International Information and Engineering Technology Association
Advancing the World of Information and Engineering

Ingénierie des Systèmes d'Information
Vol. 27, No. 2, April, 2022, pp. 303-311
Journal homepage: http://iieta.org/journals/isi

CAPTCHA Robustness- AI Approach Using to Web Security

Abhishek Sharma[1], Shilpi Sharma[2*], Saksham Gulati[3], Tanupriya Choudhury[4]

[1] My Concierge Private Limited, 08-71, Vertex Tower, Singapore
[2] Department of Computer Science and Engineering, Amity University, Uttar Pradesh 201313, India
[3] Illinois Institute of Technology, Chicago, Illinois, IL 60616, US
[4] Informatics Cluster, School of Computer Science, University of Petroleum and Energy Studies (UPES), Dehradun 248007, Uttarakhand, India

Corresponding Author Email: ssharma22@amity.edu

https://doi.org/10.18280/isi.270214

Received: 7 January 2022
Accepted: 14 April 2022

Keywords:
human verification, CAPTCHA, neural engine, gesture recognition

ABSTRACT

Human verification is important to avoid spamming over the internet. Internet has grown tremendously in the last decade. Introduction to preference personalization make the users experience and attract more people. This has deemed it necessary to verify that some of the actions are carried out by a human rather than a computer program to generation of invalid traffic and prevent fraudulence. Methods like Captcha and ReCaptcha have extensively used to overcome this challenge but with the advancements in machine leaning and artificial intelligence, these techniques have started to become obsolete. So, to address this issue we are proposing a phase based human verification process using a combination of neural network and machine learning.

1. INTRODUCTION

Verifying a user to be human has been a long requirement in the computer world and information technology. Advancements in technology have eased the process to create automation scripts and programs that can perform varied operations. Automation makes the process doing tedious and repetitive tasks quick, error free and accurate, but it can often be used for malicious purposes. An automation program can be executed to sign up a lot of accounts on some social networking platform and use them to grow someone's followers. Hence it is vital to detect if an action or a transaction is being caried out by a machine or a human.

Several strategies have been invented over the years to verify if the user is a human or a bot. The most popular of those is CAPTCHA which distinguishes bot from humans by creating words in camouflage in an image that can only be understood by a person. Another variation of it is called reCAPTCHA, which used images to realize the same target but using asking users to select images of a particular object or thing.

Although effective but these techniques have now started to become outdated and can be outsmarted by the modern bots. There are already many tutorials and videos posted by various people on how these verification programs can be bypassed. Also, the significant growth in artificial intelligence and machine learning in the last decade, will rendered these methods obsolete in not-so-distant future and will need to be replaced.

There have been some inventions to create alternatives to the existing human verification methods. The author [1] lists down the difficulties with the exiting Captcha methods and also discusses other techniques of human verification. One such techniques called is called rtCAPTCHA [2], that uses facial and voice to distinguish a genuine user from a program.

In this paper we propose a new method for human verification through digital image processing. We aim build a method to confirm that the interaction is taking place with a human by leveraging the cameras and webcams on their devices.

2. LITERATURE REVIEW

Firstly, we aim to study and understand how the current version in actually attacked. The concept of captcha was introduced in 2000 and technology has taken a massive leap since then. Even before the machine learning era, all that was needed to overcome a traditional text-based captcha was a decent enough OCR.

An attack on Microsoft's captcha was conducted to test its strength [3]. It was found that it can be attacked using a cheap attack based on segmenting. A more efficient attack on the yahoo captcha is described in the reference [4]. Although the attack specified in this publication is performed on the captcha from one provider, the concept can be applied to another captcha provides like google and Microsoft.

The only way captchas are able to defend against these attacks is applying background noise and make the characters more unreadable. This might work against a simple OCR but against a program that leverages machine learning, even these enhancements may not be enough. An experimental study is done conducted by Alqahtani & Alsulaiman [5], where attacks using machine learning were performed on Google reCAPTCHA. The experiment showed how easily machine learning can be used to attack existing captcha. Wang et al. [6] show how a deep CNN program can be trained to identify different variation of a captcha. The most complete solution is presented in Wang et al. [7]. The neural network developed by this method was tested on 20+ captcha variations and yielded a result of over 95%.

As mentioned before there have been techniques developed to overcome this challenge. Another attempt for captcha is mentioned by Almazyad et al. [8], where a combination of an image and text are used to make the process difficult for a bot. Another variation is designed to use characters in 3D shapes in reference [9]. Although it may be effective against an OCR but a NN can easily be designed to beat this variation.

A text-based captcha are basically testing the user's ability to comprehend distorted text, but a some other implementation have been proposed to test a user's cognitive skills. The captcha requires a user to solve a simple jigsaw puzzle [10]. The user is asked to map semantically similar images [11]. A user is asked to select face of real person from a set of pictures that also included animated and cartoon faces [12].

Our implementation will be using the face/hand detection for human verification. It will consist of two phases - Human Detection Phase and Verification Phase. For the Human Detection Phase, one layer will be implemented using HAAR cascades and another later with a Neural Network. One of the most effective implementations of face detection is MTCNN [13], which is able to simultaneously also detect face features with high performance. The authors [14] test 4 different CNN algorithms for face detection. An RCNN is used in Sun et al. [15] and Jiang & Learned-Miller [16], that yields quick results for face detection. A different approach is described by Farfade et al. [17], which does not rely on face feature to detect faces, in fact it only requires a single model to be able to detect face in different orientations.

For gesture recognition, instead of using a library we will be designing our own model. A survey was conducted on the many techniques available for gesture detection that can be used for a number of applications [18]. They looked primarily at four methods – HMMs, FSMs, particle filtering and condensation algorithm. The proposed solution is a hybrid of FSMs and HMMs, which will result in a system that is highly reliable and accurate.

In reference [19], the authors achieve gesture recognition using Scale Invariance Feature Transform (SIFT). SIFT provides a faster processing speed which allow for a much better result in detection. A similar implementation to what we do in our detection phase is used in Ismail et al. [20] for gesture detection. The pitfall of this method is that each gesture needs to be fed into the system separately, even for different skin color and palm size.

So, to bring efficiency to the system, a neural network is deployed. Once such type of execution [21] used Artificial Neural Network that required little computational cost. But one limitation of this technique was its incompetence to differentiate similar looking gestures. For example, it often confused letter 'G' with 'A' or 'D'. A similar solution is implemented [22] that is able to recognize gestures in real time. It describes a new way for extraction of features from an image.

A method to carry out hand gesture recognition using a Convolution Neural Network (CNN) is discussed by Nagi et al. [23]. The proposed method yielded a success percentage of almost 96%. Gaussian Mixture model (GMM) is used to train the skin model and achieve robustness. In reference [24], a CNN is implemented but in a combination with stacked denoising encoder (SDAE). In reference [25], the author used a Micro-Doppler CNN (DCNN) and was able to achieve the required solution but not to great affect because the signature of the gesture varied according to the distance and angle of the radar. A comparative study on the different techniques for gesture recognition is conducted by Sonkusare et al. [26]. In their study they underlined that rate of recognition is trade-off with the time rate.

3. METHODOLOGY

The technique put forward in this paper has been contains two phases namely - Detection Phase and Verification Phase. The image will be captured frame by frame through a computer connected camera and will be sent to detection and verification engine in a live environment. Detection phase uses HAAR cascade methodology of line and edge detection through xml files. HAAR cascade xml files faces are prepared and stored before pushing the data into the detection engine. Details on HAAR cascade detection are given below.

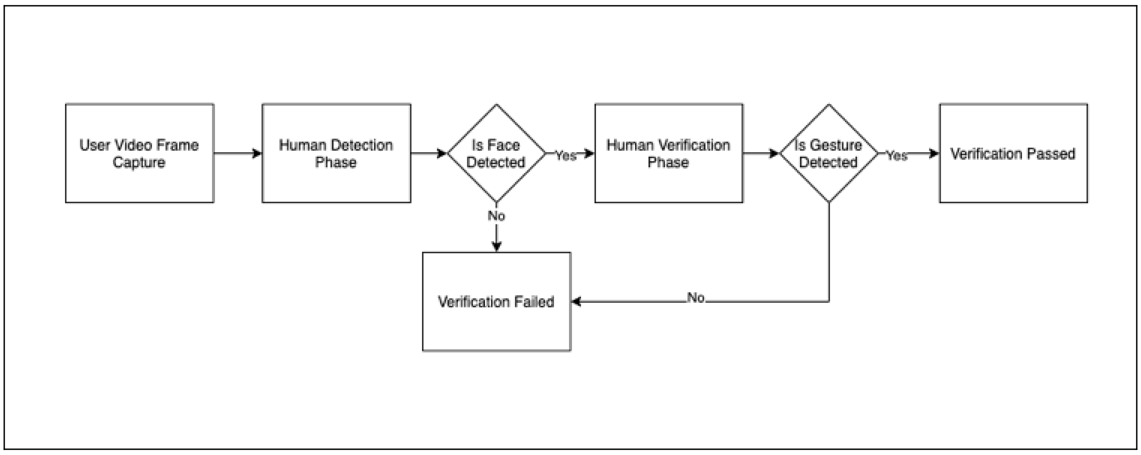


Figure 1. System flowchart

Proposed Method for Human Verification [Figure 1]:

1. Human Detection - In this phase intense digital image processing will be used to detect the object in camera to be a human through facial recognition.
2. Human Verification – In this phase we will ask the user to make a hand sign. Once the user makes that gesture, we will run it against our model to see if it's the same gesture that was requested.

3.1 Human detection phase

Detection phase will consist of multiple layers of detection and initial face verification [Figure 2]. It will consist of two mandatory layers through which each captured frame must pass through with a certain threshold to qualify for the next phase. The thresholds are chosen at each stage as per algorithms implemented on that stage. For HAAR cascade the comparison is done through html and the threshold is generally above 90% while in neural engine face detection.

In human detection phase we will be verifying a human face using digital image processing using the following process:

3.1.1 HAAR face detector

In this phase, a face will be detected on screen using a HAAR cascade face detector. We will use the trained classifiers that is provided by OpenCV for face detection that has been trained with positive and negative images to build a classifier based on the learning algorithm.

We capture images from the user source input and then convert them to grayscale and then analyze them with HAAR cascades for a face [Figure 3]. These images are converted to greyscale because they are easier to process computationally. Grayscale images contain single-channel and the total process of human verification can be reduced greatly using grayscale images.

To understand the implementation of HAAR cascades we first need to understand the Viola-Jones Face Detection Technique. The first part we need to extract feature from the image. This is achieved using the HAAR features [Figure 4]. These features help us to recognize changes in pixels intensity that corresponds to an edge in the image. The integer matrix corresponding to the feature is determined if the feature is an edge or not. In haarcascade this detection matrix is usually manual.

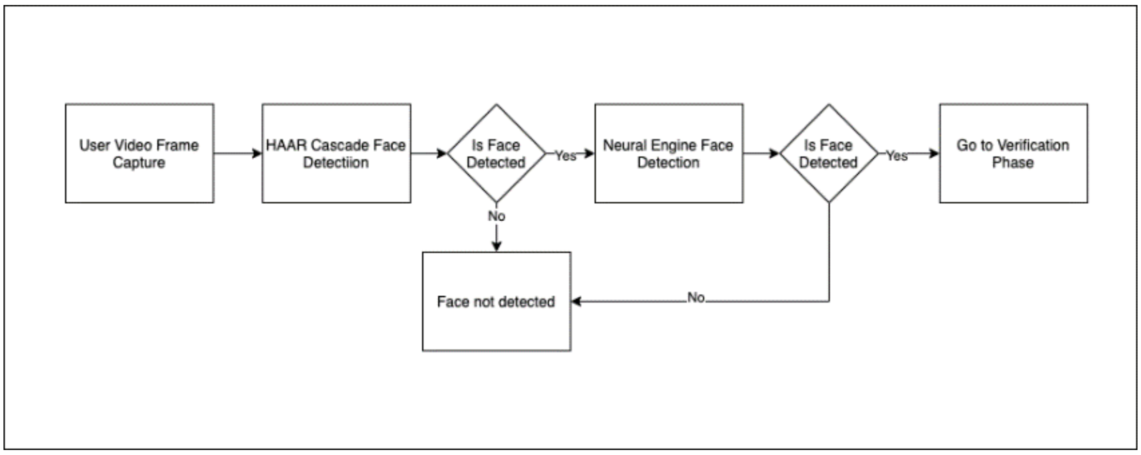


Figure 2. Detection phase flowchart

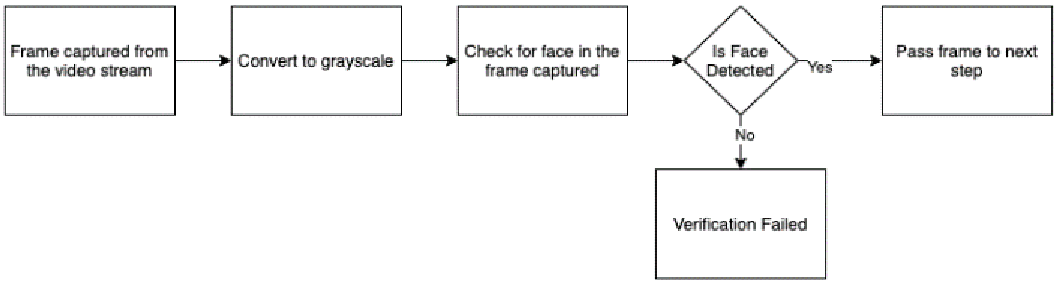


Figure 3. Flowchart for HAAR cascades face detection

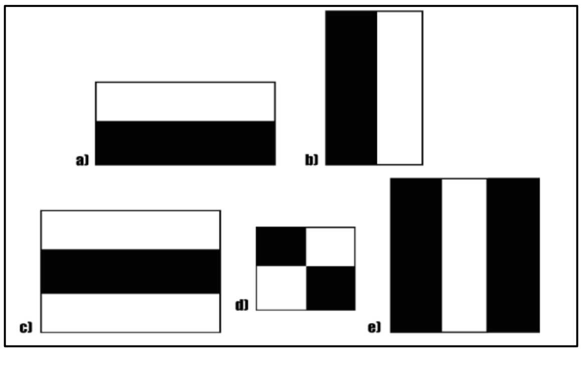


Figure 4. HAAR Features. Image Downloaded from https://towardsdatascience.com/face-detection-with-haar-cascade-727f68dafd08

Below is an example on how these HAAR features are made use of. All pixel's values are converted to a range from 0.0 to 1.0 with 0 being completely white and 1 being completely black using the following formula.

$$pixelValue_{new} = \frac{(255 - pixelValue_{old})}{255}$$

We calculate the avg intensity of pixels in the dark and light region of the HAAR feature once it is plotted on the image [Figure 5]. If the deviation of the intensity is approximately to 1, it means that it is an edge. In the below example since the deviation is close to 0, hence it is not an edge.

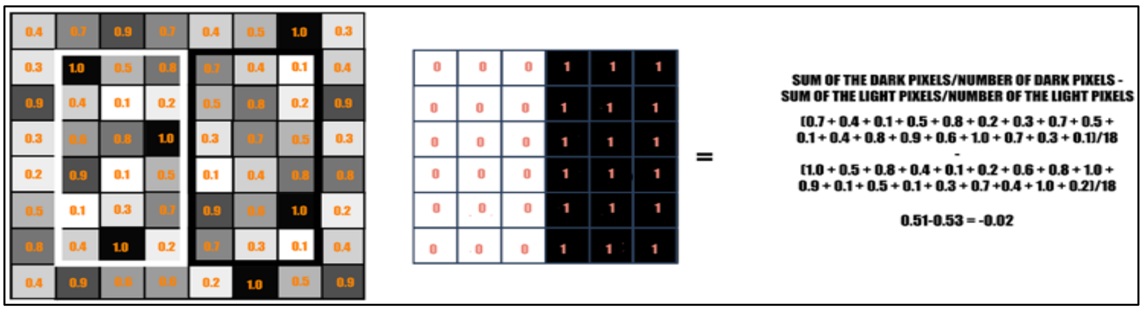


Figure 5. HAAR Feature Calculation. Image Downloaded from https://towardsdatascience.com/face-detection-with-haar-cascade-727f68dafd08

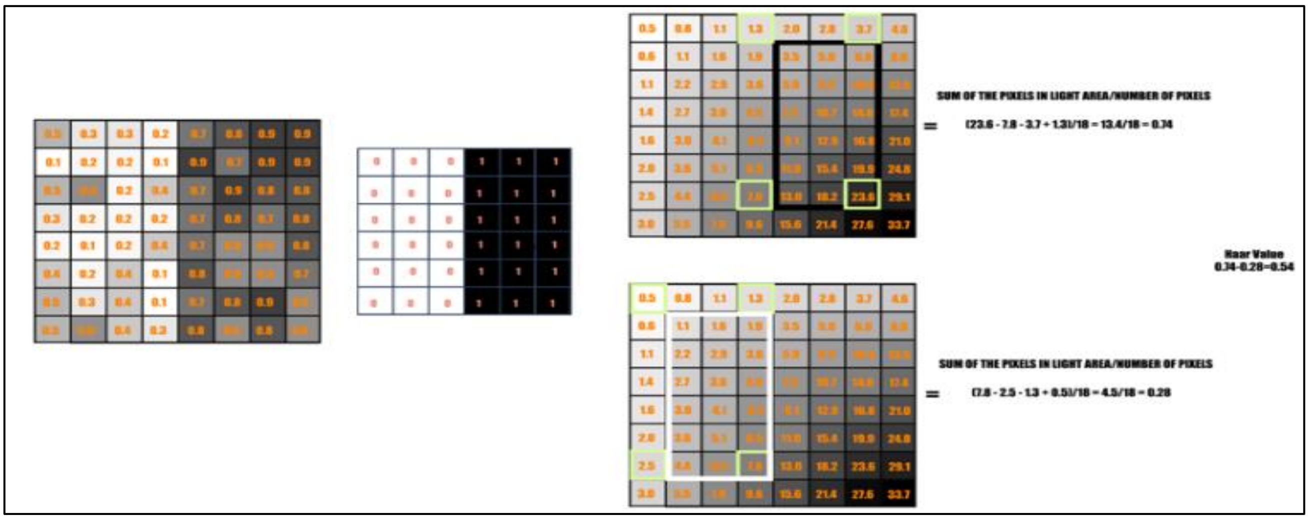

SUM OF THE PIXELS IN LIGHT AREA/NUMBER OF PIXELS
Haar Value
SUM OF THE PIXELS IN LIGHT AREA/NUMBER OF PIXELS

Figure 6. Calculations using Integral Image. Image downloaded from https://towardsdatascience.com/face-detection-with-haar-cascade-727f68dafd08

All 5 HAAR features are used to identify different kind of features in an image. The HAAR feature is used to identify a vertical edge [Figure 6]. Each feature is running through the complete input image to recognize as many features as possible to be able to recognize and object. But doing this needs a lot of computation since there are a huge number of pixels in any image. Hence to make this algorithm more efficient, the image is converted to an *Integral image* before feature calculation. In an *Integral image* value for each pixel is calculated as sum of intensity values at top and before that pixel. We start with the pixel at (0,0) and move sequentially to other pixels in the same row, computing the value of each pixel as the sum of its own value and value of the pixel to its left. Once all the values of the first row are computed we move to the next to row. This time to calculate the value the pixel we add the value of the pixel above. This process is continued till the last pixel. The value of the last pixel will be the sum of all pixel values in the image.

Once we have the integral image. The calculation of variation in intensities for each transversal becomes much faster since the addition operation has already been performed as show in image above.

3.1.2 Face detection using neural engine

Detection phase will consist of multiple layers of detection and initial face verification. Ife will consist of two mandatory layers through which each captured frame must pass through with a certain threshold to qualify for the next phase.

In human detection phase we will be verifying a human face using digital image processing using the following process:

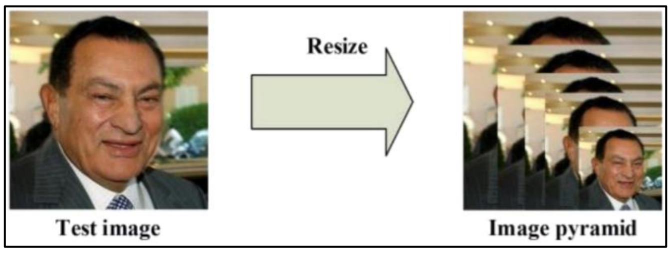


Figure 7. Image Pyramid. Image downloaded from https://towardsdatascience.com/how-does-a-face-detection-program-work-using-neural-networks-17896df8e6ff

Firstly we create an image pyramid by resizing the image to different scales [Figure 7]. Once we have images in different scales, a cropped section of 12x12 pixels from the beginning of the image at (0,0) is captured from each scaled image and passed to the P-Net. The P-net's job is to calculate the probability of how much this cropped image passed matches a face. It then returns a value of confidence reaching from 0 and 1 and the relative coordinates of a rectangle enclosing the face in the cropped image [Figure 8].

This process is carried out on the complete image for each scaled image by moving the filter by a stride of 2. After all the transversal is done, we get a list of coordinates of filters on the image that has some probability of containing a face. But right now, these coordinates are not equivalent since they most probably have been captured on the different scaled images. Hence to make them consistent, the coordinates are scaled up to the size of the original image.

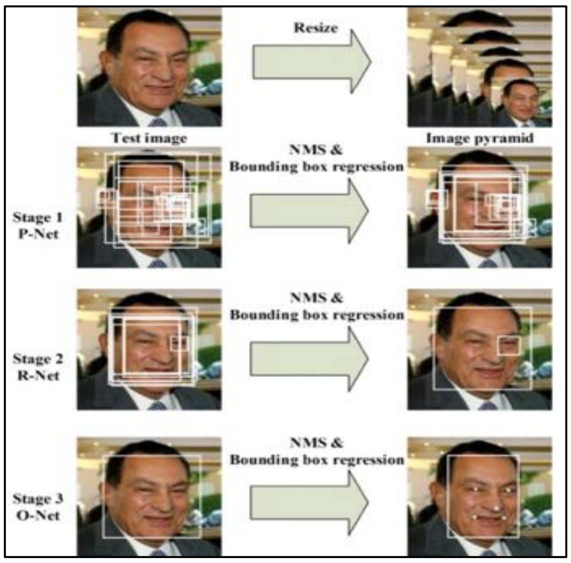


Figure 8. Image transversal by MTCNN. Image downloaded from https://medium.com/dummykoders/face-detection-using-mtcnn-part-1-c35c4ad9c542

Now that we have a list of probable locations of faces in the image, we need a process to get one single accurate result. This is achieved by an elimination process. The coordinate with the highest probability value is taken for image of each scale and are weighed up against the other coordinates in the same scale. If the overlapping area is high enough then the coordinate is eliminated. This step is repeated for the surviving coordinates from each scaled image.

The last surviving coordinates are then sent to the R-net, which works in very similar to the P-net, but it consists of a fully connected CNN that detects faces more accurately than R-net [Figure 8]. The results from the R-net are, which are coordinates of faces in the image, are then padded if need and then passed to the O-net. The O-net in the end detects the face features in the passed coordinates.

The outputs of O-Net are slightly different from that of P-Net and R-Net [Figure 8]. O-Net provides 3 outputs: the coordinates of the bounding box (out[0]), the coordinates of the 5 facial landmarks (out[1]), and the confidence level of each box (out[2]).

Once again, we get rid of the boxes with lower confidence levels, and standardize both the bounding box coordinates and the facial landmark coordinates. Finally, we run them through the last NMS. At this point, there should only be one bounding box for every face in the image.

3.2 Human verification phase

The human verification phase [Figure 9] will be as follows

1. An arbitrary hand gesture will be picked up from a list of gestures which will be shown on to the user to copy.
2. One notified, we will check the check frames from the video capture to check for hand gestures.
3. Gesture picked pseudo randomly from the list will be compared with a gesture made by the user in camera through neural network comparison.

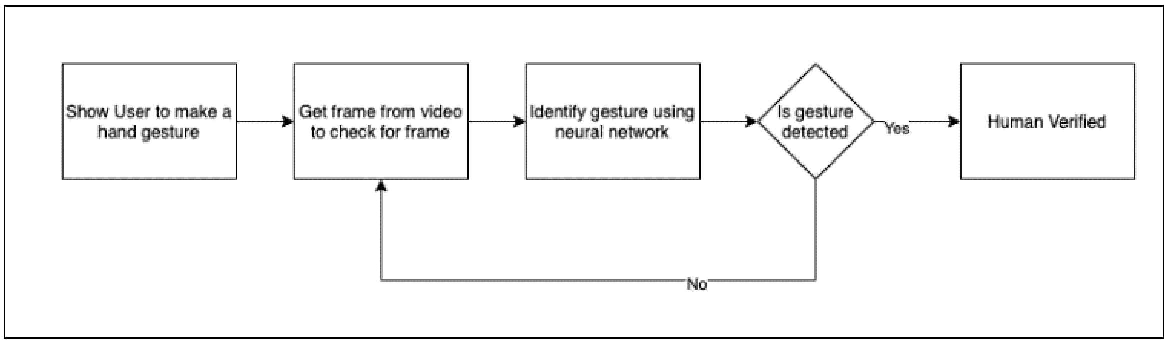

Show User to make a hand gesture
Get frame from video to check for frame
Identify gesture using neural network
Is gesture detected
Yes
Human Verified
No

Figure 9. Flowchart for verification phase

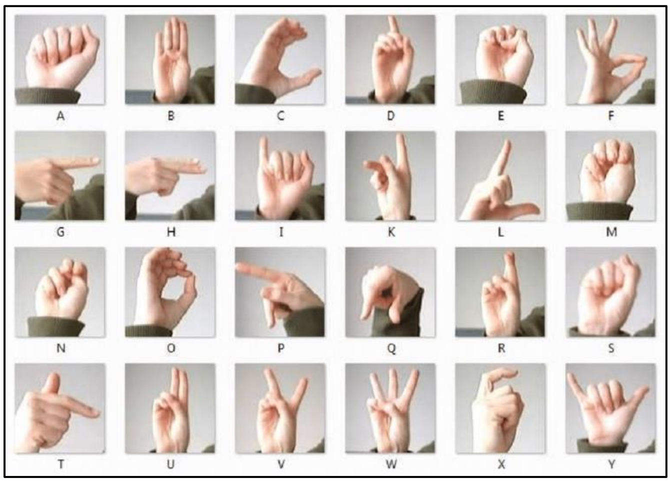


Figure 10. American sign language gestures. Image downloaded from https://analyticsindiamag.com/hands-on-guide-to-sign-language-classification-using-cnn/

In order to perform gesture recognition, we will build a CNN model using TensorFlow. This model was taught on the 24 static gestures in the American sign language [Figure 10], on a cluster of 21000+ images.

4. RESULTS

4.1 Model for gesture recognition

To build our model using the Keras API in python.

4.1.1 Data pre-processing

We have a dataset of almost 35000 samples, out of which 7000 are reserved for testing the model. The dataset is in csv format, so our first step is to augment the data from the csv so that it can be passed to our model. The csv contains 785 columns where the first column is for the label of the image and rest of the column represent the pixel value of a 28x28 image. The label of the image is a number ranging from 0 to

24 representing alphabets from A to Y, with an exclusion of 9 which represents 9. The model we created excludes the alphabet J and Z since they are not static hand gestures.

Now that we understand our dataset. The next step is to transform the data in a form in which it can be processed. To achieve that, we remove the first column from our dataset which is the ‘Label’ column and store it in a separate array. After that the dataset is transformed into an array of 28x28 matrix. Right now, the pixel values are in the range of 0-255. We rescale it to a range of 0-1 which make the processing of image during the learning phase easier. Lastly, we binarize our labels array by using LabelBinarizer. This transforms each element in the labels array into an array that contains 24 values representing each alphabet in our model. All the values in this array are 0, except the value for the alphabet that corresponds to the label of the image. The value of this alphabet is represented by 1.

This process is applied to both the test and training data.

4.1.2 Data augmentation

We use data augmentation to improve the performance of our model. The idea is to train the model on data which is a slight modification of our existing dataset. For example, we can apply rotation, zooming, vertical/ horizontal flips etc. to improve our database. This helps the model to generalize it’s leaning and prevent overfitting. We use the ‘ImageDataGenerator’ from the Keras api to achieve this.

```
data_generator = ImageDataGenerator(
    featurewise_center=False,
    samplewise_center=False,
    featurewise_std_normalization=False,
    samplewise_std_normalization=False,
    zca_whitening=False,
    rotation_range=10,
    zoom_range=0.1,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=False,
    vertical_flip=False
)
```

Figure 11. Code snippet for data augmentation

The code in [Figure 11] is used to augment our data.

4.1.3 Building model

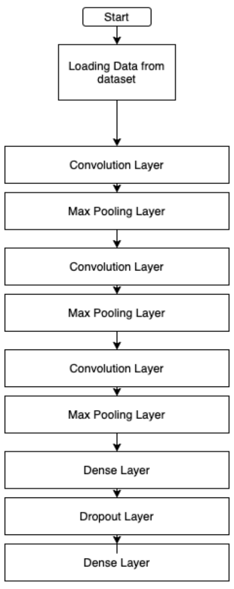


Figure 12. CNN layers

Figure 12 represents our sequential model. We start with a convolution layer with 75 filters, a kernel of size 3x3 and stride of 1. Our model contains 3 convolutional layers and all three have a same parameters except the number of filters which we decrease by 25 in each layer. The first one is followed by a Batch Normalization and a MaxPool2D layer of size 2x2 and stride 1 after which the output is transformed to 14x14. The first 3 layer are repeated with an inclusion of a Dropout layer to exclude some filters. And our third repetition of the convolutional layer we add a dense layer. Our last layer is a Dense layer with 24 units, one for each gesture.

4.1.4 Training model

We use ‘adam’ as our optimizing function and ‘categorical crossentropy’ as loss function [Figure 13].

```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

Figure 13. Code Snippet for training model

We define our learning rate as a plateau that reduces the learning rate if the metric has stopped improving [Figure 14].

```
learning_rate_reduction=ReduceLROnPlateau(monitor='val_accuracy', patience=2, verbose=1, factor=0.5, min_lr=0.00001)
```

Figure 14. Code Snippet for learning rate

We train the model using 20 Epocs. Once the model is created it is able to evaluate the test data with an 100% accuracy.

4.2 Implementation

Firstly, we implement face detection using OpenCV. We used the “*haarcascade_frontalface_default.xml*” provided by the OpenCV library [Figure 15]. Before passing for human face detection, we convert the image to grayscale. It is then passed for processing, which returns the coordinates of the face with respect to the input frame.

```
# convert JS response to OpenCV Image
img = js_to_image(js_reply["img"])

# create transparent overlay for bounding box
bbox_array = np.zeros([480,640,4], dtype=np.uint8)

# grayscale image for face detection
gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

# get face region coordinates
faces = face_cascade.detectMultiScale(gray)
```

Figure 15. Code snippet for HAAR cascades face detection

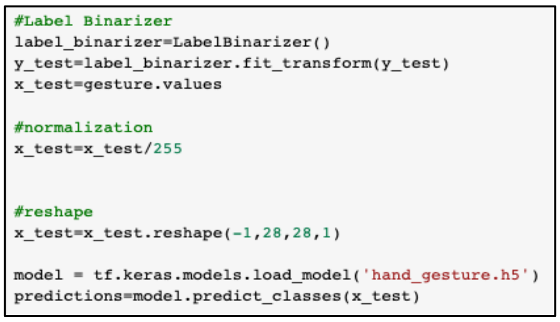
```
# convert JS response to OpenCV Image
img = js_to_image(js_reply["img"])

# create transparent overlay for bounding box
bbox_array = np.zeros([480,640,4], dtype=np.uint8)

# grayscale image for face detection
gray = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)

mtcnn_face_detector_model = MTCNN()
face_locations = mtcnn_face_detector_model.detect_faces(img)
```

Figure 16. Code snippet for MTCNN face detection



```
#Label Binarizer
label_binarizer=LabelBinarizer()
y_test=label_binarizer.fit_transform(y_test)
x_test=gesture.values

#normalization
x_test=x_test/255


#reshape
x_test=x_test.reshape(-1,28,28,1)

model = tf.keras.models.load_model('hand_gesture.h5')
predictions=model.predict_classes(x_test)
```

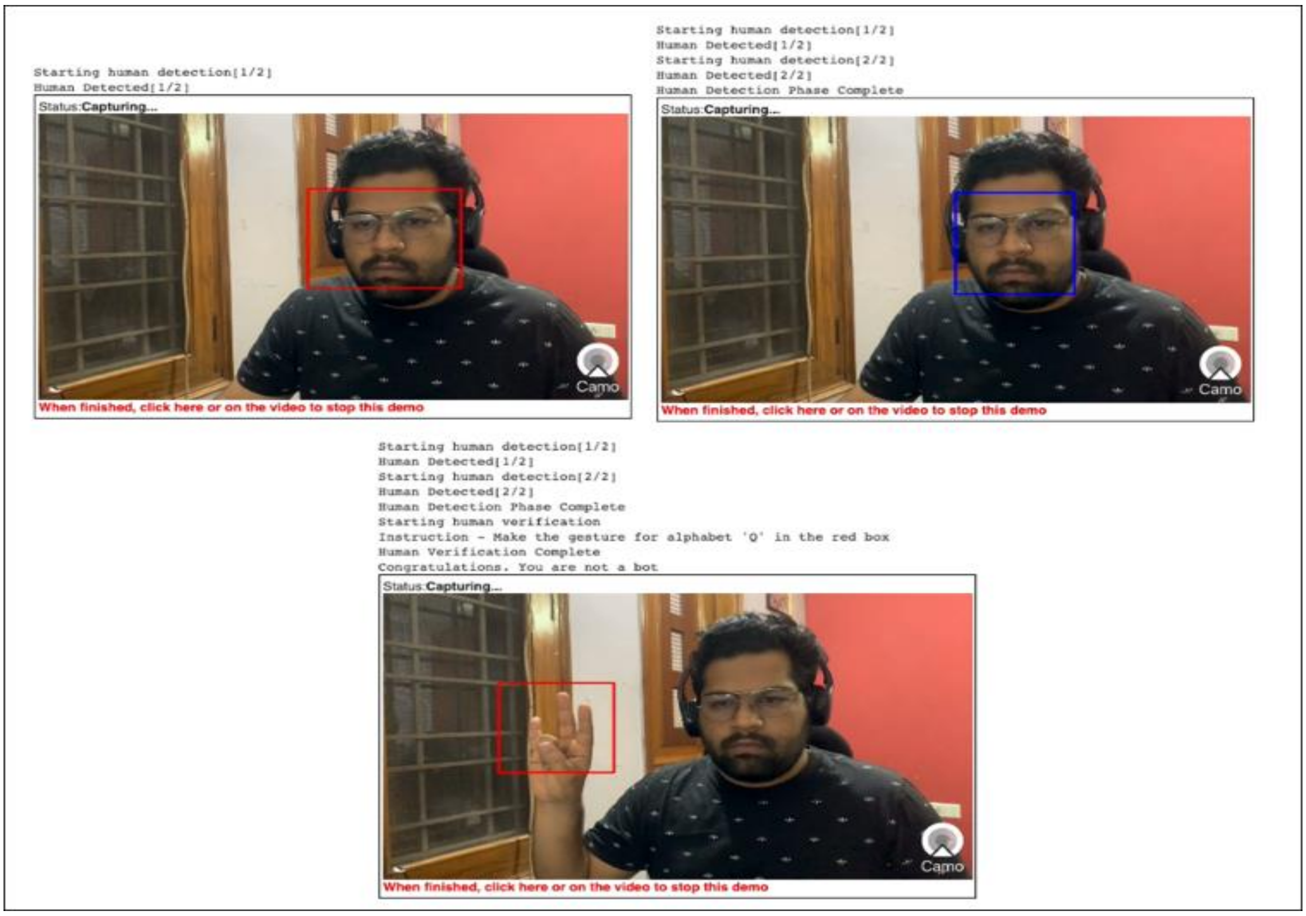
Figure 17. Code Snippet for gesture recognition

In the next step, we similarly pass a grey image from the user input source to the MTCNN library [Figure 16]. It returns the face coordinates as well as the face feature coordinates, but we only make use of the face coordinates.

We use our model created earlier for getting match for the gesture made by the user on the screen [Figure 17]. We leave out the signs of alphabet ‘J’ and ‘Z’ because they require hand movements.

Now that we have our 3 implementations, we put them together to create a complete flow for human verification.

Figure 18 shows the screenshots from the program. The blue and red boxes on the face represent the detection by MTCNN and HAAR cascades respectively. Once we have verified, we move to the verification phase. We present a red box on the camera screen and ask the user to their hand in that box making the required random gesture. The program completed once it identifies the required gesture.



```
Starting human detection[1/2]
Human Detected[1/2]
```

```
Starting human detection[1/2]
Human Detected[1/2]
Starting human detection[2/2]
Human Detected[2/2]
Human Detection Phase Complete
```

```
Starting human detection[1/2]
Human Detected[1/2]
Starting human detection[2/2]
Human Detected[2/2]
Human Detection Phase Complete
Starting human verification
Instruction - Make the gesture for alphabet 'O' in the red box
Human Verification Complete
Congratulations. You are not a bot
```

Figure 18. Screenshots from Program

5. CONCLUSION

The program developed is able to successfully identify a person. For a computer program to bypass this test, it will be required to replace the camera stream with a video. Although that maybe easy but to get the required gesture at the required location will be very though. And to make it even tougher, we can ask the user to perform 2 or 3 different gestures at different locations on the screen.

Although the program realizes its goal, but the performance is not very good in the detection phase, particularly in the MTCNN detection. It could just be down to the environment. But it can easily be changes with some other more mature and advance method of face detection CNN. The gesture recognition on the other hand performs decently due to the fact that we ask the user to place the gesture in a box instead of trying to detect it on the complete frame.

6. FUTURE WORK

This test has a lot of potential for improvement. For instance, to improve the detection phase we can ask a user to do some head movements to confirm a live subject. In the verification phase we can add face gesture recognition. In the verification phase, we can also add face gestures to our model. Also, we can develop model for dynamic hand gestures.

As mentioned above there is a need to improve the execution of the face detection techniques implemented using MTCNN. Right now, the models can only give their predictions as true or false. They can be enhanced to give a result of the percentage match, and then based on that percentage, we can choose whether to pass the test or not.

REFERENCES

[1] Moradi, M., Keyvanpour, M. (2015). CAPTCHA and its alternatives: A review. Security and Communication Networks, 8(12): 2135-2156. https://doi.org/10.1002/sec.1157

[2] Uzun, E., Chung, S.P.H., Essa, I., Lee, W. (2018). rtCaptcha: A real-time CAPTCHA based liveness detection system. In NDSS. 1-15. http://dx.doi.org/10.14722/ndss.2018.23253

[3] Yan, J., El Ahmad, A.S. (2008). A low-cost attack on a Microsoft CAPTCHA. In Proceedings of the 15th ACM Conference on Computer and Communications Security, pp. 543-554. https://doi.org/10.1145/1455770.1455839

[4] Gao, H., Wang, W., Fan, Y. (2012). Divide and conquer: An efficient attack on Yahoo! CAPTCHA. In 2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications, pp. 9-16. https://doi.org/10.1109/TrustCom.2012.131

[5] Alqahtani, F.H., Alsulaiman, F.A. (2020). Is image-based CAPTCHA secure against attacks based on machine learning? An experimental study. Computers & Security, 88: 101635. https://doi.org/10.1016/j.cose.2019.101635

[6] Wang, J., Qin, J., Xiang, X., Tan, Y., Pan, N. (2019). CAPTCHA recognition based on deep convolutional neural network. Math. Biosci. Eng, 16(5): 5851-5861. https://doi.org/10.3934/mbe.2019292

[7] Wang, P., Gao, H., Shi, Z., Yuan, Z., Hu, J. (2020). Simple and easy: Transfer learning-based attacks to text CAPTCHA. IEEE Access, 8: 59044-59058. https://doi.org/10.1109/ACCESS.2020.2982945

[8] Almazyad, A.S., Ahmad, Y., Kouchay, S.A. (2011). Multi-modal captcha: A user verification scheme. In 2011 International Conference on Information Science and Applications, pp. 1-7. https://doi.org/10.1109/ICISA.2011.5772421

[9] Imsamai, M., Phimoltares, S. (2010). 3D CAPTCHA: A next generation of the CAPTCHA. In 2010 International Conference on Information Science and Applications, pp. 1-8. https://doi.org/10.1109/ICISA.2010.5480258

[10] Gao, H., Yao, D., Liu, H., Liu, X., Wang, L. (2010). A novel image based CAPTCHA using jigsaw puzzle. In 2010 13th IEEE International Conference on Computational Science and Engineering, pp. 351-356. https://doi.org/10.1109/CSE.2010.53

[11] Vikram, S., Fan, Y., Gu, G. (2011). SEMAGE: A new image-based two-factor CAPTCHA. In Proceedings of the 27th Annual Computer Security Applications Conference, pp. 237-246. https://doi.org/10.1145/2076732.2076766

[12] Goswami, G., Powell, B.M., Vatsa, M., Singh, R., Noore, A. (2014). FaceDCAPTCHA: Face detection based color image CAPTCHA. Future Generation Computer Systems, 31: 59-68. https://doi.org/10.1016/j.future.2012.08.013

[13] Zhang, K., Zhang, Z., Li, Z., Qiao, Y. (2016). Joint face detection and alignment using multitask cascaded convolutional networks. IEEE Signal Processing Letters, 23(10): 1499-1503. https://doi.org/10.1109/LSP.2016.2603342

[14] Navabifar, F., Emadi, M., Yusof, R., Khalid, M. (2011). A short review paper on Face detection using machine learning. In Proceedings of the International Conference on Image Processing, Computer Vision, and Pattern Recognition (IPCV), 1.

[15] Sun, X., Wu, P., Hoi, S.C. (2018). Face detection using deep learning: An improved faster RCNN approach. Neurocomputing, 299: 42-50. https://doi.org/10.1016/j.neucom.2018.03.030

[16] Jiang, H., Learned-Miller, E. (2017). Face detection with the faster R-CNN. In 2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017), pp. 650-657. https://doi.org/10.1109/FG.2017.82

[17] Farfade, S.S., Saberian, M.J., Li, L.J. (2015). Multi-view face detection using deep convolutional neural networks. In Proceedings of the 5th ACM on International Conference on Multimedia Retrieval, pp. 643-650. https://doi.org/10.1145/2671188.2749408

[18] Mitra, S., Acharya, T. (2007). Systems man and cybernetics part C: Applications and reviews. IEEE Transactions on Gesture Recognition, 37(2007): 311-324. https://doi.org/10.1109/TSMCC.2007.893280

[19] More, S.P., Sattar, A. (2016). Hand gesture recognition system using image processing. In 2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT), pp. 671-675. https://doi.org/10.1109/ICEEOT.2016.7754766

[20] Ismail, A.P., Abd Aziz, F.A., Kasim, N.M., Daud, K. (2021). Hand gesture recognition on python and OpenCV. In IOP Conference Series: Materials Science and Engineering, 1045(1): 012043. https://doi.org/10.1088/1757-899X/1045/1/012043

[21] Nguyen, T.N., Huynh, H.H., Meunier, J. (2013). Static hand gesture recognition using artificial neural network. Journal of Image and Graphics, 1(1): 34-38. https://doi.org/10.12720/joig.1.1.34-38

[22] Ahmed, T. (2012). A neural network based real time hand gesture recognition system. International Journal of Computer Applications, 59(4): 17-22.

[23] Nagi, J., Ducatelle, F., Di Caro, G.A., Cireşan, D., Meier, U., Giusti, A., Gambardella, L.M. (2011). Max-pooling convolutional neural networks for vision-based hand gesture recognition. In 2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA), pp. 342-347. https://doi.org/10.1109/ICSIPA.2011.6144164

[24] Lin, H.I., Hsu, M.H., Chen, W.K. (2014). Human hand gesture recognition using a convolution neural network. In 2014 IEEE International Conference on Automation Science and Engineering (CASE), pp. 1038-1043. https://doi.org/10.1109/CoASE.2014.6899454

[25] Kim, Y., Toomajian, B. (2016). Hand gesture recognition using micro-Doppler signatures with convolutional neural network. IEEE Access, 4: 7125-7130. https://doi.org/10.1109/ACCESS.2016.2617282

[26] Sonkusare, J.S., Chopade, N.B., Sor, R., Tade, S.L. (2015). A review on hand gesture recognition system. In 2015 International Conference on Computing Communication Control and Automation, pp. 790-794.

https://doi.org/10.1109/ICCUBEA.2015.158

NOMENCLATURE

OCR	Optical Code Reader
CNN	Convolutional Neural Network
NN	Neural Network
MTCNN	Multi-task Cascaded Convolutional Networks
HMM	Hidden Markov model
FMS	Finite State Machine
SIFT	Scale Invariance Feature Transform
GMM	Gaussian Mixture model
SDAE	stacked denoising encoder
DCNN	Deep CNN