Journal Homepage: - www.journalijar.com

INTERNATIONAL JOURNAL OF ADVANCED RESEARCH (IJAR)

Article DOI: 10.21474/IJAR01/11121
DOI URL: http://dx.doi.org/10.21474/IJAR01/11121

RESEARCH ARTICLE

IMPLEMENTATION OF HUMAN FACE AND SPOOFING DETECTION USING DEEP LEARNING ON EMBEDDED HARDWARE

R. Abhishek
Department of Electronics and Communication (ECE) Vnit-Nagpur.

Manuscript Info

Manuscript History
Received: 05 April 2020
Final Accepted: 07 May 2020
Published: June 2020

Key words:-
Deep Learning, Computer Vision, Image Classification, Face Detection, Face Spoofing

Abstract

Human face and spoofing detection is of prime importance in many security verification and law enforcement applications. State-of-the-art human face and spoofing detection applications and publications have recorded low accuracy, specificity and sensitivity. 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 realtime detection. It accurately predicts whether or not a real human face is present in the frame of the recognizing/detecting camera.

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 [5], [6] can easily be fooled by an interloper who can use photographs of a person to log in successfully. There have been solutions developed in the past to tackle this issue. Face live-ness detection systems as described in [7], [8] and spoof detection system [9] are good examples.

However, 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.

Corresponding Author:- R. Abhishek
Address:- Department of Electronics and Communication (ECE) Vnit-Nagpur.

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. Our real-time face detection system will work as a security key to further recognize faces.

There have been traditional face detectors readily available for use in various computer vision applications such as detection using Viola Jones face detector [2] and Haar cascades [3]. CNN cascade classifier [4] can also be used.

In this paper we introduce a binary classification approach using CNN for real human face detection task. 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 [10], AlexNet [11], and MobileNet [12] that have been trained on huge datasets have achieved formidable accuracy in classifying images into multiple classes. It is desirable to classify two classes. Thus a binary classifier has been designed using CNN. A dataset containing more than 15k images was prepared that was used to train and test the CNN.

The rest of the paper is organized as follows: Section 2 provides the glimpse on the basics of Convolutional Neural Network, Section 3, 4 and 5 describes our work in brief, Section 6 discusses about the results and Section 7 concludes the work.

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 is a class of neural network. LeNet-5 [1] was the first CNN architecture designed. It 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.

All parameters such as input size, number of layers, size and number of filters, batch size and optimizing algorithm’s learning rate are called ‘hyper parameters’. Hyper parameter tuning is very important process in order to obtain a sophisticated model for the proposed work.

Dataset

Dataset Design

We have prepared a dataset of 15.6k images. The dataset consists of two classes- 'Face' and 'No Face'. We have downloaded images from various sources on the internet and also added images from publicly available datasets. 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

Table 1:- Dataset Statistics.

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

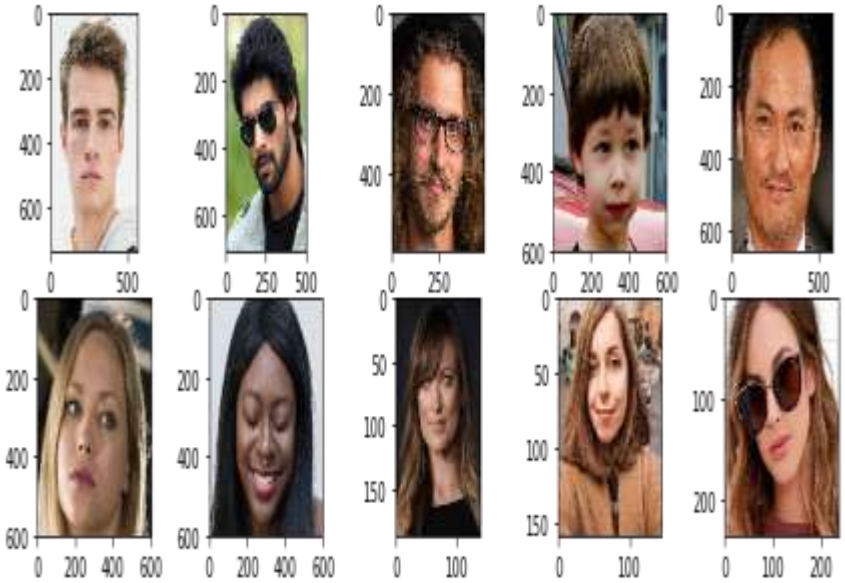


Fig. 1:- Training Samples of 'Face' class.

Fig. 2:- Training Samples of 'No Face' class.

beard and/or moustache or not (for men), with headwear or not. Fig. 1 shows some training samples of the 'Face' class.

In addition to the above, we collected 7.6k images 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 our dataset have varied sizes. We divided the dataset into three different parts for training, validation, and testing. The dataset statistic for both the classes is shown in Table 1. It was ensured that the training and validation sets were prepared from the same distribution while the training and testing sets were mutually exclusive.

Data Pre-processing

To improve the accuracy and enhance the speed of training, all the images were first reshaped to a fixed size. All the pixel values in each image were divided by 255 and the mean and standard deviation of all the pixel values in the three channels across all training images was computed.

The obtained values of mean were 0.5519, 0.4811 and 0.4498 respectively for R, G and B channels. The obtained values of standard deviation were 0.3125, 0.2979, 0.2946 respectively for R, G and B channels. 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.

Proposed CNN Architecture

In this section, we describe the architecture of our proposed CNN model which we have trained on our dataset. The layout of this CNN model is shown in Fig. 3. 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 2.

A single block consists of a stack of two convolutional layers having the same number of filters followed by a max-pooling layer. This kind of block structure has been seen in the architecture of VGGNet as described in [10]. The kernel for all the filters in our network is of size 3 × 3 and 5 × 5 in alternate layers. Before performing convolution with the input in each layer, 'zero' padding has been done to the input. Also, the filters are shifted by a stride of 1 in both the direction. Batch normalization [16] has been applied after each convolutional and the fully connected layer. The importance of applying batch normalization has been discussed in Section 6.

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

Table 2:- A Block of Layers.

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

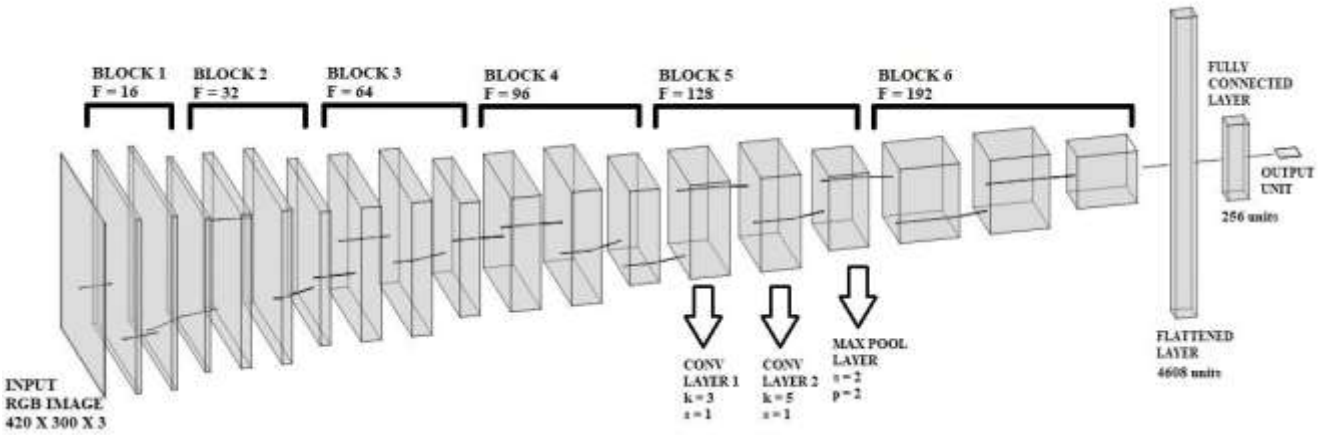


Fig. 3: Layout of proposed CNN Architecture [F=number of filters, k=kernel size, s=strides, p=pool size]

Table 3:- Complete Architecture.

Block/Layer	Number of Filters	Output Shape
Input Image	3	420 × 300 × 3
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
Flattened	-	4608
Fully Connected	-	256
Output Unit	-	1

or a dense layer) followed by an output layer of a single unit. In the proposed CNN layout shown in Fig. 3 and Table 3, the number of filters increases from 16 in the first block to 192 in the last block. In other words, we increase the number of feature maps 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. Our 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, our CNN consumes 35 MB space only.

Training

In the previous section, we have discussed the CNN architecture. In this section, we will discuss about training and testing/evaluating the CNN on the designed dataset.

We have made use of deep learning frameworks Keras and Tensorflow for training the CNN. Python's image processing library OpenCV was used for pre-processing the samples as described in the Section 3.

Weight initialization plays an important role in obtaining a better model. We have initialized all weights using 'He' initialization [13] which is a little modification to the 'Xavier' initialization [14] commonly used. We have applied the ReLU activation function 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. We have used L2 regularization 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. We have used batches of 32 samples 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 as described in [17]. Here, the Adam optimization technique with an initial learning rate of 0.001 was used. Default values of other parameters $\beta 1$, $\beta 1$ and were taken as mentioned in [15]. 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. Keras has an inbuilt callback for learning rate decay.

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.

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, we will show the results of testing the trained weights of the CNN model on images that the network had not come across while training. Also, the results of real-time evaluation using webcam interfaced with an embedded device are provided.

Training results and experiments

Before arriving at our 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 and 5 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. We trained the model till the 30th epoch but we selected the model weights obtained after the 22nd epoch. This was because the validation loss obtained here was the least.

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. We trained the CNN with 256, 512 and 1024 neurons.

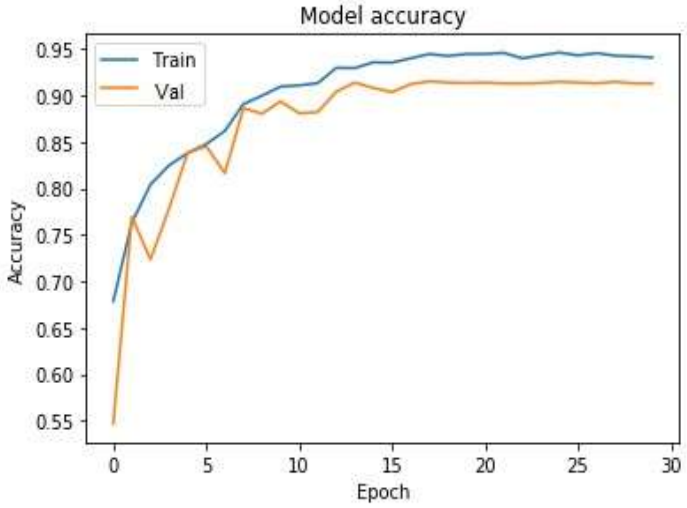


Fig. 4:- CNN Training and Validation accuracy.

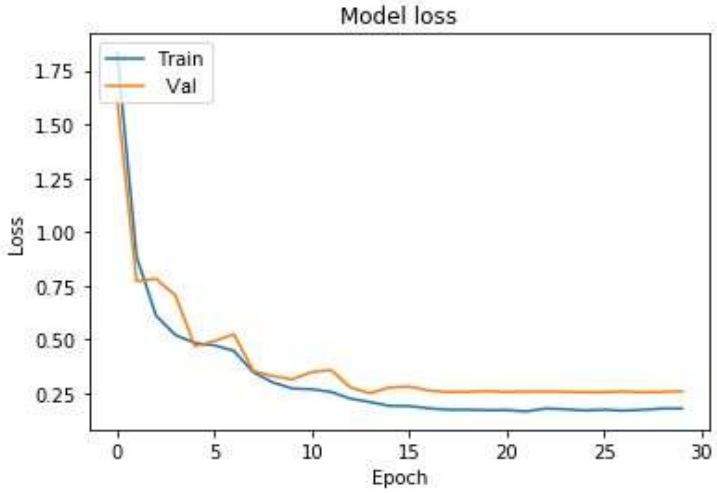


Fig. 5:- CNN Training and Validation loss.

There were no substantial differences in model performance. So, we decided to keep 256 neurons in the fully connected layer.

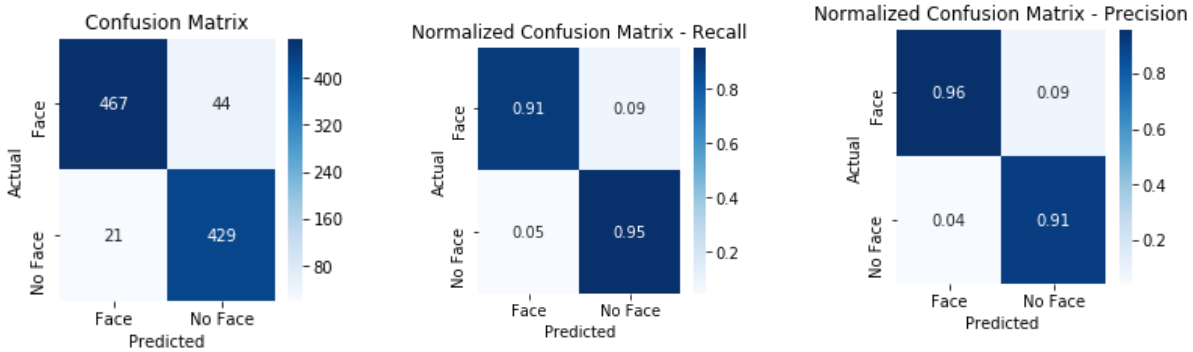
Testing

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. 6 (a).

Every time a model was trained, we analyzed which types of images were wrongly classified i.e. all the false positives and true negatives. Adding more samples of those in the training set helped improve the model performance. Some test samples and their obtained model probabilities (accuracy in scale of 1) along with their predicted labels have been illustrated in Figure 7 and 8. Considering 'No Face' as our 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. 6 (b) and Fig. 6 (c).

Hardware Implementation

After testing the CNN model, we implemented it on hardware. Raspberry Pi 3 Model B+, an ARM (advanced RISC



(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.

machine) microcontroller (embedded device) was used. We connected the Logitech C310 webcam to a USB port of the device. Raspberry Pi has a Fig. 8: Test results of 'No Face' class

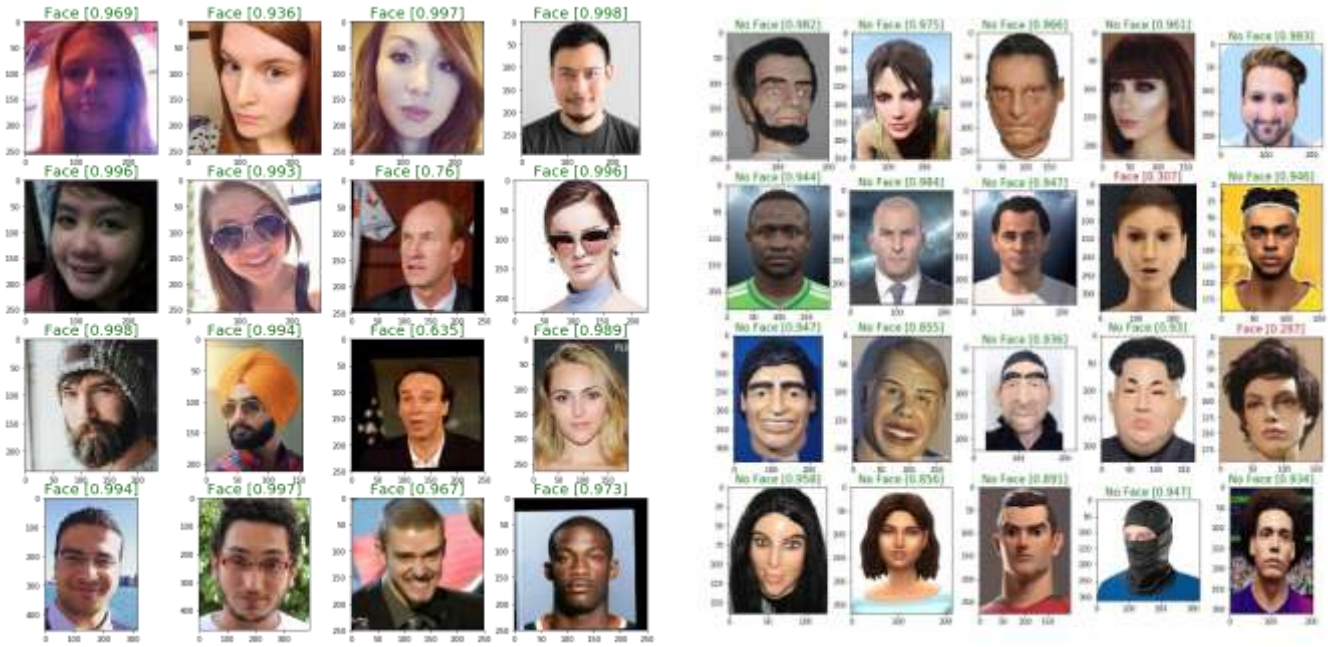


Fig. 7: Testresults of 'Face' Class.

1.4 GHz 64-bit quad-core ARM Cortex A53 processor. It works on Linux based OS Raspbian. We installed all the required python packages on it.

Table 4: Time taken for evaluating a frame on different processors

Processor	Time taken(in seconds)
Intel Core i7-6500U CPU	0.31319
Tesla K80 GPU	0.01223 ARM Cortex A53 0.52341

(a) (b)

Fig. 9:- Real-time prediction results on webcam. (a) shows that frame has 0.93 probability that real human face is present (b) shows that frame has 0.08 probability that real human face is present that is 0.92 probability that no real human face is present.

For real-time testing on webcam, we have used 3D plastic mask of a human face and 25 color printouts of high-resolution images of computer graphics generated face and latex masks.

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 we 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 'Face Detected' or 'No Face Detected'. This is repeated for every incoming frame. Real-time prediction results have been depicted in Fig. 9. The time taken to pre-process and evaluate a frame on various processors including Raspberry Pi's processor were calculated . The results have been tabulated in Table 4.

Conclusion:-

In this paper, we have proposed a CNN based classification model for real-time human face and spoofing detection. We have designed our 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. In addition to accuracy, satisfactory level of performance on test set was obtained by the trained CNN after evaluation on metrics such as precision, recall and F1-score.

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. However, the model hasn't been trained for liveness detection. Hence, we can extend our model so in addition it can detect liveness of the human face in the frame as well. A combined model for both human face/realistic mask detection (which we have mentioned in this paper) and liveness detection can be sought as a future scope.

References:-

1. LeCun, Yann, et al., "Object recognition with gradient-based learning.", Shape, contour and grouping in computer vision. Springer, Berlin, Heidelberg, 319-345(1999)
2. Viola, Paul, and Michael J. Jones. "Robust realtime face detection." International journal of computer vision 57.2, 137-154 (2004)
3. Wilson, Phillip Ian, and John Fernandez. "Facial feature detection using Haar classifiers." Journal of Computing Sciences in Colleges 21.4, 127-133(2006)
4. 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)
5. Turk, Matthew A., and Alex P. Pentland. "Face recognition using eigenfaces." Proceedings. 1991 IEEE Computer Society Conference on Computer
6. Vision and Pattern Recognition. IEEE, (1991)
7. Parkhi, Omkar M., Andrea Vedaldi, and Andrew Zisserman, "Deep face recognition.", bmvc. Vol. 1.
8. No. 3., (2015)
9. 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,
10. 504-517(2010)
11. Zhang, Zhiwei, et al., "Face liveness detection by learning multispectral reflectance distributions." FG. 436-441(2011)
12. Chingovska, Ivana, et al., "Face recognition systems under spoofing attacks." Face RecognitionAcross the Imaging Spectrum. Springer, Cham,165-194 (2016)
13. Simonyan, Karen, and Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition.", arXiv preprint arXiv:1409.1556 (2014)
14. Krizhevsky, Alex, IlyaSutskever, and Geoffrey E. Hinton, "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems, (2012)
15. Howard, Andrew G., et al., "Mobilenets: Efficient convolutional neural networks for mobile vision applications.", arXiv preprint arXiv:1704.04861(2017)
16. He, Kaiming, et al., "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification.", Proceedings of the IEEE international conference on computer vision, (2015)

17. Glorot, Xavier, and YoshuaBengio, ”Understanding the difficulty of training deep feedforward neural networks.”, Proceedings of the thirteenth international conference on artificial intelligence and statistics, (2010)
18. Kingma, Diederik P., and Jimmy Ba, ”Adam: A method for stochastic optimization.”, arXiv preprint arXiv:1412.6980 (2014)
19. Ioffe, Sergey, and Christian Szegedy, ”Batch normalization: Accelerating deep network training by reducing internal covariate shift.”, arXiv preprint arXiv:1502.03167 (2015)
20. Ruder, Sebastian, ”An overview of gradient descent optimization algorithms.”, arXiv preprint arXiv:1609.04747 (2016)
21. Wu, Haibing, and XiaodongGu, ”Towards dropout training for convolutional neural networks.” Neural Networks 71, 1-10 (2015).