CS-678 Topics in Internet Research
Mid term Report

Azmeer Faisal - 24100164
Muhammad Nameer Anjum - 24100270
Mustafa Bin Amir - 24100084

Friday, 31 March 2023

Github Project Link

1 Objectives

The aim of this project is to develop a mobile app that employs machine learning algorithms to identify potholes on roads in real-time. The app will use the smartphone's camera to detect potholes and then display augmented reality markers on the detected potholes. These markers will be visible to other users in the vicinity, allowing them to take appropriate action.

2 Introduction

This project involves the development of a mobile application that has two main sub-applications. The first sub-application is focused on the use of a machine learning algorithm to detect potholes on roads using the smartphone camera in real-time. The second sub-application involves the use of augmented reality to create shared experiences for multiple users, allowing them to view AR objects.

3 Pothole Detection

This part discusses the planning and experiment done for the pothole detection part.

3.1 Planning

Learning Android Development
Before commencing with the programming, we studied `Kotlin` to acquaint ourselves with the fundamental structure of the application.

General Object Detection Algorithm
To develop our object detection system, we conducted a thorough review of relevant research papers[3] and also watched various tutorials to explore different algorithmic approaches. We found relevant algorithms used for object detection such as `CNNs`, `SSDs` and `YOLOs`.

TensorFlow Lite Pre-Trained portable models
YOLOs and other CNNs are computationally intensive algorithms, hence cannot be used on resource limited mobile phones. Further research introduced us to TensorFlow Mobile or `TensorFlow-lite` models which are based on portable CNN architecture[3] such as `MobileNet` and `EfficientDet`.

Most Suitable portable Models
We identified `6` most suitable algorithms for our project such as

MobileNetv1	EfficientDetv0
EfficientDetv1	EfficientDet2
Inception	YOLOv4-tiny

These models provide different levels of average precision and inference time.

3.2 Experiment

First Sample App
We developed a basic app which was able to detect objects using pre-trained tflite models. However, the detection was not realtime rather the user had to upload images on the app to detect object.
This gave us an idea of the working model but our requirement of realtime camera was not met.

Sample App - "Spothole"
We used and modified an example `Kotlin` application that can recognise objects in `realtime` based on `TensorFlow` documentation[5]
We used model such as `MobileNet` and `EfficientDet` to detect objects. These models are pre-trained on `COCO dataset 2017`[4] which allows detection for ∼80 objects. The app is able to detect humans, cars etc but doesn't work on potholes hence we have to create our custom model. We will base our app "Spothole" on this sample app.

Generating Train/Test Data
We scrapped 369 unique images of potholes from google and YouTube videos.
We manually labelled them using LabelMe software to create Labelled data.
Extracted another ∼1300 lablled data of potholes from Kaggle. We decided on 80-20 Train/Test split to improve accuracy of model.

Custom trained tflite Model
We used the idea of Transfer training[3] to train our model, which allows us to train pre-trained models effectively, saving us a lot of time. TensorFlow documentation provided the colab link[2] which can be modified to train custom model.
We tried training EfficientDetv0 and EfficientDetv1 with custom dataset but the process failed due to incorrect file formats of dataset.

3.3 Failures

Unfortunately, there were several failures in the project that prevented us from achieving our milestone. These failures include:

1. Incompatible Dataset format: Our model training failed.
2. New Model integration: We tried using another pre-trained tflite model EfficientDetv1 to replace EfficientDetv0 in the app but app crashed. We did this to ensure the steps needed inorder to put custom model to app.

3.4 Results

Currently the sample TensorFlow app is able to recognise object efficiently at the distance of 3.7m, which is ideal for our usecase as this serves as the minimal distance between car (where is app is placed) and pothole.

4 Augmented Reality

4.1 Experiment

In order to create a shared augmented reality experience the `Google ARCore SDK` was utilised to develop an Android application on Android Studio that employs the Cloud Anchoring API. When AR Core creates a local anchor in the environment to identify the position of the AR object it hosts that anchors data to the `ARCore Cloud Anchor API`[1] which then returns a unique ID. Another user with that unique ID is able to recreate the same anchor in the same local environment hence recreating AR experience on a different device.
The application is written in Java and was recreated with the ar-core cloud anchor sample app on google ar's public repository on github. The flow of the application is as follows:

1. The application accesses the smartphone's camera and recognizes the surfaces using GLSurfaceView

2. When the user taps on screen the 3D object is placed on the respective position on the surface, and creates a new anchor

3. On pressing the Host Button on top of the screen a new Room Code is generated to identify the local environment and and the data along with the anchor data is hosted on a Firebase Cloud

4. On a separate device (or restart the app on the same device) in the same local environment press the Resolve button. Provide the appropriate Room Code

5. On pressing the Ok button the application fetches the cloud anchor data of the respective room and recreates the 3D objects on those anchors.

6. This user can now participate in the same AR experience as the first user.

4.2 Testing and Results

We have tested the application in closed private spaces. The application successfully resolves within 2-3 seconds on scanning the environment if the camera viewing angle is similar to that of the user who hosted the anchors. The application at times does not work when the resolver's viewing angle is completely opposite of that of the host's. The application works better in static environments. For dynamic environments we tried changing the setting of the room after hosting. The application would successfully resolve in similar time durations if the change in setting is not too drastic i.e few small items added to the environment. If change in the setting is too drastic i.e the surface on which the 3D object was placed is removed, then the application is unable to place the anchor successfully.

5 Future Work

We have decided to take a serial approach now instead of working on both sub applications in parallel.

- Our next step will be to successfully train the machine learning model on a considerable size dataset of pothole images to achieve a desired accuracy of greater than 70

- We then want to integrate this mobile app in a client server model where the application(client) estimates the distance to the pothole and sends this data along with the device's current location to the server.

- The server would then be able to send this data to other users warning them of a pothole and estimate how far it is located.

- The machine learning algorithm will then be implemented on the server side.

- If all the above steps are implemented successfully the next phase of the project would focus on using Google's Geospatial API instead of the cloud anchoring api which claims to work better on road environments and utilise it to place Augmented Reality objects as markers on the potholes.

References

[1] Cloud anchors - arcore.

[2] Model maker object detection tutorial - colaboratory.

[3] Oscar Alsing. Mobile object detection using tensorflow lite and transfer learning, 2018.

[4] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13*, pages 740–755. Springer, 2014.

[5] TensorFlow. Tensorflow documentation, 2022.