SecureWatch using Tensorflow and Google Colab(YOLO & EasyOCR)

Vroom Vroom Team
Members:
YAP EN THONG
MUHAMMAD HARIZ BIN ZAABAH
GREGORY HONG

1.0 Introduction

In the present day, many vehicles traverse our roads, leaving us uncertain about the status of these vehicles—whether they are stolen or associated with individuals with criminal backgrounds. Imagine if a sophisticated system enabled law enforcement to swiftly identify stolen vehicles and track persons of interest, including those evading the law. Such a system promises to streamline investigative procedures, bolstering public safety and fortifying security measures.

In this project, TensorFlow will be employed to develop advanced image recognition models that can accurately identify car colour and car type, as well as extract information about the car's colour and type to store in a database. The primary objective is to enhance road safety and security. When a vehicle passes through a designated monitoring point, the system will capture real-time footage and the TensorFlow model will analyse this data to extract the required information. The recognized details, including the car colour and car type, will then be saved to a secure database, providing valuable insights into the types of vehicles passing through specific areas. This data can be used for various road safety and traffic management purposes, aiding law enforcement and emergency response teams in identifying vehicles involved in incidents, tracking traffic patterns, and ensuring a safer and more efficient road network. So we trained TensorFlow Lite models for the car type recognition model and car colour detection model, which are connected to a database. While for the car plate recognition model, functioned using footage on Google Colab. The model is trained using YOLO and OCR.

2.0 Progress Record

2/11/2023	Grouping
3/11/2023	First draft proposal
4/11/2023	SecureWatch idea approved
5/11/2023	Discussion about the environment used to create our models. In our project, we chose YOLO (You Only Look Once) for quickly spotting objects, especially for detecting license plates in real time. To read the text on these plates, we utilized EasyOCR, a tool for Optical Character Recognition (OCR). TensorFlow models were trained to recognize car types and colours. For efficient data management, we turned to SQLite, setting up tables to neatly store information like detected car colours and types. To showcase our models, we could easily use recorded footage or even a regular PC camera for live demonstrations. This approach allowed us to seamlessly integrate different technologies and bring our project to life.
6/11/2023	The car plate recognition model using EasyOCR and YOLO can be found on GitHub by inserting a custom dataset on Google Colab. The model efficiently processes videos, displaying bounding boxes along with the corresponding car plate numbers. It's a straightforward solution for recognizing car plates with ease.
7/11/2023	We had a team meeting to figure out how to make a TensorFlow model. We thought about using Roboflow's model website but ended up using their dataset for training images. We got busy preparing and labelling our own set of training images, using tools like LabelImg. It was a key step in our mission to create a solid TensorFlow model.
11/11/2023	En Thong encountered an issue with a Python script while splitting training images into train, validation and test sets, specifically when using the JPEG format. To address this, she edited the Python script to include support for the JPEG format. Additionally, En Thong successfully trained a car colour detection model using TensorFlow Lite on Google Colab. The model demonstrated effective performance, running seamlessly on En Thong's webcam and processed footage.
12/11/2023	En Thong implemented a deployment function to facilitate the model's execution. Upon running the model, the generated video is saved to a

	designated path named "output_video.mp4." Notably, the existing video at this path is replaced each time the model is re-run.
16/11/2023	During a team meeting, we discussed creating another model. Unfortunately, Hariz first attempt at training a Car Type Detection model didn't work well due to low accuracy. We realized the need to find ways to improve the model's performance. This involved reevaluating our training approach and exploring different strategies to enhance accuracy.
19/11/2023	Hariz trained the TensorFlow Lite car type detection model using Google Colab with a higher mAP score.
26/11/2023	We had a brief online meeting to discuss the project's progress. We focused on debugging the code for releasing the video, incorporating a function ('c') to capture images and save them to the database. Pressing 'q' stopped the video playback and closed it, with the video automatically saved in the download/output_video.mp4 directory.
27/11/2023	We successfully integrated the car type detection and car colour detection models with a database, allowing us to view the database using SQLite.
7/12/2023	Writing report
8/12/2023	Project Submission

3.0 Environment Set Up

We studied a few videos, we found out that the best and alternative way to create a TensorFlow model is by using Google Colab. By working through Google Colab, we can create and download a TFLite model that we can run on our PC, an Android phone or an edge device like the Raspberry Pi.

This is how we train our car type recognition model and car colour detection model respectively using the same method:

To begin with, we searched for the code for TensorFlow Lite object detection on GitHub and we ran it using Google Colab. Before we start training, we need to gather and label images that will be used for training the object detection model. We decided to gather the training images on Roboflow and Google Chrome. For better running of the model, we try to search training images that are similar to what the camera will see in the actual application. For each model, we used at least 200 images to train an initial model. At this point, we had a folder full of pictures of the object, they are mostly in jpeg and jpg format. Then we need to download LabelImg which is an open-source labelling tool to label the images with the correct classes and location of the object in the images and save the file. LabelImg is used to draw bounding boxes and indicate their classes. The training algorithm will use this data to teach the model what your object looks like and how to locate it in featured images. When we finished gathering and labelling images, we had a folder full of images and a corresponding .xml data annotation file for each image.

LabelImg download page for Windows:
https://github.com/HumanSignal/labelImg/releases

Next, we installed TensorFlow Object Detection Dependencies in the Google Colab instance. The version of TensorFlow we used here is TF v2.8.0. Then, we started to upload the image dataset and prepare the training data. We zipped all our training images and XML files into a single folder called "images.zip" and uploaded it to the Google Colab. After uploading the images.zip, we tried to split the images into train, validation and test sets. Here's what each set is used for:

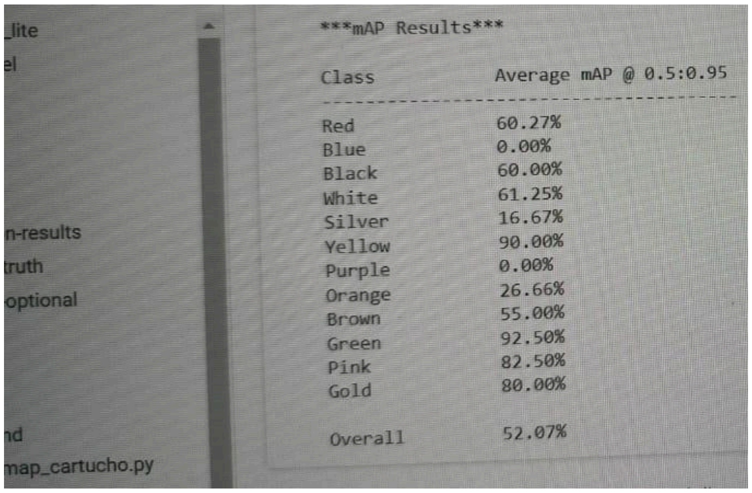
- **Train**: These are the actual images used to train the model. In each step of training, a batch of images from the "train" set is passed into the neural network. The network predicts the classes and locations of objects in the images. The training algorithm calculates the loss (i.e. how "wrong" the predictions were) and adjusts the network weights through backpropagation.
- **Validation**: Images from the "validation" set can be used by the training algorithm to check the progress of training and adjust hyperparameters (like learning rate). Unlike "train" images, these images are only used periodically during training (i.e. once every certain number of training steps).
- **Test**: These images are never seen by the neural network during training. They are intended to be used by a human to perform final testing of the model to check how accurate the model is.

There is an existing Python script we found online to randomly move 80% of the images to the "train" folder, 10% to the "validation" folder and 10% to the "test" folder. We downloaded the script and executed it but found out it couldn’t work with a JPEG format file. Many images we prepared weren't spilted into the train, validation and test folder. So we edited the script to make it compatible with our usage. Luckily it worked.

Finally, we created a labelmap for the detector and converted the images into a data file format called TFRecords, which are used by TensorFlow for training. We'll use Python scripts to automatically convert the data into TFRecord format. Before running them, we need to define a labelmap for our classes. It is a txt file.

By running the code, TFRecord files are created for the training and validation datasets, as well as a labelmap.pbtxt file which contains the labelmap in a different format. Then, we set up the training configuration.We specified which pre-trained TensorFlow model we want to use from the TensorFlow 2 Object Detection Model Zoo. For our project, the two models use 'ssd-mobilenet-v2-fpnlite-320'. We also set the number of steps and batch size to control training steps. For the number of steps, we set it as 20000 and observed the TensorFlow Board's graph to determine whether to stop training earlier before it reached 20000 steps because we were scared it overtrain the model and the result would not be accurate. Next, we'll rewrite the config file to use the training parameters we just specified. Before we started training, we loaded up a TensorBoard session to monitor the training progress. Then, we ran the code for training and the training for the two models took around 2 hours+ respectively. Lastly, we converted the trained model to TensorFlow Lite-compatible format.

Also, there is a testing code in the Google Colab. To know how well it actually performs at detecting objects in images, this is where the images we set aside in the **test** folder come in. The model never saw any test images during training, so its performance on these images should be representative of how it will perform on new images from the field. We ran a code that defines a function to run inference on test images. It loads the images, loads the model and labelmap, runs the model on each image and displays the result. We had a visual sense of how our model performs on test images, but how can we quantitatively measure its accuracy? One popular method for measuring object detection model accuracy is "mean average precision" (mAP). Basically, the higher the mAP score, the better your model is at detecting objects in images. We used the mAP calculator tool at https://github.com/Cartucho/mAP to determine our model's mAP score. Ideally, mAP score should be above 50% (0.50). For example, our car colour model successfully got 52.07% of the average mAP at the end of training. Sadly, we got a 0% mAP score for the colours blue and purple in the testing stage. We guess probably it is because we only interfere with 10% of images in the test folder(no purple and blue colour labelled images in it).



```
***mAP Results***

Class          Average mAP @ 0.5:0.95
----------------------------------------
Red            60.27%
Blue           0.00%
Black          60.00%
White          61.25%
Silver         16.67%
Yellow         90.00%
Purple         0.00%
Orange         26.66%
Brown          55.00%
Green          92.50%
Pink           82.50%
Gold           80.00%

Overall        52.07%
```

After converting our model, we downloaded our TFLite model by copying the labelmap files into the model folder, compressing it into a zip folder and then downloading it. The zip folder contains the detect.tflite model and labelmap.txt labelmap files that are needed to run the model in your application. Overall, we changed some codes in the existing Google Colab file to make it compatible with our needs. Normally, we will copy the Python code to Visual Studio Code to edit and save it as a new file then replace the older version in the Google Colab to run it and for some situations, we just directly edit the code in the Google Colab cells, trial and error. We will explain how to deploy our model in '4.0 Installation'.

For car plate detection model, we dived in an existing GitHub code related to license plate detection and recognition using YOLO V8 and Easy OCR which we found it suitable to run with custom dataset like using different footage. After installation of requirements.txt like

```
# Base ----------------------------------------
ultralytics==8.0.3
hydra-core>=1.2.0
matplotlib>=3.2.2
numpy>=1.18.5
opencv-python>=4.1.1
Pillow>=7.1.2
PyYAML>=5.3.1
requests>=2.23.0
scipy>=1.4.1
torch>=1.7.0
torchvision>=0.8.1
tqdm>=4.64.0
Easyocr

# Logging -------------------------------------
tensorboard>=2.4.1
```

```
# Plotting ------------------------------------
pandas>=1.1.4
seaborn>=0.11.0

# Extras --------------------------------------
ipython  # interactive notebook
psutil  # system utilization
thop>=0.1.1  # FLOPs computation

# HUB -----------------------------------------
GitPython>=3.1.24
```

, we ran the cell on Google Colab to run our model. The source of the video we provided is replaceable in mp4 format. Then, after the cell finishes running, the video generated by this model will be saved to runs/detect/train. By opening the file, we can get the same video we provided but with labelled class and bounding box to the object detected while the video is running.

4.0 Installation

Our three models run separately. To run the car type detection model and car colour detection model which are TensorFlow Lite base, we set up a virtual environment using Anaconda to run our model's zip file which contained the tflite file. First, we need to download and install Anaconda. Secondly, we have to set up a virtual environment and directory. We go to the Anaconda Prompt and click to open a command terminal. We'll create a folder called `tflite1` directly in the C: drive. So we can retrieve it easily. The folder names depend on ourself.

```
mkdir C:\tflite1
cd C:\tflite1
```

Next, create a Python 3.9 virtual environment by issuing:

```
conda create --name tflite1-env python=3.9
```

Enter "y" when it asks if you want to proceed. Activate the environment and install the required packages by issuing the commands below. We'll install TensorFlow, OpenCV, and a downgraded version of Protobuf. TensorFlow is a pretty big download (about 450MB), so it will take a while.

```
conda activate tflite1-env
pip install tensorflow opencv-python protobuf==3.20.*
```

Then, download the detection scripts from this repository by issuing:

```
curl
https://raw.githubusercontent.com/EdjeElectronics/TensorFlow-Lite-Object-Detect
ion-on-Android-and-Raspberry-Pi/master/TFLite_detection_image.py --output
TFLite_detection_image.py
curl
https://raw.githubusercontent.com/EdjeElectronics/TensorFlow-Lite-Object-Detect
ion-on-Android-and-Raspberry-Pi/master/TFLite_detection_video.py --output
TFLite_detection_video.py
curl
https://raw.githubusercontent.com/EdjeElectronics/TensorFlow-Lite-Object-Detect
ion-on-Android-and-Raspberry-Pi/master/TFLite_detection_webcam.py --output
TFLite_detection_webcam.py
curl
https://raw.githubusercontent.com/EdjeElectronics/TensorFlow-Lite-Object-Detect
ion-on-Android-and-Raspberry-Pi/master/TFLite_detection_stream.py --output
TFLite_detection_stream.py
```

Next, take the custom TFLite model that was trained and downloaded from the Colab notebook and move it into the C:\tflite1 directory.

```
move C:\Users\User\Downloads\custom_model_lite\custom_model_lite.zip
```

Once it moved, unzip it using:

```
tar -xf custom_model_lite.zip
```

At this point, you should have a folder at C:\tflite1\custom_model_lite which contains at least a `detect.tflite` and `labelmap.txt` file.

Alright! Now that everything is set up, running the TFLite model is easy. Just call one of the detection scripts and point it at your model folder with the `--modeldir` option. For example, to run your `custom_model_lite` model on a webcam, issue:

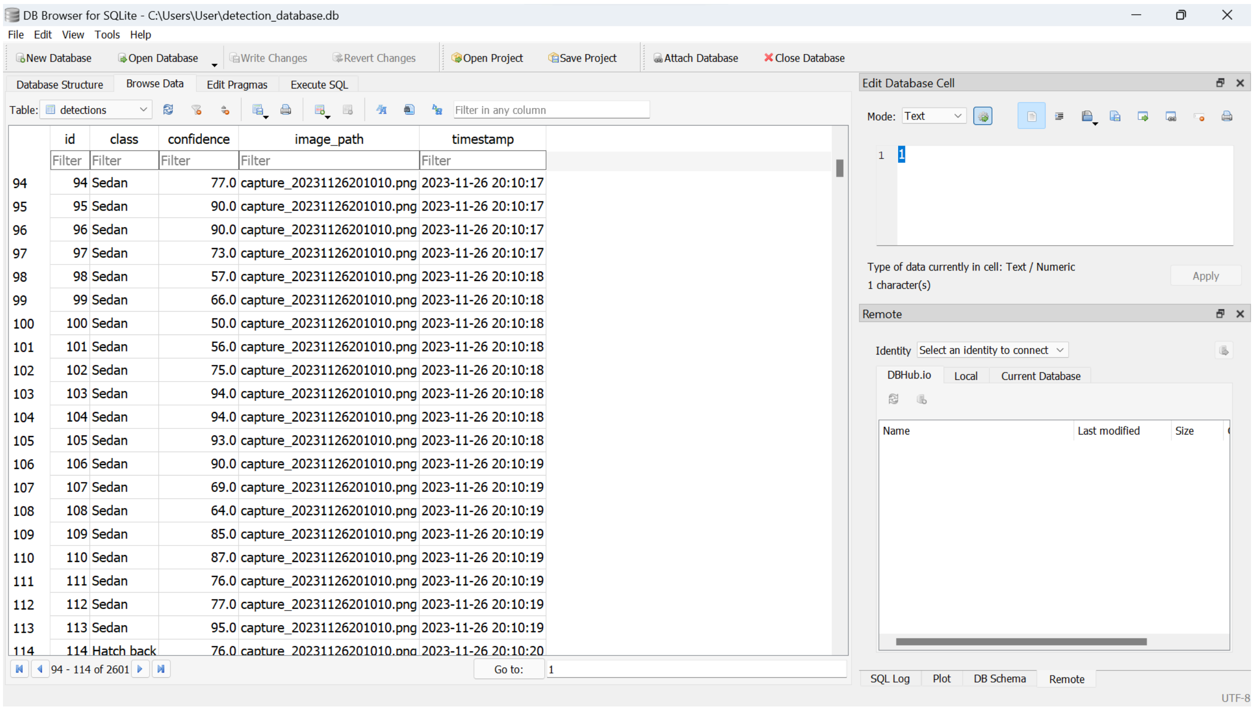
```
python TFLite_detection_webcam.py --modeldir=custom_model_lite
```

A window will appear showing detection results drawn on the live webcam feed.

Also to run the custom_model_lite model using the footage provided, before that copy your video path and replace the following code, issue:

```
python TFLite_detection_video.py --modeldir=custom_model_lite
--video="C:\Users\User\Downloads\demo.mp4"
```

For our case, we need to connect it to a database. So En Thong rewrote the code for TFLite_detection_video.py many times and added some functions. We add in a function like the video released will automatically saved to a defined output video path. So every time this code runs, the output video path will be replaced by the latest video run. Next, we found out that to connect the model to a database, we should add in function to capture the images while the video is released. So En Thong edited the code again. It will print “Frame captured and information saved to the database.” while we press the key ‘c’ when the video is running. Parameters like capture_time, class_name, confidence_level and image_path will be returned to the database.



	id	class	confidence	image_path	timestamp
	Filter	Filter	Filter	Filter	Filter
94	94	Sedan	77.0	capture_20231126201010.png	2023-11-26 20:10:17
95	95	Sedan	90.0	capture_20231126201010.png	2023-11-26 20:10:17
96	96	Sedan	90.0	capture_20231126201010.png	2023-11-26 20:10:17
97	97	Sedan	73.0	capture_20231126201010.png	2023-11-26 20:10:17
98	98	Sedan	57.0	capture_20231126201010.png	2023-11-26 20:10:18
99	99	Sedan	66.0	capture_20231126201010.png	2023-11-26 20:10:18
100	100	Sedan	50.0	capture_20231126201010.png	2023-11-26 20:10:18
101	101	Sedan	56.0	capture_20231126201010.png	2023-11-26 20:10:18
102	102	Sedan	75.0	capture_20231126201010.png	2023-11-26 20:10:18
103	103	Sedan	94.0	capture_20231126201010.png	2023-11-26 20:10:18
104	104	Sedan	94.0	capture_20231126201010.png	2023-11-26 20:10:18
105	105	Sedan	93.0	capture_20231126201010.png	2023-11-26 20:10:18
106	106	Sedan	90.0	capture_20231126201010.png	2023-11-26 20:10:19
107	107	Sedan	69.0	capture_20231126201010.png	2023-11-26 20:10:19
108	108	Sedan	64.0	capture_20231126201010.png	2023-11-26 20:10:19
109	109	Sedan	85.0	capture_20231126201010.png	2023-11-26 20:10:19
110	110	Sedan	87.0	capture_20231126201010.png	2023-11-26 20:10:19
111	111	Sedan	76.0	capture_20231126201010.png	2023-11-26 20:10:19
112	112	Sedan	77.0	capture_20231126201010.png	2023-11-26 20:10:19
113	113	Sedan	95.0	capture_20231126201010.png	2023-11-26 20:10:19
114	114	Hatch back	76.0	capture_20231126201010.png	2023-11-26 20:10:20

Also to close the video released earlier, we added in function press ‘q’ to quit. The video will closed and saved to the output_video.mp4 path as defined.

We applied this code for both environments we created for car type detection and car colour detection model so in the end the table of this database is common for both car colour detection and car type detection model. This database function is only applied for the footage method which is called `TFLite_detection_video.py` , for the real live webcam method(`TFLite_detection_webcam.py`), we didn’t build the functions for it.

5.0 Execution/Running

To run the car colour and car type models:

1. Open a terminal using the Anaconda Prompt.

2. Activate the previously built environment. For example:
   ```
   conda activate tflite4-env
   ```

3. Install the required dependencies using pip:
   ```
   pip install tensorflow opencv-python protobuf==3.20.*
   ```

4. Move the model zip file to the current directory and unzip it. For example:

```
move C:\tflite1\custom_model_lite.zip
tar -xf custom_model_lite.zip
```

5. Run the following code using the existing TFLite_detection_video.py script in the C: environment:

```
python TFLite_detection_video.py
--modeldir=custom_model_lite
--video="C:\Users\User\Downloads\demo.mp4"
```

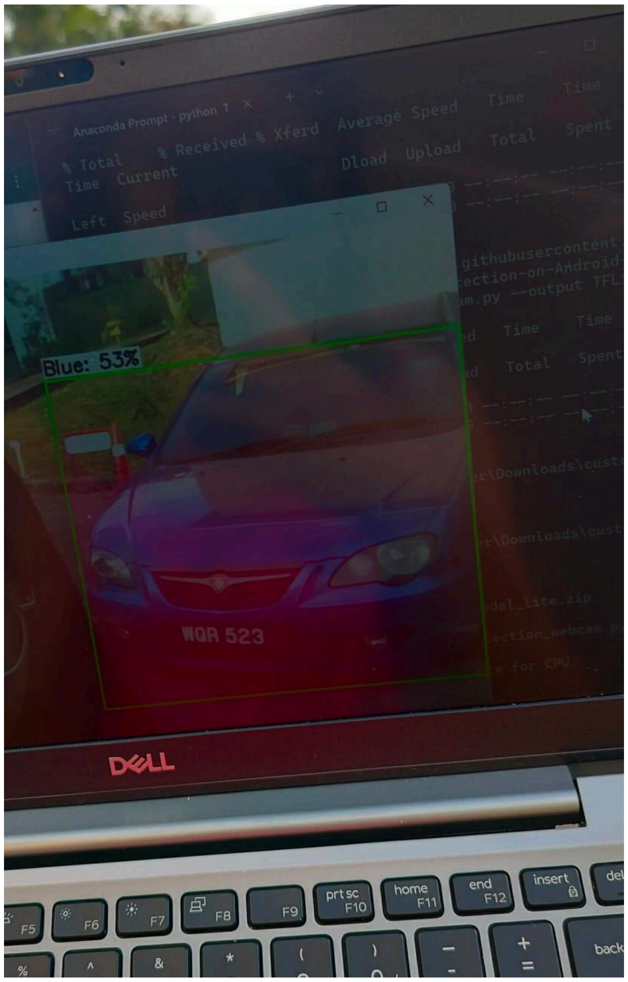


Figure 5.1 Running car colour detection model using a webcam

To run the car plate recognition model, follow these steps:

1. Click on the provided Google Colab link: Car Plate Recognition Colab.

https://colab.research.google.com/drive/1pqU8iA2SvzAxq7E0R1BCEwS7IxA4-LlT?usp=sharing

2. Once the Colab notebook opens, you can execute each cell one by one. To execute a cell, click on the play button located on the left side of the cell or press Shift + Enter.
3. Follow any instructions or comments provided in the Colab notebook. Ensure that any necessary dependencies are installed and the required files are uploaded or accessible.
4. Monitor the output of each cell for any errors or warnings. If needed, address any issues that arise during execution.
5. After running all the cells, you should have the car plate recognition model up and running.

6.0 Conclusion

Embarking on this project with minimal knowledge of TensorFlow models and EasyOCR YOLO-based applications presented a challenging yet rewarding experience for our team. The iterative process of trial and error became a continuous learning opportunity, with each obstacle providing valuable insights into model creation. Group collaboration emerged as a cornerstone, allowing us to leverage collective knowledge and troubleshoot issues effectively.

Despite the difficulties we faced due to the demanding internship program, our team persisted. We completed the project and refined it afterwards, demonstrating our resilience and problem-solving abilities. Debugging sessions were crucial in increasing our comprehension of the code and fine-tuning our abilities.

Furthermore, our involvement with the Airost internship program provided an additional layer of education, introducing us to the intricacies of being a member of a vibrant community. Looking back, the active and practical aspect of the project made it an enjoyable and thrilling experience. We are proud that we were able to overcome challenges and obstacles, and this project is proof of our commitment and teamwork. To sum up, even we can't make this project what we want it to be, the process still enhanced our technical skills and also strengthened our ability to be resilient and adaptable.

7.0 References

1. Licence Plate Recognition with YOLO V8 and Easy OCR using Custom Dataset

   https://youtu.be/k4OIaCOH9NY?si=sHqQQJF1a4Om3SkW

2. Train TensorFlow Lite Model for Custom Object (License Plate) Detection with Custom Dataset

https://youtu.be/kwrLK71fP1Q?si=BrabMpMznr4vgJtt

3. How to Capture and Label Training Data to Improve Object Detection Model Accuracy

   https://youtu.be/v0ssiOY6cfg?si=2BExCuxiiwU15z9M

4. How to Train TensorFlow Lite Object Detection Models Using Google Colab

   https://youtu.be/XZ7FYAMCc4M?si=RN3uySDEptM4ysQh

5. Learn how to deploy TensorFlow Lite model

   https://github.com/EdjeElectronics/TensorFlow-Lite-Object-Detection-on-Android-and-Raspberry-Pi/blob/master/deploy_guides/Windows_TFLite_Guide.md