RESEARCHERID
THOMSON REUTERS

[ICRTCET-2018]

ISSN 2348 – 8034
Impact Factor- 5.070

GLOBAL JOURNAL OF ENGINEERING SCIENCE AND RESEARCHES

PROVIDING VISION TO THE VISUALLY IMPAIRED USING IMAGE PROCESSING

Raj Chandak[*1], Parth Doshi[2], Bhavin Sarvaiya[3] & Manasi Kulkarni[4]

[*1,2&3]Bachelor of Technology, Computer Engineering, Veermata Jijabai Technological Institute Mumbai, India

[4]Assistant Professor, Computer Engineering, Veermata Jijabai Technological Institute, Mumbai, India

ABSTRACT

Ability to see and recognize is one of the most important and rudimentary skills that any human needs. However, not everyone is blessed with this ability, which puts them at a serious disadvantage. Hence, to assist the visually impaired, we created an application that is designed to aid them in their everyday state of affairs. This research paper describes a project is tailor made to provide the ability to identify and recognize objects. It includes three features: Object Recognition, Emotion (facial expression) Recognition and Barcode Scanning. Building on the principles of image processing, convolutional neural networks and data science, we have accomplished a working version of this application which can classify 100 objects through the object recognition module, 7 emotions through the facial expression recognition module and scanning and identification of bar codes of 1000 objects through the barcode scanning module.

Keywords: *Image Processing, Neural Network, Machine Learning, Object Recognition.*

I. INTRODUCTION

With a recent surge in Artificial Intelligence systems, applications of the same competent in recognizing objects and emotions (facial expressions) have been of much research interest and have spanned across domains such as interactive customer marketing, robotic interfaces, health monitoring, image & video stabilization, optical character recognition and many more. In light of the substantial role that object and emotion recognition systems have played across such varied domains, it has been a common mistake to generalize the ease of use of such systems for all community groups. Such computer vision systems have not been able to make any substantial impact on those groups which lack the rudimentary skills required to use them. This has served as a motivation for us to develop an application which is specifically configured to cater to the needs of one such group (visually impaired). Our objective is to use deep learning and image processing to span the repertoire of objects that a visually challenged individual is capable to recognizing. In the following sections, we will discuss the currently available systems and solutions, research done, give an overview and description of the modules that we have implemented, analyze and conclude from the results that we have obtained and will finally discuss the future scope of this project.

II. LITERATURE SURVEY

The history of Convolutional Neural Networks(CNN) can be traced back to its formulation by Yann LeCun. However, the first successful application was the development of AlexNet Architecture by Alex Krizhevsky, Ilya Sutskever and Geoff Hinton [3]. AlexNet won the ILSVRC challenge 2012, out-performing the runner up by a substantial margin. This opened flood gates and led to the development of ZFNet and GoogleNet, the later of which led to the development of inception module capable of classifying 1000 classes. Since then, CNNs have been used for classifying images. But, for a long period, they were capable of classifying only a single object in the image. In order to classify multiple objects, one approach is to slide the classifier from left to right, evaluating the corresponding windows and integrate the results together. However, this approach is not feasible for the proposed system as we need faster processing and lesser response times. Thus, we decided to use tiny YOLO (you only look once) which predicts class probabilities from image pixels and creates bounding boxes.

Recent applications of CNNs also include their use in facial expression recognition analyses. EmotiW is a popular competition where participants use their model to train and test on a common dataset. Yu and Zhang were the first

ones to show that state-of-the-art accuracies can be achieved by using a convolutional dataset [11]. Since then, CNNs have been used frequently used in Facial Expression Recognition (FER). There are multiple datasets available such as CK, CK+ and Kaggle which offer a train set, test set and validation set. We have used Kaggle's Facial Expression Recognition Challenge Dataset [1] as we found the dataset to be more robust in terms of the number and type of images available in the training set. Accuracies ranging from a lower threshold of 0.48 to a higher threshold of 0.61 has been achieved on this dataset so far.

Another aspect of the proposed system is the use of barcodes on commercial products. A barcode is a machine-readable code that is used to represent information about the object that carries it. Barcodes such as UPC & EAN have been adopted by manufacturers from all over the world across all industrial domains as the standard to represent product information. Commercial/retail products carry 1-Dimensional (1D) barcodes generally of the following 2 types – Universal Product Code (UPC) & European Article Number (EAN). Commonly used variations of UPC include UPC-A & UPC-E, whereas those of EAN include EAN-13, EAN-8, ISBN & ISSN. Table 1 shows the variations among these barcodes.

TABLE 1. Types of Barcodes

Type of Barcode	Digits Represented
UPC-A	12
UPC-E	6
EAN-13	13
EAN-8	8
ISBN	13
ISSN	8

For the purpose of detection of these barcodes, the native Android SDK has been used that packages a Barcode Detector class by default. This class can scan in real-time, detect multiple barcodes at once and has the capability of scanning both 1-D and 2-D Barcodes (QR Codes).

Recently Microsoft has released an application – Seeing AI [7], which can recognize people and can predict their emotion. The application also offers scanning of barcodes, documents and is capable of recognizing US currency. The proposed system is capable of performing these functionalities in addition to object recognition.

III. PROPOSED SYSTEM

The Object Recognition module, Facial Expression Recognition module and Barcode Scanning module, integrated together will facilitate the user in recognizing a diverse variety of objects spanning across multiple domains. In order to facilitate easy use, each of these modules can be activated with a single swipe. – right swipe for object and facial expression recognition modules and left swipe for barcode scanning module. Fig. 1 shows the Data Flow Diagram for the proposed system.

ISSN 2348 – 8034
Impact Factor- 5.070

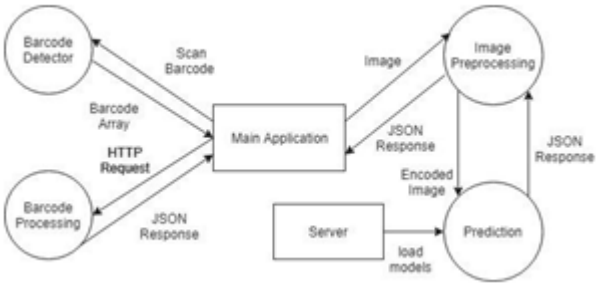

Figure 1: DFD Level 1

The proceeding subsections: (1), (2) and (3) capture the dataset, architecture and implementation details of Object Recognition, Facial Expression Recognition and Barcode Scanning modules.

1. Object recognition

Open Images Dataset (V3)
In the proposed system, the Open Images Dataset V3, which has approximately 9 million URLs to images of more than 1000 images [10], has been used. Out of the several options available, the human -verified-image -level annotations for training have been used. In addition to the already pre- trained tiny YOLO model which is capable of classification of 20 class of objects, we trained the model for 80 more object classes. The selection of these classes has been done by taking into consideration factors such as frequency of encounter, resemblance to similar objects and relevance in frequent scenarios. For each class of object, we have used a minimum of 2000 and a maximum of 3000 images for training.

Tiny YOLO Architecture
The proposed system uses the tiny YOLO architecture instead of the original YOLO architecture since the former has faster processing power than the later. As the model is to be deployed on an android device, we need an architecture that could process an image at a higher fps to give back results with minimal latency. Tiny YOLO is lightweight with the number of layers being reduced to 15. Table 2 shows the Tiny YOLO architecture.

TABLE 2. Tiny YOLO Architecture [12

Layer	Kernel	Stride	Output Shape
Input			(416,416,3)
Convolution	3X3	1	(416,416,16)
MaxPooling	2X2	2	(208,208,16)
Convolution	3X3	1	(208,208,32)
MaxPooling	2X2	2	(104,104,32)
Convolution	3X3	1	(104,104,64)
MaxPooling	2X2	2	(52,52,64)
Convolution	3X3	1	(52,52,128)
MaxPooling	2X2	2	(26,26,128)
Convolution	3X3	1	(26,26,256)
MaxPooling	2X2	2	(13,13,256)
Convolution	3X3	1	(13,13,512)
MaxPooling	2X2	1	(13,13,512)
Convolution	3X3	1	(13,13,1024)

Convolution	3X3	1	(13,13,1024)
Convolution	1X1	1	(13,13,125)

Object Recognition: Implementation Details

The Open Images Dataset assigned a unique code to each object category. The first step was to get the mappings between objects and corresponding codes. Every image in the dataset is given a unique ID. Using this, the text files for each image were obtained, which contained multiple rows of text with each row containing the class name, xmin, xmax, ymin, and ymax. However, YOLO requires all its metadata to be in xml format. Thus, these text files were converted into corresponding xml files while also using the urls to scrape the internet and fetch the corresponding images. Also, tiny YOLO needs the coordinate values in absolute terms but the dataset had values in relative terms. Hence the following conversion was used to bring the data in the required format.

$$Xmin/Xmax = Xmin/Xmax * width\ of\ image \quad (1)$$

$$Ymin/Ymax = Ymin/Ymax * height\ of\ image \quad (2)$$

The images were then fed into the tiny YOLO neural network. The system is divided into 7*7 grids with each box containing 2 bounding boxes and corresponding confidence scores. Each bounding box consists of 5 predictions: x, y, w, h, and con¿dence. Each grid cell also predicts 80 conditional class probabilities, Pr(Classi|Object). Thus, the class specific confidence scores are given by [9]:

□□□ = (3)

□□□

2. Facial expression recognition

FER Dataset

The Fer Dataset was used for facial expression recognition. It consists of 48x48 pixel grayscale images of faces [11]. Each image (containing a person's face) has been classified into one of the 7 predefined categories. The training images are in the form of pixels separated by spaces in one column of a csv file while the corresponding emotion (a number in the range 0-6) is in the next column. The training set consists of 28,709 examples while the test set consists of 3,589 examples [1].

Proposed Architecture

Table 3 shows the proposed architecture for Facial Expression Recognition.

TABLE 3. Proposed Architecture

Layer	Kernel	Stride	Output Shape
Input			(48,48,1)
Convolution	5X5	1	(48,48,128)
MaxPooling	2X2	2	(24,24,128)
Convolution	5X5	1	(24,24,256)
MaxPooling	2X2	2	(12,12,256)
Convolution	5X5	1	(12,12,512)
MaxPooling	2X2	2	(6,6,512)
Flat Layer			(6*6*512)
Dropout Layer			(0.3*6*6*512)

Flat Layer			(2048*1)
Flat Layer			(3072*1)
Flat Layer			(7*1)

Facial Expression Recognition: Implementation Details
A total of seven CNN models were tried and a class was written for each. The training was done on a google cloud server with 8 vCPUs and 30 GB ram memory. The training was initially set for 200 epochs but the model which gave the best accuracy reached a stabilization point after 30 epochs. The training took a minimum of 1 to a maximum of 3 days depending on how heavy the model was. The rectified linear unit activation function has been used for the all the layers except for the last one.

$$f(x) = max(0,x) \quad (4)$$

For the last layer in our best model, a softmax activation function was used:

$$\iota: {}^{K} \dashrightarrow (0,1)^{K} \quad (5)$$

$$\iota(z)_j = \frac{}{\sigma} \text{ for } j = 1...K \quad (6)$$

3. Barcode detection

Product Database
In order to retrieve the information of a product corresponding to a particular barcode, a database of nearly 1000 products was created that stored the barcode number and all relevant information about that product. This database was created using MySQL. Table 4 depicts the structure of this database.

TABLE 4. Database structure

#	Name	Type	Collation	Null	Default
1	Barcode Number	Varchar(32)	Latin1_swedish_ci	No	None
2	Product Name	Varchar(256)	Latin1_swedish_ci	No	None
3	Manufacturer Name	Varchar(256)	Latin1_swedish_ci	No	None
4	Price	float		No	None
5	Volume	float		No	None
6	Weight	float		No	None

System Design

ISSN 2348 – 8034
Impact Factor- 5.070

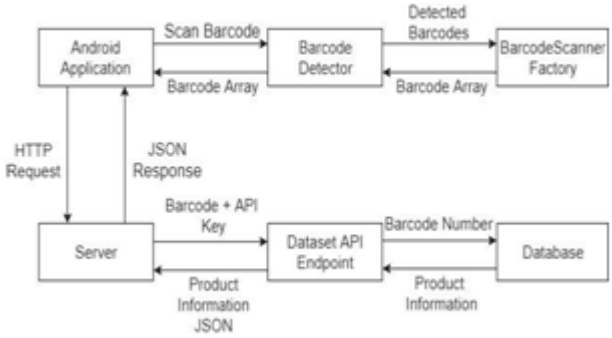

Figure 2: High Level Architecture

Barcode Detection: Implementation and Output

Barcode Detector
The BarcodeDetector class of the native Android SDK was used to scan barcodes using the Smartphone's camera. The default BarcodeTrackerFactory class was used to create a set of BarcodeGraphic classes, one for each detected barcode. Each BarcodeGraphic retrieved the value stored in the barcode, i.e. the barcode number and returned the result back to the BarcodeTrackerFactory. Finally, the BarcodeDetector combined the results retrieved from the BarcodeTrackerFactory and returned the result in the form of an array that stored all the barcode numbers captured from the field of view of the camera. Fig. 3 explains the BarcodeDetector processing pipeline.

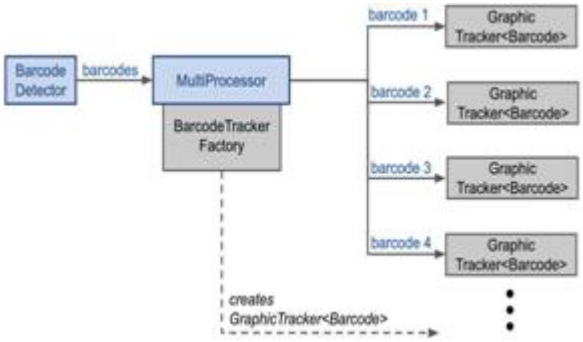

Figure 3: Barcode Detector Processing Pipeline [5]

Input: Camera's field of view that has commercially packaged products with barcodes on their packaging.

Output: Barcode Array of all the barcode values present in the input field of view.

API Interfacing
Due to the absence of an openly available dataset that provides relevant information about products with respect to their barcode, we created one using the aforementioned database structure. In order to fetch this data from the database, an API End-Point was created that acted as an interface between the mobile application and the MySQL database, as seen in Fig.4 below:

ISSN 2348 – 8034
Impact Factor- 5.070

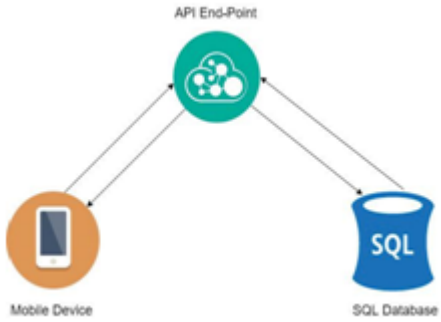

Figure 4: API Interfacing

To create this API End-Point, the Slim PHP Micro Framework was used, which is a very light weight and clean framework that provides a middle layer architecture, useful for filtering incoming requests and verifying the API Key that is generated for each user. After the barcode value was fetched by the application, it sent an HTTP request to the API with the API Key in the message header. After fetching the API Key using the Authorization field, we searched the database for a matching API key and got the appropriate user. If the API key was not present in users table, then the execution was stopped and the JSON error was echoed. Once the key was validated, the request was sent to the database to fetch the product information corresponding to the scanned barcode. The response was received by the application in JSON format.

Input: API Key & Barcode Number
Output: Product Information

JSON Parsing
Finally, the response received by the application was parsed using Android SDK's standard JSON parser. This response was parsed into String format and was then spoken out using the in-built Text-To-Speech Engine that is part of every Android Smartphone by default.

Input: JSON Response

Output: Spoken out product information

IV. OUTPUT AND RESULTS

The model, developed for **Object Recognition** was used to predict the test images present in the Open Images dataset. A collection of 962 images where chosen in such a way that each contained at least one of the classes that the model was trained for. Predicting objects in each of those 962 images, there were 899 instances or images where the predictions about the objects made overlapped with the human verified labels assigned to the images. Thus, an accuracy of 93.45% was achieved with the trained model. The model failed to converge with it getting stuck around a local optimum after a few initial epochs. Fig 5 shows a graph where loss is plotted as a function of epochs.

ISSN 2348 – 8034
Impact Factor- 5.070

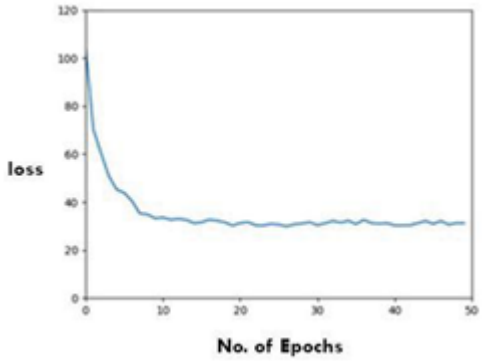

Figure 5: Loss as a function of Epochs (Learning Rate = 0.001) for Object Recognition

Using tensorflow's tensorboard, a visualization of kernel values and scalar values such as learning rate, accuracy and error loss was created. This led to the modification of the parameters which resulted in a model that converged and reached stabilization point after 60 epochs. The graph shown in Fig. 6, where loss is plotted as a function of epochs illustrates the same:

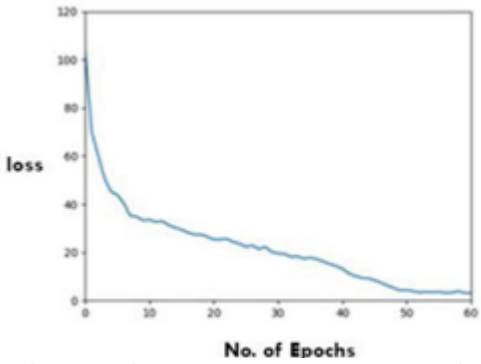

Figure 6: Loss as a function of Epochs (Learning Rate = 0.01) for Object Recognition

For Facial Expression Recognition, a total of 7 CNN models were tried, with models from 1-6 having three convolutional and three max pooling layers followed by two 3 fully connected layers while model 7 had 4 fully connected layers instead of 3. For all 7 models, a rectified linear unit activation function was used for all the layers except the last layer and we varied the number of kernels. For the 1st two models, tanh was used as the activation function for the last layer. We observed that as the number of kernels were increased from Model 1 to Model 2, the accuracy increased; as more kernels meant more detailed features. However, when the kernels were increased further, the model became resource and

time intensive. Thus, we concluded that a maximum accuracy of 53.38% was possible with Model2. From Models 3-6, the softmax activation function was used for the last layer and more filters were added progressively in multiples of 2 to the preceding layers. The accuracy increased linearly with the number of filters used while not making the models resource intensive. However, when another full connected layer was added, the accuracy decreased. For all the models described above, changes in some of the parameters were done on a trial and error basis. Fig 7. Summarizes the accuracies for all the models that have been tried:

ISSN 2348 – 8034
Impact Factor- 5.070

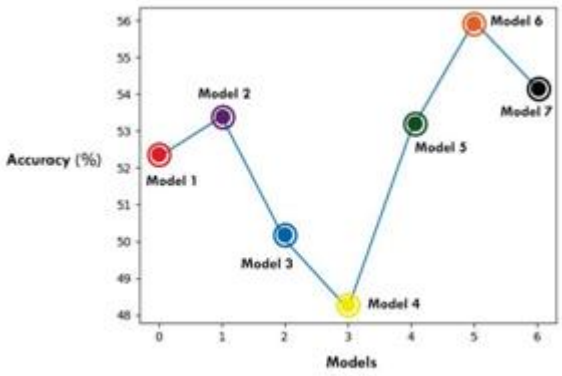

Figure 7: Accuracy of models tried

The learning rate was initially set to 0.001. But, we observed that the model got stuck at a local optimum without being able to escape from the same. As we know, learning rate decides how big a step the model takes. Since there were no improvements for many iterations (local optimum), we decided to increase the learning rate to 0.01. The model was able to go beyond the local optimum and finally converged and stabilized around the global optimum. Thus, the best accuracy of 55.97% was achieved with Model 6 where the learning rate was set to 0.01. Fig. 8 depicts loss function values as a function of epochs is plotted as follows:

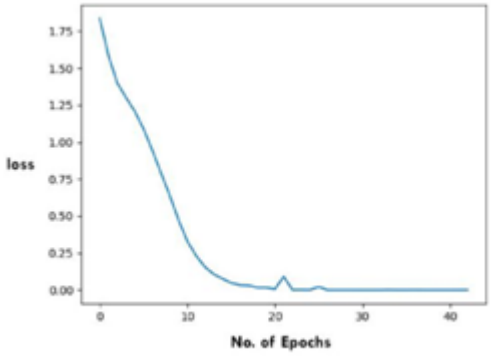

Figure 8: Loss as a function of Epochs for Facial Expression Recognition

V. CONCLUSION

This paper has described a project with three modules, all aimed at helping and facilitating the visually impaired to recognize a wide range of objects and human emotions. For **Object Recognition**, the tiny-yolo architecture was used to train and classify a total of 100 objects which are most likely to be used or seen in everyday life, and an accuracy of 93.45% was achieved with the test set. For **Facial Expression Recognition**, our proposed architecture, which was trained on the FER dataset, could achieve an accuracy of approximately 56% on the test set. For **Barcode Detection**, our system was able to detect a total of 1000 commercially packaged objects that were included in the database.

The aforementioned modules integrated together facilitates the user in recognizing more than 1000 day-to-day objects as well as facial expressions simply by using his smart phone. The application does not have any external dependencies and be easily installed and used on an android device.

The above modules need internet connection as the processing is done on a remote server. Although, the off premises processing and storage facilitates higher processing and response times, latency becomes a limitation as the response time depends on the relative positions of the device and server.

VI. FUTURE SCOPE

The project intends to expand the number of objects that the system is currently able to recognize through **Object Recognition** and **Barcode Scanning** modules. The proposed architecture for **Facial Expression Recognition** can be further enhanced to achieve a higher accuracy. However, enhancing the architecture may lead to slower processing time and hence a trade-off between the two may have to be made. For all three modules, the response time limitation induced by latency can be minimized by using a cloudlet.

REFERENCES

1. *Challenge, K. F. (May, 2013). Challenges in Representation Learning: Facial Expression Recognition Challenge. Retrieved from https://www.kaggle.com/c/challenges-in-representation-learning-facial-expression-recognition-challenge*
2. *Deshpande, A. (July, 2016). A Beginner's Guide To Understanding Convolutional Neural Networks. Retrieved from https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/*
3. *Deshpande A. (August, 2017). A Beginner's Guide to Understanding Convolutional Neural Networks. Retrieved from: https://dzone.com/articles/the-9-deep-learning-papers-you-need-to-know-about-1?fromrel=true*
4. *Ghosh, S. K. (June, 2017). Cloud Computing Course. Retrieved from Nptel: http://nptel.ac.in/courses/106105167/32*
5. *Google. (February, 2017). Mobile Vision. Retrieved from https://developers.google.com/vision/android/multi-tracker-tutorial*
6. *Huang J, R. V. (August, 2017). Speed/accuracy trade-offs for modern convolutional object detectors. Retrieved from https://github.com/tensorflow/models/tree/master/research/object_detection*
7. *James-Vincent. (July, 2017). Microsoft's new iPhone app narrates the world for blind people. Retrieved from https://www.theverge.com/2017/7/12/15958174/microsoft-ai-seeing-app-blind-ios*
8. *Joseph Redmon, S. D.-Y.-T. (January, 2018). Retrieved from https://pjreddie.com/media/files/papers/yolo.pdf*
9. *Khan, B. (October, 2015). Android Upload Image to Server using Volley Tutorial. Retrieved from Simplified Coding: https://www.simplifiedcoding.net/upload-image-to-server/*
10. *Krasin I., D. T.-E.-H. (November, 2017). Open Images V3 Dataset. Retrieved from Github: https://github.com/openimages/dataset*
11. *Raghuvanshi, A. a. (March, 2016). Facial Expression Recognition with Convolutional Neural Networks. Retrieved from http://cs231n.stanford.edu/reports/2016/pdfs/023_Report.pdf*
12. *Redmon, J. a. (January, 2018). YOLO:v3, An Incremental Improvement . Retrieved from https://pjreddie.com/darknet/yolo/*
13. *Redmon J. ,Farhadi A. (December, 2016). YOLO9000: Better, Faster, Stronger. Retrieved from: https://arxiv.org/pdf/1612.08242v1.pdf*
14. *Trieu. (August, 2016). Translating Darkflow to Tensorflow. Retrieved from https://github.com/thtrieu/darkflow*
15. *Wijaya, A. A. (April, 2016). Upload file with Multipart Request Volley Android. Retrieved from Github: https://gist.github.com/anggadarkprince/a7c536da091f4b26bb4abf2f92926594.*