AnAssistive Model for Visually Impaired People using YOLO and MTCNN

FerdousiRahman,IsratJahanRitun,NafisaFarhin, JiaUddin
Department of Computer Science & Engineering
BRAC University
66 Mohakhali, Dhaka, Bangladesh
+88- 01780372177, 1686966990, 1621472151, 1817205771
{ferdousirahman707, isratritun,nafisafarhin}@gmail.com, jia.uddin@bracu.ac.bd

ABSTRACT
Visually impaired people face difficulties in safe and independent movement which deprive them from regular professional and social activities in both indoors and outdoors. Similarly they have distressin identification of surrounding environment fundamentals. This paper presents a model to detect brightness and major colors in real-time image by using RGB method by means of an external camera and then identification of fundamental objects as well as facial recognition from personal dataset. For the Object identification and Facial Recognition, YOLO Algorithm and MTCNN Networking are used, respectively. The software support is achieved by using OpenCV libraries of Python as well as implementing machine learning process. The major processor used for our model, Raspberry Pi scans and detects the facial edges via Pi camera and objects in the image are captured and recognized using mobile camera. Image recognition results are transferred to the blind users by means of text-to-speech library. The device portability is achieved by using a battery. The object detection process achieved 6-7 FPS processing with an accuracy rate of 63-80%. The face identification process achieved 80-100% accuracy.

CCS Concepts
• **Computing methodologies → Classification and regression trees**

Keywords
Visually Impaired; OpenCV; Object Identification; YOLO Algorithm; Deep Learning.

1. INTRODUCTION
According statistical analysis study of WHO (World Health Organization) [1], approximately 285 million people around the world are blind or have amblyopia, 246 million of whom have serious vision problems. Visually impaired people usually face difficulties in movement as well as identifying people and avoiding obstacles in their day to day activities. The conventional solutions to these situations are often seen to be usage of guide canes to detect obstacles in front of them or relay on vocal guessing for identification of persons. As an outcome, visually impaired people cannot predict the exact environment features about what types of objects lies in front of them or whom they are facing presence.

In this paper, the approach is to construct a module to feed the user with vital data such as individual identification and obstacle detection with better accuracy. Our proposed method is based on Deep Learning Algorithms and compatible and user friendly hardware for practical implementation. Our objective is to implement two different algorithms individually for face identification and object identification as well as establish a hardware setup for hands-on result of the executed software program. We will also state the detailed description of the steps involved for the coding execution as well as result analysis with accuracy rate after the system training and testing session.

The rest of the paper is organized as follows. Chapter 1 is for introduction to our proposed concept and implemented algorithm, motivation and objective overview. Chapter 2 holds the background analysis of previous works related to the concept and methods. Next Chapter 3 describes the extensive explanation of the software implementation of the system followed by Chapter 4 that explains practical implementation of the hardware setup and experimental results. Finally conclude the paper in Chapter 5.

ICCSP 2019, January 19–21, 2019, Kuala Lumpur, Malaysia

ACM ISBN 978-1-4503-6618-2/19/01…$15.00
https://doi.org/10.1145/3309074.3309114

2. BACKGROUND ANALYSIS

2.1 Literature Review
The basic concept of facial and object detection and identification system is a very commonly known factor. Not only for the aiding of visually impaired people, this notion is in implementation in many sectors such as security and industrial manufacturing. Different software system models are designed as such that it firstly takes the input images fetching from the database and implement the deep learning process to classify and then specifically identify the required result such as objects, facial identity or expression, forgery etc. for the real-time circumferences, the captured input images contain several entities and needs more efficient program to extract and derive the specified category and sometimes for real-time analysis, multiple detection are identified and it is a challenge to identify correctly.

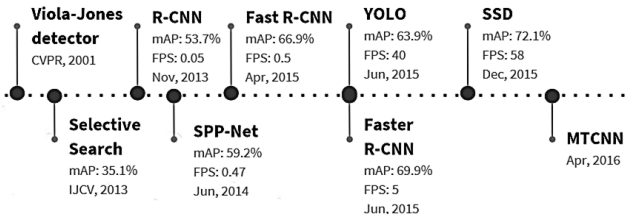

Figure 1. Comparative Analysis of different Algorithms in the field of Image Processing and Detection.

In the paper [2], Spatial Pyramid Pooling (SPP-net) has been executed that the feature map computation of the input is done only once and repetition is avoided, thus faster than R-CNN method still better accuracy achieved on Pascal VOC 2007. In [3], using the OpenCV libraries and basing on AdaBoost algorithm, the images are classified and pattern recognition feature of this algorithm gives advantage to use Haar Like features and output proper results. In [4], using Single Shot Detector (SSD), for the detection accuracy, predictions of different measures from feature maps of different scales are projected which are explicitly separate predictions by aspect ratio. Figure 1 shows the comparative analysis of different image processing algorithms.

2.2 Algorithms

2.2.1 YOLO (You only Look Once)

You Only Look Once[5] is a real-time object detection algorithm that uses a single convolutional network and is much more faster compared to the other identification systems, even in real-time, it performs 150fps, thus it is capable of processing live streaming video with less than 25 milliseconds of latency. Along with the fast processing time efficiency, this method still holds a respectable measurement of accuracy in identification. In this process, a single neural network predicts bounding boxes and class probabilities directly from full images in one evaluation.

In [5], YOLO has been established particularly fast system as it can process 45 frames per second in real-time also it has been outpacing the previous identification systems for its time-efficiency. The paper also represents the system as a single convolutional network instantaneously forecasting multiple bounding boxes and class probabilities for those boxes. YOLO trains on full images, straight optimizing detection performance. For training purposes, Darknet Framework has been implemented. A more rationalized and efficient approach was introduced in 2017, YOLO9000, which could detect and identify more than 9000 categories if objects in real-time [6]. This system was trained with both COCO detection dataset and the ImageNet classification dataset which gave the system advantage of more precise identification. This version implemented Batch Normalization which resulted in performance improvement for approximately 2% [6]. Also the hierarchical view of object classification permits to combine different datasets together.

2.2.2 MTCNN (Multi-task Cascaded Neural Networking)

Convolutional Neural Network is an artificial neural networking structure intended to evaluate imageries and visual dataset. MT-CNN or Multitask Cascaded Neural Networking is an advanced version of CNN [7,8]. In [9], the proposed model of MTCNN exploits the inherent correlation between detection and alignment of input frames to increase the performance. This major research gave us a lot of idea about the concept of MTCNN. Multi Task Cascaded Convolutional Network brings relation between detection and alignment. This process achieves superior accuracy over the stateof-the-art techniques on the challenging FDDB and WIDER FACE benchmarks for face detection, and AFLW benchmark for face alignment, while keeping real time performance. It involves bounding box calibration from face detection with additional computational expense and overlooks the inherent correlation between facial landmarks localization and bounding box regression. Paralleled to other multi-class objection detection and classification tasks, face detection is a challenging binary classification assignment, so it may need less numbers of filters per layer; by decreasing the filter values, this algorithm process can increase the depth so it will accelerate the performance of this algorithm.

3. ALGORITHM IMPLIMENTATION

3.1 Object Detection and Identification Methodology

YOLO is a jointly trained method for object detection and classification for real time video. Using this method YOLOv2 is trained simultaneously on the COCO detection dataset and ImageNet classification dataset. It is basically offering easy tradeoff between speed and accuracy and for this reason it is one of the most efficient algorithms for object detection in real time. YOLOv2 gets 40 FPS at 78.6 mAP on VOC 2007 whichis really fast for real time [10].

Yolo divides up the image into a grid of 13x13 cells and each of the cells is responsible for predicting 5 bounding boxes. A bounding box describes the rectangle that encloses an object. Yolo also outputs a confidence score that tells how certain it is that the predicting bounding box actually encloses some objects. For each bounding box the cell also predicts a class. The confidence score for the bounding box and the class prediction are combined into one final score that tells the probability that this bounding box contains a specific type of object. Since there are 13 ×13=169 grid cells and each cell predicts 5 bounding boxes, it ends up with 845 bounding boxes but in final result there will be just those bounding boxes which have higher score value than the threshold value. Even though there were 845 separate predictions, they are all made at the same time and the neural network just runs once and that is the reason of YOLO being so powerful and fast.

The original version of YOLO was implemented on Darknet [11] which a Deep learning framework written in C programming language and uses CUDA. But as this language is not user friendly so we have executed our program in python language which is compatible with hardware implementation and easy to execute. For program development, we used Darkflow framework, Tensorflow version. As we are using tiny-yolov2 model, we collected the weight file and required .cfg file from the website of YOLO [12].

3.1.1 Detection and Identification Processing in Real-time

For the computation of our model, firstly we imported a program instance from Darkflow and another library called numpy. The instance was specified with model, weights load and a threshold which is to specify the confidence factor for generating the bounding box window within an object named ‘Options’. Initially we intended to process a previously saved video input with a threshold value of 0.3 without training the system. Here we are setting a lower value of threshold to get more bounding boxes captured from the input video and as we are using CPU, we are processing 6-7 frames per second time. For the input captured

from the camera, a specific value of width and height of the captured data is declared in the system parameters. Then we have passed this information as parameter through TFNet in darkflow by creating tfnet. This is going to initialize our model that going to make prediction. The processing result is structured to contain the co-ordinates of the frame bounded in the input video images along with the confidence value and label name of the framed object.

In this model, we have used cv2.VideoCapture( ) to connect our webcam for real time input with the support of OpenCV. For identifying the object tfnet plays a vital role. tfnet is mainly where yolo does all the training process. After using all the libraries the TfNet class works with different methods. _TRAINER has all the optimizer in a directory. All there optimizer are used to optimize the performance of the Network. _get_fps method will get the Frame Rate which is the frequency of the video. The greater the FPS the smoother the video motion will be. For displaying our output we called capture.read () function which gives Boolean values. When the camera is on it will give true value and frame it. This frame has sent to TFNet as perameter through tfnet.return_predict ( ) function. In this way we will get result of identifying objects. Then for each object we set different boxes of different colors.

3.1.2 Custom-made Dataset Construction

For our system, we intended to develop our own personalized dataset created for the elected desired objects for which we collected the images from Google images. Python has a built-in library dedicated to generate such databases. After importing the library, we generated instances for each subject with argument of keyword and limitation number provided as parameters.

3.1.3 Annotation and generating .xml file

After creating our customized dataset we created our annotation files. In our annotation files we have crated xml file of every pictures. For creating xml file we have specified image folder directory, save directory where our annotated file will be saved and object name. First we display our image using *ax.imshow ()* function. Then we created a rectangular drag able selector tor select the object we want to detect by selecting the requiredobject from top left to bottom right.By selecting object with this drag able bounding box we will get 3 top left coordinates and 3 bottom right coordinates. Here we specified name of the object so every time we draw box it put that object name in the list**.**For getting xml file*cElementtTree* library and this is our main xml generating library that we gone use to generate our xml file. In this xml file we keep record path of the image, height, width, depth of the images.

3.1.4 Machine Training

For training our own dataset we have changed the number of classes and filter number of our .cfg file. Then we have also changed our label.txt file where we put all the labels of our classes.

3.1.5 Accuracy Analysis and Development

Initially after the model establishment, we run the model with the mobile camera connected to the computer via an application; we passed the captured data to the processing unit and successfully identified the object in real-time. The initial runtime accuracy without the training phase is shown in Figure 2.

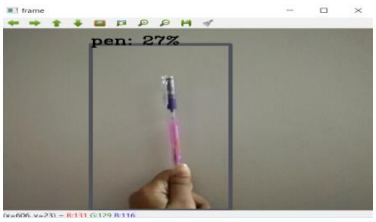

Figure 2.Accuracy at the initial stage of compilation of identification.

3.2 Face Detection and Identification Methodology

Based on the analysis of previous research works and implementation of system based on dissimilar algorithms, for our robust face identification, we have focused on implementing MT-CNN. The MTCNN algorithm basically mechanisms in three steps and uses one neural network for each. The first part is usually called a proposal network. It will predict potential face positions and their bounding boxes quickly through a shallow CNN. The outcome of this stage is a large number of face detections and lots of false detections. The second part uses images and outputs of the first prediction and improves the result to eliminate most of false detections and aggregate bounding boxes through a more complex CNN. The last part perfects even more the predictions and adds facial landmarks predictions.

3.2.1 Argument Parsing

For user-to-user customization, we have designed the system to develop with its own dataset collected from the user interface. In our project, a parameter of mode that controls the decision of when the camera will take new input data to store or only identify.

For the alignment, we implemented *Dlib* face alignment strategy. The distinctive fact in this method or approach is that it doesn't deform the original image. Here the whole face portion gets positioned in three parts, Right, Left, and Center. As a new input list, a loop runs to make 2 dimensional matrixes and the created matrix is transposed. The shapes gets computed by *Mean* and *Coefficient of Variable(cov)*. Then *Affine transformation* is applied on the matrix which basically refers to combination of linear transformations and translations. The Align face is done in *BGR format*. BGR format is the convention for OpenCV. So the term BGR (and BGRA) is a leaky abstraction where the graphics library is explaining how the integer is logically ordered. This makes our code more readable as that is directly accessing the color components individually.

We had to import *tensorflow* and *inception_resnet_v1*. The main hallmark of this architecture is the improved utilization of the computing resources inside the network. This was accomplished by a wisely constructed strategy that allows for increasing the depth and width of the network while keeping the computational budget constant. Here for tensorflow, *tf.placeholder*('float', [None,160,160,3]) is provided which is the default input for tf.nn.l2_normalize. Here we get the data that are already previously trained and stored in model folder. Previously trained images go through the matrix process and returned to main.py.

3.2.2 Detection using MTCNN

At this stage, *mtcnn_detect*function in called by main class. In this file the images uses Tensorflow implementation of the mtcnn face detection algorithm. The images go through image pyramid which helps in smoothing and subsampling images. Here for pyramid

representation by default it uses the value .709 as a parameter. Model path of the previously trained data also linked in this part. The threshold is of 70%, factor for smoothing is saved .709 and also scale_factor for scaling images.

3.2.3 Loading MTCNN Face Detection Model

The actual processing portion works on three stages: Proposal Network (P-Net), Refine Network (R-Net) and Output Network (O-Net).***Proposal Network (P-Net)***to obtain the candidate facial windows and their bounding box regression vectors. Then candidates are calibrated based on the estimated bounding box regression vectors. After that, we employ non-maximum suppression (NMS) to merge highly overlapped candidates.All candidates are fed to another CNN, called ***Refine Network (R-Net),***which further rejects a large number of false candidates, performs calibration with bounding box regression, and conducts NMS**. *Output Network (O-Net)*** is similar to the second stage, but in this stage we aim to identify face regions with more supervision. In particular, the network will output five facial landmarks positions.

After each convolutional layer, it is convention to apply a nonlinear layer (or ***activation layer***) immediately afterward. The purpose of this layer is to introduce nonlinearity to a system that basically has just been computing linear operations during the conv layers (just element wise multiplications and summations).In the past, nonlinear functions like tanh and sigmoid were used, but researchers found out that ***ReLU layers*** work far better because the network is able to train a lot faster (because of the computational efficiency) without making a significant difference to the accuracy. Another method called *max pooling* uses the maximum value from each of a cluster of neurons at the prior layer. This serves two main purposes- first is that the amount of parameters or weights is reduced by 75%, thus lessening the computation cost and second is that it will control ***overfitting***. This term refers to when a model is so tuned to the training examples that it is not able to generalize well for the validation and test sets [13].

3.2.4 Loading New Face Data

A function called *create_manual_data()*is used to create new dataset. Camera will open and wait till it gets the input name of the person. After getting the input it goes to the *facerec_128D.txt* and with the support of json.load parameters of a person, face image is stored. While storing the features of face, it maintains 3 positions which are left, right and center. But if data of a person already exists in the dataset then by *json.dumps(),* the previous features will be removed and new features will be added. When the user is done registering their data, they will press "q" to close the window.

If there is no "*input*" portion it will go to *camera_recog()* to detect face and will try to identify if the dataset has the person's data. If not then it will show "unknown". To recognize, the window will open with cv2 and will show "*camera sensor warming up*".

3.2.5 Compilation and Result Display

In this portion of the code previously explained codes are being compiled. So if the *face_features* matches with any dataset it will show the user name and with accuracy or distance between points saved in dataset and shown in camera in real time. This function basically does a simple linear search for the 128D vector with the min distance to the 128D vector of the face on screen. Code compilation is shown in Figure 3.

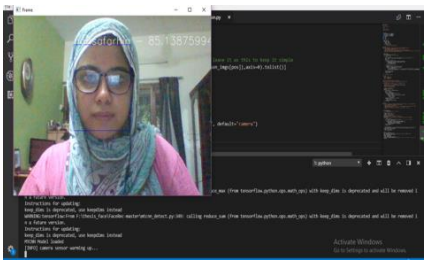

Figure 3.Code compilation and Face identification with Name Label and Accuracy Rate.

4. EXPERIMENTAL SETUP AND RESULT

The central processing unit of our proposed system setup is Raspberry Pi 3 Model B [14]. It is a single board operating system unit with built-in wireless LAN and Bluetooth connectivity. It is a vastly used efficient device for the implementation of our proposed method. For the input data collection of real-time image capturing, Raspberry Pi Camera Module V2 has been used. This module is resourceful for capturing high definition instantaneous video as well as still photographs. Raspberry Pi is powered using a portable battery or power bank.The upgraded switched Micro USB power source up to 2.5A is much efficient and user approachable. For our project, the output result has to be an audio output as the target users are visually impaired. So for user convenience we have used an earphone. Our project has two major individual identifications from which we are implementing hardware for the face recognition first. Figure 4 shows the element connection for our model.

Figure 4.Elements with basic connection wiring and device model.

For the video capture installation, we imported *picamera*which is a package for interfacing Raspberry Pi camera module and it is used companionable with python 3 versions which we have used in our identification processing system. Upon construction, this class initializes the Pi camera. *camera.start_preview()* is used to initial start the camera viewing. The resolution and frame rate are declared. Using *PiRGBArray*we are collecting the camera input with a proper size specified which is running in a loop for real-time video capturing.

For the audio output, we have to use a function that can covert text result into speech as our target uses cannot see the label of identified person. A *text-to-speech (TTS)* system converts normal language text into speech.*eSpeak* is a compact open source software speech synthesizer for English and other languages. After importing the package, it can compile speech from text. For our system, we firstly run the main identification code set up in the system with proper library and environment installation, then the identified result label is passed via an object variable to espeak

for text to speech conversion from which the result is passed into the audio device connected with the raspberry pi.

4.1 Object Identification Result

Figure 5 shows the result of our object detection. Here we have seen the bounded box surrounded a specific object. This specifies the machine has the capability to identify object from its surrounding. The code has its portion which has the capability to match the confidence factor with the previously trained dataset. By matching the confidence factor it finds the most appropriate confidence factor from the pre trained model and thus it represents the name shown in the Figure 5.

As the object's name is showing correctly so the identification has done properly. Next to the name of the object, a percentage number is shown; this number represents the accuracy rate of object identification. This percentage number is subject to maximum matching of the confidence factor in real time video frame with previously trained model confidence factor. Also there are values of Frame per Second (FPS) which shows how fast our Algorithm (YOLO) is able to detect object in real time. The percentage that shows the accuracy of identification of an object is low and the FPS is little bit low because the machine needs to train more on the dataset it has. For not having a GPU supported machine the system is taking more time to process the whole dataset.

Figure 5. Detected object with name label and accuracy rate.

4.2 Face Identification Result

Figure 6 shows the accuracy of face identification. Faces that are in the dataset that was taken as input matches with the faces in front of the camera. As the machine could match one of its dataset values with the bounded box it shows the output with accuracy which generally ranges within 78-100% depending on the light and facial circumstances. The bounded box shows the proper face detection.

Figure 6. Detected face with name label and accuracy rate.

The percentage is the accuracy that was calculated through code and the name with the percentage was shown according to the input user typed while giving their face data. How well the machine is able to detect and recognize depends on the percentage number that is shown in the figure. As the System uses Deep Learning so more time and data will be needed to provide more accurate result. As the system is still on improvement the dataset is small for now, it will have multiple people's faces in its set of data and will be able to detect and recognize multiple people at a time while the video is on real time.

5. CONCLUSIONS

This paper presents anobject detection model using YOLO algorithm for visually impaired peoples. MTCNN is utilized for building model. We have used Raspberry Pi to process the data and Pi camera to capture the images. To validate the model we use a dataset with different types of images. Experimental results demonestrate that the proposed model achieves 63-80% accuracy for 6-7 FPS. In addition, the face identification process achieved 80-100% accuracy. In future, we have planned to implement the proposed model in parallel environments.

6. REFERENCES

[1] Global Data on Visual Impairments 2010. Available online: http://www.who.int/blindness/GLOBALDATAFINALforweb.pdf (Aaccessed on 23 April 2017).

[2] He K., X. Zhang, S. Ren , J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition", "Fleet D., Pajdla T., Schiele B., Tuytelaars T. (eds) Computer Vision", Lecture Notes in Computer Science, vol 8691. Springer.

[3] Prof. P Y Kumbhar, Mohd Attaullah, S. Dhere, S. Kumar Hipparagi, "Real Time Face Detection and Tracking Using OpenCV," Int. Journal For Research In Emerging Science and Technology, Vol.4, No. 4, Apr-2017.

[4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, F.Cheng-Yang, C. Alexander Berg, "SSD: Single Shot MultiBox Detector," ver. 5, 2016,Cornell University Library.

[5] J. Redmon, S. Divvala, R. Girshick, A. Farhadi; "You Only Look Once: Unified, Real-Time Object Detection," IEEE Conference on Computer Vision and Pattern Recognition; Published on: 12 December 2016.

[6] Joseph Redmon, Ali Farhadi; YOLO9000: Better, Faster, Stronger; 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 09 November 2017.

[7] [Online]. https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/ (Accessed on 19th July 2018).

[8] M. Nakib, R. T. Khan, M. S. Hasan and J. Uddin, "Crime Scene Prediction by Detecting Threatening Objects Using Convolutional Neural Network," *Int. Conf. on Computer, Communication, Chemical, Material and Electronic Engineering*, pp. 1-4, 2018.

[9] K. Zhang, Z. Zhang, Z. Li and Y. Qiao, "Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks," in IEEE Signal Processing Letters, vol. 23, no. 10, pp. 1499-1503, Oct. 2016.

[10] S. Billotta, G. Bonanno, S. Garozzo, A. Grillo, D. Marano, G. Romeo, "You Only Look Once:Unified, Real-Time Object Detection," Nuclear Instruments and Methods in Physics Research, Section A: Accelerators, Spectrometers, Detectors and Assoc. Equip., vol. 794, pp. 185–192, 2015.

[11] J. Redmon, "Darknet: Open source neural networks in c." [onloine]. http://pjreddie.com/darknet/, 2013–2016.

[12] [Online]. https://pjreddie.com/darknet/yolo/ (Accessed on: 14th July 2018).

[13] M.S. Akbar, P. Sarkar, A.M. Ashray, A.T. Mansoor, J. Uddin,"Face Recognition and RFID Verified Attendance System," Int. Conf. on Emerging Technologies in Computing 2018, London, UK.

[14] [Online]. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ (Accessed on : 10th July 2018).