# MOS-16798: Tech story for Sprint FIT 3 - Face Recognition for Authentication

This user story is for the implementation of face recognition mechanism for Authentication purpose.

As a part of working on this User Story, We have tried to implement Face Recognition using Open IMAJ Library which we have used for face detection.

Below is the PoC done on Open IMAJ library for Face Recognition:

**package** facedetection;

**import** java.awt.image.BufferedImage;

**import** java.io.File;

**import** java.io.IOException;

**import** java.net.URL;

**import** java.util.Map;

**import** java.util.Map.Entry;

**import** javax.imageio.ImageIO;

**import** org.openimaj.feature.FloatFV;

**import** org.openimaj.feature.FloatFVComparison;

**import** org.openimaj.image.DisplayUtilities;

**import** org.openimaj.image.FImage;

**import** org.openimaj.image.ImageUtilities;

**import** org.openimaj.image.processing.face.detection.HaarCascadeDetector;

**import** org.openimaj.image.processing.face.detection.keypoints.FKEFaceDetector;

**import** org.openimaj.image.processing.face.detection.keypoints.KEDetectedFace;

**import** org.openimaj.image.processing.face.feature.FacePatchFeature;

**import** org.openimaj.image.processing.face.feature.FacePatchFeature.Extractor;

**import** org.openimaj.image.processing.face.feature.comparison.FaceFVComparator;

**import** org.openimaj.image.processing.face.similarity.FaceSimilarityEngine;

**import** org.openimaj.math.geometry.shape.Rectangle;

**public** **class** FaceSimilarity {

/\*\*

\* Main method for the example.

\*

\* **@param** args

\* Ignored.

\* **@throws** IOException

\*/

**public** **static** **void** main(String[] args) **throws** IOException {

// first, we load two images

**final** URL image1url = **new** URL(

"http://s3.amazonaws.com/rapgenius/fema\_-\_39841\_-\_official\_portrait\_of\_president-elect\_barack\_obama\_on\_jan-\_13.jpg");

BufferedImage image = ImageIO.*read*(**new** File("Singlephoto.jpg"));

**final** FImage image1 = ImageUtilities.*readF*(image1url);

**final** FImage image2 = ImageUtilities.*createFImage*(image);

// then we set up a face detector; will use a haar cascade detector to

// find faces, followed by a keypoint-enhanced detector to find facial

// keypoints for our feature. There are many different combinations of

// features and detectors to choose from.

**final** HaarCascadeDetector detector = HaarCascadeDetector.BuiltInCascade.***frontalface\_alt2***.load();

**final** FKEFaceDetector kedetector = **new** FKEFaceDetector(detector);

// now we construct a feature extractor - this one will extract pixel

// patches around prominant facial keypoints (like the corners of the

// mouth, etc) and build them into a vector.

**final** Extractor extractor = **new** FacePatchFeature.Extractor();

// in order to compare the features we need a comparator. In this case,

// we'll use the Euclidean distance between the vectors:

**final** FaceFVComparator<FacePatchFeature, FloatFV> comparator = **new** FaceFVComparator<>(

FloatFVComparison.***EUCLIDEAN***);

// Now we can construct the FaceSimilarityEngine. It is capable of

// running the face detector on a pair of images, extracting the

// features and then comparing every pair of detected faces in the two

// images:

**final** FaceSimilarityEngine<KEDetectedFace, FacePatchFeature, FImage> engine = **new** FaceSimilarityEngine<>(

kedetector, extractor, comparator);

// we need to tell the engine to use our images:

engine.setQuery(image1, "image1");

engine.setTest(image2, "image2");

// and then to do its work of detecting, extracting and comparing

engine.performTest();

// finally, for this example, we're going to display the "best" matching

// faces in the two images. The following loop goes through the map of

// each face in the first image to all the faces in the second:

**for** (**final** Entry<String, Map<String, Double>> e : engine.getSimilarityDictionary().entrySet()) {

// this computes the matching face in the second image with the

// smallest distance:

**double** bestScore = Double.***MAX\_VALUE***;

String best = **null**;

**for** (**final** Entry<String, Double> matches : e.getValue().entrySet()) {

**if** (matches.getValue() < bestScore) {

bestScore = matches.getValue();

best = matches.getKey();

}

}

// and this composites the original two images together, and draws

// the matching pair of faces:

**final** FImage img = **new** FImage(image1.width + image2.width, Math.*max*(image1.height, image2.height));

img.drawImage(image1, 0, 0);

img.drawImage(image2, image1.width, 0);

img.drawShape(engine.getBoundingBoxes().get(e.getKey()), 1F);

**final** Rectangle r = engine.getBoundingBoxes().get(best);

r.translate(image1.width, 0);

img.drawShape(r, 1F);

// and finally displays the result

DisplayUtilities.*display*(img);

}

}

}

But this is not able to recognize the face accurately and is giving inaccurate results. So, we have searched for some other SDK which will provide much accurate results. We have found one, Java Machine Learning for Computer Vision which implemented Face Recognition as follows:

**package** ramo.klevis.ml.recogntion.face;

**import** lombok.extern.slf4j.Slf4j;

**import** org.bytedeco.javacpp.opencv\_core;

**import** org.bytedeco.javacpp.opencv\_imgcodecs;

**import** org.datavec.image.loader.NativeImageLoader;

**import** org.deeplearning4j.nn.graph.ComputationGraph;

**import** org.deeplearning4j.nn.graph.vertex.GraphVertex;

**import** org.deeplearning4j.nn.workspace.LayerWorkspaceMgr;

**import** org.nd4j.linalg.api.ndarray.INDArray;

**import** org.nd4j.linalg.factory.Nd4j;

**import** org.nd4j.linalg.indexing.NDArrayIndex;

**import** java.io.File;

**import** java.io.IOException;

**import** java.util.HashMap;

**import** java.util.Map;

*@Slf4j*

**public** **class** FaceRecognition {

**private** **static** **final** **double** ***THRESHOLD*** = 0.57;

**private** FaceNetSmallV2Model faceNetSmallV2Model;

**private** ComputationGraph computationGraph;

**private** **static** **final** NativeImageLoader ***LOADER*** = **new** NativeImageLoader(96, 96, 3);

**private** **final** HashMap<String, INDArray> memberEncodingsMap = **new** HashMap<>();

**private** INDArray transpose(INDArray indArray1) {

INDArray one = Nd4j.*create*(**new** **int**[]{1, 96, 96});

one.assign(indArray1.get(NDArrayIndex.*point*(0), NDArrayIndex.*point*(2)));

INDArray two = Nd4j.*create*(**new** **int**[]{1, 96, 96});

two.assign(indArray1.get(NDArrayIndex.*point*(0), NDArrayIndex.*point*(1)));

INDArray three = Nd4j.*create*(**new** **int**[]{1, 96, 96});

three.assign(indArray1.get(NDArrayIndex.*point*(0), NDArrayIndex.*point*(0)));

**return** Nd4j.*concat*(0, one, two, three).reshape(**new** **int**[]{1, 3, 96, 96});

}

**private** INDArray read(String pathname) **throws** IOException {

opencv\_core.Mat imread = opencv\_imgcodecs.*imread*(**new** File(pathname).getAbsolutePath(), 1);

INDArray indArray = ***LOADER***.asMatrix(imread);

**return** transpose(indArray);

}

**private** INDArray forwardPass(INDArray indArray) {

Map<String, INDArray> output = computationGraph.feedForward(indArray, **false**);

GraphVertex embeddings = computationGraph.getVertex("encodings");

INDArray dense = output.get("dense");

embeddings.setInputs(dense);

INDArray embeddingValues = embeddings.doForward(**false**, LayerWorkspaceMgr.*builder*().defaultNoWorkspace().build());

***log***.debug("dense = " + dense);

***log***.debug("encodingsValues = " + embeddingValues);

**return** embeddingValues;

}

**private** **double** distance(INDArray a, INDArray b) {

**return** a.distance2(b);

}

**public** **void** loadModel() **throws** Exception {

faceNetSmallV2Model = **new** FaceNetSmallV2Model();

computationGraph = faceNetSmallV2Model.init();

***log***.info(computationGraph.summary());

}

**public** **void** registerNewMember(String memberId, String imagePath) **throws** IOException {

INDArray read = read(imagePath);

memberEncodingsMap.put(memberId, forwardPass(*normalize*(read)));

}

**private** **static** INDArray normalize(INDArray read) {

**return** read.div(255.0);

}

**public** String whoIs(String imagePath) **throws** IOException {

INDArray read = read(imagePath);

INDArray encodings = forwardPass(*normalize*(read));

**double** minDistance = Double.***MAX\_VALUE***;

String foundUser = "";

**for** (Map.Entry<String, INDArray> entry : memberEncodingsMap.entrySet()) {

INDArray value = entry.getValue();

**double** distance = distance(value, encodings);

***log***.info("distance of " + entry.getKey() + " with " + **new** File(imagePath).getName() + " is " + distance);

**if** (distance < minDistance) {

minDistance = distance;

foundUser = entry.getKey();

}

}

**if** (minDistance > ***THRESHOLD***) {

foundUser = "Unknown user";

}

***log***.info(foundUser + " with distance " + minDistance);

**return** foundUser;

}

}

But it is also giving inaccurate results in some cases. Still tried with some other libraries like Open CV and others, but we are unable to find a perfect library for the Face Recognition.