Perception System: Object and Landmark Detection for Visually Impaired Users

Chenguang Zhang

University of Massachusetts Amherst

Follow this and additional works at: https://scholarworks.umass.edu/masters_theses_2

Part of the Systems and Communications Commons

Recommended Citation
https://doi.org/10.7275/19015521 https://scholarworks.umass.edu/masters_theses_2/1001

This Open Access Thesis is brought to you for free and open access by the Dissertations and Theses at ScholarWorks@UMass Amherst. It has been accepted for inclusion in Masters Theses by an authorized administrator of ScholarWorks@UMass Amherst. For more information, please contact scholarworks@library.umass.edu.
PERCEPTION SYSTEM: OBJECT AND LANDMARK DETECTION FOR VISUALLY IMPAIRED USERS

A Thesis Presented

by

CHENGUANG ZHANG

Submitted to the Graduate School of the University of Massachusetts Amherst in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN ELECTRICAL AND COMPUTER ENGINEERING

September 2020

Electrical and Computer Engineering
PERCEPTION SYSTEM: OBJECT AND LANDMARK DETECTION
FOR VISUALLY IMPAIRED USERS

A Thesis Presented

by

CHENGLANG ZHANG

Approved as to style and content by:

________________________________
Aura Ganz, Chair

________________________________
C. Mani Krishna, Member

________________________________
Beatriz Lorenzo, Member

Christopher V. Hollot, Department Head
Electrical and Computer Engineering
ABSTRACT

PERCEPTION SYSTEM: OBJECT AND LANDMARK DETECTION FOR VISUALLY IMPAIRED USERS

SEPTEMBER 2020

CHEN GUANG ZHANG

B.A., SHANDONG UNIVERSITY OF TECHNOLOGY
M.A., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Aura Ganz

This paper introduces a system which enables visually impaired users to detect objects and landmarks within the line of sight. The system works in two modes: landmark mode, which detects predefined landmarks, and object mode, which detects objects for everyday use. Users can get audio announcement for the name of the detected object or landmark as well as its estimated distances. Landmark detection helps visually impaired users explore an unfamiliar environment and build a mental map.

The proposed system utilizes a deep learning system for detection, which is deployed on the mobile phone and optimized to run in real-time. Unlike many other existing deep-learning systems that require an Internet connection or specific accessories. Our system works offline and only requires a smart phone with camera, which gives the advantage to avoid the cost for data services, reduce delay to access the cloud server, and increase the system reliability in all environments.
# LIST OF CONTENTS

<table>
<thead>
<tr>
<th>ABSTRACT</th>
<th>iii</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF TABLES</td>
<td>vi</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>vii</td>
</tr>
</tbody>
</table>

## CHAPTER

1. INTRODUCTION ................................................................. 1

2. LITERATURE SURVEY ............................................................ 3

   2.1 General Object detection .............................................. 3

      2.1.1 Problem definition .................................................. 3
      2.1.2 Development of object detection algorithm ...................... 3
      2.1.3 Speed/Accuracy trade-off of object detection algorithm ........ 5

   2.2 Perception assist systems using computer vision ..................... 6

      2.2.1 Research works ....................................................... 7
      2.2.2 Commercial applications ............................................ 10

3. SYSTEM OVERVIEW ................................................................... 12

4. OBJECT DETECTION .................................................................. 15

   4.1 Network Construction ....................................................... 15

      4.1.1 SSD ........................................................................... 16
      4.1.2 MobileNet .................................................................... 17
      4.1.3 Improvement of network construction ................................ 19

   4.2 Dataset Construction ........................................................ 21

      4.2.1 Data Collection .......................................................... 21
      4.2.2 Data Annotation .......................................................... 22
      4.2.3 Dataset Transformation ................................................ 23

   4.3 Model Training ................................................................... 24
   4.4 Model Optimization ........................................................... 27
LIST OF TABLES

Table 1 Performance and speed comparison ......................................................... 5
Table 2 Comparison of Perception assist system using computer vision .................. 8
Table 3 Detection speed on mobile devices ......................................................... 40
LIST OF FIGURES

Figure 1 Two phase architecture of the proposed system .................................................. 12
Figure 2 Detailed data flow of the proposed system ............................................................. 13
Figure 3 Architecture of offline training module ................................................................. 15
Figure 4 SSD network architecture ...................................................................................... 16
Figure 5 Default box generated in 4 × 4 feature map ............................................................. 17
Figure 6 Computation complexity of convolutional layer ...................................................... 17
Figure 7 Computation complexity of depthwise convolutional layer ................................... 18
Figure 8 Aspect ratios of bounding box ground truth distribution in PASCAL VOC .... 19
Figure 9 Aspect ratios of bounding box ground truth distribution in proposed dataset ... 20
Figure 10 Aspect ratio cluster results using k-means when k=5 ........................................... 20
Figure 11 Model optimization pipeline ................................................................................. 28
Figure 12 Pinhole camera model .......................................................................................... 30
Figure 13 2D pinhole camera model .................................................................................... 30
Figure 14 Mobile Implementation pipeline ........................................................................... 32
Figure 15 Architecture of distance calculation module ......................................................... 34
Figure 16 Object Detection mAP ......................................................................................... 38
Figure 17 Object Detection Recall ....................................................................................... 38
Figure 18 Detection accuracy at different distances ............................................................... 39
Figure 19 Real distance and estimation distance for a fixed object ....................................... 41
Figure 20 Real distance and estimation distance in object detection application ............... 41
CHAPTER 1

INTRODUCTION

According to the World Health Organization, eye conditions are remarkably common [1]. Those who live long enough will experience at least one eye condition during their lifetime. Globally, at least 2.2 billion people have a vision impairment or blindness [1]. In the United States, as documented by the American Diabetes Association, diabetes is the leading cause of new cases of blindness among adults aged 20–74 years. In 2005-2008, 4.2 million (28.5%) people with diabetes aged 40 years or older had diabetic retinopathy, and of these, almost 0.7 million (4.4% of those with diabetes) had advanced diabetic retinopathy that could lead to severe vision loss [2]. According to the National Institution of Health, 3.2 million Americans had visual impairment in 2015 [3]. Another 8.2 million had vision problems due to uncorrected refractive error. In the next 35 years, legal blindness will increase by 21 percent each decade to 2 million by 2050 [3]. Visual impairment is becoming one of the major global health issues.

Vision is one of the most important perception systems for people to find objects or landmarks which help them build a mental map of an unfamiliar environment.

The proposed application which runs in Android smartphones will enable visually impaired users to detect objects and landmarks within the line of sight. The system works in two modes, landmark mode or object mode. Landmark mode detects predefined landmarks and object mode detects objects they need in everyday life. Users will get verbal notifications about the name of the detected object or landmark and their estimated distances from the user. Landmark detection will help visually impaired users explore an unfamiliar environment and build a mental map.

The main contributions of the thesis include:

- Build an object detection network that runs on a mobile platform (e.g. Smartphone).
• Deploy the trained model on a mobile platform and optimize the inference to obtain real-time detection

• Build the perceptual system to give users verbal instructions that help them understand the environment.

The document is organized as follows: In Chapter 2, we survey object detection algorithms as well as environment perception systems. In Chapter 3, we introduce the proposed system and in Chapter 4, we describe the proposed object detection method. In Chapter 5, we discuss the implementation of an Android application. The proposed work is presented in Chapter 4.
CHAPTER 2
LITERATURE SURVEY

2.1 General Object detection

2.1.1 Problem definition

The term detection includes three-levels of visual abilities: identify, categorize and discriminate. According to the ability level of an object detection system, it can be grouped into two major types: detection of instances and detection of categories. Classify the object into categories (for example human, cars, animals, cups) are the most common focus for today’s research.

2.1.2 Development of object detection algorithm

a. Early stage

At the early stage of object detection, researches handcrafted invariant features. Among these works, Scale Invariant Feature Transform (SIFT)[4] features published in 1999 gained significant success and marks the beginning of the object detection field. Another milestone of object detection is the use of local descriptors such as Histogram of Oriented Gradients (HOG)[5].

b. Deep Learning for object detection

According to the survey Li Liu et al.[6] is a turning point of object detection with the work of Krizhevsky et al.[7]. The work of deep convolutional neural network(DCNN) named AlexNet provided excellent image classification accuracy. Following the publication of DCNN classifier, the first deep learning object detector was OverFeat by P. Sermanet et al.[8] This approach uses a combination of CNN and sliding windows. It uses the sliding window to cover each part of the
image and classifies every window to be an object or not. Then, it integrates the windows together and provides the final prediction result.

The following approaches in the past five years can be categorized into two categories:

- **Two-step methods.** This approach separates the task into two parts. First called region proposal: find the regions in the image that have a high possibility to be objects. And the second part classifies all the regions.

- **Single-step methods.** Without using a specialized region proposal, it makes the process one-stage which makes a fixed number of predictions on the grid.

Generally, two-step models get better accuracy and single-step models have faster operating speed and better memory efficiency.

The two-step models include R-CNN[9], SPPNet[10], fast-R-CNN[11], faster-R-CNN[12]. The faster R-CNN was considered to be the final stage of R-CNN object detection family. R-CNN is a combination of region proposal method and deep convolutional network. It consists of several stages: first, it uses selective search to get all the object candidates (about 2000 for each image). Then crops and warps each candidate region into the same size[9]. Using CNN it extracts features of each candidate, and using SVM it classifies each candidate. In the end, a bounding box regression is applied to fix the bounding box. SPPNet introduces two major improvements compared to RCNN. It introduced “spatial pyramid pooling” to solve the accuracy loss caused by using fixed-size CNN. Also, it computes a feature map to avoid computing each candidate object repeatedly[10]. As a result, SPPNet achieved 24-102 times faster processing time than RCNN.

Fast RCNN introduces an end-to-end method for object detection. It includes a region of interest (RoI) pooling which can be treated as a simplified version of spatial pyramid pooling in SPPNet. RoI pooling reduces computation times while it maintains the same accuracy. Although fast RCNN
significantly improved the object detection part, it still uses the selective search method to get object candidates. Faster RCNN introduces a deep convolutional network called region proposal networks to generate object candidates. The network fully relies on CNNs.

Single-step methods do not use a specialized region proposal which significantly speeds up detection. The YOLO[13] group of architectures were constructed in the same way as the SSD architectures. The image was run through a few convolutional layers to construct a feature map. The concept of anchors was used here too, with every grid cell acting as a pixel point on the original image. The YOLO algorithm-generated two anchors for each grid cell. Single Shot Detector came out in 2015, boasting state of the art results at the time and real-time speeds. The SSD uses anchors to define the number of default regions in an image. These anchors predict the class scores and the box coordinates offsets. A backbone convolutional base (VGG16[16]) is used and a multitask loss is computed to train the network.

2.1.3 Speed/Accuracy trade-off of object detection algorithm

According to J. Huang[15], SSD models with Inceptionv2 and MobileNet feature extractors are the most accurate of the fastest models. [15] compared several different combinations of object detection frameworks and feature extractors using mean Average Precision(mAP) to evaluate their accuracies (see Table 1). Note that if we ignore post-processing costs, MobileNet seems to be roughly twice as fast as Inception v2 while being slightly worse in accuracy.

<table>
<thead>
<tr>
<th>Model summary</th>
<th>minival mAP</th>
<th>test-dev mAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Fastest) SSD w/MobileNet (Low Resolution)</td>
<td>19.3</td>
<td>18.8</td>
</tr>
<tr>
<td>(Fastest) SSD w/Inception V2 (Low Resolution)</td>
<td>22</td>
<td>21.6</td>
</tr>
<tr>
<td>(Sweet Spot) Faster R-CNN w/Resnet 101, 100 Proposals</td>
<td>32</td>
<td>31.9</td>
</tr>
<tr>
<td>(Sweet Spot) R-FCN w/Resnet 101, 300 Proposals</td>
<td>32</td>
<td>31.9</td>
</tr>
<tr>
<td>(Most Accurate) Faster R-CNN w/Inception Resnet V2, 300 Proposals</td>
<td>30.4</td>
<td>30.3</td>
</tr>
<tr>
<td>---------------------------------------------------------------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td></td>
<td>35.7</td>
<td>35.6</td>
</tr>
</tbody>
</table>

Where minimal mAP describes the mean Average Precision on COCO minival dataset, test-dev mAP describes the mean Average Precision on COCO test-dev dataset.

In our application, we care more about computation speed than accuracy as we are trying to implement the model on mobile devices and run it in real-time.

2.2 Perception assist systems using computer vision

Many research projects are offering visually impaired people assistants using computer vision. Generally, the system that offers visually impaired assistance can be classified into two parts: user navigation and user perception guidance. The first part focus on offering users directly the action instruction. For example, systems [17] generate navigation instruction about what action visually impaired users should make. Users get the instruction without understand the environment. Another approach otherwise focuses on help visually impaired users understand the environment instead of helping users make the decision, like system [28]. Our approach will focus on the perception guidance and we will discuss the different approaches differ in the following features:

- Algorithms
- Objectives
- Accessible features
- Internet requirement
- Technical implementations which include:
  - Platform
  - Hardware requirements
  - Software implementation
2.2.1 Research works

In [28] the author proposed a system that guides the user to reach the object via audio output with speech command control. The whole system is wearable and can be placed on the user’s chest. The system is implemented on Raspberry Pi based on OpenCV library. The speech control feature is implemented through Google API and an object is detected using Haar cascade classifiers and color-based object detection techniques in OpenCV. Although the Haar-cascade classifier achieves quick detection on objects with clear edges, it requires manual determination of Haar-features which means it limits the number of objects the detector can detect.

In [29] the authors introduce a system based on Optical Character Recognition(OCR) and text-to-speech (TTS). A sign detection module integrated with the localization feedback system guides users to find signs and reads the text to users. The whole system is based on Android system. The paper uses shape detection to detect signs by focusing on rectangular signs. The test results show that the system can help visually impaired people read signs. However, the system features are limited to rectangular signs in the environment and localization feedback is only usable when part of signs get detected. In most situations, it can be difficult for visually impaired people to find signs.

In paper [30] the authors presented a system based on Android platform which detects objects in images captured through a camera and outputs the name of objects according to the relative position of the objects. The whole system mainly has two parts: object detection and binaural audio transformation. The author uses Google Vision API to detect objects in the image which requires an internet connection. The stereo matching algorithm was used to calculate the distance between the user and the object and using OpenAL to produce binaural audio. Binaural audio can be a great method to help visually impaired to understand the environment. However, it
has two main drawbacks. First, the binaural audio requires headphones which can isolate the user from the real environment. Second, users need pretraining to fully understand the position represented in 3D audios.

In [31] authors introduce a framework for obstacle detection to help visually impaired people detect danger ahead. Obstacle detection starts by selecting a set of interest points extracted from an image grid and tracked using the multiscale Luca-Kanade algorithm. Then, they apply HOG descriptor extraction, visual vocabulary computation, and SVM classifier to classify the objects. This approach can operate on the device offline but it only classifies the object as an obstacle or not, which limits the use of the application.

The system introduced in [32] offers visually impaired people functions of voice assistant, object recognition, currency recognition, and text reader through an Android smartphone. The system implemented relies on Google APIs. All cloud APIs use a JSON Rest interface and request using HTTP callback. Though the implementation is responsive in a stable internet environment, it can be unusable when the user loses internet connection.

In [33] the author introduces a system that produces binaural audio of objects. It utilizes several sensors including Depth-of-Field camera, stereo RGB camera, and Inertial Measurement Unit to get the 3D information of the environment. The special hardware platform improves the accuracy of 3D representation of the environment. Also, it increases the cost of the system.

We propose an Android application that uses computer vision and deep learning to detect intended objects using TensorFlow Lite.

<table>
<thead>
<tr>
<th>Paper</th>
<th>Implementation</th>
<th>Algorithm</th>
<th>Objective</th>
<th>Accessibility</th>
<th>Internet required</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>System</td>
<td>Implementation Method</td>
<td>Feature</td>
<td>Voice Control</td>
<td>Result</td>
</tr>
<tr>
<td>---</td>
<td>------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------</td>
<td>---------------</td>
<td>--------</td>
</tr>
<tr>
<td>1</td>
<td>Raspberry Pi based wearable device with sonic sensor, buzzer, and headphones and implemented using OpenCV</td>
<td>Haar Cascade Algorithm</td>
<td>guild user to object detected</td>
<td>voice control</td>
<td>Yes</td>
</tr>
<tr>
<td>2</td>
<td>Android software</td>
<td>color filter, Optical Character Recognition (OCR), TTS</td>
<td>sign detection and read text on the sign</td>
<td>text to speech</td>
<td>No</td>
</tr>
<tr>
<td>3</td>
<td>Android software using OpenCV for segmentation, Google API to classify an object, OpenAL to produce binaural audio</td>
<td>Canny’s edge detection, Stereo matching algorithm, Binural audio transform</td>
<td>object detection and output binaural audio</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td>4</td>
<td>software</td>
<td>multiscale Lucas-Kanade algorithm, SVM</td>
<td>obstacle detection and classification</td>
<td>N/A</td>
<td>No</td>
</tr>
<tr>
<td>5</td>
<td>Android software</td>
<td>Google Cloud API</td>
<td>voice assistant, image recognition, currency recognition, e-book, chatbot</td>
<td>voice control</td>
<td>Yes</td>
</tr>
<tr>
<td>6</td>
<td>Customized wearable device including a stereo camera, depth camera, and IMU sensor</td>
<td>hardware (3D acquisition system) and software (3D processing pipeline)</td>
<td>3D representation of the environment and converted obstacles and depth information to voice</td>
<td>N/A</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Our system</td>
<td>Android software based on TensorFlow</td>
<td>Object detection and transfer to audio</td>
<td>Talkback</td>
<td>No</td>
</tr>
</tbody>
</table>
There are a number of commercial applications that provide assistance to visually impaired people using computer vision. These projects can be classified into three categories:

- Computer vision-based object identification software
- Wearable visual enhancement devices
- Peer to Peer help software

2.2.2 Commercial applications

1. Computer vision based object identification software: Seeing AI, Aipoly, TapTapSee

   Seeing AI[34]
   
   This project developed by Microsoft offers 9 channels of detection. Users can obtain different kinds of recognition including short text, document, product by barcode, person, scene, currency, light, color and handwriting. The application is available on iOS devices.

   TapTapSee[35]
   
   TapTapSee’s interface is concise and returns detection results after the user takes a picture. The process may take 5-10 seconds to process.

   Aipoly[36]
   
   Aipoly offers almost real-time identification of objects and can switch the categories of detection. The identification is picture based so if there are multiple objects in one picture, it may give a wrong result. Aipoly could work offline but the accuracy is significantly lower than Seeing AI and TapTapSee.

2. Wearable visual enhance device: Orcam

   Orcam [37] empowers users to read text, recognize faces and identify products. It is a set of camera and headphone which can be attached on user’s glasses. However, people and products that need identification can only be used after saving them in the device manually. It supports
saving 100 faces and 150 products. This decrease the usability for users in an unfamiliar environment. Moreover, the hardware is specialized and therefore expensive.

3. Peer to Peer help software: Aira, Be My Eyes

Visually impaired users request help and volunteers or paid participants with vision can offer help.

Aira[38]

Aira has a group of trained agents to help blind users for their daily requirements and charges by connection minutes. This can be costly but the agent offers efficient help.

Be My Eyes[39]

Instead of hiring trained agents, “Be My Eyes” has two groups of users, visually impaired and volunteers. Currently, it has 149,999 blind users and 2,801,199 volunteers. Once a blind user starts a request the app pairs him/her with an online volunteer.
CHAPTER 3

SYSTEM OVERVIEW

The system introduced in this paper can help visually impaired users identify their surrounding objects or landmarks as well as estimate their distance from the user. We implement the application in Android platform and will not require Internet connection.

A detailed figure of the proposed system is provided in Figure 1.

Figure 1 Two phase architecture of the proposed system
**Offline processing** – The system uses computer vision and deep learning to detect intended objects. We use SSD (Single Shot Detector)[14] as the main architecture of the object detection algorithm. In order to improve the processing speed, MobileNet feature extractor was chosen to replace the VGG[16] feature extractor in the original SSD algorithm. Also, we performed extensive optimization to enable us to run the application even in a budget cell phone.

We built several datasets for different environments using Pascal VOC format which includes images, annotations in XML architecture and train/test set indication files.

The model was trained in Google Cloud platform because of the limitation of the computation power of a personal computer. The trained model was transformed into a frozen graph so it can be used on a mobile platform. Furthermore, in order to take advantage of inference speedup offered by TensorFlow lite API, the model was further transformed into TensorFlow Lite flatbuffer format.
**User interface** – As we focus on users with visual impairment the user interface was developed using Android accessibility features.

**Result translation module** – This module was designed to further improve detection accuracy without changing the model. Also, each detection returns ten top possible results and it is possible that several objects exist in one frame. This module selects the most confident result and transfers the detection results to understandable verbal instructions for users.

**Camera calibration** – In order to calculate the distance from the user to an object, we have to calibrate the camera to learn the camera matrix.

**Distance calculation** – We assume that the user uses the phone at the same height as they calibrate the camera. The distance calculation algorithm will be discussed in detail in Chapter 6.

**Camera module** – The system was implemented in Android platform and specifically in Android Studio. We used Java to develop the application and TensorFlow Lite for deep learning inference.

**Android TensorFlow Lite API** - TensorFlow Lite is an open-source deep learning framework for on-device inference. Empowered by this framework, our application can use deep learning models running on the device.
CHAPTER 4

OBJECT DETECTION

The offline training module architecture is introduced in Figure 3:

![Architecture of offline training module](image)

**Figure 3 Architecture of offline training module**

In this chapter, we will discuss the neural network for model training, dataset construction, training model on Google cloud computing engine as well as the post-optimization process of the model.

4.1 Network Construction

As discussed in Chapter 2, SSD with MobileNet architecture is one of the fastest solutions for object detection model training.
4.1.1 SSD

SSD model introduced in 46[14] and shown in Figure 4 includes three parts, convolution neural network, object detection, and NMS filter. More details on each part are provided below.

![SSD Network Architecture](image)

**Figure 4 SSD network architecture[14]**

**Convolution:** VGG-16 convolution network was used in SSD.

**Object determination:** This part includes five convolutional layers and one average pooling layer. Results from each layer offer different predictions. This strategy allows different predictions from different scales of features. All the results detected in different scales will pass through an NMS module to obtain the final result.

**Default boxes and aspect ratios:** SSD includes the concept of default boxes which is similar to anchor theory in Faster RCNN. For each element in the convolutional feature map we determine one central point called feature map cell in the original input. Centered around the point, the network generates 4 default boxes as shown in Figure 5. Also, since most of the boxes are negative, it introduces a significant imbalance between negative and positive bounding boxes. Therefore, SSD uses a strategy called hard negative mining. This approach only uses the part of negative samples, which have the largest loss, to keep the ratio of negative and positive samples 1:3. Data augmentation is proved to be of great importance for training. Sample patches have IOU of [0.1, 0.3, 0.5, 0.7, 0.9] with ground truth bounding boxes and random patches.
**Result filter:** SSD uses a fast non-maximum suppression (Fast NMS) mechanism to filter results from different scales.

### 4.1.2 MobileNet

MobileNet is a high-efficiency neural network that can be implemented on a mobile platform [41] while its accuracy is close to VGG16, which can be used as the feature extractor in SSD.

The main idea behind MobileNet is to use depthwise convolution to replace regular convolutional layers that apply to all the channels of input (see Figure 6). In contrast, depthwise convolution separates each channel (see Figure 7). In order to obtain the same result as a traditional convolutional, a pointwise convolution layer is performed after depthwise convolution.

---

**Figure 5** Default box generated in $4 \times 4$ feature map

**Figure 6** Computation complexity of convolutional layer
So, this kind of separation reduces the traditional cost:

\[ \text{Cost}_{\text{conv}} \times k^2 \times \text{channel} \]

To:

\[ \text{Cost}_{\text{conv}} \times k^2 + \text{Cost}_{\text{conv}} \times \text{channel} \]

Where \( k \) is the size of convolution kernel, \( \text{Cost}_{\text{conv}} \) the computation cost for one standard convolution. For example, in \( 3 \times 3 \) convolutional layer \( (k = 3) \) this separation offers about 1/9 of the traditional computation cost.

For actual convolutional layer in VGG16, the convolutional layer includes:

\[ 3 \times 3 \text{Conv} \rightarrow \text{BN} \rightarrow \text{ReLU} \]

In MobileNet, one step of depthwise separable convolution includes:

\[ 3 \times 3 \text{Depthwise Conv} \rightarrow \text{BN} \rightarrow \text{ReLU} \rightarrow 1 \times 1 \text{Conv} \rightarrow \text{BN} \rightarrow \text{ReLU} \]

Where BN is a batch normalization layer and ReLU is the active function.

So the final network architecture of MobileNet looks like follows: the image is first input into a standard convolutional layer. Then followed by 13 layers of depthwise separable convolution. Finally, the result will be processed by an average pooling layer followed by a fully connected layer.

When we use MobileNet in SSD object detection, we use the output of depthwise separable convolution at layer 13 as feature map.
4.1.3 Improvement of network construction

As we have discussed the default box chooses SSD strategy which separates each feature map into \( m \times m \) cells, where \( m \) is the size of each feature map. So every cell in the feature map has default boxes of predefined aspect ratio. One problem with this strategy is that the aspect ratio is determined based on experience. So, inspired by YOLO, we decided to use K-Means cluster to determine the default box to generate boxes more suitable to our own dataset.

In Pascal VOC dataset:

![Figure 8 Aspect ratios of bounding box ground truth distribution in PASCAL VOC dataset](image)

Aspect Ratios:

\[
[2.0118939 \ 0.83291479 \ 0.81014323 \ 0.62323348 \ 1.11272729]
\]

Follows is bounding box distribution of our own dataset:
We use the annotation in our dataset to cluster for the aspect ratio of the default box. Where $k$ cluster is a number of default boxes of each feature map cell. Entering the coordinate of every annotation and clustering the ratio of width and height of each bounding boxes.

When clustering the aspect ratios for ground truth in our dataset to 5 centers:

Where aspect ratios are $[1.64563807, 0.53292845, 0.43668083, 0.48733092, 0.88960759]$. 
4.2 Dataset Construction

As discussed in former chapters, we build an object detection system on a mobile platform that combines accuracy and speed and used by visually impaired users. The user will use our application to find the surrounding objects and landmarks and their distance from the user.

Our environment is North Station, Boston in which we determine eight classes of important landmarks: ATM machine, bench, Dunkin Donut store, elevator, escalator, fare gate, fare machine, police assistance sign.

4.2.1 Data Collection

Since the application will be used on a mobile device, we decided to collect the images using the mobile phone camera. Also, during the data collection process, we capture the images when the phone is placed in front of the chest. Then, we choose and annotate images that contain the desired objects. We arrange the dataset in Pascal Visual Object Classes (VOC) structure [40] as follows:

1. VOCtemplate
   2. └── VOC
      3. ├── Annotations
      4. │   └── 000001.xml
      5. └── ImageSets
         6. │   └── Main
         7. │       └── train.txt
      8. └── JPEGImages
         9. └── 000001.jpg
Where “Annotations” includes annotation files in XML format, “ImageSets-Main” has TXT files that indicate the file name of the training set and testing set, and “JPEGImages” has all the images in JPG format.

4.2.2 Data Annotation

For each standard annotation file, we use XML file to store all the information of the source images.

An annotation example of one image is shown below:

```
1. <annotation>
2.   <folder>faregate</folder>
3.   <filename>00001.jpg</filename>
4.   <path>C:\Users\zcg19\Downloads\VOC\VOC2012\8classes\faregate\00001.jpg</path>
5.   <source>
6.     <database>Unknown</database>
7.   </source>
8.   <size>
9.     <width>450</width>
10.    <height>600</height>
11.    <depth>3</depth>
12.   </size>
13.   <segmented>0</segmented>
14.   <object>
15.     <name>FareGate</name>
16.     <pose>Unspecified</pose>
17.     <truncated>0</truncated>
18.     <difficult>0</difficult>
19.     <bndbox>
20.       <xmin>151</xmin>
```
Each annotation file corresponds to one image in the dataset, and can contain more than one object in one annotation. The file starts with tag `<annotation>`, and uses `<object>` tag for each object in the image. The tag `<size>` indicates the size of the original image, and `<bndbox>` in `<object>` section shows the x, y coordinates (in pixel) of top left point and bottom right point of a bounding box.

4.2.3 Dataset Transformation

We set our model using TensorFlow and train the model on Google Cloud. We transfer the image dataset to TFRecord format that can help us read data efficiently. The TFRecord format is
a simple format for storing a sequence of binary records. It can be helpful to serialize our data and store it in a set of files that can be read linearly. This can also be useful for caching any data-preprocessing.

4.3 Model Training

The TensorFlow Object Detection API is an open-source framework built based on TensorFlow platform. This framework makes the process of building an object detection network easy and convenient. Also, due to the limited computation power in a desktop computer, we trained the model in Google cloud platform.

Instead of starting over from zero, we use transfer learning to accelerate the training process. We use SSD MobileNet checkpoint to fine tune our model, i.e. we take the weight that has already been trained on large amounts of data for similar tasks. The checkpoint we use in this project is pretrained from COCO dataset which includes a larger range of objects in images.

Data augmentation

Data pre-processing is important to improve the training accuracy. There are two kinds of data augmentation: 1. Geometric shift 2. Color jittering. These include affine transformations, perspective transformations, contrast changes, dropout of regions, saturation changes, cropping, padding, blurring and so on. In our situation, intended objects are less likely to be rotated so we do not use the geometric shift like rotation. In contrast, since lighting of indoor environments can change rapidly, we add more saturation based data augmentation. We use random_adjust_brightness, random_horizontal_flip, ssd_random_crop_pad for data augmentation.

optimizer

As suggested in [43], we use cosine decay learning rate
1. cosine_decay_learning_rate {
2.     learning_rate_base: 0.02
3.     total_steps: 200000
4.     warmup_steps: 0
5. }

Loss

1. loss {
2.     classification_loss {
3.         weighted.sigmoid_focal {
4.             alpha: 0.75,
5.             gamma: 2.0
6.         }
7.     }
8.     localization_loss {
9.         weighted.smooth_l1 {
10.            delta: 1.0
11.         }
12.     }
13.     classification_weight: 1.0
14.     localization_weight: 1.0
15. }

batch normalization

Unsimilar to SSD, we add batch normalization in all layers and use truncated normalization to initialize the weight.

1. initializer {
2.     truncated_normal_initializer {
3.         stddev: 0.03
4.     }
5. }

25
Anchor generator

In this part, we use the result of K-Means cluster to determine the aspect ratios of anchor generators.

Feature extractor

For the configuration of MobileNet network, we use RELU_6 as the activation function.
2. regularizer {
3.   l2_regularizer {
4.     weight: 0.00004
5.   }
6. }

4.4 Model Optimization

**Use quantized model training**

Quantization compresses the weights and activations in our model to an 8-bit fixed point representation. Referring to TensorFlow document, the size of quantization model is only ¼ of unquantized one and does not affect the final accuracy.

**Freeze the graph**

The model training process generates the model checkpoint file(.ckpt) and serialized graph. These files contain all the information we need for training. That is how we fine-tune our model from a pretraining model. However, it also contains many unnecessary parts for inference. In the inference process, we only need to feed the image in the network and compute results using the trained weight. All parts for adjusting weight become unnecessary.
Therefore, we freeze the checkpoint variable based on our graph. We can use TensorFlow API to freeze the graph. Also, we use optimization tools offered by TensorFlow to further optimize the model by removing training-only operations. These operations include:

- Stripping out parts of the graph that are never reached.
- Removing debug operations like CheckNumerics.
- Folding batch normalization ops into the pre-calculated weights.
- Fusing common operations into unified versions.

After this step, we get a smaller graph that can only be used for inference.

**Convert to TFLite**

Until now, we can recover the compute graph using TensorFlow on mobile devices. However, in order to take advantage of TensorFlow Lite API in Android studio and further accelerate inference speed, we transfer the model to TFLite model. Using TOCO, the TensorFlow Lite Optimizing Converter, we can transfer the model to TFLite format.
CHAPTER 5
DISTANCE CALCULATION

We seek to also provide the distance of the objects from the user which will help the users reach the object/landmark as well as build a mental map of the environment. Calculating distance means we are trying to extract 3D information from 2D images. A more accurate distance calculation using computer vision requires a stereo camera. But trading off between accessibility and accuracy, we choose to use single camera for distance estimation. An estimation is enough for visually impaired users to understand the environment.

This section will first introduce the calculation algorithms (Section 5.1) and talk about camera calibration in section 5.2.

5.1 Algorithm

Following assumptions are made for distance estimation:

1. Size of the detected object is known
2. The relative position between camera and objects are fixed

Under both assumptions, we would abstract the camera on the user’s mobile phone to be the pinhole camera. In the pinhole camera, objects from the scene reflect light in all directions. The size of the aperture is so small that only one of the rays of an object in the scene can enter the camera.
Figure 12 Pinhole camera model

Abstract the model in 2D plane:

\[ d = \frac{f}{w'} \cdot w \]

Figure 13 2D pinhole camera model

The vertical line passing through the middle is the main optical axis of the camera, \( d \) is the distance from the measured object to the lens, \( f \) is the focal length of the camera lens, \( w \) is the actual width (height) of the measured object, and \( w' \) is the width (height) on the imaging plane.

\[ d = \frac{f}{w'} \cdot w \]

can be derived directly from the similarity of triangles.

In our assumption, the actual width/height of the measured object is known. All we need to know are the focal length and actual size of the image sensor for the camera, which is a preset parameter of each phone. However, a mobile phone camera will contain installation error and
usually exhibit significant lens distortion, especially radial distortion. So we need geometric camera calibration to estimates the parameters of a lens and image sensor of the camera we use.

5.2 Camera calibration

According to [31], we calibrate the camera by viewing a chessboard pattern. Taking 11 images of the chessboard under different orientations by moving the camera. The image resolution is 640×480. The model plane contains a pattern of 7 × 9 squares. The size of the pattern is 20cm×20cm.
CHAPTER 6
MOBILE IMPLEMENTATION

Object detection implementation has already been introduced in Chapter 4. In this chapter, we will focus on the Android system design and how to infer the object detection model on mobile devices.

The mobile implementation includes the following modules: camera module, frame process, model inference, result translation, and user interface.

![Diagram of Mobile Implementation pipeline]

Figure 14 Mobile Implementation pipeline

6.1 Camera Module

Camera module controls the camera hardware on the mobile phone using Android framework APIs. First, we request user permission to use the camera:

1. `<uses-permission android:name="android.permission.CAMERA" />`
2. 

```
After we get permission, we use Camera2 API to get preview and image information. We use `CameraManager` to get the camera manager and `openCamera` method to open the camera. Within `CameraDevice.StateCallback` we use `onOpen` method to get the camera device instance and set the preview by creating a new `CameraCaptureSession`. When the session is created, `onConfigured` method will be called. In the `onConfigured` method, we use `setRepeatingRequest` to capture the image in real-time.

Also, we create the `ImageReader` for the preview frames. Using `setOnImageAvailableListener` method to create a listener.

```java
1. previewReader =
2.     ImageReader.newInstance(
3.         previewSize.getWidth(), previewSize.getHeight(), ImageFormat.YUV_420_888, 2);
4.
5. previewReader.setOnImageAvailableListener(imageListener, backgroundHandler);
```

The callback method `onImageAvailable` will return the captured image to `ImageReader`. Call `acquireLatestImage()` method on `ImageReader` instance will get the image data, which is YUV format.

**Frame process**

Images used for training our detection model are JPG images with RGB color model. So, we would transfer the YUV image to RGB image before on-device inference using image processing utilities offered by TensorFlow.
6.2 Distance module

The architecture of the distance calculation module is provided in Figure 15.

Figure 15 Architecture of distance calculation module

We briefly describe each module below.

**Camera Calibration Module:**

Camera calibration is a necessary step in computer vision to extract metric information from 2D images. Through camera calibration, we can get camera matrixes to include intrinsics, extrinsics, and distortion coefficients. To estimate the camera parameters, 3D world points and their corresponding 2-D image points are required. Zhang [42] provides an efficient camera calibration method using a chessboard. After camera calibration, we can get the camera matrices.

**Distance Calculation Module**

There are different ways to calculate the distance between the camera and the objects using the following assumptions:

1. Detected objects size is known
2. Fixed relative position between camera and objects
Under both assumptions, we would abstract the camera on the user’s mobile phone to be the pinhole camera. We implement the system on the mobile platform and test the distance accuracy and detection time for different environmental conditions such as diverse lighting conditions.

### 6.3 Model inference

To enable TensorFlow Lite in our app, we use ARR hosted at JCenter. As suggested by TFLite document, we can specify the dependencies by adding declaration in `build.gradle:

```gradle
1. dependencies {
2.     implementation fileTree(include: ['*.jar', '*.aar'], dir: 'libs')
3.     implementation 'org.tensorflow:tensorflow-lite:0.0.0-nightly'
4. }
5.
6. defaultConfig {
7.     ndk {
8.         abiFilters 'armeabi-v7a', 'arm64-v8a'
9.     }
10. }
```

This AAR includes all the binaries for ABIs. In order to reduce the size of the application, we constrain the ABIs to 'armeabi-v7a' and 'arm64-v8a'.
CHAPTER 7
TESTS AND RESULTS

7.1 Detection accuracy

7.1.1 Performance metrics

COCO is a large-scale object detection, segmentation, and captioning dataset. We use detection metrics presented by COCO to evaluate the performance of our object detection model.

**Average Precision (AP):**
- AP: AP at IoU=.50:.05:.95 (primary challenge metric)
- AP\textsuperscript{IoU=.50}: AP at IoU=.50 (PASCAL VOC metric)
- AP\textsuperscript{IoU=.75}: AP at IoU=.75 (strict metric)

**Average Recall (AR):**
- AR\textsuperscript{max=1}: AR given 1 detection per image
- AR\textsuperscript{max=10}: AR given 10 detections per image
- AR\textsuperscript{max=100}: AR given 100 detections per image

Where some important definitions including IoU, precision, recall is explained as follows.

**Intersection over Union (IoU)** is a measure based on Jaccard Index that evaluates the overlap between two bounding boxes. It requires a ground truth bounding box \( Box_{gt} \) and a predicted bounding box \( Box_{p} \). By applying the IoU we can tell if one detection is valid (True Positive) or not (False Positive).

IoU is given by the overlapping area between the predicted bounding box and the ground truth bounding box divided by the area of union between them:

\[
\text{IoU} = \frac{\text{area}(Box_{p} \cap Box_{gt})}{\text{area}(Box_{p} \cup Box_{gt})}
\]

In COCO definition, IoU = 0.5 means using IoU threshold to be 0.5. Based on the different thresholds, True Positive, False Positive, False Negative and True Negative are defined. For each detection result:
• When \( IoU \geq threshold \), defined as True Positive(TP)
• When \( IoU < threshold \), defined as False Positive(FP)
• When ground truth was not been detected, defined as False Negative(FN)
• True Negative: everywhere else. Ignored.

**Precision** is the percentage of true positive predictions:

\[
Precision = \frac{TP}{TP + FP} = \frac{TP}{all\ detection}
\]

**Recall** is the percentage of true positive detected among all relevant ground truths:

\[
Recall = \frac{TP}{TP + FN} = \frac{TP}{all\ ground\ truth}
\]

**Average Precision** is calculated by the area under the precision-recall curve. To simplify the average precision, an 11-point interpolation method is used by averaging the precision at a set of 11 equally spaced recall levels \([0, 0.1, 0.2, \ldots, 1]\).

\[
AP = \frac{1}{11} \sum_{r \in \{0.0, 0.1, \ldots, 1\}} \rho(r)
\]

Where \( \rho(r) \) is precision at recall \( r \).

### 7.1.2 Object Detection Accuracy

The detection accuracy is tested by using 200 images taking in the campus center to simulate scenarios when the user using the application.
As we can see in Figure and Figure, our model achieves 0.308 overall mAP and 0.63 mAP @.50IoU. While the baseline mAP for SSD with MobileNet model in [15] is 0.2 overall mAP and
0.38 mAP @ .50IoU. Through the model is trained from different datasets, we can tell the performance of our model is relatively well.

7.1.3 Detection Accuracy vs. Distance

In order to determine practical performance in the user’s mobile application, we tested the detection result at different distances. We use ten frames to determine the accuracy at each distance.

As expected, as the distance increases we observe that the accuracy is reduced. And the detection result shows that it is more reliable within 10 meters from the desired object, which is quite enough in practical especially for the indoor environment.

7.2 Detection speed

Without considering the time training the detection model, we use frame test time delay to evaluate the detection speed. The time delay measures whole processing time we take for each frame, including image pre-processing, object detection and distance calculation.

As discussed in Chapter 4.4, we use the quantization model to improve inference time on the mobile application.
Table 3 Detection speed on mobile devices

<table>
<thead>
<tr>
<th>Model ID</th>
<th>Device</th>
<th>Quantize</th>
<th>Model size</th>
<th>Speed (per frame)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>OnePlus 5</td>
<td>unquantized</td>
<td>20.8MB</td>
<td>130 - 160 ms</td>
</tr>
<tr>
<td>2</td>
<td>OnePlus 5</td>
<td>quantized</td>
<td>5.5MB</td>
<td>40 - 70 ms</td>
</tr>
<tr>
<td>3</td>
<td>Samsung s7</td>
<td>unquantized</td>
<td>20.8MB</td>
<td>200 - 280 ms</td>
</tr>
<tr>
<td>4</td>
<td>Samsung s7</td>
<td>quantized</td>
<td>5.5MB</td>
<td>60 - 110 ms</td>
</tr>
</tbody>
</table>

Our quantized model improves detection speed by 70% and reaches detection speed at 40-70ms per frame (15-25 frames per second). Oneplus 5 android smartphone is announced in 2017 and currently sale for a price less than 200 dollars. We can fairly say that the model can reach real-time detection on most user’s mobile devices.

7.3 Distance Estimation Accuracy

To evaluation the distance estimation accuracy, we compare actual and measured distance in the following scenarios:

- Distance estimation for a fixed object
- Distance estimation in object detection application

Distance estimate accuracy for fixed objects shows the accuracy of the desired distance estimate algorithm.
Figure 19 Real distance and estimation distance for a fixed object

Figure 19 shows the distance estimation algorithm gives acceptable accuracy for distance estimation.

For distance estimation in object detection application:

![Real and estimated distances](image-url)

Figure 20 Real distance and estimation distance in object detection application

We observe that the estimated distance error is significantly larger after combined with object detection. The error rate increases as expected because of the deviation of detected bounding boxes. Even at the same distance, the detected bounding box can change -10% to +10%, which
would significantly affect the final distance estimation and enlarge the error rate of distance estimation.

However, the distance estimation is only designed to provide a percept to landmarks for the visually impaired user and help them build their mind map, so the error rate is acceptable.
CHAPTER 8

CONCLUSION AND FUTURE WORK

The main contribution of the thesis is to build a perceptual system for visually impaired users based on computer vision. The system can give verbal instructions that help them understand the environment. The system works in two modes, landmark mode or object mode. Landmark mode detects predefined landmarks and object mode detects objects they need in everyday life. Users will get verbal notifications about the name of the detected object or landmark and their estimated distances from the user. Landmark detection will help visually impaired users explore an unfamiliar environment and build a mental map.

In system design and implementation, the thesis contributed to the following aspects: 1) Build an object detection network running on mobile platform. 2) Deploy the trained model on the mobile platform and optimize the inference to obtain real-time detection and 3) Calculate the distance from the user to detected objects.

The object detection network is built based on SSD mate architecture and utilizing MobileNet as feature extractor with an optimized default bounding chosen strategy. The detection model could run in real-time and reach 0.308 overall mAP. After deploying the detection model in the mobile platform, the system was tested in different scenarios. The performance result in terms of accuracy and detection time is excellent within a distance of 10 meters. It offers estimate distance from user to detected objects. We have also developed a vision free user interface using Android accessibility features.

However, several parts can be improved in the future. First, the number of objects that can be detected is limited by our model. If we want to enlarge the objects to be detected, we need to
retrain the model. Although it can be done more quickly using fine tuning strategy, training an object detection model is time consuming.

Secondly, the detection model provides accurate detection for objects name and localization on each frame, but it is not accurate enough for us to calculate the precise distance from objects and users. Also, for the distance estimation module, we use single camera to calculate the distance which requires that the size of the object is known. We may use a stereo camera to estimate the distance for objects of any size.
REFERENCES


[38] Aira, https://aira.io/


