FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Mobile App using Object Detection for Car Driving

Filipe Campos • Francisco Gonçalves Cerqueira • Vasco Alves

Bachelor in Informatics and Computing Engineering

Supervisor: Ricardo Cruz

July 15, 2023

Contents

Chapter 1

Introduction

A technology race is "on" between car manufacturers to produce automatic systems to help human drivers. Currently, this technology goes from simple warnings and momentary assistance (SAE level 0) to fully self-driving systems under certain conditions (SAE level 3). However, these systems are expensive and the vast majority of the car fleet does not yet incorporate any of these systems. These systems typically use either RGB cameras or LiDAR sensors, or both. Tesla has been focusing mostly on RGB cameras, while Bosch/Daimler has focused more on LiDAR sensors. Neural networks are then used to translate these sensory inputs into semantic representations of the surroundings. Object detection systems already exist that provide high accuracy, even for mobile phone RGB cameras.

The goal of this project was to produce an Android application that generates an audible sound to warn the user in certain situations: when another car is in front of the user's car, when a pedestrian or another obstacle is on the road, and to notify on horizontal and vertical signs. The neural network should run on the mobile phone itself to mitigate problems that accompany the usage of a network connection to an external server. The mobile phone needs to be fixed in the dashboard so that it moves as little as possible and is positioned in such a way that it provides a good front-view.

This application has been available for Android (≥ 9) at the Play Store through the following link: https://play.google.com/store/apps/details?id=pt.up.fe.mobilecardriving for Android.

1.1 Objectives

- Develop a model that can detect several possible objects; namely, cars, passengers, traffic signs.
- Deploy the model in an Android application that users can use in their cars.

1.2 Report Structure

This report is structured as:

- **Chapter 2:** an overview of alternative solutions and models.
- **Chapter 3:** how the project was planned.
- **Chapter 4:** development and results from the project.
- **Chapter 5:** concluding remarks, including possible suggestions for improvement.

Chapter 2

State of the Art

The state of art first addresses possible alternatives to our project and also what the state of the art is in terms of object detection.

2.1 Autonomous Driving Systems

The ISO standard SAE divides autonomous driving systems into 6 levels:

- **Level 0:** no driving automation
- **Level 1:** driver assistance
- **Level 2:** partial driving automation – steering and accelerating/decelerating
- **Level 3:** conditional driving automation (driver may need to intervene at any moment)
- **Level 4:** high driving automation (in some situations, such as high-ways)
- **Level 5:** full driving automation

Cars such as Tesla are considered Level 2, which uses RGB camera sensors. The most advanced autonomous car currently being sold in the European market is the Mercedes S-Class [1], which is the first Level 3. This car uses LiDAR and RGB camera sensors.

In terms of driver assistance, many applications exist, such as Google Maps and others. Yet, as far as we know, none of these applications try to monitor to the environment and help the user in the navigation.

2.2 Object Detection

Object detection is a computer vision technique that consists in the detection of instances of objects in an image and their class. Currently there are two main approaches used for object detection: two-stage and one-stage object detection.

[1] https://europe.autonews.com/automakers/mercedes-opens-sales-level-3-self-driving-system-s-class-eqs

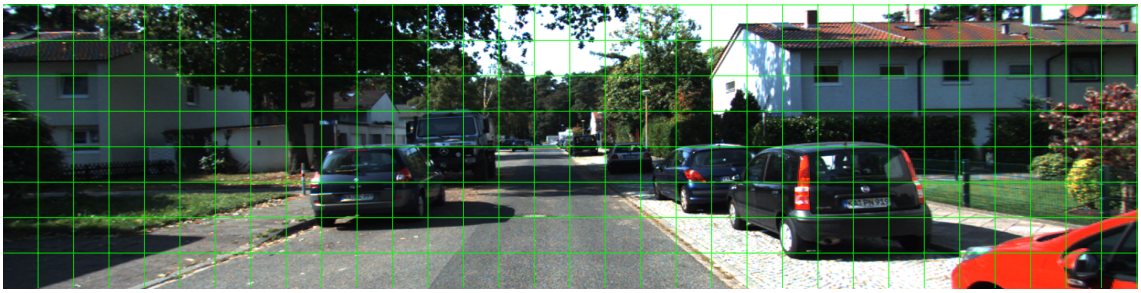


Figure 2.1: Image grid.

2.2.1 Two-stage Object Detection

R-CNN (Girshick et al., 2013) was the first modern neural network object detector. It comprised of two stages: (Stage 1) a region proposal stage that creates proposals of areas that might contain an object, (Stage 2) for each region a second stage predicts the presence or absence of an object and its type.

In the original paper, a neural network was only used for the second stage. This was improved by Faster R-CNN (Ren et al., 2015), which also used a neural network for the first stage.

These models are typically slower than one-stage models and have an inconsistent inference time, due to the dependency on the number of regions proposed for each image.

2.2.2 One-stage Object Detection

YOLO (Redmon et al., 2015) and SSD (Liu et al., 2016) introduced a new family of object detection models which skip the region proposal stage entirely, performing predictions on a single pass through the model. The original image is split into a grid, and, for each cell, the model predicts whether an object exists, and, if so, the respective class and bounding box. See Figure 2.1 for an illustration.

A post-processing step is still necessary because naturally the same object may be detected by multiple cells. This is usually done using the non-maximum suppression algorithm.

These models have been subsequently improved by such models as FCOS (Tian et al., 2019) which do not only predict *one* grid, but multiple grids using different scales to better catch objects of different sizes.

Chapter 3

Planning

At the beginning of the project, we established four main development phases to guide the progress of the project, that were iteratively adapted. This are illustrated in the chronogram in Figure 3.1.

	March				**April**				**May**				**June**			
	W1	**W2**	**W3**	**W4**	**W1**	**W2**	**W3**	**W4**	**W1**	**W2**	**W3**	**W4**	**W1**	**W2**	**W3**	**W4**
Phase 1: Experiments 1.1: Explore datasets 1.2: Package experimentation (torchvision, detectron2, objdetect)																
Phase 2: Android 2.1: Torch Mobile 2.2: Synthetic voice, UI, store																
Phase 3: Neural Network Improvements 3.1: Train with more outputs 3.2: Re-think detection & challenges																
Phase 4: Finalization 4.1: Presentation 4.2: Report/paper																

Figure 3.1: Chronogram of the planned work.

3.1 Experiments

This first exploratory stage served to acquaint ourselves with the available computer vision datasets and the state of the art object detection technology.

(a) KITTI. (b) GTSDB.

Figure 3.2: Illustration of the two datasets that were used (the ratio varies between the two datasets).

3.1.1 Dataset exploration

Our goal was too choose a dataset that had all the necessary labels for our purpose while containing enough data to train our model. To this effect we compared 7 different datasets, the label comparison table [Table A.1] shows that all the analysed datasets contained, at the very least, vehicle and pedestrian labels which are the most important classes for our project:

- The KITTI Dataset (Geiger et al., 2013) was chosen because it contains 7,481 fully annotated images with a 1242×375 pixels resolution, which is enough for our purpose, and due to its ease of use.
- Additionally, for the purpose of sign detection, we utilized the GTSDB Dataset (Houben et al., 2013) which contains 600 annotated images with a 1360×800 pixels resolution. The reason why this dataset was necessary is because, although KITTI also contains traffic signs, the signs are not annotated.

An image from the two datasets is shown in Figure 3.2 for illustrative purposes.

3.1.2 Package experimentation

Using the **Torchvision** (Marcel and Rodriguez, 2010) package, we tested multiple built-in pre-trained models, SSD (Liu et al., 2016), Faster R-CNN (Ren et al., 2015) and FCOS (Tian et al., 2019). Among all the tested models, FCOS yielded the best accuracy, but its inference time proved to be too slow for our use case.

Using an early version **objdetect** package (Cruz, 2022) we created three different one-stage models, a built-in model and two models with different pre-trained backbones from torchvision, ResNet-50 (He et al., 2015) and MobileNetV3 (Howard et al., 2019). These models yielded significantly worse results than their torchvision (Marcel and Rodriguez, 2010) counterparts and were deemed unusable. Table 3.1 shows some initial tests for the KITTI dataset using a 75-25 train-test split.

Models	mAP	mAP_{50}	mAP_{75}
Torchvision, FCOS	52.01	77.07	61.35
Objdetect, ResNet-50 backbone	8.95	27.05	3.90

Table 3.1: Mean Average Precision comparison between the best models of each package.

Despite the fact that we have planned to explore the **Detectron2** (Wu et al., 2019) library, this is a two-stage oject detection framework, therefore, we decided the time investment would not be worth it.

3.2 Android Application

A mobile application was developed, using Java, to use the trained model in real life scenarios.

3.2.1 Prediction Pipeline

Since the moment the image is captured until the moment the analysis results are displayed, such as boxes and warnings, the data goes through 4 steps:

1. **Pre-processing:** the image is cropped and adjusted to the required aspect ratio (4:1);
2. **Model Evaluation:** the image is evaluated by the model so it can return which classes were detected, along with the corresponding position and score;
3. **Post-processing:** the results returned by the model are now post-processed, returning a last analysis result that includes the collection of objects and warnings that should be displayed, as well as other relevant info, like the device's current speed according to the GPS.
   This analysis uses not only the information of the last frame, but also the last 3 frames, in order to increase the accuracy;
4. **Prediction Output:** the results obtained on the previous step are sent to the view class and sound warnings are played if necessary, using Synthetic Voice (TextToSpeech).

3.2.2 UI, Synthetic Voice, Store

The results of the analysis mentioned in the previous section are handled by the object detection view, whose purpose is to display, when requested, the results in a friendly and efficient way to the user. The view, constantly updated, allows the user to visualize in real-time the position and type of objects that are being detected by the model. It also shows the current movement state of the device.

To avoid distracting the driver, the application also comes with a Synthetic Voice system, in which the warning messages are played to inform the driver of any caution situation.

This application is available for Android (≥ 9) at the Play Store through the following link: https://play.google.com/store/apps/details?id=pt.up.fe.mobilecardriving

3.2.3 Torch Mobile

The conversion of the tested models to torch mobile (Paszke et al., 2019) took much longer than planned, due to some restrictions of the torch mobile interpreter which did not support the models that were created. Initially, the idea was to use models already bundled with torchvision (Marcel and Rodriguez, 2010) – yet, they were found to be incompatible with torch mobile. The models that are bundled with Torchvision make use classes that are not sub-classes of `nn.Module`, which makes them impossible to convert to the mobile interpreter.

To overcome this difficulty, we developed our own models – for that purpose, we have used the objdetect package (Cruz, 2022). This change costing us around two additional weeks over the planned schedule.

3.3 Neural Network Improvements

To improve our application, we added sign detection and recognition, this required some changes to the model, namely adding two additional outputs (scores and classes) for signs. Additionally, to obtain a faster and more accurate model, the class labels “Car”, “Truck” and “Van” were merged into a singular “Vehicle” label.

3.4 Finalization

In the last phase, we proceeded to the development of the deliverables, i.e. the report and the poster, taking the necessary screenshots to show the final product. We also made final adjustments to the application, most of them aesthetic related. It was also in this phase that we published our app at the Play Store.

Chapter 4

Development & Results

This project is divided into two main modules, the model, where a neural network model is created, trained and deployed, and the application.

4.1 Model

The final model was developed utilizing the objdetect package (Cruz, 2022) and a pretrained torchvision (Marcel and Rodriguez, 2010) MobileNet-v3 (Howard et al., 2019) was used as the model backbone due to performance constraints, even though ResNet-50 (He et al., 2015) proved to be have more accurate results.

Table 4.1 and Table 4.2 depict the metrics measured with different model backbones, for the GTSDB and KITTI Dataset, respectively.

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
mobilenet_v3_small	99.36384	96.80851	62.75862	76.15063
mobilenet_v3_large	99.58333	94.90291	81.12033	87.47203
resnet50	99.84375	99.26108	91.17646	95.04716

Table 4.1: GTSDB metrics.

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
mobilenet_v3_small	98.45818	97.96800	93.67720	95.77456
mobilenet_v3_large	98.61410	98.88839	93.68030	96.21391
resnet50	99.05568	98.94281	95.99198	97.44506

Table 4.2: KITTI metrics.

The largest model, with a resnet50 backbone, has the best results on all metrics, but it comes with a trade-off of increased inference time and size in storage, as shown in Table 4.3.

4.1.1 Dataset

As previously mentioned, the KITTI (Geiger et al., 2013) dataset was used together with the GTSDB (Houben et al., 2013) dataset. To read them two `Dataset` classes were created, GTSDB was completely built from the ground up while KITTI was adapted from the sample objdetect implementation with a minor change to convert different types of vehicle labels into a singular label.

- **GTSBD labels:** stop sign, give way, prohibited, no overtaking, end of no overtaking prohibition.
- **KITTI labels:** vehicle and pedestrian.

4.1.2 Training

This model was trained on the KITTI and GTSDB datasets using Albumentations (Buslaev et al., 2020) to perform data augmentation by adding random crops, flips, brightness and contrast transformations to the original image. The loss functions we use are sigmoid focal loss and cross entropy loss for scores and classes, respectively.

4.1.3 Inference

This model receives a $3\times256\times1024$ image and outputs two sets of scores and classes arrays, one for car and pedestrian detection and the other for sign detection as depicted in Figure 4.1. A score array contains 8×32 elements, each a float with a score between $[0,1]$, that predict whether or not that grid space contains an object. The classes array is composed of $K\times8\times32$, where K is the number of classes, by iterating for each grid element the K we can determine which type of object is more likely to be present.

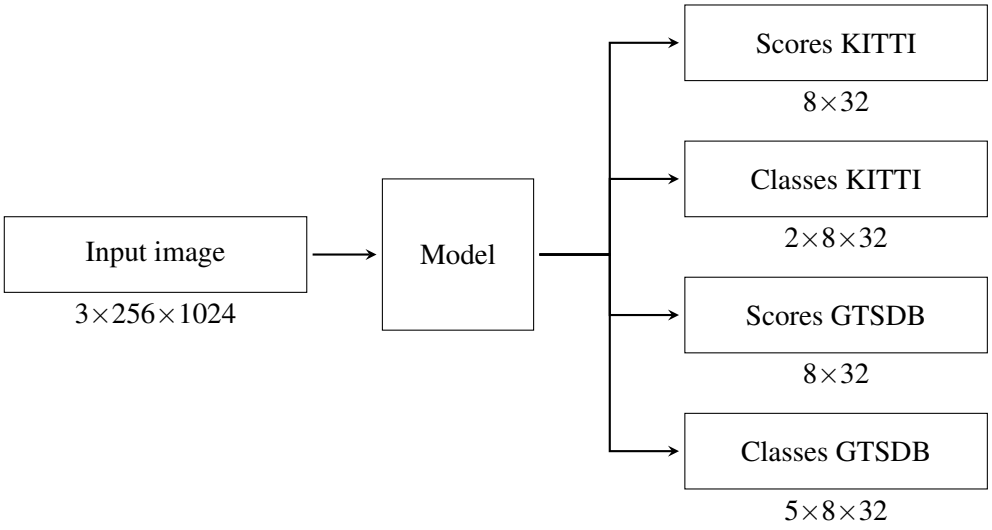


Figure 4.1: Model usage.

4.1.4 Torch Mobile Deployment

To deploy to Torch Mobile we utilize the workflow described in the official documentation, as illustrated by Figure 4.2. [1]

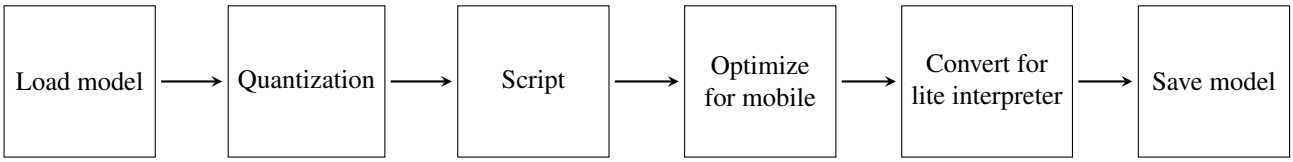


Figure 4.2: Torch Mobile deployment pipeline.

After converting the models to mobile, we compared their inference time and size, as shown in Table 4.3.

Model	Average Inference time (ms) [2]	Average FPS	Size (MB)
mobilenetv3_small	130.73845	≈ 7.6	3.6
mobilenetv3_large	502.39804	≈ 2.0	11.4
resnet50	2477.47302	≈ 0.4	89.7

Table 4.3: Mobile model comparison.

Using the model with a ResNet-50 (He et al., 2015) backbone is unfeasible due to its inference time, the MobileNet-v3 small (Howard et al., 2019) backbone proved to be the best compromise between speed, size and accuracy.

4.2 Code Structure

Since the beginning, the code was developed to be scalable, modular and efficient. To achieve that, several modules were created to keep it organized:

- **Activity**: Activities (or menus) of the application. Contains two abstract classes to utilize the camera and background threads and two classes the implement the main menu and object detection menu;
- **Analysis**: Contains a model to store the analysis information and two classes that are used at the post-processing;
- **Detection**: Contains a model to store the model evaluation information, a class to manage the dataset and relevant classes to the perform the model evaluation;
- **Motion**: Module responsible for obtaining GPS data that will be used to calculate the device's speed and notifying the listeners;

[1] https://pytorch.org/mobile/home/#deployment-workflow
[2] Measured on Xiaomi Mi A3, running Android 11

- **Speech**: Module responsible for interacting with the synthetic voice and notifying the listeners when the execution was completed;
- **Util**: Utility classes that were used in other modules
- **View**: Contains classes that are responsible for managing and updating the activity view;
- **Warning**: Models used to store information about warnings, such as the speech phrase and priority.

Figure 4.3 shows an overview of the class structure we used.

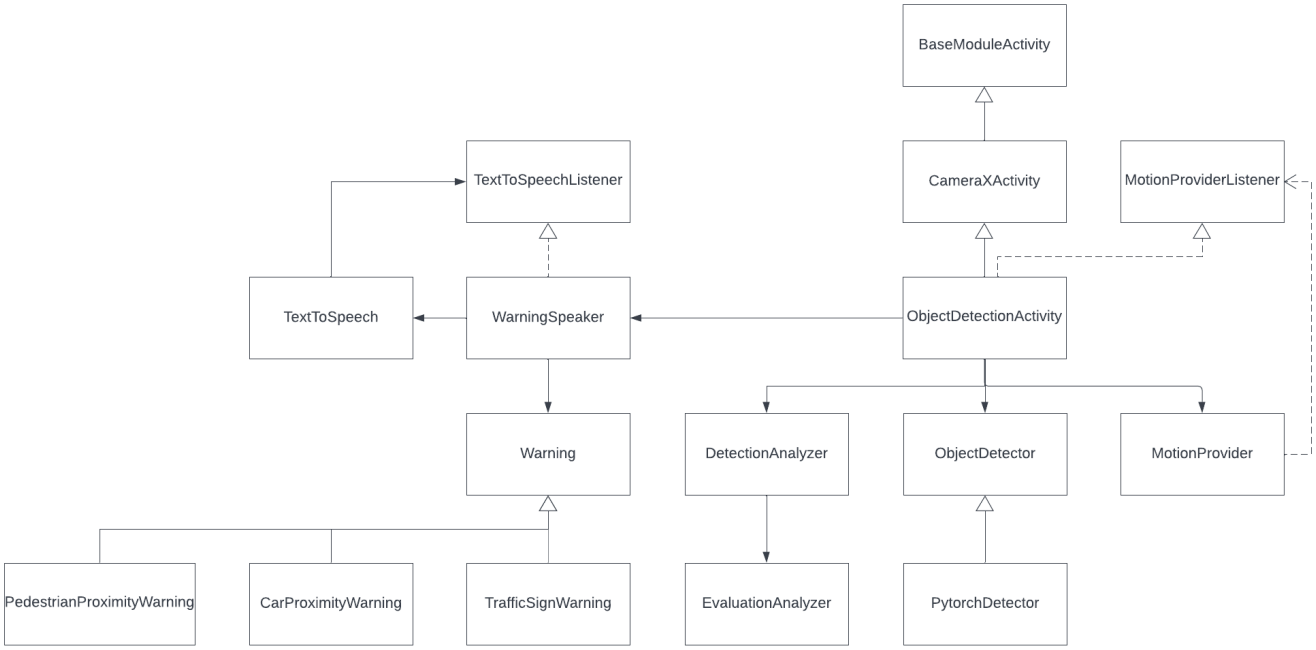


Figure 4.3: Class diagram.

4.3 User Interface

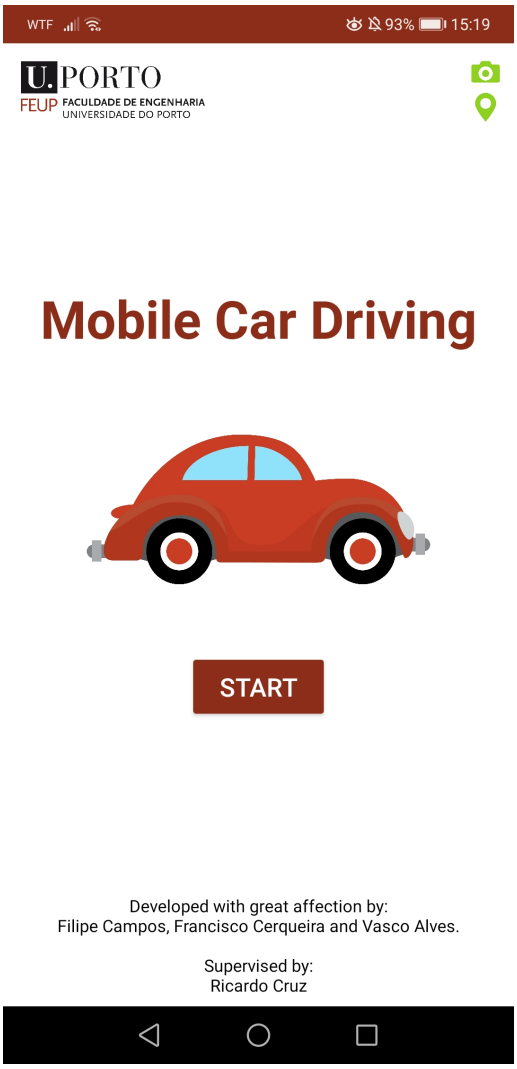
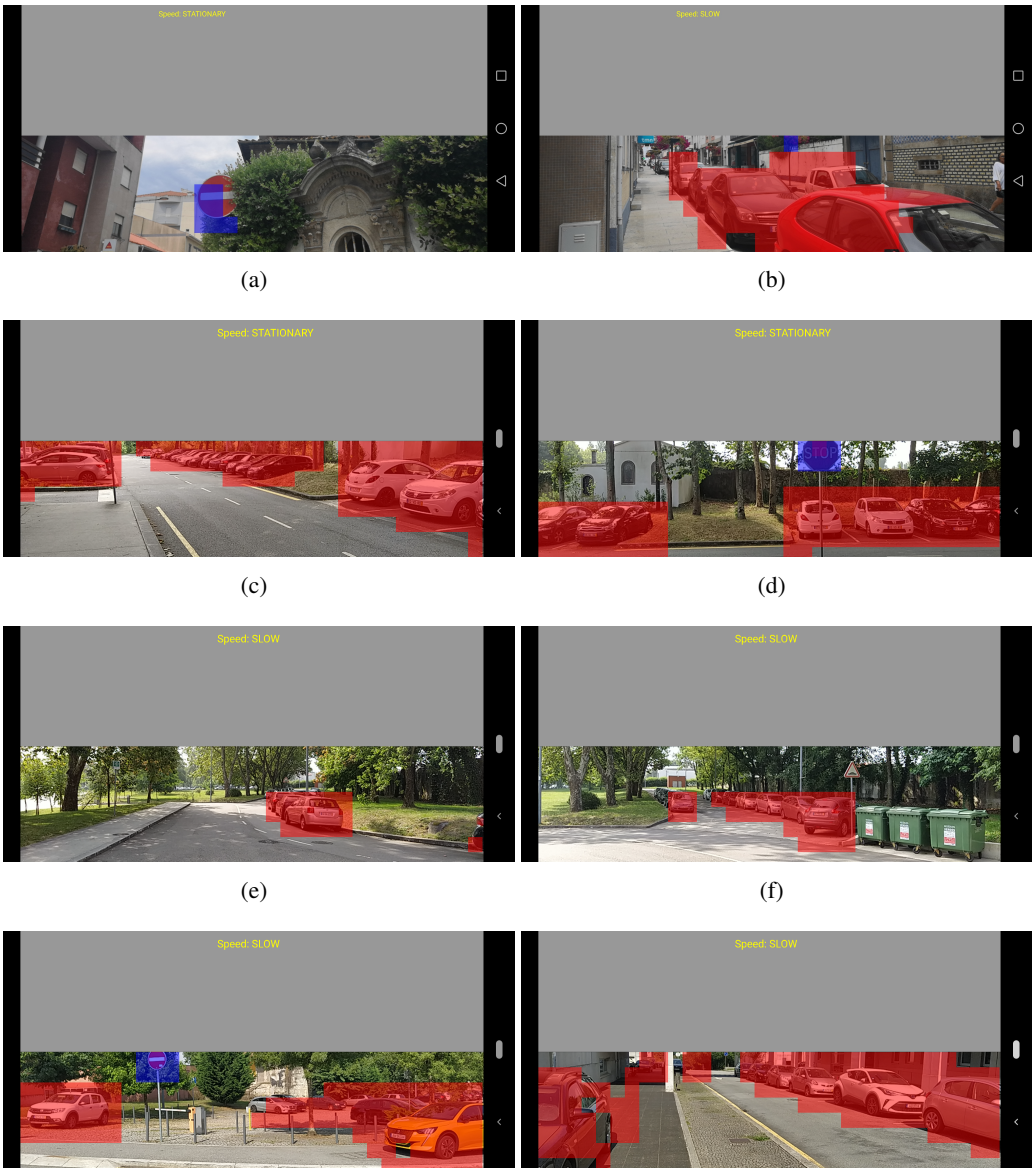


Figure 4.4: Menu screenshot.



(a) (b)

(c) (d)

(e) (f)

(g) (h)

Figure 4.5: Object detection screenshots.

4.4 Synthesizer

For the synthesizer we decided to use the American English language and reduced the speech rate to 0.7, as we believe it provides a clearer understanding of what is being said. The message used for each type of warning is shown in Table 4.4.

Warning Type	**Message**
Car	"Car nearby"
Pedestrian	"Pedestrian nearby"

Warning Type	**Message**
Stop	“Stop”
Give Way	“Give way”
Prohibited	“Prohibited way”
Prohibited Overtaking	“Overtaking is prohibited”
End of overtaking prohibition	“Overtaking is allowed”

Table 4.4: Voice messages.

4.5 Real Use Case

To use the application to its fullest extent the user should have a holder to hold the cellphone firmly, which does not block the camera and provides a good view of the road as exemplified in Figure 4.6.

Figure 4.6: Real-use scenario

Chapter 5

Conclusions

Overall the project was successful as we managed to create an Android application capable of warning its users about nearby vehicles, pedestrians and traffic signs. Unfortunately the end result was constrained by the current state of the Torch Mobile (Paszke et al., 2019) interpreter, which limited our ability to develop better and more interesting object detection models.

There are some improvements that can be developed in the future:

- Create a proper object detection model, possibly based on FCOS (Tian et al., 2019), that can be successfully ported to Torch Mobile (Paszke et al., 2019).
- Limit the scope of supported devices and utilize hardware acceleration through NNAPI[1] or Vulkan[2]. This could allow the usage of more complex prediction models, leading to a overall better accuracy without increasing the inference time.
- The user interface does not provide useful information for the end user, therefore the app can be modified to instead run in the background or be integrated with existing driver assistance applications such as Google Maps.

This project was as an excellent learning experience that allowed us to acquaint ourselves with some state computer vision technology and develop crucial skills in that same area.

[1] https://developer.android.com/ndk/guides/neuralnetworks
[2] https://source.android.com/devices/graphics/implement-vulkan

References

Alexander Buslaev, Vladimir I. Iglovikov, Eugene Khvedchenya, Alex Parinov, Mikhail Druzhinin, and Alexandr A. Kalinin. Albumentations: Fast and flexible image augmentations. *Information*, 11(2), 2020. ISSN 2078-2489. doi: 10.3390/info11020125. URL https://www.mdpi.com/2078-2489/11/2/125.

Holger Caesar, Varun Bankiti, Alex H. Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. nuscenes: A multimodal dataset for autonomous driving. In *CVPR*, 2020.

Ricardo Cruz. Objdetect package. https://github.com/rpmcruz/objdetect, 2022.

Jean-Emmanuel Deschaud. Kitti-carla: a kitti-like dataset generated by carla simulator, 2021. URL https://arxiv.org/abs/2109.00892.

Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation, 2013. URL https://arxiv.org/abs/1311.2524.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015. URL https://arxiv.org/abs/1512.03385.

Sebastian Houben, Johannes Stallkamp, Jan Salmen, Marc Schlipsing, and Christian Igel. Detection of traffic signs in real-world images: The German Traffic Sign Detection Benchmark. In *International Joint Conference on Neural Networks*, number 1288, 2013.

John Houston, Guido Zuidhof, Luca Bergamini, Yawei Ye, Long Chen, Ashesh Jain, Sammy Omari, Vladimir Iglovikov, and Peter Ondruska. One thousand and one hours: Self-driving motion prediction dataset, 2020. URL https://arxiv.org/abs/2006.14480.

Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3, 2019. URL https://arxiv.org/abs/1905.02244.

Xinyu Huang, Peng Wang, Xinjing Cheng, Dingfu Zhou, Qichuan Geng, and Ruigang Yang. The ApolloScape open dataset for autonomous driving and its application. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(10):2702–2719, oct 2020. doi: 10.1109/tpami.2019.2926463. URL https://doi.org/10.1109%2Ftpami.2019.2926463.

Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot MultiBox detector. In *Computer Vision – ECCV 2016*, pages 21–37. Springer International Publishing, 2016. doi: 10.1007/978-3-319-46448-0_2. URL https://doi.org/10.1007%2F978-3-319-46448-0_2.

Sébastien Marcel and Yann Rodriguez. Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM International Conference on Multimedia*, MM '10, page 1485–1488, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781605589336. doi: 10.1145/1873951.1874254. URL https://doi.org/10.1145/1873951.1874254.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection, 2015. URL https://arxiv.org/abs/1506.02640.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks, 2015. URL https://arxiv.org/abs/1506.01497.

Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in perception for autonomous driving: Waymo open dataset. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

Zhi Tian, Chunhua Shen, Hao Chen, and Tong He. FCOS: Fully convolutional one-stage object detection. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9627–9636, 2019.

Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. Detectron2. https://github.com/facebookresearch/detectron2, 2019.

Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning, 2018. URL https://arxiv.org/abs/1805.04687.

Appendix A

Dataset Label Comparison

Table A.1: Dataset label comparison.

Dataset	KITTI(Geiger et al., 2013)	BDD100k(Yu et al., 2018)	Waymo(Sun et al., 2020)	nuScenes(Caesar et al., 2020)	Lyft(Houston et al., 2020)	KITTI-CARLA(Deschaud, 2021)	ApolloScape(Huang et al., 2020)
ambulance							
animal							
bus							
car							
construction vehicle							
cyclist							
debris							
generic vehicle label							
misc							
motorcycle							
pedestrian							
person_moving							
person_sitting							
person_standing							
police officer							
police vehicle							
rider							
road							
sidewalk							
stroller							
traffic barrier							
traffic_cone							
traffic light							
traffic sign							
trailer							
train							
tram							
truck							
tunnel							
van							
wheelchair							
wall							