**Training Face Landmark Detector**

**PROJECT REPORT**

**GROUP MEMBERS:**

**MUJTABA SHAHID FAIZI**

**M.YASIR HAYAT KHAN**

**ZIYAB IKRAM**

**MUHAMMAD ATIF**

**Goal of the Project:**

**This application helps to train your own face landmark detector. You can train your own face landmark detection by just providing a bunch of images of yours of different poses and expressions at different places.**

**Implementation:**

**Training:**

CascadeClassifier face\_cascade;

bool myDetector( InputArray image, OutputArray ROIs );

bool myDetector( InputArray image, OutputArray ROIs ){

Mat gray;

std::vector<Rect> faces;

if(image.channels()>1){

cvtColor(image.getMat(),gray,CV\_BGR2GRAY);

}

else{

gray = image.getMat().clone();

}

equalizeHist( gray, gray );

face\_cascade.detectMultiScale( gray, faces, 1.1, 3,0, Size(30, 30) );

Mat(faces).copyTo(ROIs);

return true;

}

The facemark API provides the functionality to the user to use their own face detector to be used in training.The above code creartes a sample face detector. The above function would be passed to a function pointer in the facemark API.

vector<String> filenames;

glob(directory,filenames);

The above code creates a vector filenames for storing the names of the .txt files. It gets the filenames of the files in the directory.

Mat img = imread(image);

face\_cascade.load(cascade\_name);

FacemarkKazemi::Params params;

params.configfile = configfile\_name;

Ptr<Facemark> facemark = FacemarkKazemi::create(params);

facemark->setFaceDetector(myDetector);

The above code creates a pointer of the face landmark detection class. The face detector created above has to be passed as function pointer to the facemark pointer created for detecting faces while training the model.

vector<String> imagenames;

vector< vector<Point2f> > trainlandmarks,Trainlandmarks;

vector<Mat> trainimages;

loadTrainingData(filenames,trainlandmarks,imagenames);

for(unsigned long i=0;i<300;i++){

string imgname = imagenames[i].substr(0, imagenames[i].size()-1);

string img = directory + string(imgname) + ".jpg";

Mat src = imread(img);

if(src.empty()){

cerr<<string("Image "+img+" not found\n.")<<endl;

continue;

}

trainimages.push\_back(src);

Trainlandmarks.push\_back(trainlandmarks[i]);

}

The above code creates std::vectors to store the images and their corresponding landmarks. The above code calls a function loadTrainingData to load the landmarks and the images into their respective vectors.

If the dataset you downloaded is of the following format :

version: 1

n\_points: 68

{

115.167660 220.807529

116.164839 245.721357

120.208690 270.389841

...

}

This is the example of the dataset available at https://ibug.doc.ic.ac.uk/resources/facial-point-annotations/

Then skip the above code for loading training data and use the following code. This sample is provided as sampleTrainLandmarkDetector2.cpp in the face module in opencv contrib.

std::vector<String> images;

std::vector<std::vector<Point2f> > facePoints;

loadTrainingData(imagesList, annotations, images, facePoints, 0.0);

In the above code imagelist and annotations are the file of following format :

example of contents for images.txt:

../trainset/image\_0001.png

../trainset/image\_0002.png

example of contents for annotation.txt:

../trainset/image\_0001.pts

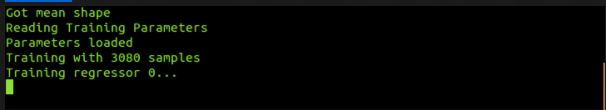
../trainset/image\_0002.pts

The above code scales images and landmarks as training on images of smaller size takes less time. This is because processing larger images requires more time. After scaling data it calculates mean shape of the data which is used as initial shape while training.

Finally call the following function to perform training :

facemark->training(Trainimages,Trainlandmarks,configfile\_name,scale,modelfile\_name);

In the above function scale is passed to scale all images and the corresponding landmarks so that the size of all images can be reduced as it takes greater time to process large images. This call to the train function trains the model and stores the trained model file with the given filename specified.As the training starts successfully you will see something like this :



**Face Landmark Detection in an Image:**

****

This application lets you detect landmarks of detected faces in an image. You can detect landmarks of all the faces found in an image and use them further in various applications like face swapping, face averaging etc. This functionality is now available in OpenCV.

// Command to be typed for running the sample

./sampleDetectLandmarks -file=trained\_model.dat -face\_cascade=lbpcascadefrontalface.xml -image=/path\_to\_image/image.jpg

**Description of command parameters:**

model\_filename f : (REQUIRED) A path to binary file storing the trained model which is to be loaded [example - /data/file.dat]

image i : (REQUIRED) A path to image in which face landmarks have to be detected.[example - /data/image.jpg]

face\_cascade c : (REQUIRED) A path to the face cascade xml file which you want to use as a face detector.