WTP - IT Security

Android App for Detecting Morphed Passport Images

Nikhilkumar Italiya
Computer Science, DE
nikhilkumar.italiya@st.ovgu.de

Darshit Paresh Shah
Computer Science, DKE
darshit.shah@st.ovgu.de

Ranjiraj Rajendran Nair
Computer Science, DKE
ranjiraj.nair@st.ovgu.de

Endi Haxhiraj
Computer Science, DKE
endi.haxhiraj@st.ovgu.de

***Abstract*—Face Morphing Attacks (FMA) pose a serious threat and other concerns to current travel regulations security. In this piece of work, we demonstrate how a live image of the user face acquired at uncontrolled environment, can be used to restore the de-morphed image from the morphed image stored in the travel document. A series of steps carried out on a data set gives the benchmark for further analysis and results.**

Index Terms: De-morphing, Android Application, Face Morphing, Attack, Detection, Face Match, Bio-metrics, CNN, Deep Learning

I. Introduction

Biometrics is a branch in the computer science and forensics, where a typical characteristic trait of a person is exhibited by two broad modalities:
1. Behavioural
2. Physical

Although these traits are exhibited by birth or develop over a period of time, there is a possibility of an external threat which can pose serious threat not only to the individual but to the entire community. In this experimental work we focus in the area of physical modality, particularly with Face and the possible attacks caused due to morphing.

In a biological perspective since face is the largest feature of a human body this modality is exhibited by almost everyone. The reason being it is fast, requires no additional user assistance, and it's contact-less, thereby providing faster capturing. Face is such an discriminatory aspect in humans, that it can reveal the person's gender as the basic feature. In addition to this it can convey derived features such as emotion, age, and health conditions (including skin diseases). Moreover, different ethnic group and cultural practices uncover many details of an individual in the society.

With the growth of technology and the increasing number of devices there also is an increase in security risks on these different assets possessed by each individual. Perhaps, there exits top security in travel documents, however, under some conditions the security becomes compromised and vulnerable to attack and face modality it's not an exception. A person who willfully performs an activity illegally may morph the face of a legitimate user with his/her face and can spoof the system, getting the rights that the legitimate person has. In the remaining section of our report we prove how morphing and successful de-morphing is a challenge considering international travel security.

Fig. 1. Face Morph example, Kramer [3]

II. Concept

The application follows a generic workflow as described below.

For developing the Android application to detect potentially morphed passport images, we started by portraying a layout for the application followed by implementing the features one by one. The steps are as follows:

- Read the travel document data (image in our case) using NFC
- Capture a live image with device camera
- Perform a face match
- If necessary perform De-morphing process
- Visualization and Evaluation of Results

Basically our approach scans the electronic passport through Near Field Communication (NFC) and collects the relevant information of a person with a photograph of the facial image. Then a live image is captured of that person and is then matched with image extracted from the passport. If the matching score is less then the predefined threshold then the process stops as clearly the persons being compared are not the same so the assistance of a human agent is required. If the matching score is above the predefined threshold, then the next step is performed.

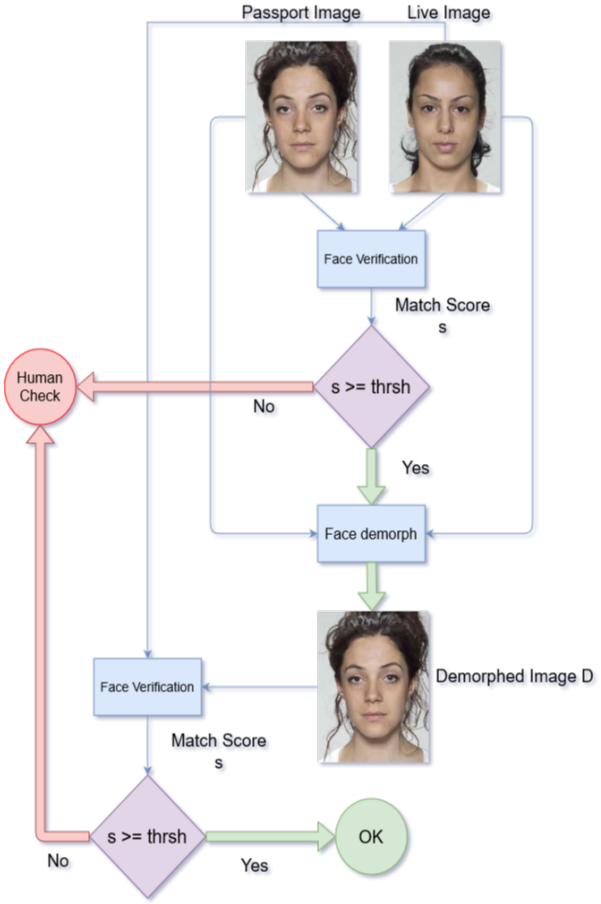

Fig. 2. Schema for face verification we use in our application, based on the work from Ferrara [2]

A demorphing process will take place if the first step of face matching is successful. In this step we perform a demorphing of the travel document image, with the live image as a model. The pseudo-code is as follows:

```
BEGIN
INPUT passport_image, live_image
score = face_match(passport_image,
live_image)
IF (score >= threshold_1) THEN
    result = demorph(passport_image,
                     live_image)
    final_score = face_match(result,
                             live_image)
    IF (final_score >= threshold_2) THEN
        OUTPUT: 'WARNING'
    ELSE
    OUTPUT: 'OK'
    ENDIF
ELSE
    OUTPUT: 'WARNING'
ENDIF
END
```

III. Implementation

In this section we explain in more detail every step of our app implementation

A. NFC Reading

For the NFC reading we make use of the Machine-Readable Travel Document (MRTD) Reader SDK. We used some of the code from Passport Reader Sample Code by AppliedRecognitation. It uses Ver-ID Person SDK which detects the face from travel document and lets Android applications read structured information in biometric travel documents (passports). In addition to the document and document holder information most biometric travel documents contain an image of the holder, which is what we need to perform the face verification in our application.

Fig. 3. Travel Document to be Scanned Example

In order to perform a successful read of travel documents from our device we need to specify some Basic Access Control (BAC) variables in our application. These variables typically are **document number, date of birth**, and **date of expiry**, and of course the device must be NFC enabled to perform a successful scan. If NFC is disabled then the application will directly prompt the user to turn it on but in some cases user might need to enable NFC explicitly.

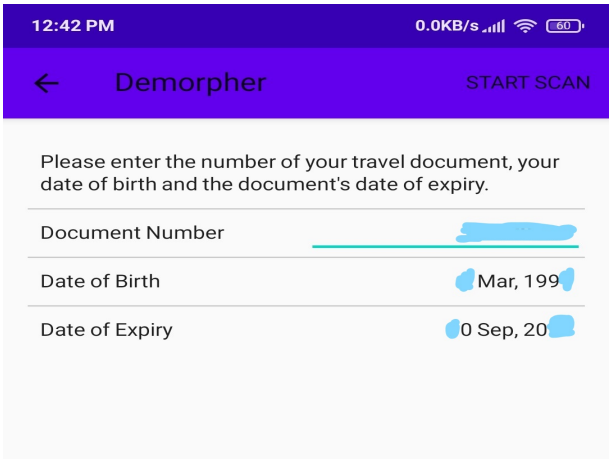

Fig. 4. NFC Prerequisite Information to Start Scan

After the users enters this information, the user scans the travel document with its device and saves the image and the other data located in the travel document. The saved images do not appear on the Gallery app of the phone or by searching it in the File Manager. However the images can be accessed if user knows location of the folder where it is being saved. So it provides moderately secure environment.

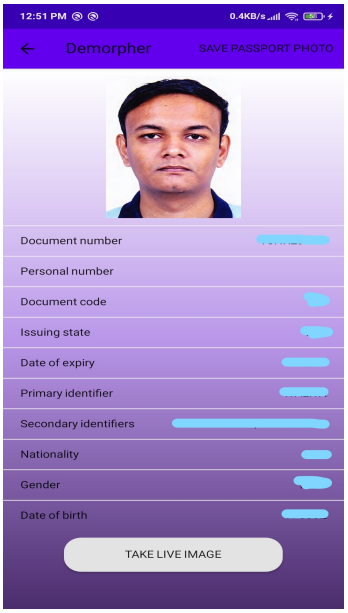

Fig. 5. NFC Scan Results

One security concern that may arise now is that the information scanned is saved in the device until another scan is performed. Then the previous information is overwritten with the new one and so on. This can be solved by including a button to erase the information scanned, or by removing them automatically if the device has been idle for a specific amount of time. We did not implement this feature as we needed the data for testing purposes.

Note: For testing purposes, both NFC reading page and live image capture page have an Import image function that makes it easier to feed the app with the required images

B. Live Camera Capture

For the live capture of the face we make use of the CameraX Library. CameraX is an addition that makes it easier to add camera capabilities to android apps. The library provides a number of compatibility fixes and workarounds to help make the developer experience consistent across many devices.

The app will ask for permissions the first time in order to access the device camera for capturing the image, and storage permission in order to store the captured image locally. If permissions are not granted the camera capture cannot start so the acceptance of permissions is a requirement.

We have implemented some guidelines when the user wants to capture a live image in order to make it easier to use. The user can decide to recapture the image if it is not up to standard or something else might have interfered. There also exists the capability to switch for a frontal camera so the subjects might take their own photos but this is not a good idea in practice as they can actively try to sabotage the process.

Fig. 6. Camera Capture with Guidelines

Again, as with the NFC scanning part, one security concern that may arise is that the information captured is saved in the device until another capture is performed. Then the previous information is overwritten with the new one and so on. This can be solved by including a button to erase the information scanned, or by removing them automatically if the device has

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1×1	-	1280	1	1
$7^2 \times 1280$	avgpool 7×7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1×1	-	k		

TABLE 1
MOBILEFACENETV2 ARCHITECTURE

been idle for a specific amount of time. We did not implement this feature as we needed the data for testing purposes.

C. Face Match

For effective face matching of the passport image and the photo taken of the person by camera, we planned to use A Convolution Neural Network Architecture. As we were using smartphone as a device, training a Neural Network with around a million parameters would require more GPU power and time. Thereby, we decided to use a pre-trained CNN model. We used MobileFaceNet from Sheng [1], a pre-trained model on the Insightface Dataset Zoo.

The Architecture is described in Table 1. The described architecture used from MobileNetV2 in the provided pre-trained model. Each row represents layers repeated n times. c is the number of output channels in each layer. with expansion factor t applied to input size, the architecture uses 3×3 kernel for every convolutions. Strides s used in each layer is also described. $bottleneck$ is a residual block containing 1×1 conv2d, 3×3 depth wise and 1×1 linear conv2d. from Sandler [4] The model was trained using tensorflow and then converted into a small sized .tflite file to run on smartphones. The tensorflow lite library was used for this purpose. The model takes two images of faces as input and gives the output value between 0 to 1 representing similarity between the provided faces.

The MobileFaceNet architecture gives promising results, if the given input of faces are aligned properly. While the image from passport following standard guidelines, the alignment can be proper. But the image taken from camera might not be in proper alignment. To counter this issue, we added guidelines that overlays on the screen while camera is opened. Apart from this, to cope with human error we implemented MTCNN from Zhang [5] to detect the faces and align them properly. The MTCNN uses 3 CNN architectures to detect the face from image and applies 5 landmarks on face. It takes two images as input and returns bounding boxes which is then processed and two images with alligned face are generated to provide as an input to MobileFaceNet model.

Face Matching gives around 98.58 % accurate results. [1] It is not required to process the images for demoprphing, if the face matching module gives lower similarity value. We added a threshold value of 85 %. If the face matching result gives similarity value more than this threshold, we process the images for demorphing.

The entire face matching module runs offline and gives result in less than 3 seconds.

D. Demorphing

For the demorphing part, we first try to implement the idea presented by Ferrara [2], however seeing as we had limited time we were not successful to implement a working example derived from his paper. That left us with the option of using a working preexisting webservice that does the demorphing process for us. The webservice was given to us, courtesy of Advanced Multimedia and Security Lab group at Otto-von-Guericke University, Magdeburg.

The way the webservice works is that the user posts a request with the URL of the websevice, a JSON object which containts two images, the live, genuine image captured by the camera and the image subject to the morphing process, usually taken from the travel document, and some header giving some extra information to the webservice, which are of no interest to us.
One remark is that the images sent through the webservice need to be encoded as a base64 binary data, as the webservice only reliably suports text contect.
In the case of Java implementation we must first convert the image into a ByteArray, then convert it into a base64 binary data. The response from the webservice would again send among the evaluation result and score, the resulting image as base64 binary data which we need to decode in order to visualize the result.
We were getting a lot of bad response messages at first because we were only using libraries provided by Android Studio itself, which apparently does not handle JSON data very well. After a lot of trial and error we found a solution by using a mix of **OkHttpClient, Retrofit, and UserService** libraries in order to correctly send the POST request with the JSON object data.
Unfortunately we do not possess any information regarding the process of the demporphing done by the webservice, as we have no access to the source code behind it, so we cannot provide any further information besides that it needs a base64 data inside a JSON object, and a connection with the Otto-von-Guericke University network in order to work.

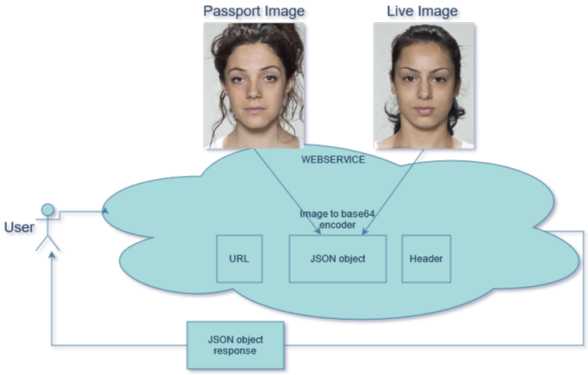

Fig. 7. Webservice Diagram

IV. Evaluation

From the tests we performed regarding the performance of the app, we got different results. For the tests we used the ANANAS dataset provided to us by Otto-von-Guericke University, and our own images.

Firstly, as expected, for some images the app performed really well, where it would detect morphed images even though on first impression the pictures looked very similar, and with True Positives it never gave any problems.

In some cases, the morphed image would not pass our offline face match step process, so the demorph process was not required.

Something else we noticed is that if the person being face matched does not use a relaxed face in their live image capture (or a smiling face if they are smiling in their travel document) then the face match process might throw a false negative match.

Another interesting observation is that sometimes the system itself will be deceived into thinking the images are the same, and the noMorph message will appear, even though looking at the deMorphed image one can clearly see there are very noticeable differences in the faces presented.

Finally, the demorphing process counts wearables as part of body and not as accessories, so if for instance one person is wearing glasses in one image and not in the other one, the demorph process will count it as a morphed image.

Even though we performed a lot of tests, and in most cases the app performed quite well (i.e it detected most morphed images) we cannot give a final verdict as we do not know the parameters that the demorpher webservice uses, we can only assume.

Lastly, in most cases the travel document image is taken with a high quality camera, and lens that are specialized for portrait photos. For this reason we recommend that if the app is to be used for face authentication the live capture need to be done done with a similar camera and lens attached to the device, rather than using the camera of the device itself. This is for the sole purpose of having both images being taken in as similar environment as possible.

A. True Negative and True Positive

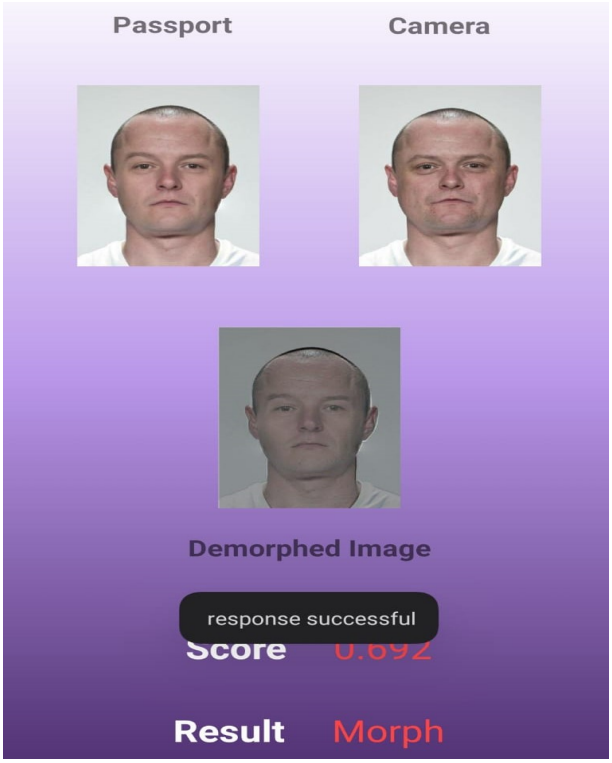

Fig. 8. Morphed Image 1

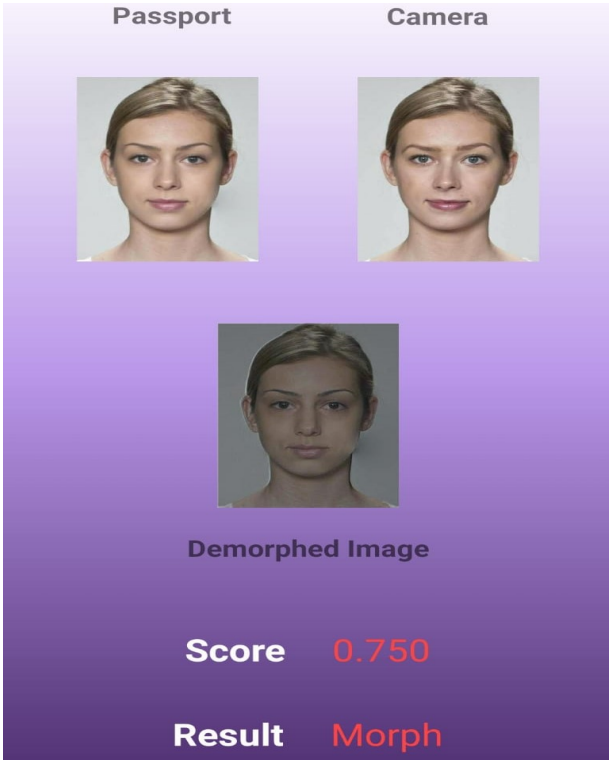

Fig. 9. Morphed Image 2

Fig. 10. No Morph Image

Fig. 11. No Match

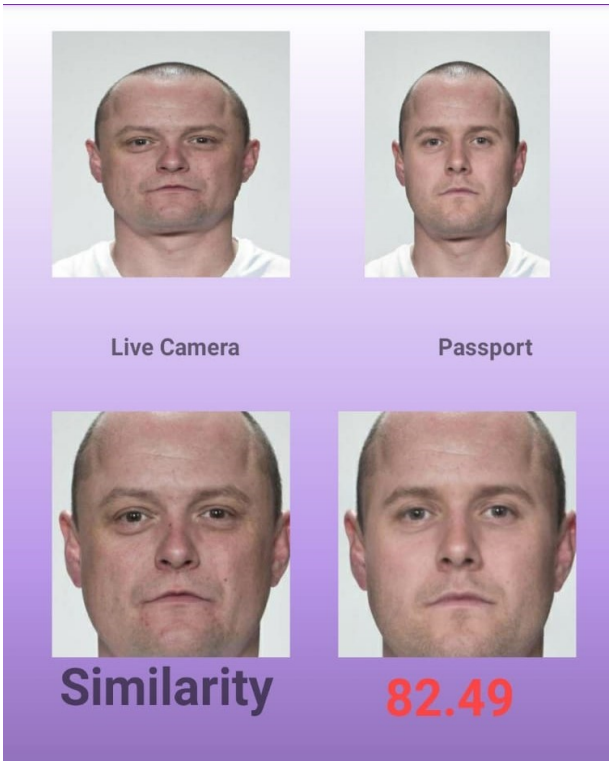

Fig. 12. No Match 2

B. No Match

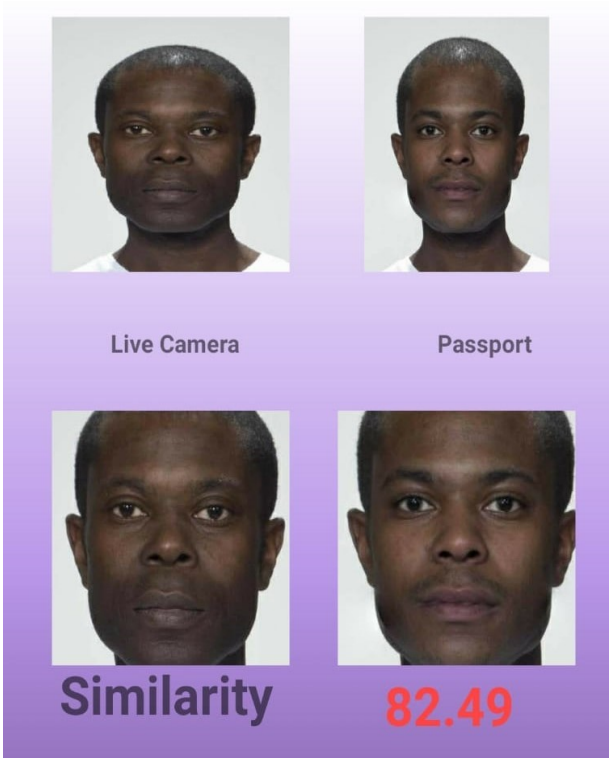

Fig. 13. No Match 3

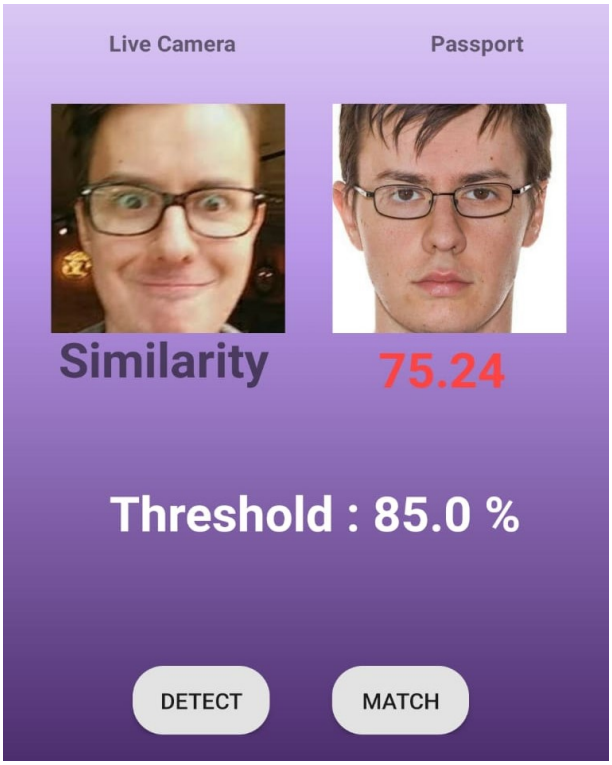

Fig. 14. False Negative Match

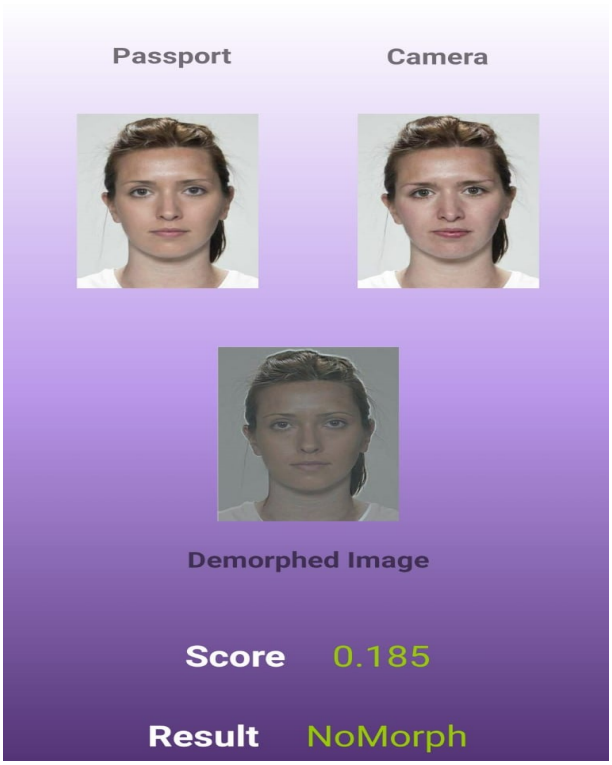

Fig. 16. False Positive Match 2

C. *False Positive*

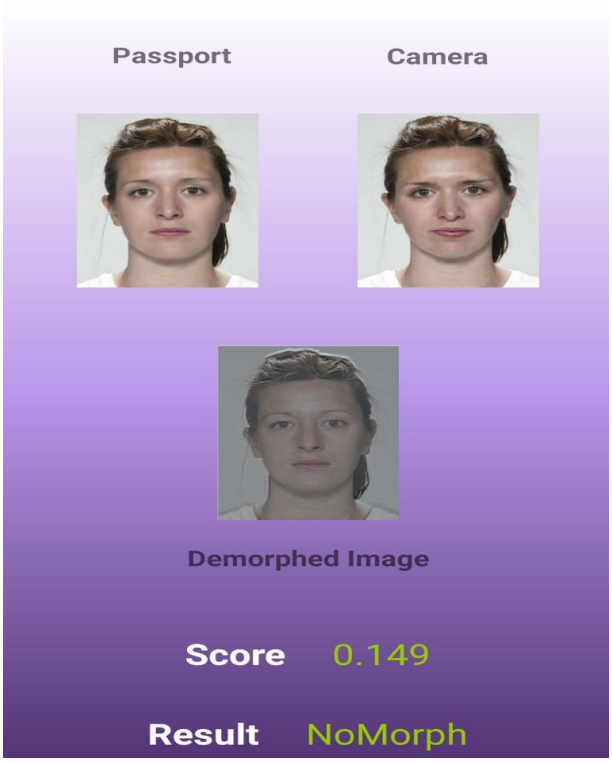

Fig. 15. False Positive Match 1

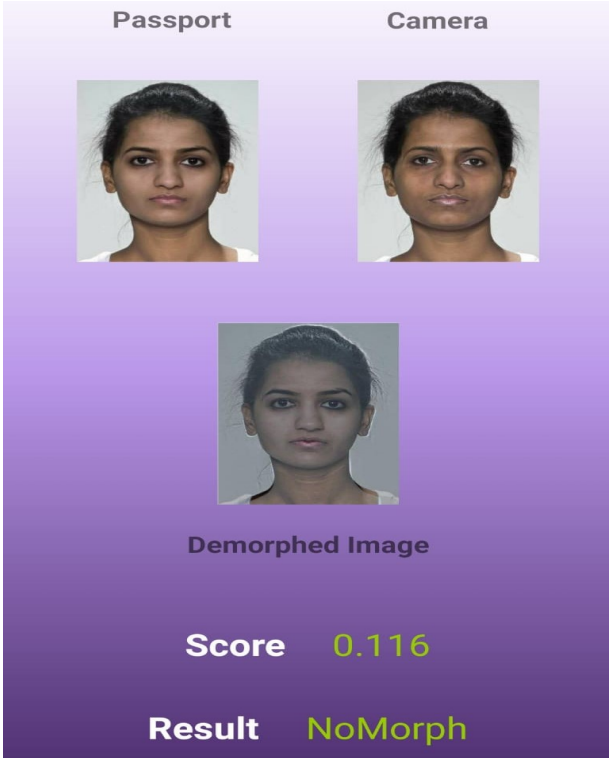

Fig. 17. False Positive Match 3

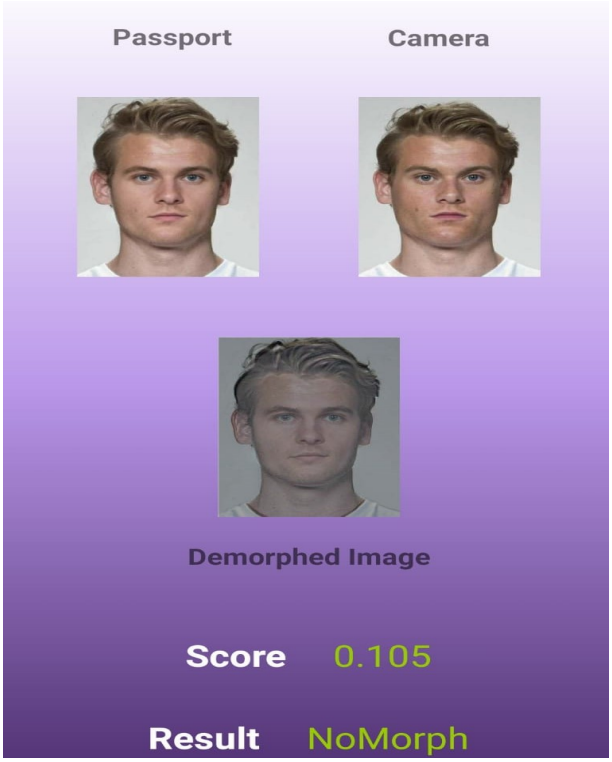

Fig. 18. False Positive Match 4

D. Wearable

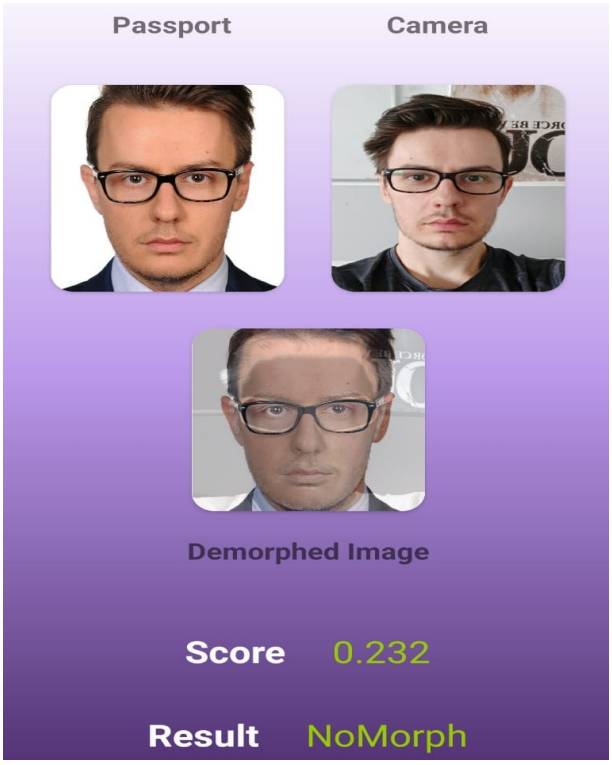

Fig. 19. With Glasses

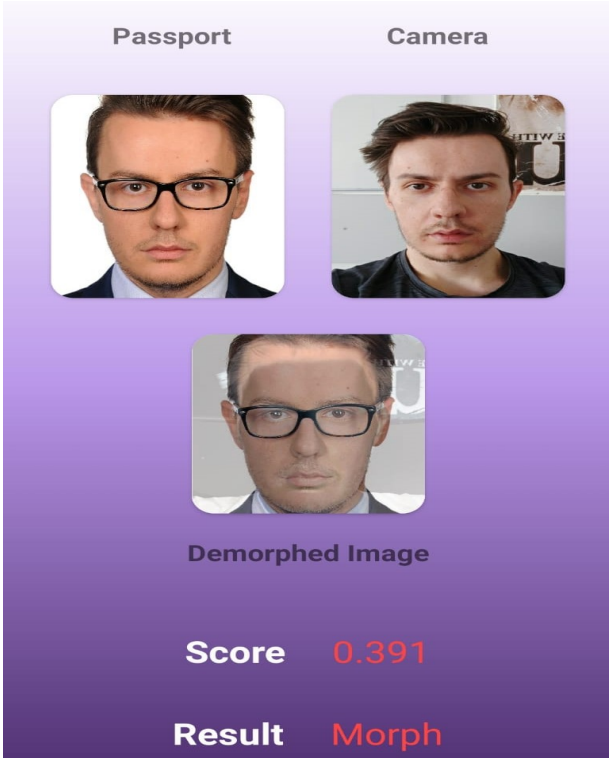

Fig. 20. Without Glasses

V. Conclusion

Biometrics is a branch in computer science and forensics where a typical characteristic trait of a person is exhibited by different modalities. Possibilities exist where external agents can pose serious threat to the individual. In our work we focused on the Face modality and the possible risks caused due to morphing attacks.

Face is such an discriminatory aspect in humans, that it can reveal a lot of information without much analysis, that is why a successful attack towards face modality is a dangerous thing. With the growth of technology and the increasing number of devices there also is an increase in security risks on these different assets possessed by each individual.

In this report we showed how a morph attack can be successful even though preventing measures can be taken. For the time being these preventive measures are not always effective,especially under some conditions the security becomes compromised and vulnerable to face modality attacks. A person who willfully performs an activity illegally may morph the face of a legitimate user with his/her face and can spoof the system, getting the rights that the legitimate person has.

VI. Source Code and other Links

GitHub repository APK Download

REFERENCES

[1] Sheng Chen, Yang Liu, Xiang Gao, and Zhen Han. Mobilefacenets: Efficient cnns for accurate real-time face verification on mobile devices. *CoRR*, abs/1804.07573, 2018.

[2] Matteo Ferrara, Annalisa Franco, and Davide Maltoni. Face Demorphing.

[3] Robin S. S. Kramer, Michael O. Mireku, Tessa R. Flack, and Kay L. Ritchie. Face morphing attacks: Investigating detection with humans and computers.

[4] Menglong Zhu Andrey Zhmoginov Liang-Chieh Chen Mark Sandler, Andrew Howard. MobileNetV2: Inverted Residuals and Linear Bottlenecks.

[5] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao. Joint face detection and alignment using multitask cascaded convolutional networks. *IEEE Signal Processing Letters*, 23(10):1499–1503, Oct 2016.