing dataset for classification (Figure 10).



```
import pandas as pd
df = pd.read_csv('drive/My Drive/HairStyleRec/femaleData/faceMeasurements.csv')
df.head()
```

	ID	ImageName	Label	FaceLength	Forehead	CheekboneWidth	JawLineWidth	ChinWidth
0	1	square_square_bheathcote0.obj	square	174.6682	136.3196	139.3974	116.8075	72.0099
1	2	square_square_bheathcote1.obj	square	173.6746	137.2386	139.8409	115.9868	70.8252
2	3	square_square_bheathcote2.obj	square	179.6947	132.3350	136.8794	108.9105	64.3508
3	4	square_square_bheathcote3.obj	square	177.6449	135.2761	137.7612	115.0802	70.6379
4	5	square_square_bpiper0.obj	square	177.5697	135.9317	138.2969	114.0375	69.4194

Figure 10. Input .csv File content

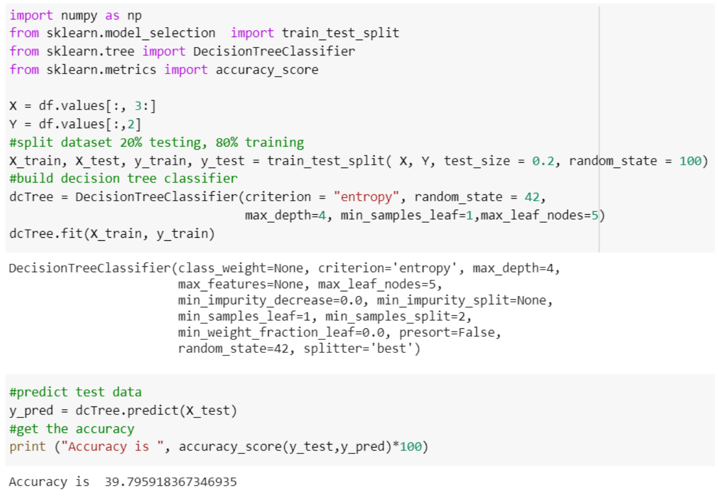
4.1 Decision Tree

It is probably the most popular algorithm in supervised learning for classification and regression problems. It uses binary tree like data structure which can map non-linear relationships unlike other linear models. It works by breaking down the dataset into smaller subsets such that each node represents feature, each branch represents decision rule and each leaf represents the category of the class. The top decision node is called root node while sub node is called decision node as it further splits up into sub-nodes. If the tree grows more deeper, the more complex will be the decision which may cause the overfitting problem. This algorithm does not require users to have detailed knowledge about the data. The deeper the tree, the more complex the decision rules and the fitter the model.

This project implements decision tree classifier using scikit-learn library. The library provides DecisionTreeClassifier class available in sklearn.tree module. This class takes following parameters:

- criterion : This measures the quality of a split. It can either be ‘gini’ or ‘entropy’. In order to gain information, entropy is used as criterion in this experiment. Information Gain is the difference between uncertainty of the starting node and weighted impurity of the two child nodes. Information gain decides which feature should be used to split the data.
- random_state**:** is the seed used by the random number generator
- max_depth: It is the maximum depth of the tree. As there are around 5 classes it is set to 4**.**
- min_samples_leaf: The minimum number of samples required to be at a leaf node.
- max_leaf_nodes: The number of leaf nodes needed. Here there are five classes of face shape, it is set to 5.

Below is the screenshot of Decision tree classifier implemented in this project. The model gave accuracy of 39% only (Figure 11).



```
import numpy as np
from sklearn.model_selection  import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score

X = df.values[:, 3:]
Y = df.values[:,2]
#split dataset 20% testing, 80% training
X_train, X_test, y_train, y_test = train_test_split( X, Y, test_size = 0.2, random_state = 100)
#build decision tree classifier
dcTree = DecisionTreeClassifier(criterion = "entropy", random_state = 42,
                                max_depth=4, min_samples_leaf=1,max_leaf_nodes=5)
dcTree.fit(X_train, y_train)
```

```
DecisionTreeClassifier(class_weight=None, criterion='entropy', max_depth=4,
                       max_features=None, max_leaf_nodes=5,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort=False,
                       random_state=42, splitter='best')
```

```
#predict test data
y_pred = dcTree.predict(X_test)
#get the accuracy
print ("Accuracy is ", accuracy_score(y_test,y_pred)*100)
```

```
Accuracy is  39.795918367346935
```

Figure 11. Decision Tree Implementation

To visualize the decision tree, scikit-learn library provides export_graphviz class to display tree within the python notebook (Figure 12).

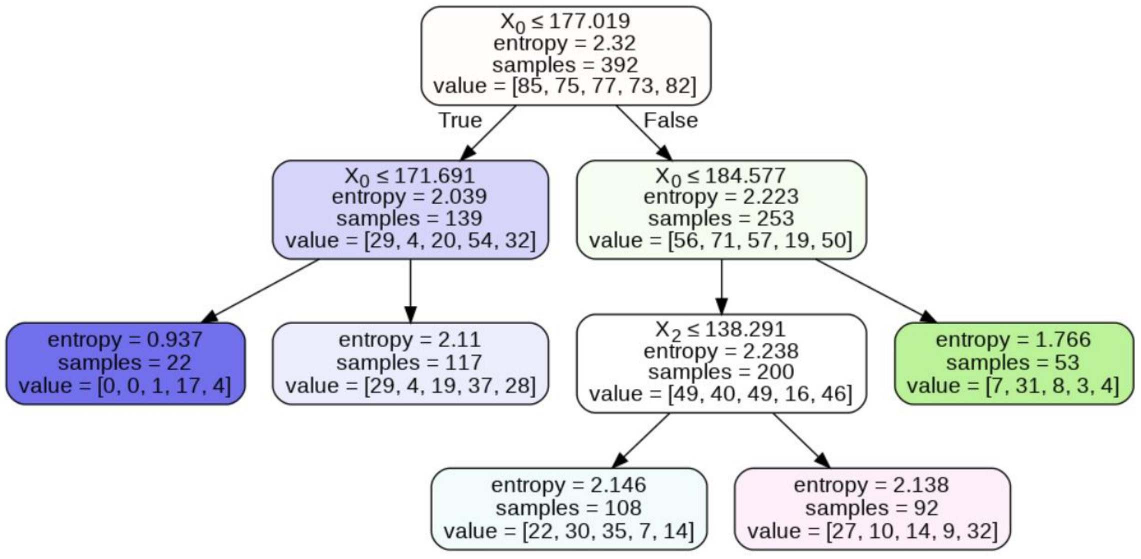


Figure 12. Decision Tree Visualization

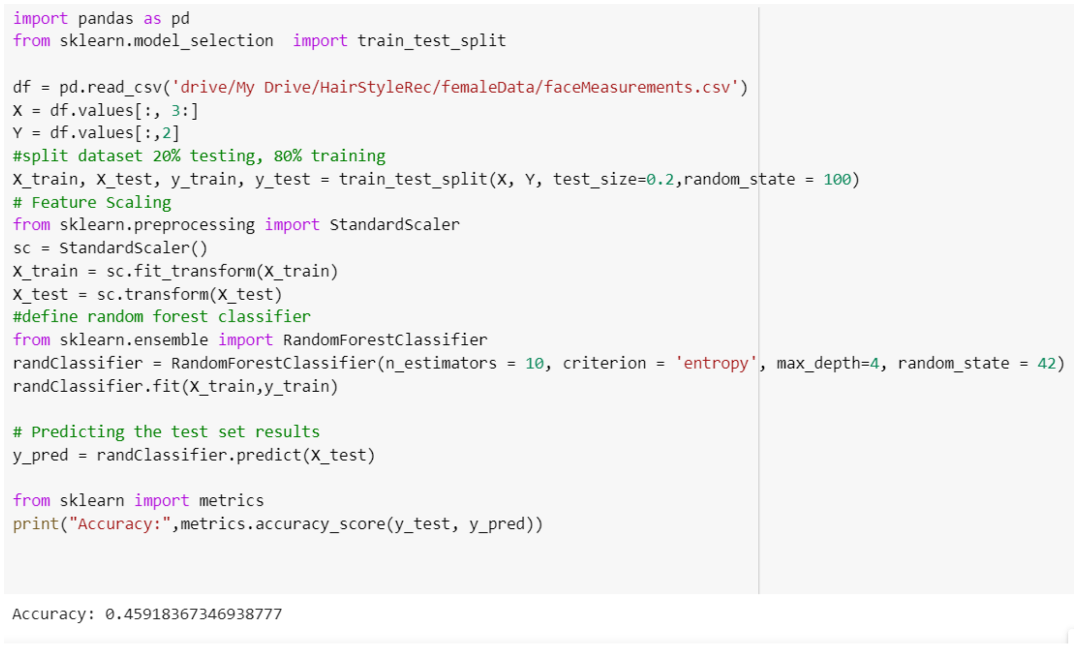
But to save it as image, pydotplus library needs to be installed. By executing below commands, it will get installed onto the system.

```
pip install -q pydot
```

4.2 Random Forest

As decision tree works on whole dataset and keeps improving at each depth but this may cause data overfitting. Also, it does not give the global optimal solution. Random Forest classifier can overcome this problem. It consists of many decision trees and gives the result as the average of the predictions made by all the decision trees. It operates as an ensemble and helps to get better performance. It works on random samples of the data and at each node random features are used for splitting and hence avoid overfitting of the data.

In this experiment, using scikit-learn library, RandomForestClassifier class is used to implement Random Forest classifier. The n_ estimators parameter of this class is set as 10 which means there are 10 number of trees to be included in the random forest. This model gave 45% accuracy (Figure 13).

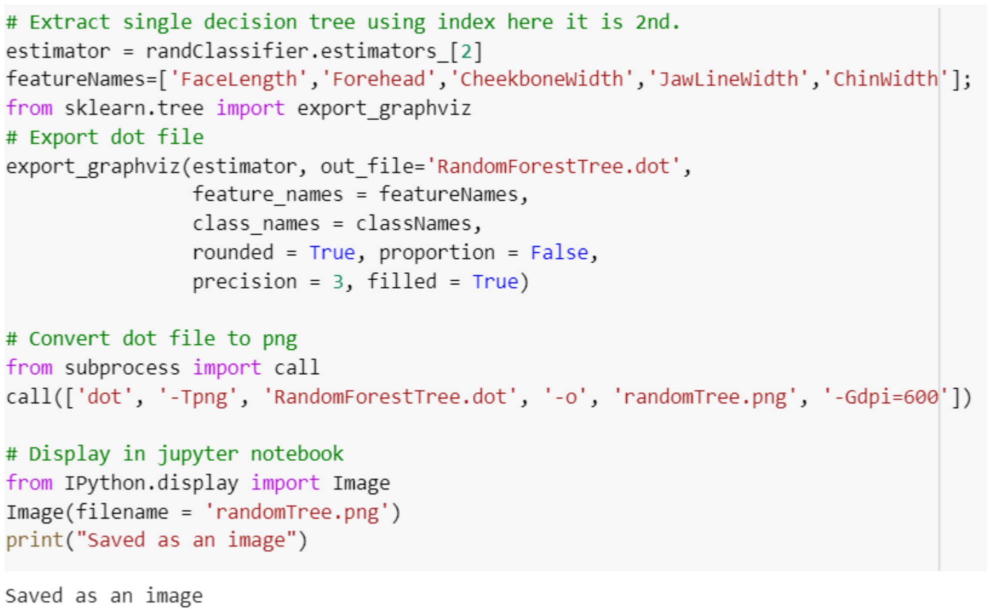


```
import pandas as pd
from sklearn.model_selection  import train_test_split

df = pd.read_csv('drive/My Drive/HairStyleRec/femaleData/faceMeasurements.csv')
X = df.values[:, 3:]
Y = df.values[:,2]
#split dataset 20% testing, 80% training
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,random_state = 100)
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
#define random forest classifier
from sklearn.ensemble import RandomForestClassifier
randClassifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', max_depth=4, random_state = 42)
randClassifier.fit(X_train,y_train)

# Predicting the test set results
y_pred = randClassifier.predict(X_test)

from sklearn import metrics
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.45918367346938777
```

Figure 13. Random Forest Implementation

To visualize a single decision tree from the random forest, use export_graphviz class (Figure 14).



```
# Extract single decision tree using index here it is 2nd.
estimator = randClassifier.estimators_[2]
featureNames=['FaceLength','Forehead','CheekboneWidth','JawLineWidth','ChinWidth'];
from sklearn.tree import export_graphviz
# Export dot file
export_graphviz(estimator, out_file='RandomForestTree.dot',
                feature_names = featureNames,
                class_names = classNames,
                rounded = True, proportion = False,
                precision = 3, filled = True)

# Convert dot file to png
from subprocess import call
call(['dot', '-Tpng', 'RandomForestTree.dot', '-o', 'randomTree.png', '-Gdpi=600'])

# Display in jupyter notebook
from IPython.display import Image
Image(filename = 'randomTree.png')
print("Saved as an image")
```

```
Saved as an image
```

Figure 14. To visualize single Decision Tree of Random Forest

Below is the image of single decision tree from the random forest:

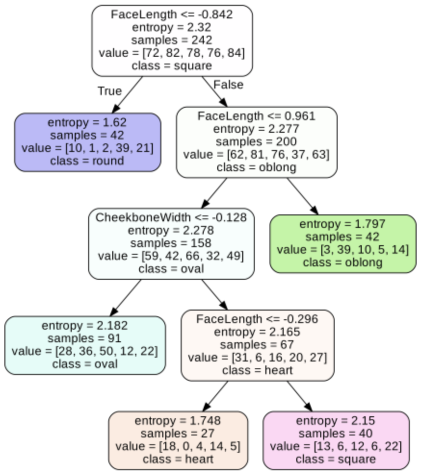
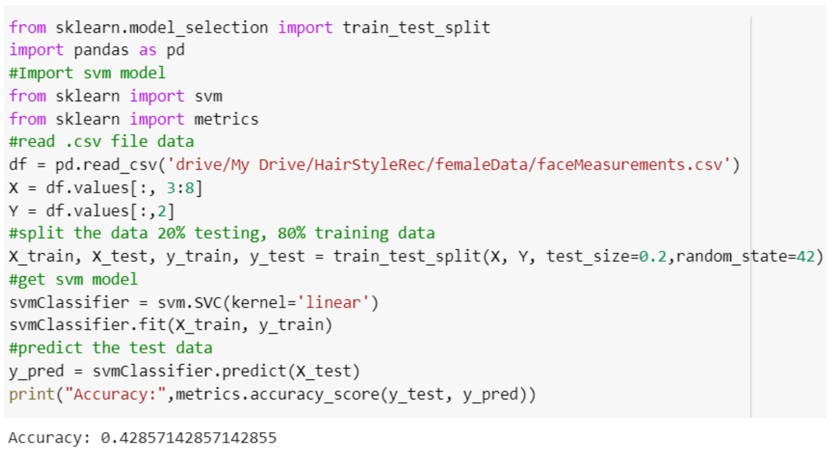


Figure 15. Single Decision Tree of Random Forest Visualization

4.3 SVM

A Support Vector Machine (SVM) is a classifier that aims to get an optimal hyperplane that can distinctly classify the data. The data points closer to the hyperplane are called support vectors that influence position of hyperplane. SVM is effective in high dimensional space. In this project, SVM is implemented using scikit-learn library for classification. To get SVM classifier use svm.SVC() class and set the kernel as 'Linear' (Figure 16). However, this model gives accuracy of 42%



```
from sklearn.model_selection import train_test_split
import pandas as pd
#Import svm model
from sklearn import svm
from sklearn import metrics
#read .csv file data
df = pd.read_csv('drive/My Drive/HairStyleRec/femaleData/faceMeasurements.csv')
X = df.values[:, 3:8]
Y = df.values[:,2]
#split the data 20% testing, 80% training data
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2,random_state=42)
#get svm model
svmClassifier = svm.SVC(kernel='linear')
svmClassifier.fit(X_train, y_train)
#predict the test data
y_pred = svmClassifier.predict(X_test)
print("Accuracy:",metrics.accuracy_score(y_test, y_pred))
```

```
Accuracy: 0.42857142857142855
```

Figure 16. SVM Implementation

CHAPTER 5: CONCLUSION

This paper presents experimental results for face shape classification using 3D facial landmarks. It presents an approach to extract facial features from 3D reconstructed face from single 2D image. However, due to the small number of female and male dataset the results obtained are not the best and little can be said about the model capabilities to classify the face shape correctly, thus much work needs to be done on data collection. This work can be considered if large dataset is available. It is found that using Random forest is more effective than SVM and Decision Tree used in the experiment. However, only few machine learning algorithms are used for this experiment. There could be a chance to improve model performance by using 3DCNN to classify 3D mesh. There are only five basic facial features are considered which might have not given adequate information about the exact face shape. Work could be done to extract more facial features. If the better model is built, then it can be useful in many beautification applications for example, hair recommendation system.

REFERENCES

[1] Bulat, Adrian, and Georgios Tzimiropoulos. “How Far Are We from Solving the 2D & 3D Face Alignment Problem? (and a Dataset of 230,000 3D Facial Landmarks).” 2017 IEEE International Conference on Computer Vision (ICCV), 2017, doi:10.1109/iccv.2017.116

[2] 1adrianb. “1adrianb/Face-Alignment.” GitHub, 31 Aug. 2019, https://github.com/1adrianb/face-alignment.

[3] Huber, P., Hu, G., Tena, R., Mortazavian, P., Koppen, W. P., Christmas, W. J., … Kittler, J. (2016). A Multiresolution 3D Morphable Face Model and Fitting Framework. Proceedings of the 11th Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications. doi: 10.5220/0005669500790086

[4] Patrikhuber. “Patrikhuber/EOS.” GitHub, 19 July 2019, https://github.com/patrikhuber/eos.

[5] Tio, A. (2019). Face shape classification using Inception v3. [online] arXiv.org. Available at: https://arxiv.org/abs/1911.07916 [Accessed 2 Dec. 2019]

[6] Adonistio. "Adonistio/Inception-Face-Shape-Classifier." GitHub, https://github.com/adonistio/inception-face-shape-classifier.

[7] Tran, Anh Tuan, et al. “Regressing Robust and Discriminative 3D Morphable Models with a Very Deep Neural Network.” 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2017, doi:10.1109/cvpr.2017.163.