OBJECT IDENTIFICATION USING MOBILE DEVICE FOR VISUALLY IMPAIRED PERSON

Thesis

Submitted to

The College of Arts and Sciences of the

UNIVERSITY OF DAYTON

In Partial Fulfillment of the Requirements for

The Degree of

Master of Computer Science

By

Deepika Akarapu

Dayton, Ohio

August 2021

OBJECT IDENTIFICATION USING MOBILE DEVICE FOR VISUALLY IMPAIRED PERSON

Name: Akarapu, Deepika

APPROVED BY:

_____
Mehdi R. Zargham, Ph.D.
Faculty Advisor

_____
Raghava Gowda, Ph.D.
Committee Member

_____
Tom Ongwere, Ph.D.
Committee Member

ABSTRACT

OBJECT IDENTIFICATION USING MOBILE DEVICE FOR VISUALLY IMPAIRED PERSON

Name: Akarapu, Deepika
University of Dayton

Advisor: Dr. Mehdi R. Zargham

The human eye perceives up to 80% of all the impressions and acts as the best shield from threat. While it is believed and accepted that vision is a predominant sense in people, as per the World Health Organization, around 40 million individuals on the planet are blind and 250 million have some type of visual disability. As a result, a lot of research and papers are being suggested to create accurate and efficient navigation models utilizing computer vision and deep learning approaches. These models should be fast and efficient, and they should be able to run on low-power mobile devices to provide real-time outdoor assistance. Our objective is to extract and categorize the information from the live stream and provide audio feedback to the user within the University campus. The classification of the objects in the stream is done by a CNN model and sent as an input for the voice feedback, which is divided into several frames using the OpenCV library and converted to audio information for the user in the real-time environment using the Google text to speech module. The results generated by the CNN model for image classification have an accuracy of over 95 percent, and real-time audio conversion is a rapid transition technique, resulting in an algorithm that performs competing with other prior state-of-art methods. We also want to

integrate the application in smartphones, into our mobile app to provide a more user-friendly experience for the end-users.

Dedicated to my Amma and Papa (Mom & Dad)

ACKNOWLEDGMENTS

I would like to express my gratitude to my advisor Dr. Mehdi Zargham for his constant support and guidance with patience throughout my coursework and thesis and for making this a worthwhile learning experience for me. I would like to thank Dr. Tom Ongwere and Dr. Raghava Gowda for serving as the committee members. Special thanks to my family and friends who helped me and made me believe in myself in every stage of this journey.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND NOTATIONS

API- Application Programming Interface

APK – Android Application Package

CNN – Convolution Neural Network

DNN – Deep Neural Network

GPU – Graphical Processing Unit

GPS – Global Positioning System

RCNN – Region-Based Convolution Neural Network

ReLU – Rectified Linear Unit

TF – TensorFlow

TFL – TensorFlow Lite

TPU – Tensor Processing Unit

gTTS – Google Text-to-Speech

CHAPTER I

INTRODUCTION

One of the most challenging aspects for a visually impaired person is gaining independence in handling daily duties and understanding the situation and the place they are present in by recognizing objects in his surroundings. For blind assistance, many algorithms are presented concentrating on multiple sensors detection, better accuracy, and low computational device models. Since the 1970s, object categorization and detection have improved and grown to the point where intriguing applications for visual replacement are becoming viable. This leads up to ever-changing and advanced methods in deep learning, implemented every year for more accurate results. Getting improved accuracy and latency, overcoming constraints from prior work are the essential characteristics of enhancement. Although these methods assist to improve the focused characteristic, they compromise on other essential criteria like setup cost, reduced dependencies within the internal layers, and also from external sources. Numerous sensors produce reliable findings, but the setup cost and real-time result derivation are compromised. The same is true for complex architecture, which may be more accurate but requires a lot of resources and more dependencies between them, and cannot be implemented on a low computing device, and vice versa.

1.1 Thesis Contribution

Taking the above-mentioned scenarios into account, the following are the contributions of this thesis:

(i) We proposed an idea for object identification through Convolutional Neural Network (CNN) using Keras framework followed by a voice alert using gTTS through the video stream processing using OpenCV, for a blind person to understand his surroundings (see figure 1.1 and 1.2 below).

(ii) We used a pre-trained model Mobilenet network for our image classification with CNN using the TensorFlow Lite framework in one of our approaches.

(iii) We compared a model built from scratch and a Mobilenet based classifier model that could best fit this application.

(iv) We created a dataset, comprised of 10 classes of different outdoor objects, gathered from the University of Dayton campus.

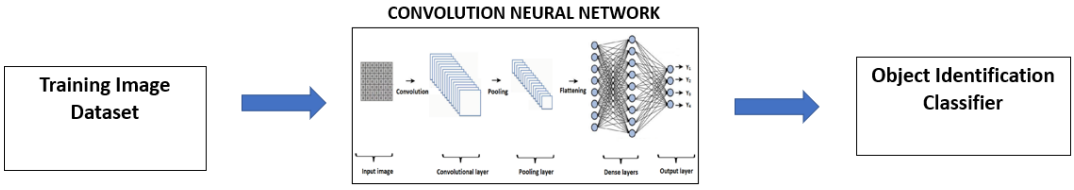

Figure 1. 1: Training Flow of the Object Identification Classifier

Figure 1. 2: Flow representing the conversion of the video stream to audio alert

1.2 Thesis Overview

This thesis is organized in the following manner: Chapter II summarizes the literature on the topic of convolutional neural networks, pre-trained models, and relevant frameworks for mobile deep learning algorithms and their applications in supervised learning. Chapter III provides a detailed implementation description of the data collection and preprocessing, algorithm development using frameworks, and inference steps. Chapter IV presents the experimental results with performance validation using statistical evaluation metrics for classification. Finally, the conclusion of the paper is drawn in Chapter V.

CHAPTER II

LITERATURE SURVEY

There are many real-time-based implementations conducted with various classification algorithms in different possible ways to make an inexpensive, non-intrusive and simple approach to result in efficient smart systems for blind assistance. Over the past decade, there are numerable technological methods such as GPS devices, echolocation, sensors and microcontrollers, and smartphones replacing the traditional methods like a blind stick, guide dogs, etc., for their mobility purposes. In this work, we discuss the approaches implemented currently being used in supporting blind people, we provide a brief CNN background, and finally provide a comparison between CNN and other traditional approaches, between the existing framework platforms, and between publicly available pre-trained models.

2.1 Previous Approaches

(i) *Conversion of the image to a sound format:* This approach was proposed by Krishnan et al., 2013 [1]. They use the image processing - canny edge detection concept that uses a multi-stage algorithm to identify objects by the wide range of edges and comparing those values with pre-trained dimensions followed by a speech sound. However, it is observed that the algorithm worked well for simple images but requires improvements in the case of complex images especially for overlapping of images [1]. We believe that though the canny edge

the detector works very well for images with their localization being immune to a noisy environment, it consumes a lot of time for computations resulting in delayed speech response for blind navigation.

(ii) *The second consideration is a voice-assisted navigation system for the blind:* This tool was designed by Noorithaya, Kumar & Sreedevi, 2014 [2], using affordable navigators, ultrasonic sensors processed by the microcontroller to identify the sudden changes in ground gradient or obstacles in the front and give voice instruction by using an mp3 module associated with the system to guide the destination path for the person, through the installment in their cane. They have GPS incorporated that guides through an android app that sets destinations to and from. With this, an obstacle of as close as 4cm can be detected. Yet, due to constraints like signal loss and navigation in indoor environments, GPS-based solutions cannot be completely reliable [3].

(iii) *The third consideration is a smartphone-based classification system:* Tapu et al., 2013 designed a system to detect obstacles in video streams in any kind of dynamic background for autonomous navigation is emphasized [3]. The obstacles are detected using the multiple Lucas-Kanade algorithm and identify background motion by Random Sample Consensus (RANSAC) algorithm. The obstacles are marked as urgent or normal depending on the distance from the camera. The identified images are resized, divided into cells to extract interesting points using the Histogram of Gradients (HOG) descriptor combined with Bag of visual words to capture the structure from the surroundings. The go

through the Support Vector Machine (SVM) classifier, and the object is labeled based on the ranking of the images from the distance seen in histograms computed using L2 normalization. This system is a smartphone attached to the user with the help of a chest-mounted harness and the technique could detect both static and dynamic obstacles and classify them based on the relevance and degree of danger to a blind person, but we are concerned that the implementation of this approach is involved with multiple algorithms creating a high rate of dependency among each other leading to chances of effect on overall performance in case of even one functional delay.

(iv) *The fourth consideration is the proposal based on Raspberry Pi:* The Raspberry pi was developed by Durga Devi et al., developed that comprises a camera and speaker [4]. Raspberry Pi is powered up by 5V DC, the camera is used to capture the images which can be interfaced by using the Camera Slot Interface with a raspberry pi board for further processing. The region of interest method is used to localize and identify the test. The python-based image processing takes place converting the image into audio signals as the output which redirects through the audio jack in which the earphones are connected. Raspberry pi has low computation power [5] due to which we suppose that the voice feedback to the users is delayed, which is the most critical part of the present application.

(v) *Finally, the smart stick:* Rohit et al. proposed a smart stick to overcome the drawback of raspberry pi discussed above by introducing a server for the computations [5]. The camera with a controller is capable of switching in any

direction to capture the images, which are processed and encoded in raspberry pi, forwarding the data to the server through Google's Gmail REST API, where the MASK RCNN algorithm runs to do object instance segmentation to detect objects in the image along with a segmentation mask over each instance. Each image would have its unique ID to avoid image replacements. Since the server can handle multiple requests simultaneously, it processes the images after the algorithm run and sends back an encoded output into the raspberry pi, the output is further decoded here, and the content is read aloud (audio) twice to the user. The experimental results yielded were good enough to reach the expectations, tested for a wide range of objects in various conditions but their primary future goal is to replace raspberry pi with a better efficient controller to avoid the dependency over small and slow computing systems which needs external power backup.

Though all the mentioned approaches work well in terms of accuracy, detection, audio signals, etc., there are certain compromises to consider on the other hand. As previously stated in the introduction chapter, complex architecture may be more accurate but requires multiple resources which cannot be implemented on a low computing device. A finer and straightforward implementation that has reduced steps of transfers between capturing, processing, sending the output is more accurate considering factors like time-saving, less complexity between connections, and their reduced dependencies resulting in good latency. Hence, a well-trained classification model that detects objects followed by voice alert,

deployed onto a mobile application is an approach that can be emphasized more for future research.

2.2 CNN Background

CNN's have proven to be very efficient in solving image-based problems and new architectures continue to progress in performance [6]. It is known to be popular, and promising for image recognition and processing, especially designed to process pixel-level data based on supervised learning with a high level of accuracy. Due to the large learning capacity, dominant expressive power, and hierarchical structure of CNNs, a high-level, semantic, and robust feature representation for each region proposal can be obtained. They are extensively applied to many research fields, such as image super-resolution reconstruction, image classification, image retrieval, face recognition, pedestrian detection, and video analysis [7].

It may owe to the contribution of Hinton's group, who proposed the Backpropagation algorithm, their continuous efforts have demonstrated that deep learning would bring a revolutionary breakthrough on grand challenges. Their success results from training a large CNN on 1.2 million labeled images together with fewer techniques [8] (e.g., ReLU operation and 'dropout' regularization). CNN is the most representative model of deep learning [8]. The classification of images is done into various categories after getting a specific conditional probability for each output neuron in the final dense layer.

2.3 Comparison of CNN Vs Traditional methods, existing frameworks, and pretrained models

The advantages of CNN against traditional methods can be summarized as follows [9].

(i) Hierarchical feature representation, which is the multilevel representations from pixel to high-level semantic features learned by a hierarchical multi-stage structure [6] and hidden factors of input data can be disentangled through multi-level nonlinear mappings.

(ii) Compared with traditional shallow models, a deeper architecture provides an exponentially increased expressive capability.

(iii) The architecture of CNN provides an opportunity to jointly optimize several related tasks together.

(iv) Benefitting from the large learning capacity of deep CNNs, some classical computer vision challenges can be recast as high-dimensional data transform problems and solved from a different viewpoint.

Compared with the traditional mobile sensing and cloud computing, the advantages of deep learning and inference on the mobile device are - the more computing is done on the mobile device, the less data needs to be sent to the cloud, the cost of maintaining/renting cloud computing resources can be prohibitive for some applications, computation on mobile devices becomes possible as mobile devices become more computationally powerful. Choosing the right platform for training and deployment can make a significant difference in development time, cost, and final system performance that can be referred from a comparison table below [9].

Table 1: Comparison of the existing platforms

ML platform	Mobile OS	DNN Architecture	GPU/CPU	Model Format
TensorFlow / TFLite	iOS, Android	Most DNNs	Yes / Yes	TensorFlow
Caffe2	iOS, Android	CNN	Yes / Yes	Caffe2, CNTK, PyTorch
CoreML	iOS	CNN, RNN, SciKit	Yes / Yes	Keras, TensorFlow
Snapdragon	Android	CNN, RNN, LSTM	Yes / Yes	Caffe, Caffe2, TensorFlow
Deep Learning Kit	iOS	CNN	Yes / Yes	Caffe

TensorFlow Lite can be deployed to a variety of platforms from CPU, GPU, TPU, to mobile and edge devices. There is an easy pipeline to bring TensorFlow models to mobile devices compatible with both android and IOS devices. TensorFlow Lite is the solution for running TensorFlow models on mobile and embedded devices. It was optimized for accuracy, low latency, small model size, speed, efficiency, and portability to both Android, iOS, and other internet of things (IoT) devices. Most (Deep Neural Networks) DNNs are paired with this tool to work, thus chosen for our work as well. The major benefits of lite models, also called mobile models are low communication bandwidth, less cloud computing resource costs, quicker response time, and improved data privacy [10].

Most of the toolkits require a good knowledge of both machine learning and software engineering. Keras is a greatly simplified library for learning and deployment. It is a python-based, and easy-to-use high-level API that hides all unnecessary and complicated machine learning and software details backend enabling users to use TensorFlow, Theano, CNTK without having to study the underlying details. PyTorch is a python open-source ML library based on Torch (Lua programming). Keras is more mature and runs in

Windows, Linux, and OSX whereas PyTorch only supports Linux, OSX and is not so optimal for product deployment comparatively. A survey says that Engineers prefer Keras providing fast development comparing to PyTorch, through Quora [11]. Therefore, the Keras framework is chosen for this work to implement classification.

Some of the conclusions that are learned from [9] include, a paper that summarized the performance of various deep learning models for image classification on iPhone 7 based on results obtained from over 10,000 mobile devices and more than 50 different mobile SoCs: (i) The easiest way to using deep learning on Android is to use TensorFlow Mobile framework, TensorFlow Lite is an option (ii) Caffe2 and other frameworks are much less popular with almost no tutorial and fewer problem descriptions, (iii) The benchmark of mobile deep learning should consider metrics such as accuracy, model size, and speed/execution time as in Table 2 [12].

Table 2: Benchmark Image Classification Model Performance on iPhone7

Model Architecture	Top1 Acc (%)	Model Size (MB)	Execution time (ms)
vgg16	71	553	208
inception v3	78	95	90
ResNet 50	75	103	64
mobilenet	71	17	32
sqeezenet	57	5	24

The selection of pre-trained model required for a task depends on two competitive criteria: namely accuracy of the model and the Speed of the model training and processing. There are many high-quality pre-trained models such as VGG16, ResNet50, Mobile Net, Inception, etc., available publicly. These models are trained prior on a large benchmark

dataset and have their weights saved for use in any other similar tasks as the starting point by fine-tuning on the domain-specific data which can avoid building everything from scratch. For most of the mobile deep learning deployments, the smallest models with good enough accuracy and processing speed are required. Mobilenet is known for its lightweight architecture and very low maintenance thus resulting in high speed. Mobilenet, proposed by Google is compatible with the TensorFlow framework and is already trained with the ImageNet dataset and gives much higher accuracy than other models. It is most suitable for mobile-based and embedded applications. Its major advantage is that it reduces the number of parameters in the neural network. MobileNetV2 wrapper also improves the performance of mobile models and can handle multiple tasks at the same time. It is the smallest model with 14-16MB of size depending on the version. Based on the reliable reviews from [12][13], Mobilenet architecture has been chosen as the pre-trained model for this work since the lesser model size and faster latency are key factors in this implementation.

We performed a comparative study to decide on the potential approaches. The key takeaways of this chapter involve choosing CNN based approach for the object classification due to its high-level feature representation and promising outputs, using the Keras and TensorFlow frameworks to make it deployable onto a smartphone, a small but efficient computing device. The pretrained model Mobilenet known for its speed and lighter architecture is chosen for another approach which is based on transfer learning using the TensorFlow Lite framework and deploy it onto the TF Lite built ready-made classify android application.

CHAPTER III

METHODOLOGY

As discussed earlier in the above chapters, to serve the purpose of blind assistance within the university campus, a CNN model approach using TensorFlow and Keras frameworks for object identification in the video stream through the smartphone followed by a voice alert is focused to implement. The plenitude of quality data is the vital element for building these effective models. Data collected ought to be understandable and relevant to the requirements and preprocessing helps to standardize the data obtained making it feasible to work with. As mentioned in [13], "Effort spent for collecting the right data is more rewarding than the effort spent on improving the algorithm. It is always a good idea to collect some real data from the kind of environment that the mobile deep learning application will be deployed. This data will be invaluable for improving the model or verifying the application performance." As a result, a good number of high-quality image data has been collected and partitioned for training, validation, and testing making sure zero overlaps of images in training and model evaluation/validation.

3. 1 Data Collection

The selection of images is based on different viewpoints to make the dataset a versatile collection and reduce overfitting on the network model. The models work more efficiently on their selective characteristics of the dataset available. Thus, a dataset plays a key role in choosing the model and the resultant accuracy. Supervised deep learning algorithms like

CNNs tend to require a good number of high dimensional data to work efficiently, though the amount of data needed depends on the complications of the task planned to perform. Since the data is as important as the algorithm, the dataset collection, and preprocessing stand out before beginning with anything else.

Dataset collection has been an ongoing process throughout the project. The initial dataset comprised of 5 classes with 100-150 images per class for training the model and measuring the accuracy and later added additional data for tuning the model to result in better performance. The final training dataset consists of 10 classes with 625 RGB images per class having the highest camera resolution setting of 4:3, concentrating majorly on outdoor environments within the University of Dayton campus premises. Stratified data sampling [14] technique has been followed to transform the dataset to balance each class distribution i.e., the total images within each class are equally divided into groups based on common characteristics that are present within the campus for the required classes to avoid classification imbalance.

The class labels in this work are Bench, Building, Bush, Campus car, Door, Person, Staircase, Trashcan, Tree, and Window. The images for Class ‘Person’ were taken from the NTU Pedestrian dataset that was collected from Nanyang Technological University Campus [15] [16] [17] and some from the Google images [18]. All the rest of the classes were collected manually with good resolution using an Android phone version 10 with 16 +20 MP Dual Camera. All the images are set to be in the JPG/JPEG format with portrait orientation. The following are the sample images of the dataset classes and their characteristic type used in this work:

Figure 3. 1: Sample images for the Class Bench with rectangular type (left), round type (right)

Figure 3. 2: Sample image for the Class Building

Figure 3. 3: Sample image for the Class Bush

Figure 3. 4: Sample image for the Class Campus Car

Figure 3. 5: Sample image for the Class Door

Figure 3. 6: Sample images for the Class Person from Google (Left), NTU dataset (right)

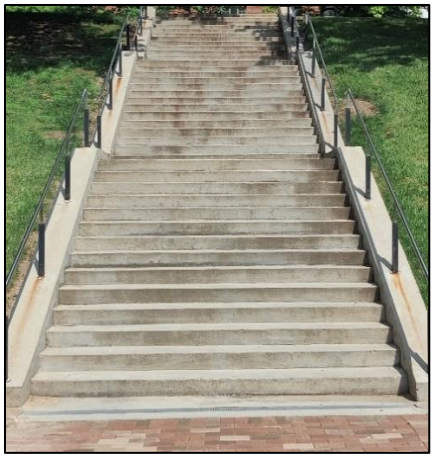

Figure 3. 7: Sample image for the Class Staircase

Figure 3. 8: Sample image for the Class Trashcan with a round type (Left), square type (right)

Figure 3. 9: Sample image for the Class Tree

Figure 3. 10: Sample image for the Class Window with type 1(left), type 2 (right)

3.2 Object Classification

The classification method is implemented using Keras in python3 script. Keras is an extremely popular high-level API that uses TensorFlow in its backend for building and training deep learning models. It is used for fast prototyping, state-of-the-art research, and production [19] [20]. This model is trained on GeForce RTX 2070, CUDA version 11.2 with 16GB memory.

3.2.1 Data Loading

The input for the model contains 6250 images belonging to 10 classes splitting 5000 images for training and 1250 images for validation in 80-20 split ratio resizing the images into 224 * 224 size with a batch size of 16. Batch size is the number of training images used in one iteration. The key concepts followed when loading the data are cache(), shuffle() and prefetch(). Buffered prefetching is used so that the data can be taken from the disk without having I/O become blocking, cache() keeps the images in memory after they have been loaded off disk during the first epoch ensuring the dataset does not become a bottleneck when the model is being trained. prefetch () will overlap the data preprocessing and model execution while training [21].

3.2.2 Data Augmentation

This preprocessing concept is utilized to increase the quantity and diversity within the existing training images called ‘Random Transformations’ by applying techniques like flipping, cropping, rotating, zooming, scaling, padding, etc., for the model to get trained better and expose to more variations while helping to avoid overfitting, that is the model getting over trained to an extent resulting in a negative impact on the performance. The augmentation techniques used in this work are horizontal flipping, random rotation at a degree of 0.1, and random zoom at a degree of 0.1. Rescaling is done to standardize the data making the input to smaller values in the range of [0,1] which is ideal for the neural networks [20].

3.2.3 Classification using CNN Network from scratch:

Data augmentation and rescaling layers are attached at the start of the model followed by Conv2D operations. This way augmentation will happen on the device, synchronously with the rest of the model execution. Data augmentation is inactive at the test time, so the input samples will only be augmented during training by the fit() function, not when calling evaluate() or predict() that does validation, making it benefit from GPU acceleration when training on GPU [20]. A sequential neural network is initialized where the Conv2D function is used to perform the convolution operations on the two-dimensionality training images and the MaxPooling2D function is used to reduce the size of the image by considering the maximum pixel value from the defined kernel.

The number of convolution layers is chosen based on the classification requirement. The parameters used within Conv2D to define the model architecture for training are ‘Number

of filters', 'Kernel size' that tells the height and width of the convolutional window, 'Stride' that tells the model by how many numbers of units the filter is supposed to move to another location over the image, 'Padding' is the pixels added to an image on all sides such that the output has same height/width as of the input when it is being processed and 'activation function' which is the non-linear transformation done over the input deciding if the neuron should be activated or not before sending it to the next layer or as an output.

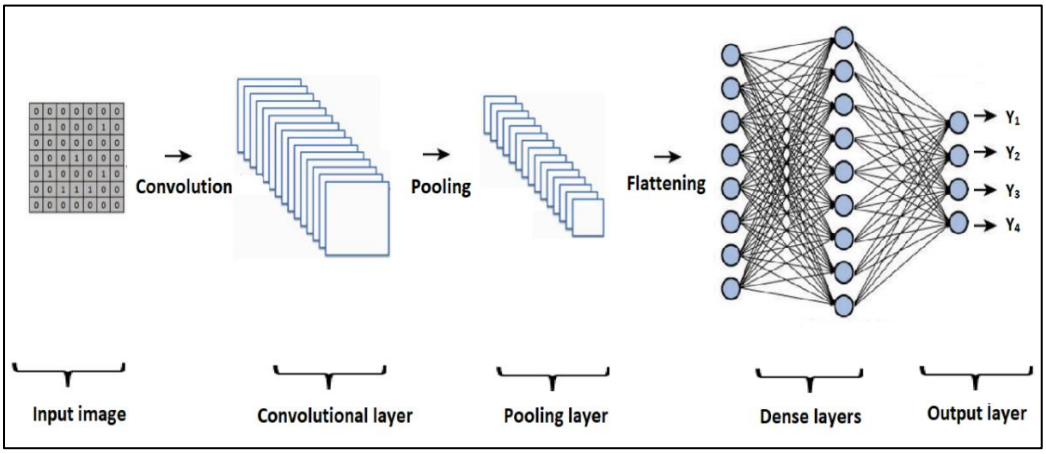

Figure 3. 11: CNN Network Architecture

The architecture of this work consists of 12 layers – four convolution layers, four max-pooling layers, one dropout layer, and two dense/fully connected layers as shown in Figure 3.12. The fine-tuning process includes the setting of the hyperparameters of the network, figuring out the number of layers, kind of activation function and optimizer, rate of learning, dropout range, batch size, and the number of epochs.

The present network uses 16 filters of kernel size 3*3 with a stride of 1-pixel unit, which scans over the image, producing 16 sets of activation maps. This continues to be with the

next layers of filters i.e., 32, 64, and 64. Padding is set to be ‘same’ that fills zeros evenly and activation function as ‘ReLU’[22] for all the convolution layers. ReLU (Rectified Linear Unit) helps in making all the negative values zero.

After each Convolution layer, Max pooling layers are present alternatively that work on each feature map resulting in subsampling. A dropout layer is present at the degree of 0.2 i.e., 20 percent of the neuron connections are randomly removed to regularize the model and prevent overfitting in the dense layers. The flattening step is done using flatten function after the dropout to convert/ flatten the 2-Dimensional arrays to a single linear vector.

Finally, the Dense layer (hidden layer) is used to create a full connection of the neural network and send the flattened linear vectors as input. ‘Dense’ function is used where the number of hidden neurons and the activation function is declared. The first dense layer has 128 hidden neurons declared with the ‘ReLU’ activation function followed by the final dense/output layer with 10 neurons representing the 10 classes of the dataset along with the ‘Softmax’ activation function. Softmax finds the probability of different target classes over all the possible classes.

The compilation step is done to compile the CNN network built. The optimizer parameter, Loss parameter, and performance metrics parameters are passed as arguments for the compile function. I am using the ‘Sparse_Categorical_Crossentropy’ loss function along with the ‘Adam’ optimizer in this network. Adam [23] is known to have the best properties of AdaGrad and RMSProp algorithms to handle the sparse gradients on noisy problems. Metrics is ‘Accuracy’. The whole network built is having 1,667,562 parameters.

```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
sequential (Sequential)      (None, 224, 224, 3)       0
_________________________________________________________________
rescaling_1 (Rescaling)      (None, 224, 224, 3)       0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 224, 224, 16)      448
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 112, 112, 16)      0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 112, 112, 32)      4640
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 56, 56, 32)        0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 56, 56, 64)        18496
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 28, 28, 64)        0
_________________________________________________________________
conv2d_7 (Conv2D)            (None, 28, 28, 64)        36928
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 12544)             0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               1605760
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                1290
=================================================================
Total params: 1,667,562
Trainable params: 1,667,562
Non-trainable params: 0
```

Figure 3. 12: Model Summary of the CNN Network

Finally, the training is initiated by fitting the model using the fit() function to the given input arguments - training dataset, validation dataset, and the number of epochs to run and returns a history callback, containing the lists of successive losses and accuracy metrics of both training and validation along with the epoch number and time taken for each epoch to run. The number of epochs ranges between 35-40 resulting in significant results. The final trained model weights are saved in the ‘hdf5’ file format. The hdf5 model can be converted into the tflite model using the ‘TFLite Converter’ [24] which can result in reduced file size.

The plots generated for graphical representation of the training and validation accuracy and loss in Tensorboard [25] can be viewed in the next chapter.

3.2.4 Classification using Mobilenet Pretrained Network

Mobilenet uses depth-wise separable convolutions which means it performs a single convolution on each color channel rather than combining all three channels and flattening it. These effects the filtering of the input channels. As explained in [13], “For Mobilenet the depth-wise convolution applies a single filter to each input channel. The pointwise convolution then applies a 1×1 convolution to combine the outputs of the depthwise convolution. A standard convolution filters and combines inputs into a new set of outputs in one step. The depthwise separable convolution splits this into two layers, a separate layer for filtering and a separate layer for combining. This factorization has the effect of drastically reducing computation and model size.”

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figure 3. 13: Mobilenet Network Architecture

Transfer Learning saves a lot of training time, which could usually take days or weeks depending on the complexity of the task, requires lesser training data because the pre-trained model would already be trained on huge amounts of data and better performances comparatively. The only requirement is to choose the right model based on the dataset.

TensorFlow Lite provides optimized pre-trained models that you can deploy in your mobile applications. The TF lite Model Maker library makes it simpler to train a TensorFlow Lite model using a custom dataset. The Model Maker library has been popular in supporting tasks like image classification, Object Detection, Text Classification, BERT Question

Answer, Audio Classification, and Recommendation. Currently, TensorFlow Lite supports models such as Efficient Net-Lite* models, MobileNetV2, ResNet50 as pre-trained models for image classification [26].

Pretrained model can be used as it is or through transfer learning by customizing the model specific to the task and dataset. There are two ways of customizing the model: one is the feature extraction, and the other is the fine-tuning [27].

(i) In the feature extraction method, the network uses the representations learned by the previous network to extract features from the new dataset. Retraining can be avoided as the base network has generically useful features required for classification. The final layer which is the classification layer alone requires to be changed specifically to the classes of the new task.

(ii) Fine-tuning method refers to unfreezing the required top layers of the frozen model which usually stands as a base and train those layers along with the newly added classifier layers. This is done for changing the representations at a higher level to make them more relevant for the task. This is done when the task is not completely related to the data the pre-trained model was originally trained on but needs initially trained weights as a starting point.

In this image classification process, the feature extraction method is used to make the classifier model. 'DataLoader' class is used to load the data and the 'from_folder()' method is used to load the data from the folder. JPEG- and PNG- encoded images are supported. The whole dataset is split at once into training, validation, and testing data in the desired

ratio using the data_split() function [26]. The customized model is defined in the required specifications using the ‘create’ method. The train data and validation data are given along with the pre-trained model and the number of epochs.

The dataset division followed for this method is 5000 images for training, 1249 images for validation, and 201 images for testing completely shuffled by the inbuilt function. The whole network sums to 3,550,285 parameters out of which 10,020 are the trainable parameters from the last dense/classifier layer which is customized to the present dataset. The default model is EfficientNet-Lite0. The model is switched to MobileNetv2 using the ‘model_spec’ parameter and is set for training for 5-10 epochs. The number of trainable parameters is less in this as we are calling in the trained weights. The trained weights are saved in the ‘.tflite’ format.

```
INFO:tensorflow:Retraining the models...
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
hub_keras_layer_v1v2 (HubKer (None, 1001)              3540265
_________________________________________________________________
dropout (Dropout)            (None, 1001)              0
_________________________________________________________________
dense (Dense)                (None, 10)                10020
=================================================================
Total params: 3,550,285
Trainable params: 10,020
Non-trainable params: 3,540,265
_________________________________________________________________
```

Figure 3. 14: Model Summary of Classification Network using Mobilenet

3.2.5 Deployment of the classifier model to the Mobile Application

TensorFlow Lite which is an open-source, cross-platformed deep learning framework is not only capable to build or convert models to ‘tflite’ format but also supports deployments onto mobile and edge devices. The final weights are saved into the required format after training the models. There are product-ready applications for TensorFlow Lite on Android and IOS. These applications can be built into a mobile device and run inference using the TensorFlow Lite Java API. The .h5 format models can also be converted to the .tflite format model and be used for classification through the TFLite Classify mobile application.

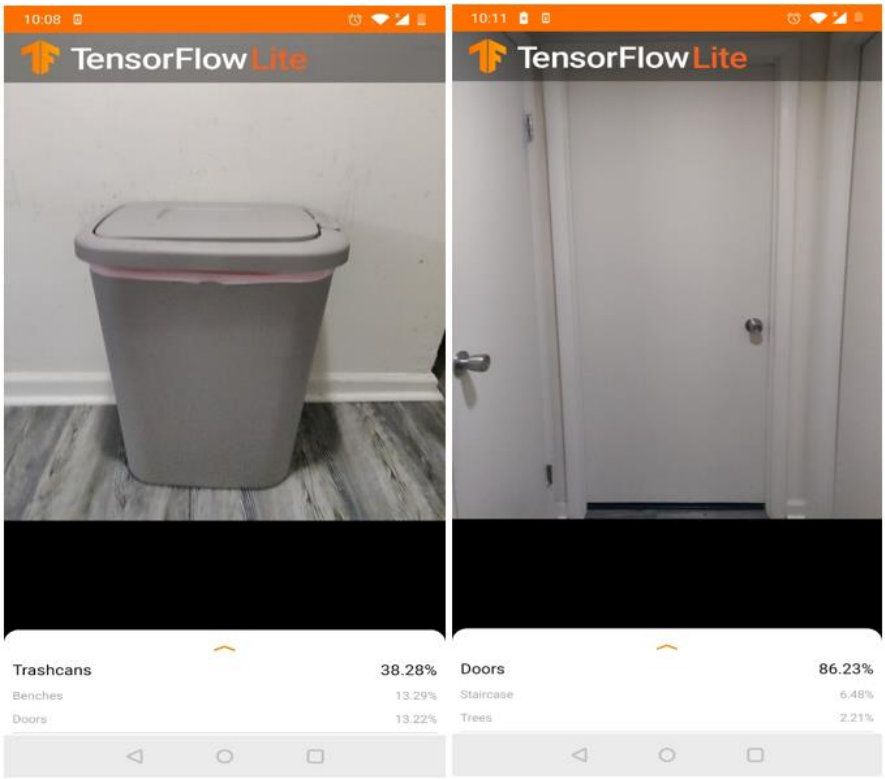

Figure 3. 15: Inference results from TFLite mobile application of the classifier trained using a pre-trained model

We developed that application following the below steps: The basic requirements for the deployment would be Android Studio 3.2, an android mobile device with USB debugging enabled, and a USB cable that is connectable to the computer [28] [29]. Android Studio is the go-to to Google's official integrated development environment (IDE) for android development.

Step 1: Clone the TensorFlow examples repository from GitHub [30].

Step 2: A project is built in the android studio making sure the android SDK (Software Development Kit) configurations are updated. Missing libraries can be known through the 'build.gradle' file prompt that can be downloaded and updated.

Step 3: The TF Lite '.lite' model files for image classification on the android device demo application example from the repository are selected and opened under the assets folder in the android studio. Out of multiple ways of implementations, the one which works for inference alone is chosen for this work. 'lib_task_api' leverages the out-of-box API from the TensorFlow Lite Task Library. The required code changes are done by clicking 'Make Project' and making sure of all the configurations set to run on the mobile device.

Step 4: The saved model which is trained on a customed dataset present in the tflite format is added along with labels text files under the assets folder of the project.

Step 5: The developer mode and the USB debugging are enabled on the smartphone which is a one-time setup. The USB setting is chosen to transfer files when the system is connected to the smartphone.

Step 6: When the project is built in android studio and the phone is connected through USB and is ready with the settings, the Run/Run App button on the toolbar is clicked to run the

project on the device.

Step 7: The permission to access the camera of the phone from TFL Classify is accepted. The TF Lite classify app is set up in the mobile device with the uploaded trained model. The inference of the model can be done using this app within the campus premises to identify objects.

3.2.6 Implementing voice alert to the video stream

Each frame of the video is accessed and processed using the OpenCV [31] library and the voice alert is initiated using the gTTS module [32] [33]. The frame grabbed from the video stream is being classified using the trained classifier model to identify the objects present in the frame. The trained model predicts the class name of the identified object in the text format which is sent to the text-to-speech converter to generate a voice alert.

Frames per second (fps) is the frequency at which consecutive images appear on the display which causes a repetition of the object in the scene and there will be a set of frames consecutively, associated with the same class name. Only the class name of the first frame is given out as a voice alert ignoring the rest of the consecutive frames with the same class name until a change in prediction with a new class name occurs. This helps in avoiding voice overlap of the alerts between the frames. The results obtained from both the trained classifier models and their corresponding prediction of the objects on the test data, the output video clippings showing identification of the object followed by voice alert inferences within the university campus can be viewed in the next chapter.

CHAPTER IV

PERFORMANCE EVALUATION

An important part of building networks specific to the task is evaluating the outputs. There are many metrics present to evaluate the classification performance, that vary depending on the problem statement. Accuracy [34] assessment is the most common metric used in classification. It is defined as the rate/percentage of right predictions. It calculates the ratio between the number of correct predictions and the total number of predictions. During the training of the model, the training accuracy and the validation accuracy are determined for each epoch or batch size, helpful in identifying the overfitting or underfitting. Though training accuracy is not a direct evaluation metric to rely on, it indicates if the backend functioning is as expected during training which saves time before digging in deeper.

Any useful and reliable model with the proper distribution between train and test would result in a score of above 90% accuracy. On the other hand, there is a loss function that measures the distance between the current and expected output. It is inversely proportional to accuracy and guides the model to move towards convergence [35] during which the loss is minimized virtually supporting model fitting of the training data and good accuracy. Both the accuracy and loss are represented graphically to visualize the training and validation of the model every epoch, given in Figures 4.1 through 4.4.

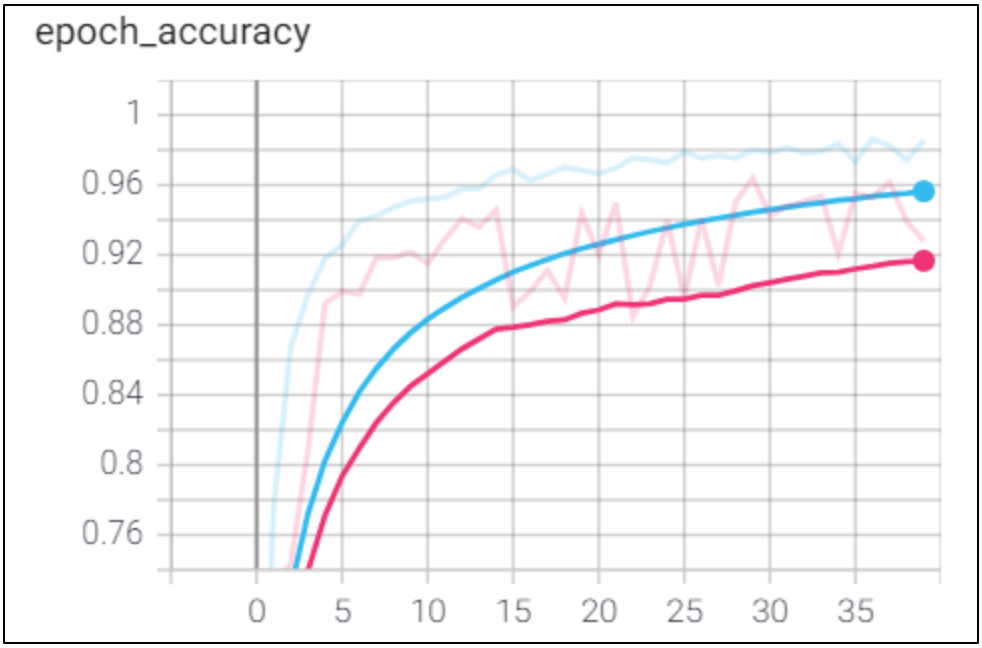

Figure 4. 1: Training (blue) and Validation (pink) accuracy plot of CNN Network

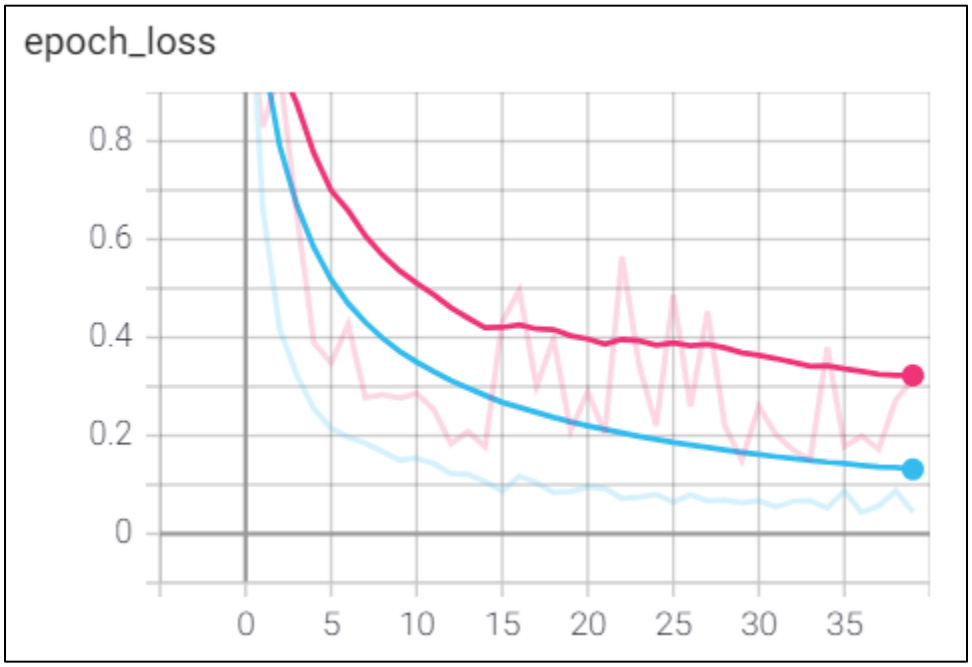

Figure 4. 2: Training and Validation loss plot of CNN Network

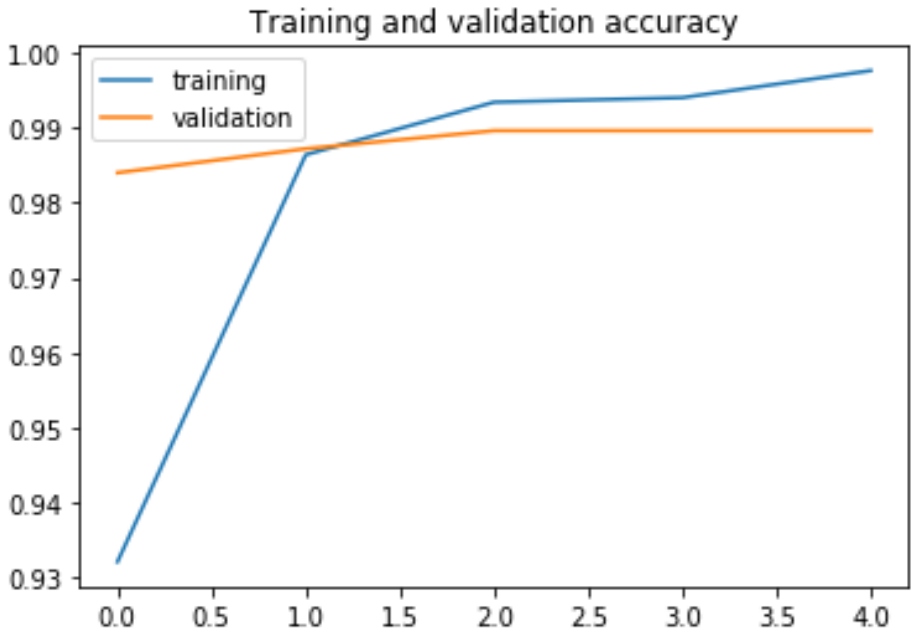

Figure 4. 3: Training and Validation accuracy plot using a pre-trained model

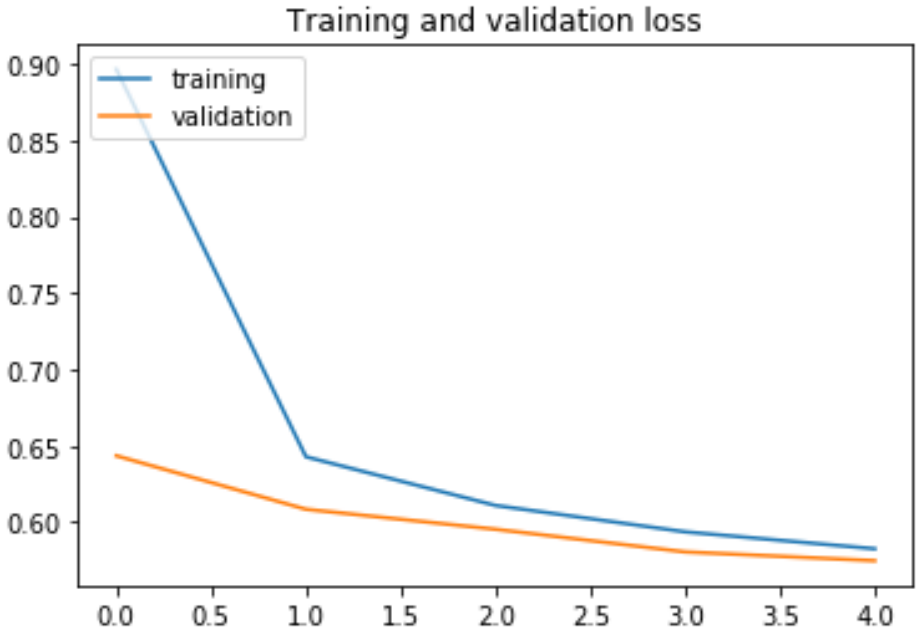

Figure 4. 4: Training and Validation loss plot of using a pre-trained model

A classification report [36] has been generated for the models which will display the precision, recall, F1 score, and support score for the model. A confusion matrix [36] is generated which is an N*N table that contains the number of correct and incorrect predictions of the classification model. The rows represent the real classes while the columns represent the predicted classes. These both solely depend on the calculation of the predictions from the model. The four types of predictions that a model could result are as follows:

True Positive (TP): outcome where the model correctly predicts the positive class.

True Negative (TN): outcome where the model correctly predicts the negative class.

False Positive (FP): outcome where the model incorrectly predicts the positive class.

False Negative (FN): outcome where the model incorrectly predicts the negative class.

Precision [37][38] is the ratio of the correctly predicted positive observations to the total predicted positive observations. The Recall is the ratio of correctly predicted positive observations to all observations in an actual class. Recall is also known as sensitivity.

$$\text{Precision} = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

$$\text{Recall} = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

F1-Score [37][38] is the weighted average of precision and recall. F1 is more useful and reliable than the accuracy since, in most cases, the class distribution might not be perfect. Support is the number of actual occurrences of the class in the specified dataset.

$$\text{F1 Score} = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}}$$

Macro average and weighted average [37][38] are calculated as an overall score to know the precision, recall, and F-1 score for all the classes together. The confusion matrix and classification report obtained from both the classifiers are discussed below.

4.1 Results from the classifier trained using Mobilenet pre-trained model

The test data for the model trained using Mobilenet consists of 201 images. There are 201 images since the inbuilt function used divides the images based on the rates we give in, to even out all the images between training, validation, and testing within 6450 images. And due to the shuffling nature of the inbuilt function, the test images are shuffled in the test dataset, and they vary in number per class making sure not to reach beyond an average value of 20. Hence, the number of test images sent for prediction for each class are as shown in Table 3:

Table 3: Test images divided per class by split() function

Class 0 (Bench)	14
Class 1 (Building)	17
Class 2 (Bush)	24
Class 3 (Campus car)	24
Class 4 (Door)	22
Class 5 (Person)	17
Class 6 (Staircase)	24
Class 7 (Trashcan)	24
Class 8 (Tree)	19
Class 9 (Window)	16

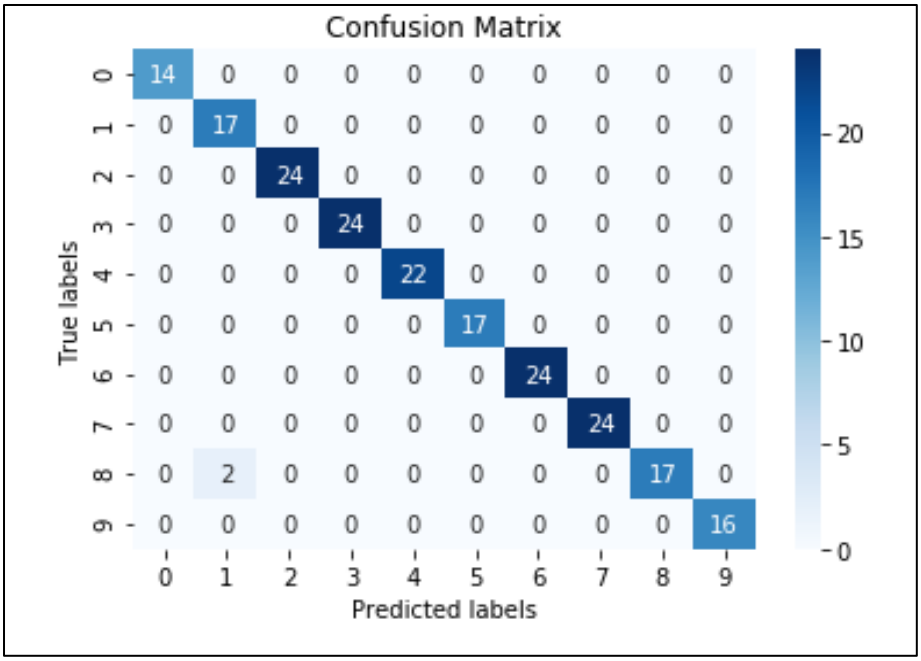

Figure 4. 5: Confusion Matrix from Network using a pre-trained model

	precision	recall	f1-score	support
Bench	1.00	1.00	1.00	14
Building	0.89	1.00	0.94	17
Bush	1.00	1.00	1.00	24
CampusCar	1.00	1.00	1.00	24
Doors	1.00	1.00	1.00	22
Person	1.00	1.00	1.00	17
Staircase	1.00	1.00	1.00	24
Trashcan	1.00	1.00	1.00	24
Tree	1.00	0.89	0.94	19
Window	1.00	1.00	1.00	16
accuracy			0.99	201
macro avg	0.99	0.99	0.99	201
weighted avg	0.99	0.99	0.99	201

Figure 4. 6: Classification Report from Network using a pre-trained model

Based on the observations from figures 4.5 and 4.6, the classifier model could correctly predict all the classes but one. Two test samples of the class tree are mispredicted to be class 1 that is building. Overall, the model trained through transfer learning performs well having an F-1 score of 100 percent for almost all classes and above 90 percent for the rest. The size of both trained networks varies due to the differences in the overall parameters involved within the network. The .h5 Keras model built from scratch holds a size of 19.17 MB. When the .h5 model is converted into the .tflite model, it comes down to 6518 KB. The size of the '.tflite' model trained using pre-trained model results to be 4144 KB.

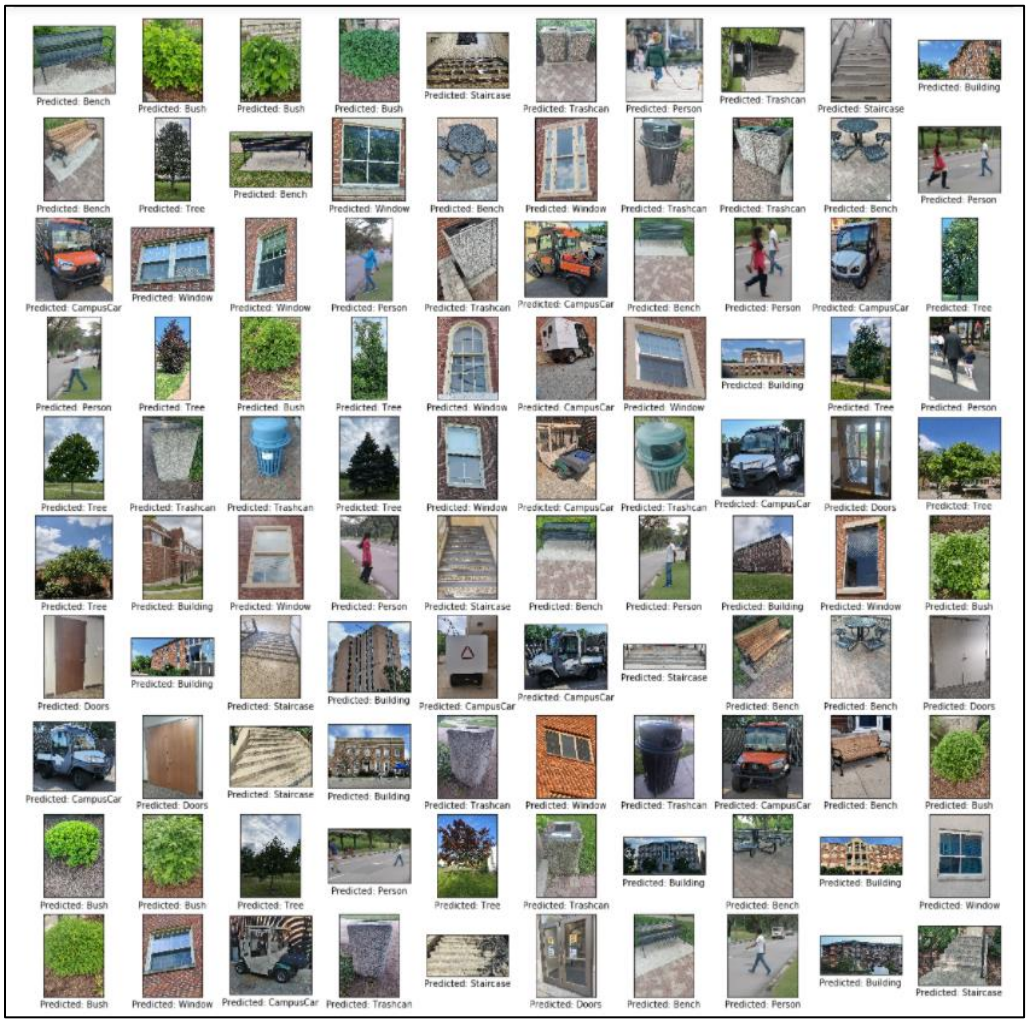

Figure 4. 7: Predictions from a model trained with Mobilenet

The predictions on images of test data from the classifier trained from pre-trained can be viewed in figure 4.7.

4.2 Results from the classifier trained using CNN network from scratch

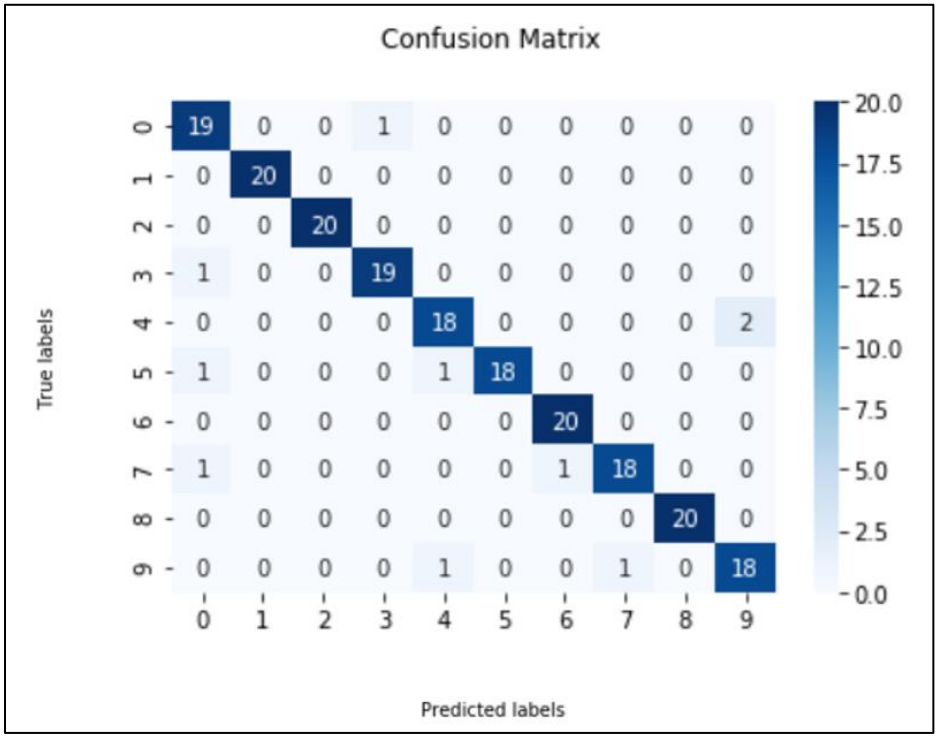

Figure 4. 8: Confusion Matrix from CNN network trained from scratch

The test data consists of 200 images in total which accommodates 20 images for each class. As seen in figure 4.8, the diagonal line elements represent the number of correct predictions which means the predicted label matches with the true labels. The off-diagonal elements show the number of mislabeled predictions for all 10 classes. In short, the diagonal elements are representing recall/sensitivity scores. The CNN model is resulting in a decent number of predictions with one or two misclassifications with an overall accuracy of 95 percent.

	precision	recall	f1-score	support
Bench	0.86	0.95	0.90	20
Building	1.00	1.00	1.00	20
Bush	1.00	1.00	1.00	20
CampusCar	0.95	0.95	0.95	20
Doors	0.90	0.90	0.90	20
Person	1.00	0.90	0.95	20
Staircase	0.95	1.00	0.98	20
Trashcan	0.95	0.90	0.92	20
Tree	1.00	1.00	1.00	20
Window	0.90	0.90	0.90	20
accuracy			0.95	200
macro avg	0.95	0.95	0.95	200
weighted avg	0.95	0.95	0.95	200

Figure 4. 9: Classification Report from CNN network trained from scratch

Generally, the classes with higher recall than the precision mean that the false positives are higher than false negatives and vice versa. Based on the results from the confusion matrix and classification report in figures 4.8 and 4.9 respectively, the model correctly predicted all the observations of classes 1,2, 6, and 8 which means the buildings, bushes, trees, and staircases have the same precision, recall, and F-1 scores as 1. For class bench, one test sample is incorrectly predicted as campus car (class 3). And for class campus cars, one test sample is incorrectly predicted as a bench (class 1). For class Door, two test samples are incorrectly predicted as the window (class 9). For the class person, one test sample is incorrectly predicted as a door (class 4) and the other sample as a bench (class 0). For the class trashcan, one sample is incorrectly predicted as staircase (class 6) and the other one as a bench (class 0). For the class window, one test sample is incorrectly predicted as a door (class 4) and the other one as a trashcan (class 7). Overall, the model performance is

rated to be pretty good to reach the expectations, having the F-1 score above 90 percent for all the classes. The predictions on images of test data from both the classifiers, the CNN model trained from scratch can be viewed in Figure 4.10.

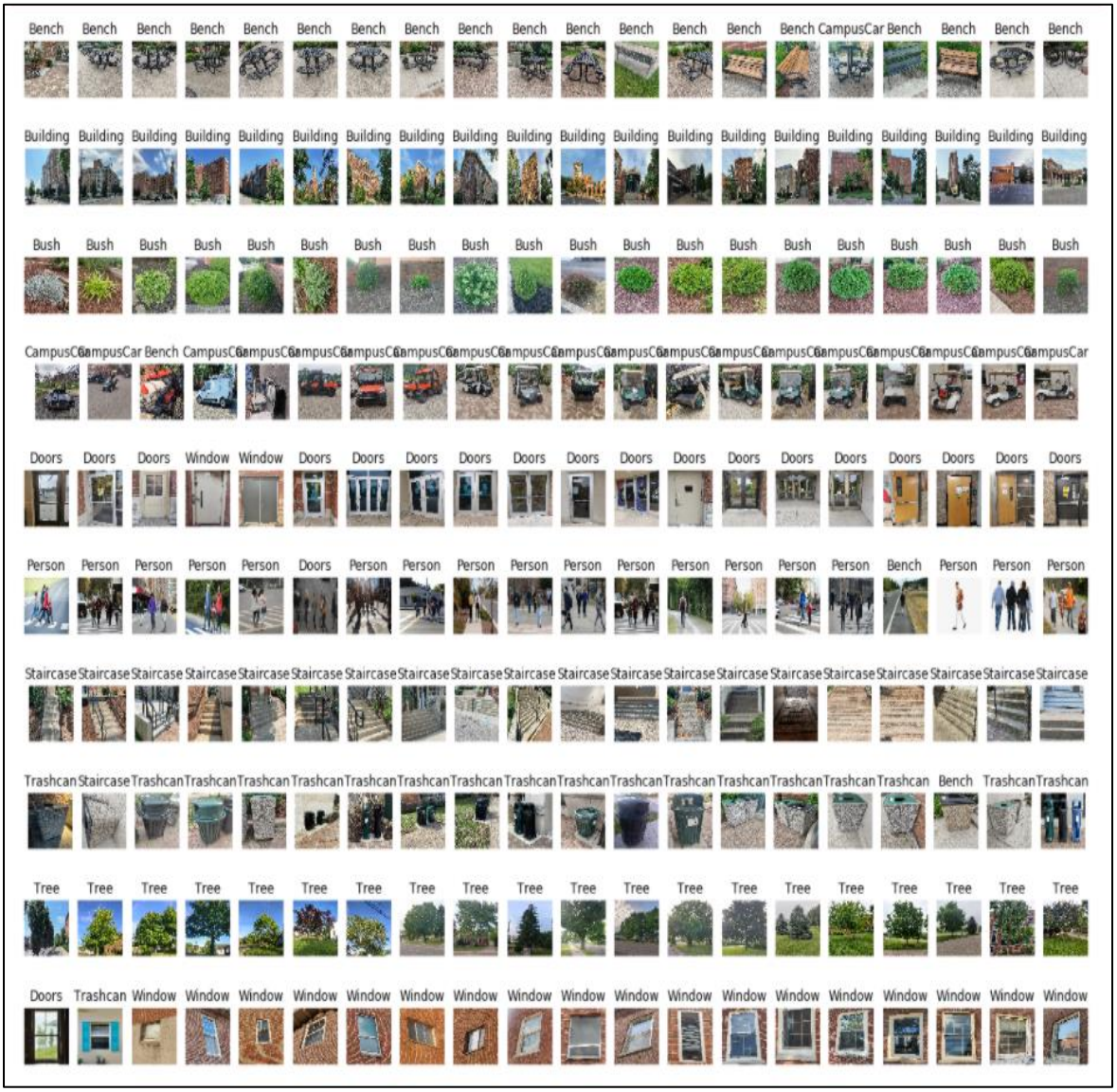

Figure 4. 10: Predictions from CNN network trained from scratch

The video stream sent to the object identification classifier will be divided into several frames, the object is identified, and the class name is displayed on every frame of the video followed by a voice alert. The timely voice alert is given for the objects which change one after the other ignoring sequence of consecutive same class names. Below are the sample screenshots of a video output showing the prediction of the objects printed on each frame.

Figure 4. 11: Inference results from video frames displaying the object predicted by the classifier trained from scratch

Therefore, the above chapters deduce the methods used in the implementation of classification and the potential results obtained. The initial work as a part of this research was an attempt to build an object detection model over the same university campus dataset using the TensorFlow 2.0 Object Detection API. All the requirements from beginning with preparing the annotations for the images, generating the tf records, configuring the training pipeline to training the model, and exporting the resultant model to detect objects were followed as per the API documentation [39]. The proper setup for TensorFlow object detection installations with python packages was time-consuming but is a one-step installation. Huge memory space and high-performance graphics card were a mandatory requirement for supporting long training hours and numerous package downloads such as pre-trained model's zoo repository and TensorFlow Object detection API along with its dependencies such as COCO API, protobuf, etc.

The GPU [40] configuration with the API for training is not well-defined in resources which challenged customizing the framework source codes. Despite of overcoming all the concerns and challenges, the training for object detection could not load in more than 300-400 images along with their annotations summing up to 700-800 files in total, requiring more memory. The idea of introducing audio processing to this object detection model would lead to higher complex architectures turning down the feasibility to deploy onto the low-computing device.

For this very reason, this chapter concludes that the object identification through CNN classification is set to be convenient with lesser complex architecture and computations, cost-effective, reliable and a user-friendly approach showing scope for more advancements in its optimization further.

CHAPTER V

CONCLUSION

Blind assistance for surrounding awareness is a societal benefitting idea and a key motivator for persons with vision impairment to participate in outdoor activities. Though many researchers took the initiative to approach this idea and implemented it with good results, they could not inspire social and universal acceptance of the work. This cause can be pinned down to the very reason that most of the prior research surveyed relatedly involves using multiple complex algorithms for processing, reliable sensors, and microcontrollers for detection and audio feedback. The constraint of having to use a lot of resources resulting in heavier architecture and incapability of implementation onto low computation devices with more computation time and delayed latency still lies in the focus for advancement. For the above-mentioned limitations identified, we implemented an approach based on the CNN model for its deep and robust extraction and learning power with lesser complexities and dependencies. When working with a huge collection of 2D images, the most essential consideration is the ability to extract significant characteristics, known as features, without redundant data or loss of crucial information, resulting in inaccurate predictions.

The present method consists of a clear and simple approach where the CNN classifier is trained on the dataset collected from the university. It performs well in identifying the objects in the video stream and giving an immediate voice alert to assist the blind person

about his surroundings within the university. The CNN model was initially trained with the benchmark deep learning pre-trained model Mobilenet using TFLite model maker library and integrated into the ready-to-use, flexible TFLite android application. It is used to perform video inference through the TFLite classify app on the smartphone which is very simple in implementation and user-friendly. Even though the pre-trained classifier operates well with a 99 percent accuracy rate by recognizing items well inside the campus and displaying reasonable probabilities for each class, it has proven to be incompatible for audio processing. This drawback was addressed by opting to train a CNN model from scratch using the Keras framework that will classify the video streams which are processed using OpenCV followed by a voice alert using the Google Text to Speech module. This scratch model also yielded good results with an average of 95 % accuracy. From the predictions observed on the test images with both the models, the transfer learned model resulted in lesser false predictions and has a slightly lesser network size with a difference of 2 MB comparatively. Though the accuracy and number of true positive predictions of the pre-trained model are slightly higher than the scratch model, it does not serve any purpose in our mobile blind assistance application without audio feedback, an essential part to consider in the real-time environment.

The best approach to building the mobile deep learning algorithms is domain-specific as the applications might be operated in different environments, use different hardware, and have different specifications. Latency, which is defined as the delay between action and the response is one of the key factors to be taken care of. Classification of the object with

a timely voice alert stands above the highest accuracy and false-positive rate in this blind assistance application. Therefore, we developed this approach of scratch CNN model which is performing very well in identifying the objects and alerting the user with a voice in the real-time environment justifies our problem statement.

The challenges faced at different stages of this work are numerous. Beginning with the dataset, collecting a huge set of data was tough because of the manual work and changing weather conditions like snow, heavy rains. The collection was an ongoing project throughout the research. The collection of images was not much favorable in the spring period as our work is focused on outdoor objects. Capturing variations in viewpoint to cover different angles of the object for better training was tedious. Data augmentation technique has been included to complement the manually taken variations. Training the model initially started with 100-150 images per class and later we gradually increased it to 625 as a part of fine-tuning validation performance to overcome the overfitting model. All the dataset images were set with a similar orientation tag or went through lossless rotation to set their metadata making sure about the portrait mode. Reading the video frames giving fast predictions and aligning the voice alert only at the required time avoiding the voice overlap between the alerts to not confuse the user was the most challenging concern. The implementation of this work relies solely on deep learning methods and computations which involves no external hardware setup units like sensors, camera modules, microcontrollers making it cost-effective, reliable, and user-friendly. The deployment of

this model onto the mobile application compatible with both android and IOS will be taken as a priority for future works in this research.

We have multiple future works aligned with this approach stated below from the application point of view. Since the present work focuses on only 10 outdoor object classes, increasing the number of classes to identify more objects within the campus premises, outdoor and indoor as well as localization of the objects will enhance the potential of the model increasing its reliability and acceptance in real-time world applications. We could also try to identify multiple objects, measure the distance between the camera and the object directly from the screen which would make it more convincing for the end-users. We are also trying to identify the object, its size, and the backdrops around it through the image segmentation technique. This would assist the user with more precision and extract more minute details from the surrounding places around the individual.

This section summarizes how an idea for blind assistance enhances the previous approach by identifying the limitation, implementing an approach more robust and accurate for low computing mobile devices, and the scope of increasing the extraction of details from the surrounding with more precision and less complexity. Therefore, this research is a unique approach to the social problems encountered for blind assistance.

BIBLIOGRAPHY

[1] K. G. Krishnan, C. M. Porkodi and K. Kanimozhi, "Image recognition for visually impaired people by sound," *International Conference on Communication and Signal Processing, pp. 943-946, doi: 10.1109/iccsp.2013.6577195.,* 2013.

[2] A. Noorithaya, M. K. Kumar, A. Sreedevi "Voice assisted navigation system for the blind. International Conference on Circuits, Communication, Control, and Computing," 2014.

[3] R. Tapu, B. Mocanu, A. Bursuc and T. Zaharia "A Smartphone-Based Obstacle Detection and Classification System for Assisting Visually Impaired People," IEEE International Conference on Computer Vision Workshops, pp. 444-451, doi: 10.1109/ICCVW.2013.65, 2013.

[4] S. Durgadevi, K. Thirupurasundari, C. Komathi, and S. M. Balaji "Smart Machine Learning System for Blind Assistance," in International Conference on Power, Energy, Control and Transmission Systems (ICPECTS), pp. 1-4, doi: 10.1109/ICPECTS49113.2020.9337031., 2020.

[5] P. Rohit, M. S. Vinay Prasad, S. J. Ranganatha Gowda, D. R. Krishna Raju, and I. Quadri "Image Recognition based SMART AID FOR VISUALLY CHALLENGED PEOPLE," International Conference on Communication and Electronics Systems(ICCES), pp.1058-1063,doi: 10.1109/ICCES45898.2019.9002091, 2019.

[6] S. Gao, M. Cheng, K. Zhao, X. Zhang, M. Yang, and P. H. S. Torr, " \Res2net: A new multi-scale backbone architecture", IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 1-1,2019.

[7] Zhong-Qiu Zhao, Peng Zheng, Shou-Tao Xu, and Xindong Wu, "Object Detection with Deep Learning: A Review", "IEEE".

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in NIPS, 2012.

[9] Anirudh Koul, "Squeezing Deep Learning Into Mobile Phones," from https://www.slideshare.net/anirudhkoul/squeezingdeep-learning-into-mobile-phones/.

[10] Yunbin Deng, "Deep Learning on Mobile Devices – A Review," FAST Labs, BAE Systems, Inc. Burlington MA 01803.

[11] "What are the pros and cons of PyTorch vs Keras?," Quora, [Online]. Available: https://www.quora.com/What-are-the-pros-and-cons-of-PyTorch-vs-Keras.

[12] "Keras Applications"," Keras" 2020 from : https://keras.io/api/applications/.

[13] Z. Menglong, H. Andrew G, C. Bo, K. Dmitry, W. Weijun, W. Tobias, A. Marco, and A. Hartwig, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision," from https://arxiv.org/pdf/1704.04861.pdf.

[14] James, "Statistics: Introduction", "people.richland.edu" . [Online]. Available: https://people.richland.edu/james/lecture/m170/ch01-not.html.

[15] Neogi Satyajit, Dauwels Justin, Hoy Michael (2019): NTU Pedestrian Dataset. figshare. Dataset from https://doi.org/10.6084/m9.figshare.11455545.v1

[16] Neogi Satyajit, Hoy Michael, K. Dang, H. Yu, Dauwels Justin, "Context Model for Pedestrian Intention Prediction using Factored Latent-Dynamic Conditional Random Fields". Accepted by IEEE Transactions on Intelligent Transportation Systems (T-ITS), 2019.

[17] Neogi Satyajit, Hoy Michael, W.Chaoqun, Dauwels Justin, 'Context-Based Pedestrian Intention Prediction Using Factored Latent Dynamic Conditional Random Fields, IEEE SSCI-2017."

[18] "Pedestrian Safety Guide and Countermeasure Selection System", Google Images, April 2014 from https://www.pedbikeinfo.org/.

[19] Reetesh Chandra, "The What's What of Keras and TensorFlow," 4 April 2019 from https://www.upgrad.com/blog/the-whats-what-of-keras-and-tensorflow/.

[20] François Chollet, "Image classification from scratch," Keras, 27 April 2020 from https://keras.io/examples/vision/image_classification_from_scratch/.

[21] "Imageclassification", "TensorFlow", 2020 from https://www.tensorflow.org/tutorials/images/classification.

[22] "Convolutional Neural Network (CNN), TensorFlow Core" Tensor Flow, 16 June 2021 from https://www.tensorflow.org/tutorials/images/cnn.

[23] J.Brownlee,13Jan2021. [Online]. Available: https://machinelearningmastery.com/-adam-optimization-algorithm-for-deep-learning/

[24] TensorFlow,28July2021.[Online].Available: https://www.tensorflow.org/lite/convert/index.

[25] "Get started with TensorBoard,TensorBoard" TensorFlow, 8 April 2021 from https://www.tensorflow.org/tensorboard/get_started.

[26] "Image classification, For Mobile & IoT " Tensor Flow, 5 May 2021 from https://www.tensorflow.org/lite/tutorials/model_maker_image_classification

[27] "Transfer learning and fine-tuning, TensorFlow Core" TensorFlow, 17 June 2021 from https://www.tensorflow.org/tutorials/images/transfer_learning.

[28] "Android quickstart, For Mobile & IoT " Tensor Flow, 30 June 2021 from https://www.tensorflow.org/lite/guide/android.

[29] "Recognize Flowers with TensorFlow Lite on Android," Codelabs, Developers. [Online]. Available: https://codelabs.developers.google.com/codelabs/recognize-flowers-with-tensorflow-on-android#4.

[30] Hoitab, "TensorFlow Lite image classification Android example application", " Github",2020.[Online].Available:https://github.com/tensorflow/examples/tree/master/lite/examples/image_classification/android.

[31] "Reading and Writing Videos using OpenCV," OpenCV, from https://learnopencv.com/reading-and-writing-videos-using-opencv/.

[32] "Module," gTTS, from https://gtts.readthedocs.io/en/latest/module.html.

[33] "Speech Recognition in Python (Text to speech)," Python programming language, 2017 https://pythonprogramminglanguage.com/text-to-speech/.

[34] "Image classification, For Mobile & IoT " Tensor Flow, 5 May 2021 from https://www.tensorflow.org/lite/examples/image_classification/overview.

[35] W. Chenrui, Y. Xinhao, Z. Ke, and Z. Jiahui, "Improved Loss Function for Image Classification","Hindawi",23Jan2021from https://www.hindawi.com/journals/cin/2021/6660961/.

[36] "How to generate classification report and confusion matrix in Python?"," ProjectPro", from https://www.dezyre.com/recipes/generate-classification-report-and-confusion-matrix-in-python.

[37] Koo Ping Shung, "Accuracy Precision, Recall or F1?" towards data science, 15 March 2018.[Online]. Available: https://www.mdpi.com/20763417/10/4/1245/htm.

[38] "4 things you need to know about AI: accuracy, precision, recall and F1 scores " towards data science, 10 October 2019. [Online]. Available: https://lawtomated.com/accuracy-precision-recall-and-f1-scores-for-lawyers/.

[39] "Exporting a Trained Model," TensorFlow 2 Object Detection API tutorial, 2020. [Online].Available:https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/training.html#exporting-a-trained-model.

[40] "GPU support, Install TensorFlow" TensorFlow, 8 June 2021 from https://www.tensorflow.org/install/gpu.