Implementation of Human Face and Spoofing Detection Using Deep Learning on Embedded Hardware

Saankhya Mondal
Department of Electronics and Communication Engineering,
Visvesvaraya National Institute of Technology,
Nagpur, India
saankhya1997@gmail.com

Abstract— **Human face and spoofing detection are of prime importance in many security verification and law enforcement applications. This paper deals with human face detection and spoofing detection using deep learning. Essential feature extraction from images is the key to achieving considerable accuracy for classification and detection. Convolutional Neural Networks (CNN) are well-equipped to extract vital features on its own from images without the need to manually select and extract features from them. In this paper, a CNN based real human face detection classifier has been proposed. The CNN classifier was implemented on an embedded system device for real-time detection. It accurately predicts whether or not a real human face is present in the frame of the recognizing/detecting camera.**

Keywords— Deep Learning, Computer Vision, Image Classification, Face Detection, Face Spoofing.

I. Introduction

In recent years, biometric identification has played a key role in security verification and access control. Some popular biometric systems include fingerprint and face recognition. Face recognition systems are in use in various organizations such as banks for customer log in, industries/company for employee login, in institutes for student attendance as well as locks in smartphones.

Face recognition systems [1], [2] can easily be fooled by an interloper who can use photographs of a person to log in successfully. In the facial recognition system, many a time it may happen that a person might easily log in to another person's account by wearing a mask that looks very real as compared to the face of the second person. The system would then wrongly predict that the second person logged in, thus allowing the intruder to get access to the second person's account or data. In an era, where computers are well equipped to generate images that look very realistic in nature as well as developments in the field of 3D printing, any intruder can easily develop a 3D mask causing problems for other users. To prevent the loss of privacy, a real human face detection system has to be in place. The importance of such type of system is that it ensures no intruder would be able to access any other person's account or phone by any means and their information and data remains protected. This system should accurately classify between a real human face and a face mask. Whenever someone tries to log in through the face detection system wearing a 3D realistic mask, the intruder would be blocked. The real-time face detection system will work as a security key to further recognize faces.

Deep learning can be applied to various computer vision applications such as image classification, object detection as well as image segmentation. Some trademark CNN architectures such as VGGNet [3], AlexNet, and MobileNet that have been trained on huge datasets have achieved formidable accuracy in classifying images into multiple classes. It is desirable to classify two classes here. In this paper a binary classification approach using CNN has been introduced for real human face detection task. A dataset containing more than 15k images was prepared that was used to train and test the CNN. Real-time detection using this model have been performed on a common embedded system device called Raspberry Pi.

The rest of the paper is organized as follows: Section II discusses about some related work. Section III provides the glimpse on the basics of CNN, Section IV, V and VI describes the work in detail, Section VII discusses the results obtained and Section VIII concludes the work.

II. Related Work

There have been solutions developed in the past to tackle the issue of spoofing. Face liveness detection systems as described in [4], [5] and spoof detection system [6] are good examples. Many traditional face detectors like Viola Jones and Haar cascades are available readily for use in various computer vision applications. However, they detect only faces and fail to detect spoofs. CNN cascade classifier [7] can also be used. Previously, many approaches have been made to solve the face spoofing problems such as appearance-based approach or motion-based approach [8], [9], [10]. Appearance-based approach requires only one frame at a time for detection. Motion-based approaches like liveness detection needs a series of frames to capitalize on movements such as eye blink or lip movement for detecting spoofing.

Multiscale Local Binary Pattern (LBP) approach for face spoofing detection has been mentioned in [11]. G. D. Simanjuntak et al. [12] proposed a technique based on color distortion. Principal Component Analysis was performed on the obtained feature vector to reduce dimensionality. P. K. Das et al. [13] combined handcrafted features (using LBP analysis) and deep features into so-called hybrid features to propose an approach to tackle this issue. The deep features were extracted by the VGG-16 pre-trained network which is a very huge CNN.

The use of deep learning facilitates extraction of important features from images without the need of handpicking them. Another anti-face spoofing system described in [14] by H. S. Lin and Y. Su also uses CNN-based mechanism and makes use of both RGB and HSV color spaces as input. The input to this CNN is a 256×256×6 3-D vector. H. Chen et al. [15] proposed a two-stream convolutional neural network which in addition to RGB color space also uses the multi-scale retinex (MSR) space. W. Sen et al. [16] also provides an approach using CNN.

The CNN model proposed considers only the RGB color space. The input image is of size 420×300×3 which has been chosen keeping in mind the aspect ratio of a human face that is around 1.5. The proposed CNN was evaluated on a dataset of 6.6k images created from video samples of the HKBU-MARs V1+ database that was introduced by S. Q. Liu et al. [17] and later extended in [18]. The main advantage is that the proposed CNN model is very light-weight and it is easier and faster to load and run. It is ideally designed for real-time detection in embedded hardware. Since, embedded system devices are designed for a specific application rather than for multiple applications, they have limited processing capabilities. It is difficult to deploy a large CNN model like VGGNet on such devices. Hence, there is a need for a light-weight CNN model which has been proposed in this paper.

III. Convolutional Neural Network

Artificial Neural networks (ANN) are models that imitate the human brain. It consists of an input layer followed by a series of hidden layers and an output layer. Each layer consists of small units called neurons shaped in the form of a 1D array. Each neuron in a layer is connected to all the neurons in the adjacent layers. Each connection is provided a weight which adapts itself while training. Each neuron takes the weighted sum of all the neurons in the previous layer and applies an activation function over it and feeds it forward. Deep learning is a class of machine learning which is inspired by the working of the human brain and the methods by which it processes data and uses it for various tasks.

A CNN follows the same layout as that of an ANN with the exception that the input is a 2D or a 3D array and each neuron is replaced by a number of 2D filters of fixed kernel size. The 2D filters in each layer are stacked together to obtain a 3D array. Generally, the number of filters (the third dimension), also called 'feature maps' in a convolutional layer are increased as we go deeper into the network while the other two dimensions of the layer are decreased. To reduce the dimensions, max pooling is performed. So, a CNN may consist of a series of convolutional and max pooling layers. This is continued until a layer of fairly small dimensions is obtained. This layer is now reshaped into a 1D array which is called 'feature vector'. This is followed by hidden layers and output layer similar to the ones used in ANN.

The filters contain weights which are learnt during training. Instead of performing a multiplication operation while taking the weighted sum, a 2D convolution operation is performed here. Convolution is performed between the layer and the filter, by shifting the filter across the layer. An activation function is applied on the obtained values which are fed to the next layer. This is continued till the output layer. At the end of one forward cycle, the difference between the obtained/predicted value/label and the correct value/label is computed and formulated into a loss function. Often the loss function is calculated over a batch of input sample. However, it can be calculated over a single input sample or over the whole dataset. The main aim is to minimize this loss function. Gradient of the loss function is computed with respect to the weights. Weights are updated in the backward direction until we reach the input layer. This process is repeated until the error stops decreasing and we obtain optimal weights.

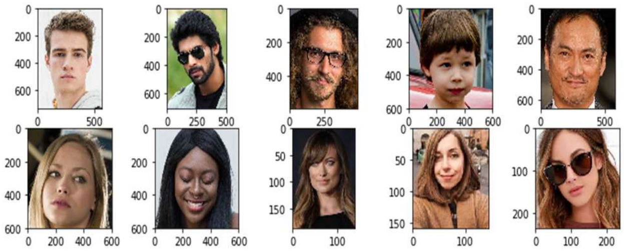


Fig. 1 Training samples of 'Face' class

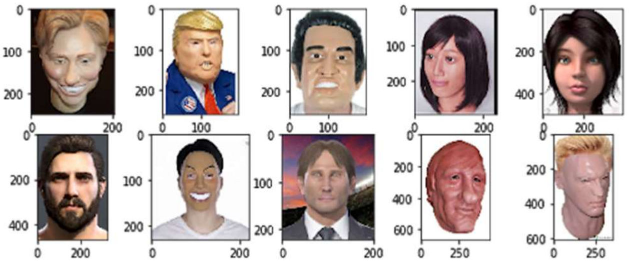


Fig. 2 Training samples of 'No Face' class

TABLE I. DATASET STATISTICS FOR BOTH THE CLASSES

	Training	Validation	Test	Total
Face	6278	1221	511	8010
No Face	6068	1144	450	7662
Total	12346	2365	961	15672

IV. Dataset

A. Dataset design

A dataset of 15.6k images has been prepared. The dataset consists of two classes- 'Face' and 'No Face'. The 'Face' class consists of 8k images of human faces in various poses (faces at different angles), of different age groups, different ethnicities, different hairstyle (both men and women), with glasses or not, with a beard and/or moustache or not (for men), with head-wear or not. Fig. 1 shows some training samples of the 'Face' class. In addition to the above, 7.6k images have been collected that don't contain human faces i.e. 'No Face' class. A major portion of the 'No Face' class includes images of 3D realistic human face mask (made up of either latex or silicone) and video game character faces (computer graphics generated human faces). Some monkey faces, cartoon faces, partially and completely covered faces of human and human caricature faces were also included. Some training examples of this class are illustrated in Fig. 2. All the images in this dataset have varied sizes. The dataset was divided into three different parts for training, validation, and testing. The dataset statistic for both the classes is shown in Table I. It was ensured that the training and validation sets were prepared from the same distribution while the training and testing sets were mutually exclusive.

B. Data pre-processing

The CNN accepts images of a fixed size. All the images were first reshaped to a fixed size. To improve the accuracy and enhance the speed of training, all the pixel values in each image were divided by 255 and the mean and standard

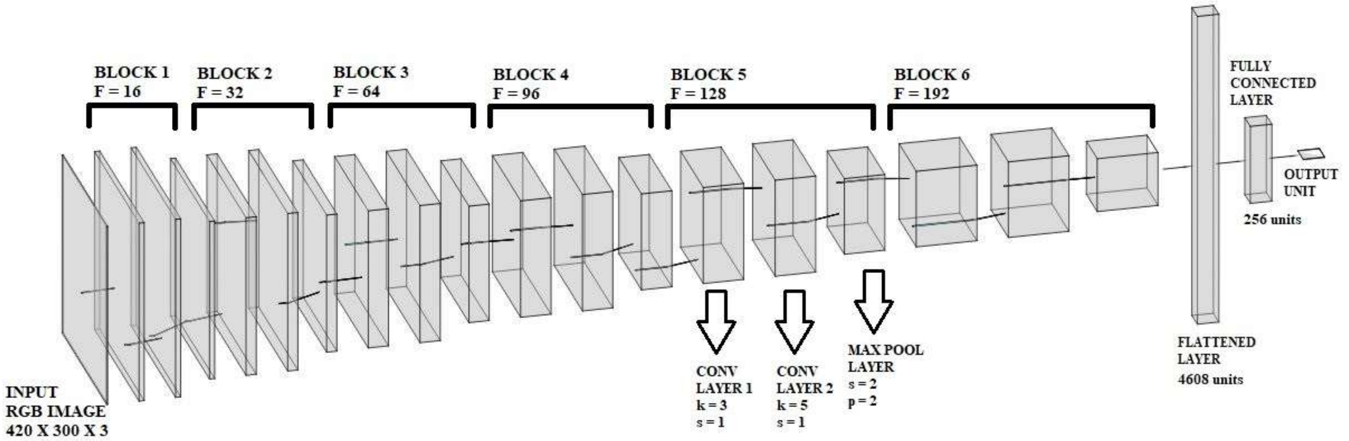


Fig. 3 Layout of the proposed CNN Architecture [F=number of filters, k=kernel size, s=strides, p=pool size]

deviation of all the pixel values in the three channels across all training images were computed. Mean normalization was then performed for each pixel of each input image.

To increase the size of the dataset, some data augmentation techniques were applied. Images were randomly rotated by a random angle of value below 30 degree to emulate tilting of face in real life. Images were horizontally flipped (mirrored). Brightness or intensity (with a scaling factor selected randomly in the range 0.5 to 1.2) of images were randomly modified. Augmentation of images in general brings robustness because it introduces the CNN to a variety of frames/images during training that would help it evaluate properly in real life scenarios.

TABLE II. ONE BLOCK OF LAYERS AND THE HYPER-PARAMETERS KERNEL SIZE(K), STRIDES(S), PADDING AND POOL SIZE(P) USED IN EACH LAYER

Layer Type	Hyper-parameters
Convolutional	k=3, s=1, padding='same'
ReLU	
Batch Norm	
Convolutional	k=5, s=1, padding='same'
ReLU	
Batch Norm	
Max Pooling	s=2, p=2

TABLE III. CNN ARCHITECTURE

Input – 420 × 300 × 3 RGB Image		
Block/ Layer	**Number of filters**	**Output shape**
Block 1	16	210×150×16
Block 2	32	105×75×32
Block 3	64	52×37×64
Block 4	96	26×18×96
Block 5	128	13×9×128
Block 6	192	6×4×192
Flatten	-	4608
Fully Connected layer	-	256
Output Unit – 1		

V. Proposed CNN Architecture

In this section, the architecture of the proposed CNN model has been described which was trained on the dataset described in the last section. The layout of this CNN model is shown in Fig. 3 as well as in Table III. The network accepts an RGB input image of a fixed size of 420×300×3. The layer configuration of a single block is outlined in Table II.

A single block consists of a stack of two convolutional layers having the same number of filters followed by a max-pooling layer. The kernel for all the filters is of size 3×3 and 5×5 in alternate layers. Before performing convolution with the input in each layer, zeros padding has been done to the input so as to keep the dimension of output same as that of the input to the layer. Also, the filters are shifted by a stride of 1 in both the direction. Batch normalization has been applied after each convolutional and the fully connected layer. The input image passes through 6 blocks of convolutional and max-pooling layers before being flattened and fed to a simple classifier consisting of a hidden layer with 256 neurons (also known as a fully connected layer or a dense layer) followed by an output layer of a single unit. In the proposed CNN layout shown in Fig. 3, the number of filters increases from 16 in the first block to 192 in the last block. In other words, the number of feature maps have been increased in the network block by block. The output of the last block was reshaped into a 1-dimensional vector to obtain a feature vector that is fed to the fully connected layer.

Overall, the CNN consists of 21 layers including the output layer. There are close to 3.3 million parameters in the network, the majority of which are in between the flattened layer and the fully connected layer. This CNN model is lightweight and thus is ideal for implementation in embedded or mobile devices. As compared to VGGNet weights that takes up more than 500 MB space and takes time to load, this CNN consumes 35 MB space only.

VI. Training

In the previous section, the CNN architecture has been described. In this section, we will discuss about training and testing/evaluating the CNN on the designed dataset. Deep learning frameworks Keras and Tensorflow have been used for training the CNN. Python's image processing library

OpenCV was used for pre-processing the samples as described in the Section IV.

Weight initialization plays an important role in obtaining a better model. All the weights have been initialized using ‘He’ initialization which is a little modification to the ‘Xavier’ initialization commonly used. The ReLU activation function has been applied for each layer in the network except the output unit where the Sigmoid activation function has been used.

To prevent the model from overfitting, spatial dropout layers have been added at the end of the last three blocks. 20% of the feature maps/filters selected randomly in each training cycle (step) were set to zero (spatial dropout). This is done to make the network less sensitive to the specific weights. This L2 regularization is another technique that is used to overcome overfitting in various machine learning algorithms. L2 regularization has been used with the regularization parameter equal to 0.001 for the weights between flattened and hidden layers. Dropout parameter was set to 0.25. CNN training was done using the batch gradient descent technique. Batches of 32 samples have been used for training the CNN. The training has been done for 30 epochs (386 steps in each epoch). After every epoch, the dataset was shuffled randomly. At the end of forward propagation, we have calculated the binary cross-entropy loss. For minimizing this loss, various optimization algorithms are available. Here, the Adam optimization technique with an initial learning rate of 0.001 was used. Default values of other parameters β1, β2 and ε were taken as mentioned in [19]. As training progressed, the learning rate was made to decay by a factor of 0.2 whenever the validation loss didn’t reduce for three consecutive epochs.

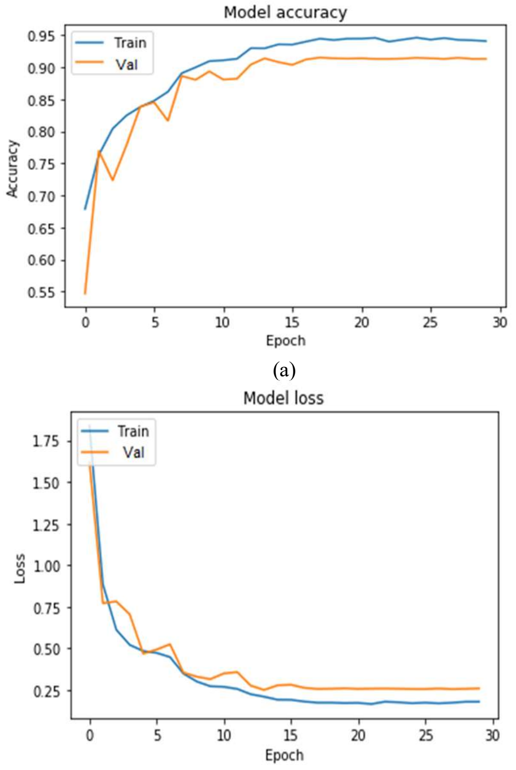
The CNN was trained on Tesla K80 GPU provided by Google’s Cloud Compute Engine. To use the cloud GPU, the dataset was uploaded on Google Drive. The Python code for training and testing the CNN was compiled and run on Google Colaboratory notebook.

VII. Results

In this section, we will first discuss about the outcomes of training the model and also about the various experiments that were performed. Next, the results of testing the trained weights of the CNN model on images that the network had not come across while training will be shown. Also, the results of real-time evaluation using webcam interfaced with an embedded device are provided.

A. *Training results and experiments*

Before arriving at the proposed CNN architecture, various architectures with a smaller number of layers and hyper parameters were tried and trained. After increasing the depth of the CNN, tuning hyper parameters, performing error analysis and adding regularization, we obtained a model whose validation loss was consistently closer to the training loss. After training for 30 epochs, we obtained a training accuracy of 95% and a validation accuracy of 93%. The variation in training and validation accuracy and loss with each epoch has been shown in Figs. 4 (a) and 4 (b) respectively. It was observed that after 20th epoch, there was no substantial change in the value of validation loss as well as the accuracy. The above-mentioned validation accuracy was obtained at the 22nd epoch. The model was trained till the 30th epoch but the model weights obtained after the 22nd epoch was selected. This was because the validation loss obtained here was the least.



(a)

(b)

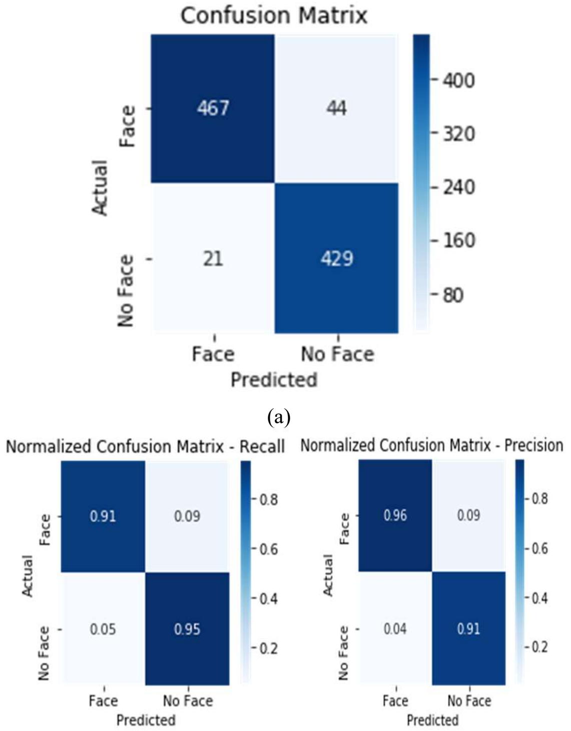
Fig. 4 CNN Training and Validation (a) accuracy and (b) loss

The effect of batch normalization was also observed. All the batch norm layers were removed and then the network was trained again. The model quickly started overfitting and reached an accuracy of 97% on the training set within a few epochs but it could only return an accuracy of 71% on the validation set. Difference in the performance with change in the number of neurons in the fully connected layer was also experimented. The CNN was trained with 256, 512 and 1024 neurons. There were no substantial differences in model performance. So, it was decided to keep 256 neurons in the fully connected layer.

B. *Testing*

After the CNN model was trained, it was evaluated on the test set. The ‘No Face’ class in the test set mainly consists of 3D realistic human masks and computer graphics generated human faces which are difficult to classify. An accuracy of 93.2% was obtained here. In addition to accuracy, the model was evaluated by calculating several other performance metrics such as precision, recall, and F1-score based on the test set confusion matrix shown in Fig. 5 (a). Some test samples and their obtained model probabilities (accuracy in scale of 1) along with their predicted labels have been illustrated in Fig. 7 and Fig. 8.

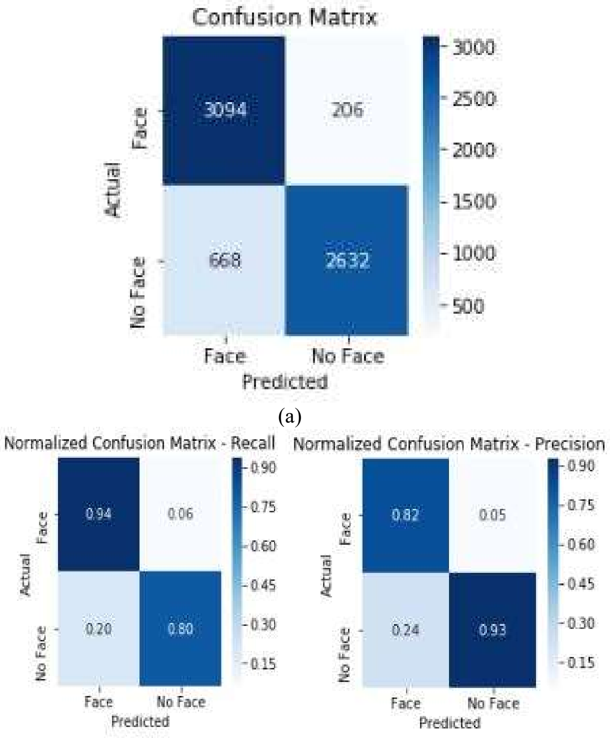
Considering ‘No Face’ as the relevant class, we obtained a precision of 0.91 and recall of 0.95. F1-score is a combined metric that is equal to the harmonic mean of precision and recall. An F1-score of 0.93 was obtained on the test set. Normalized confusion matrices based on precision and recall have been shown in Fig. 5 (b) and Fig. 5 (c) respectively.



(a)

(b) (c)

Fig. 5 (a) Confusion matrix (b) Normalized confusion matrix taking Recall as the metric (c) Normalized confusion matrix taking Precision as the metric



(a)

(b) (c)

Fig. 6 (a) Confusion matrix (b) Normalized confusion matrix taking Recall as the metric (c) Normalized confusion matrix taking Precision as the metric for the HKBU-MARS-V1+ dataset

TABLE IV. TIME TAKEN FOR EVALUATING A FRAME ON DIFFERENT PROCESSORS

Processor	Time taken (in seconds)
Intel Core i7-6500 CPU	0.18149
ARM Cortex A53	1.02341

C. Testing on the HKBU-MARs V1+ database

The HKBU 3D Mask Attack with Real World Variations Database (HKBU-MARs) is a 3D mask anti-spoofing database created keeping in mind all the variations expected in real-world application. It consists of 10 seconds video (30 frames per second) of 24 subjects, 12 of which contain real human face and the other 12 contains humans wearing 3D masks. 22 video samples have been used and properly cropped 6600 images were generated from them. Thus, the created dataset has 3300 images of real human face and 3300 images of humans wearing 3D masks. An accuracy of 86.75% was obtained on this test set. The confusion matrix and other metrics have been calculated and shown in Fig. 6. The recall, precision, and F1-score obtained for this test set considering 'No Face' as the relevant class are 0.8, 0.93 and 0.86 respectively. Some test samples have been illustrated in Fig. 9 and Fig. 10.

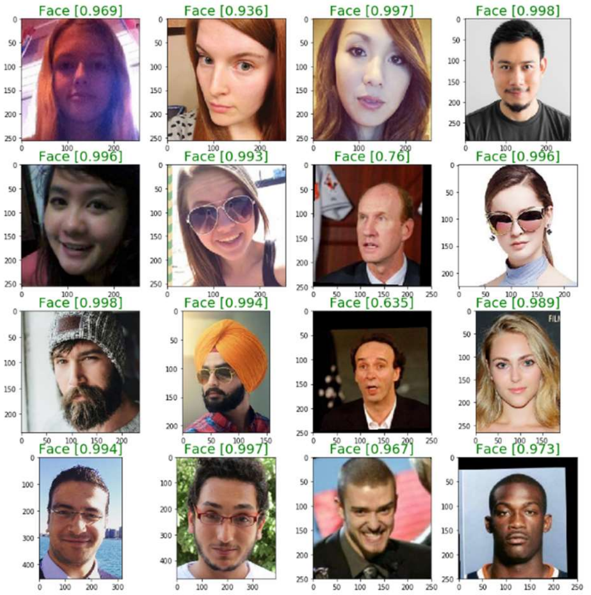


Fig. 7 Results on some test samples of 'Face' class

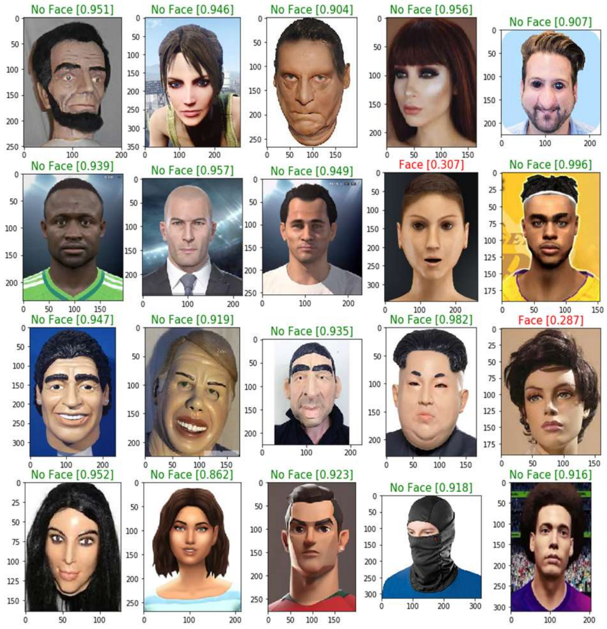


Fig. 8 Results on some test samples of 'No Face' class

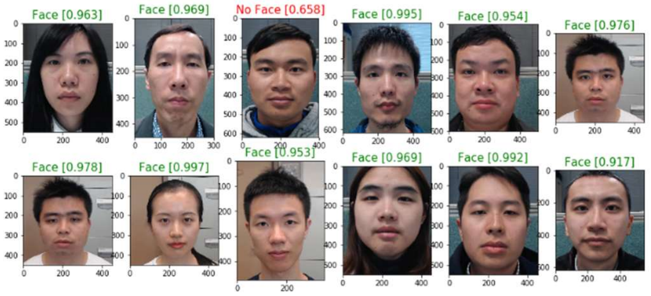


Fig. 9 Results on some HKBU-MARs V1+ dataset test samples of 'Face' class

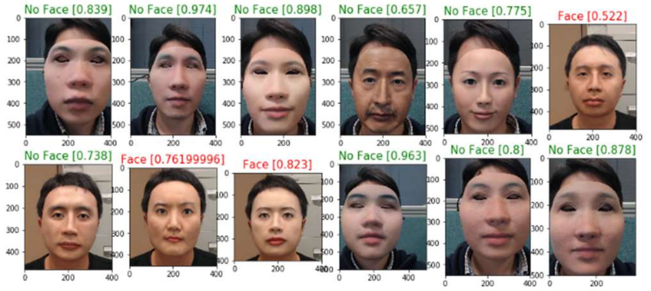


Fig. 10 Results on some HKBU-MARs V1+ dataset test samples of 'No Face' class

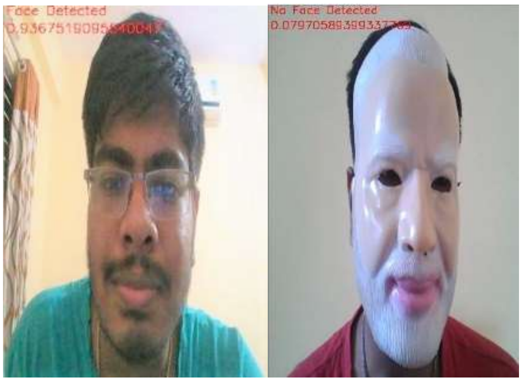


Fig. 11 Real-time prediction results on webcam

D. Hardware implementation

After testing the CNN model, it was implemented on hardware. Raspberry Pi 3 Model B+, an ARM (advanced RISC machine) microcontroller (embedded device) was used. A Logitech C310 webcam was connected to a USB port of the device. Raspberry Pi with a 1.4 GHz 64-bit quad-core ARM Cortex A53 processor works on Linux based OS Raspbian. All the required python packages were installed on it. For real-time testing on webcam, 3D plastic masks of human faces were used.

A python script was run on the device which would first load an HDF5 (.h5 extension) file where the trained weights of the CNN model mentioned earlier are saved. It will start streaming from the web camera, capture the frame and pre-process it. Then, it will predict whether or not a real human face is present in the frame of the camera and accordingly label it as 'Face Detected' or 'No Face Detected'. The model probability (accuracy in scale of 1) for the frame is also shown on it. This is repeated for every incoming frame. Real-time prediction results have been depicted in Fig. 11. The time taken to pre-process and evaluate a frame on the Intel Core i7 processor and Raspberry Pi's ARM processor were calculated. The results have been tabulated in Table IV.

VIII. Conclusion

In this paper, a CNN based classification model for real-time human face and spoofing detection has been proposed. A dataset by collecting more than 15k 'Face' and 'No Face' images. A variety of images were included in both the classes. Mean normalization was performed on each training sample. Data augmentation techniques such as mirroring, rotation and illumination were applied on training set images thus bringing in all kinds of variation. The proposed CNN performed very well on the test set as well as on the test set created from the HKBU-MARs V1+ database. In addition to accuracy, satisfactory level of performance on both the test set was obtained by the trained CNN after evaluation on metrics such as precision, recall and F1-score. The ability of this CNN to accurately classify real human faces and masked faces gives it an edge over traditional face detection algorithms where masked faces often gets wrongly classified as real human face. Moreover, the model is suitable for use in embedded or mobile devices because of it being light weighted. When implemented real-time on Raspberry Pi embedded system device using a webcam, it was correctly able to detect a human face in the frame and whether the face was masked or not.

References

[1] Turk, Matthew A., and Alex P. Pentland. "Face recognition using eigenfaces." Proceedings. 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. IEEE, pp 586-591, 1991.

[2] Parkhi, Omkar M., Andrea Vedaldi, and Andrew Zisserman, "Deep face recognition.", bmvc. Vol. 1. No. 3., pp 1-12, 2015.

[3] Simonyan, Karen, and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition.", arXiv preprint arXiv:1409.1556, 2014.

[4] Tan, Xiaoyang, et al., "Face liveness detection from a single image with sparse low rank bilinear discriminative model." European Conference on Computer Vision, Springer, Berlin, Heidelberg, pp 504-517, 2010.

[5] Zhang, Zhiwei, et al., "Face liveness detection by learning multispectral reflectance distributions." FG. 436-441, 2011.

[6] Chingovska, Ivana, et al., "Face recognition systems under spoofing attacks." Face Recognition Across the Imaging Spectrum. Springer, Cham, 165-194, 2016.

[7] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua. "A convolutional neural network cascade for face detection." IEEE Conf. Computer Vision and Pattern Recognition (CVPR), pages 5325–5334, 2015.

[8] R. Shao, X. Lan and P. C. Yuen, "Joint Discriminative Learning of Deep Dynamic Textures for 3D Mask Face Anti-Spoofing," in IEEE Transactions on Information Forensics and Security, vol. 14, no. 4, pp. 923-938, April 2019.

[9] N. Erdogmus and S. Marcel, "Spoofing Face Recognition With 3D Masks," in IEEE Transactions on Information Forensics and Security, vol. 9, no. 7, pp. 1084-1097, July 2014.

[10] K. Patel, H. Han, and A. K. Jain. Cross-database face antispoofing with robust feature representation, in Proc. CCBR, vol. 9967, pp. 611-619, 2016.

[11] T. Dhawanpatil and B. Joglekar, "Face Spoofing Detection using Multiscale Local Binary Pattern Approach," 2017 International Conference on Computing, Communication, Control and Automation (ICCUBEA), Pune, pp. 1-5, 2017.

[12] G. D. Simanjuntak, K. Nur Ramadhani and A. Arifianto, "Face Spoofing Detection using Color Distortion Features and Principal Component Analysis," 2019 7th International Conference on Information and Communication Technology (ICoICT), Kuala Lumpur, Malaysia, pp. 1-5, 2019.

[13] P. K. Das, B. Hu, C. Liu, K. Cui, P. Ranjan and G. Xiong, "A New Approach for Face Anti-Spoofing Using Handcrafted and Deep Network Features," 2019 IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI), Zhengzhou, China, pp. 33-38, 2019.

[14] H. S. Lin and Y. Su, "Convolutional Neural Networks for Face Anti- Spoofing and Liveness Detection," 2019 6th International Conference on Systems and Informatics (ICSAI), Shanghai, China, pp. 1233-1237, 2019.

[15] H. Chen, G. Hu, Z. Lei, Y. Chen, N. M. Robertson and S. Z. Li, "Attention-Based Two-Stream Convolutional Networks for Face Spoofing Detection," in IEEE Transactions on Information Forensics and Security, vol. 15, pp. 578-593, 2020.

[16] W. Sun, Y. Song, C. Chen, J. Huang and A. C. Kot, "Face Spoofing Detection Based on Local Ternary Label Supervision in Fully Convolutional Networks," in IEEE Transactions on Information Forensics and Security, vol. 15, pp. 3181-3196, 2020.

[17] Liu S., Yuen P.C., Zhang S., Zhao G. (2016) 3D Mask Face Anti-spoofing with Remote Photoplethysmography. In: Leibe B., Matas J., Sebe N., Welling M. (eds) Computer Vision – ECCV 2016. ECCV 2016. Lecture Notes in Computer Science, vol 9911. Springer, Cham, 2016.

[18] Si-Qi Liu, Xiangyuan Lan, Pong C. Yuen, Remote Photoplethysmography Correspondence Feature for 3D Mask Face Presentation Attack Detection, The European Conference on Computer Vision (ECCV), pp. 558-573, 2018.

[19] Kingma, Diederik P., and Jimmy Ba, "Adam: A method for stochastic optimization.", arXiv preprint arXiv:1412.6980. 2014.