GRAB AI 2019 :

Computer Vision Challenge

Car Classifier

Submission by Nguyen Duc Phuong

https://www.aiforsea.com/computer-vision

CAR CLASSIFIER

C2

TABLE OF CONTENTS

C2

CAR CLASSIFIER

THE GOAL:

AUTOMATICALLY RECOGNIZE CAR (USING MOBILE PHONE)

Using Tensorflow 2.0 and Android

The project's goal is to develop a model to **recognize car model and make** with **high accuracy** and be able to run that model on **mobile device** to recognize the car using the **mobile phone's camera**.

Implementation Overview

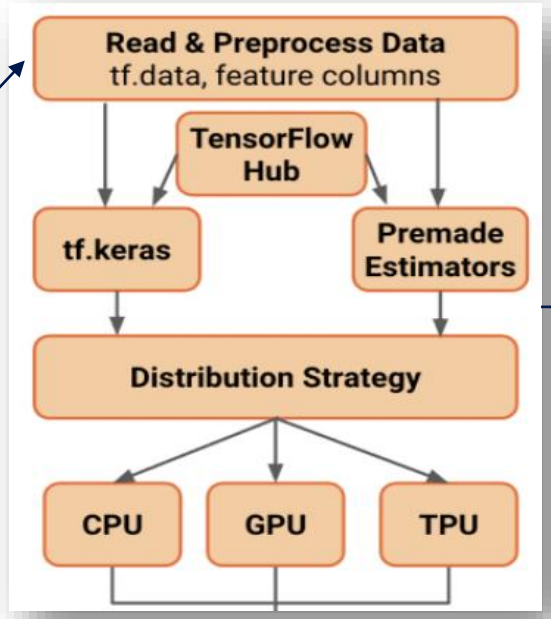
CAR CLASSIFIER

TRAINING

- Model implemented in Tensorflow 2.0

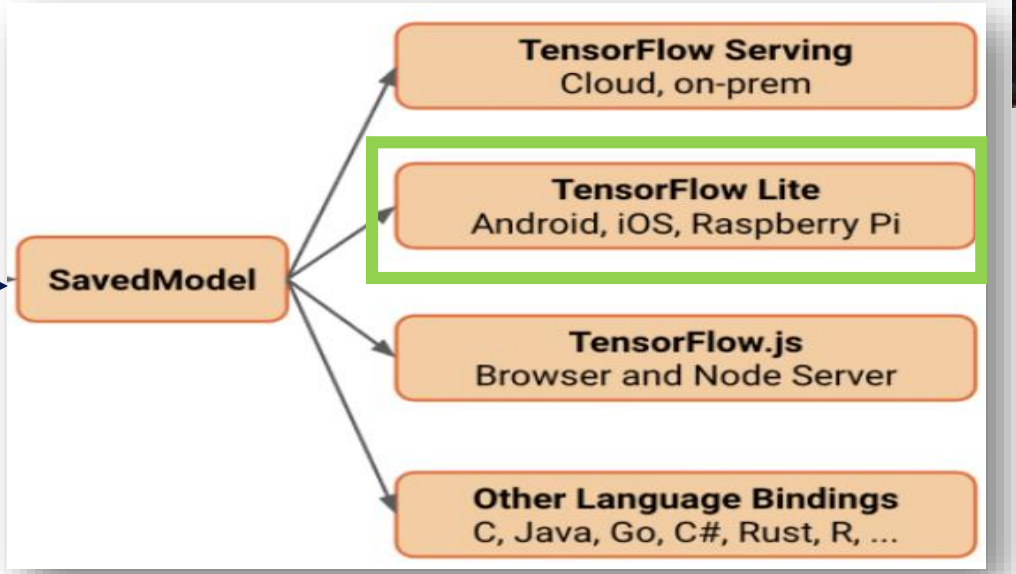
Car Dataset

https://ai.stanford.edu/~jkrause/cars/car_dataset.html



DEPLOYMENT

- Trained model are converted to Tensorflow Lite format and use on mobile phone (Android).



C2

MODELS (1/3)

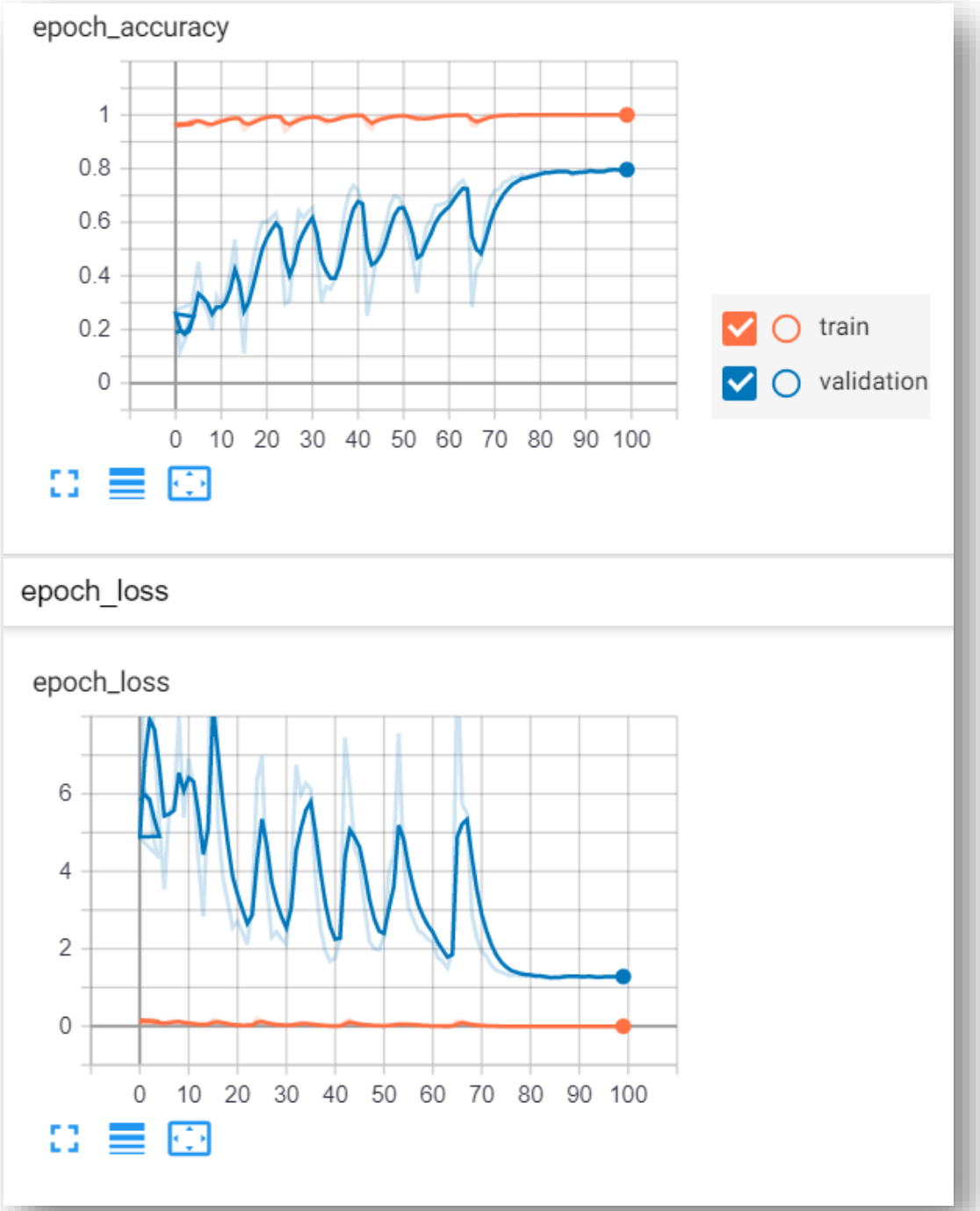
USING **MOBILENET V2** AS BASE MODEL

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
mobilenetv2_1.00_224 (Model) (None, 7, 7, 1280)        2257984
_________________________________________________________________
batch_normalization_v2_1 (Ba (None, 7, 7, 1280)        5120
_________________________________________________________________
global_average_pooling2d_1 ( (None, 1280)              0
_________________________________________________________________
dense_1 (Dense)              (None, 197)               252357
=================================================================
Total params: 2,515,461
Trainable params: 2,117,509
Non-trainable params: 397,952
_________________________________________________________________
```

Using pre-trained MobileNet V2 as base model

I am able to get after **100 epoch**:

- **100% accuracy** on training dataset
- And around **82% accuracy** on validation dataset.



MODELS (1/3)

USING **MOBILENET V2** AS BASE MODEL

Using pre-trained MobileNet V2 as base model

I am able to get after **100 epoch**:

- **100% accuracy** on training dataset
- And around **82% accuracy** on validation dataset.

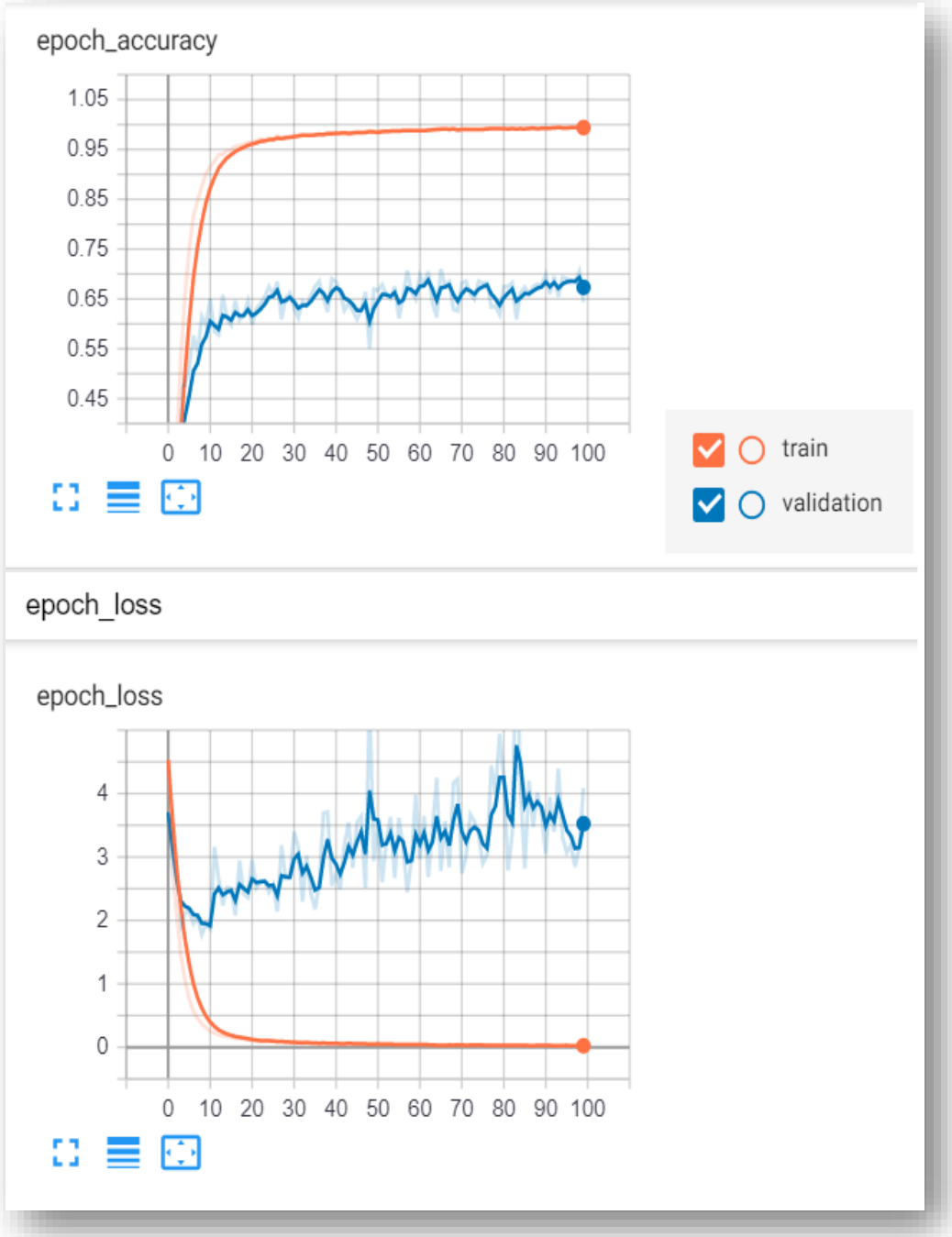
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
xception (Model)             (None, None, None, 2048)  20861480
_________________________________________________________________
conv2d_4 (Conv2D)            (None, None, None, 32)    589856
_________________________________________________________________
leaky_re_lu (LeakyReLU)      (None, None, None, 32)    0
_________________________________________________________________
batch_normalization_v2 (Batc (None, None, None, 32)    128
_________________________________________________________________
global_average_pooling2d (Gl (None, 32)                0
_________________________________________________________________
leaky_re_lu_1 (LeakyReLU)    (None, 32)                0
_________________________________________________________________
dense (Dense)                (None, 256)               8448
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 197)               50629
=================================================================
Total params: 21,510,541
Trainable params: 10,127,341
Non-trainable params: 11,383,200
_________________________________________________________________
```

MODELS (2/3)

USING **XCeption** AS BASE MODEL

Using pre-trained **XCeption** as base model I am able to get after **100 epoch**:

- **100% accuracy** on training dataset
- And around **70% accuracy** on validation dataset.



MODELS (2/3)

USING XCeption AS BASE MODEL

Using pre-trained **XCeption** as base model I am able to get after **100 epoch**:

- **100% accuracy** on training dataset
- And around **70% accuracy** on validation dataset.

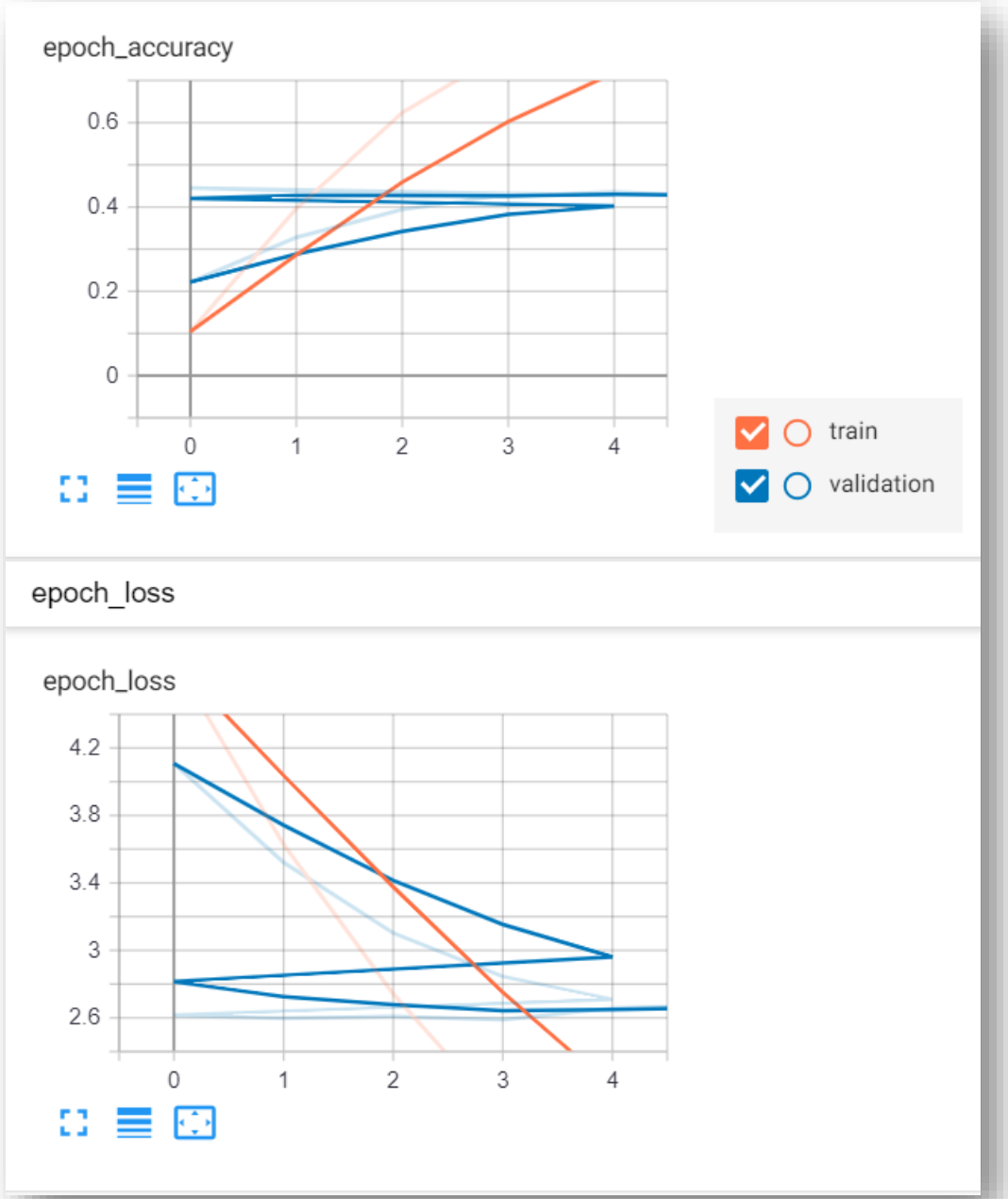
MODELS (3/3)

USING **DenseNet** AS BASE MODEL

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
densenet201 (Model)          (None, 7, 7, 1920)        18321984
_________________________________________________________________
gaussian_noise_1 (GaussianNo (None, 7, 7, 1920)        0
_________________________________________________________________
batch_normalization_v2_2 (Ba (None, 7, 7, 1920)        7680
_________________________________________________________________
global_average_pooling2d_3 ( (None, 1920)              0
_________________________________________________________________
dense_3 (Dense)              (None, 197)               378437
=================================================================
Total params: 18,708,101
Trainable params: 664,901
Non-trainable params: 18,043,200
_________________________________________________________________
```

Using pre-trained **DenseNet** as base model I am able to get after **100 epoch**:

- **100% accuracy** on training dataset
- And around **70% accuracy** on validation dataset.



MODELS (3/3)

USING **DenseNet** AS BASE MODEL

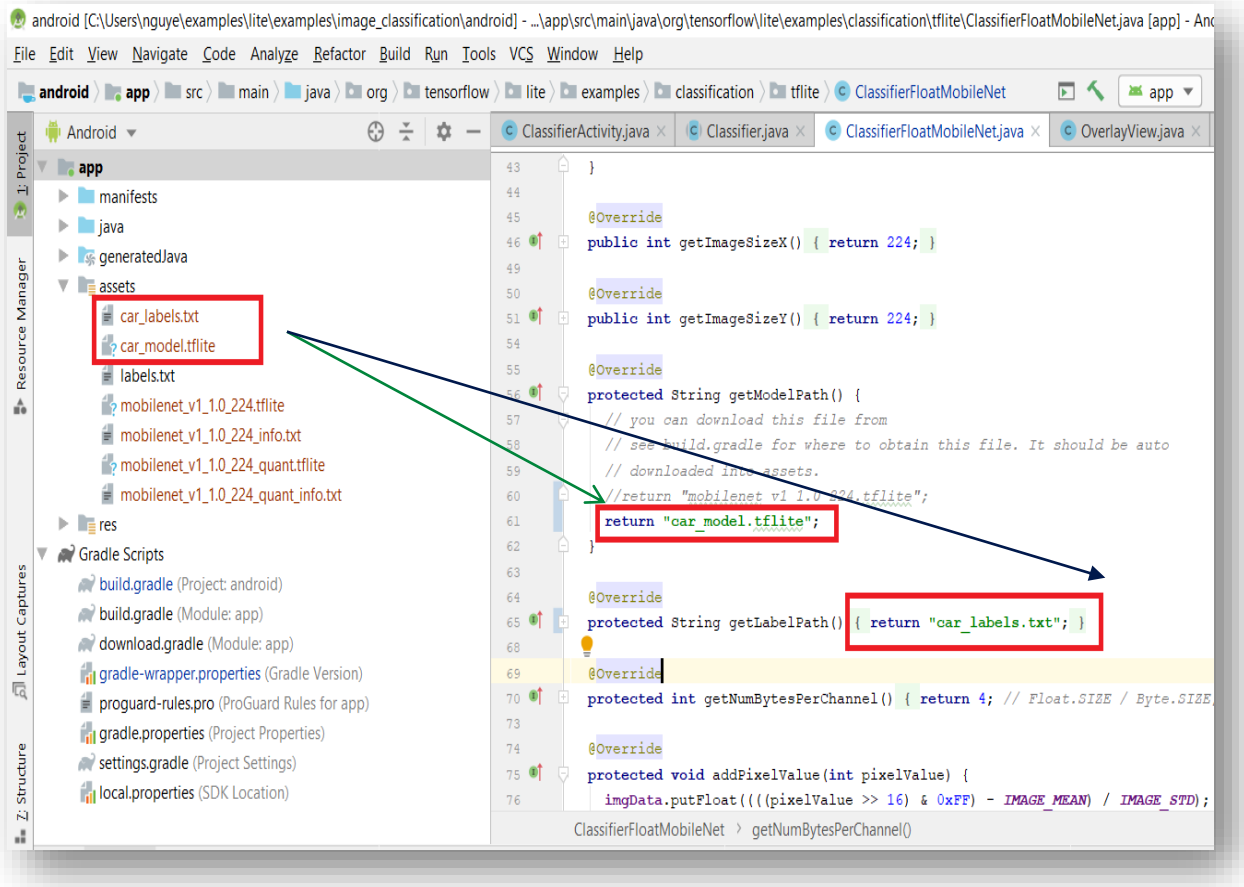
Using pre-trained **DenseNet** as base model I am able to get after **100 epoch**:

- **100% accuracy** on training dataset
- And around **70% accuracy** on validation dataset.

CAR CLASSIFIER

MOBILE APP

USING Tensorflow Lite example mobile app



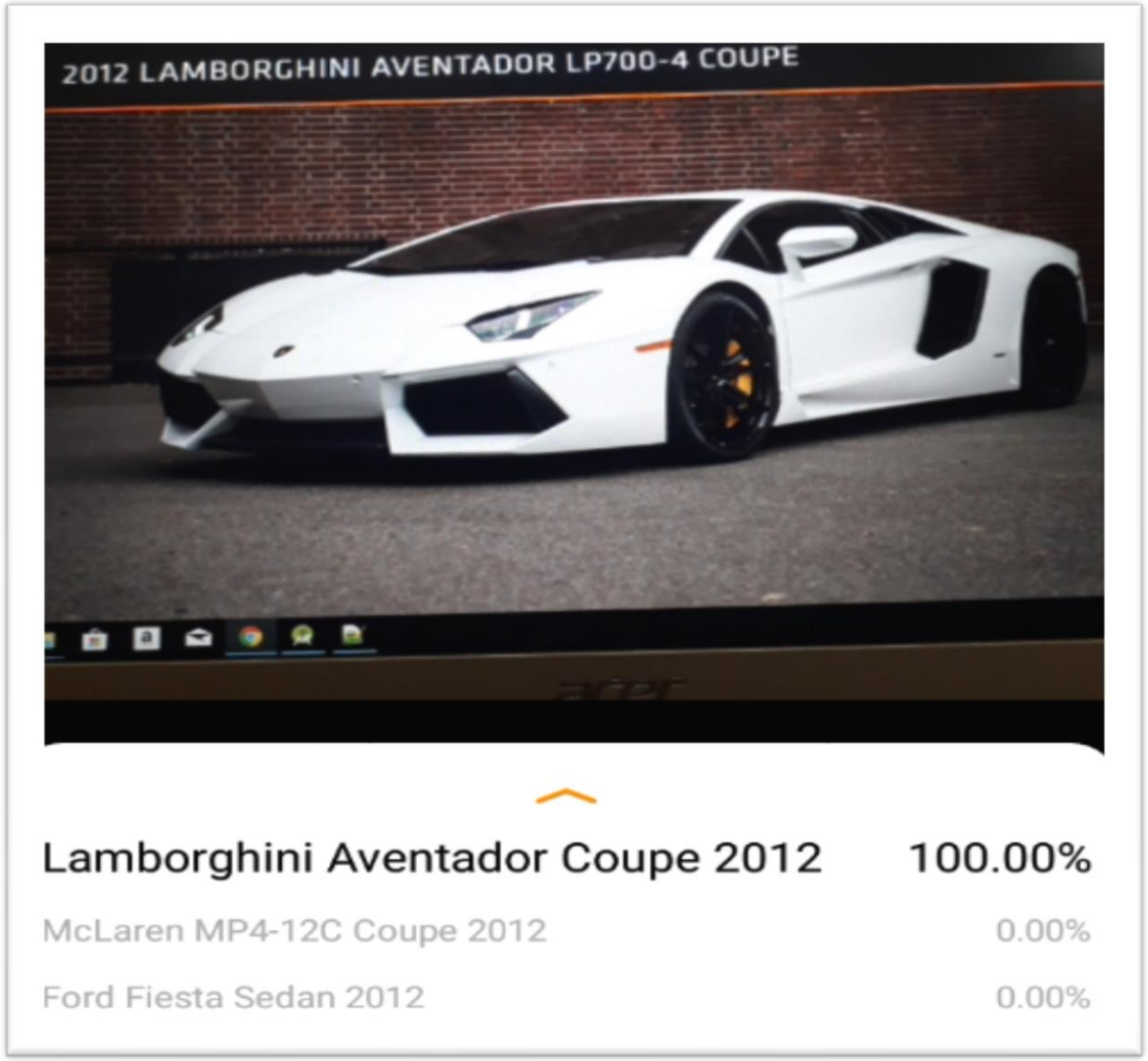
Step 1: Download the sample mobile application on Github

https://github.com/tensorflow/examples/tree/master/lite/examples/image_classification/android

Step 2: Convert the saved model to Tensorflow Lite format, using the notebook "Convert Tensorflow Lite.ipynb" on Github.

Step 3: Include the trained car model and labels into the example mobile app and run!

CAR CLASSIFIER



RESULT

USING MOBILE APPLICATION

The trained model perform quite well on the mobile application.

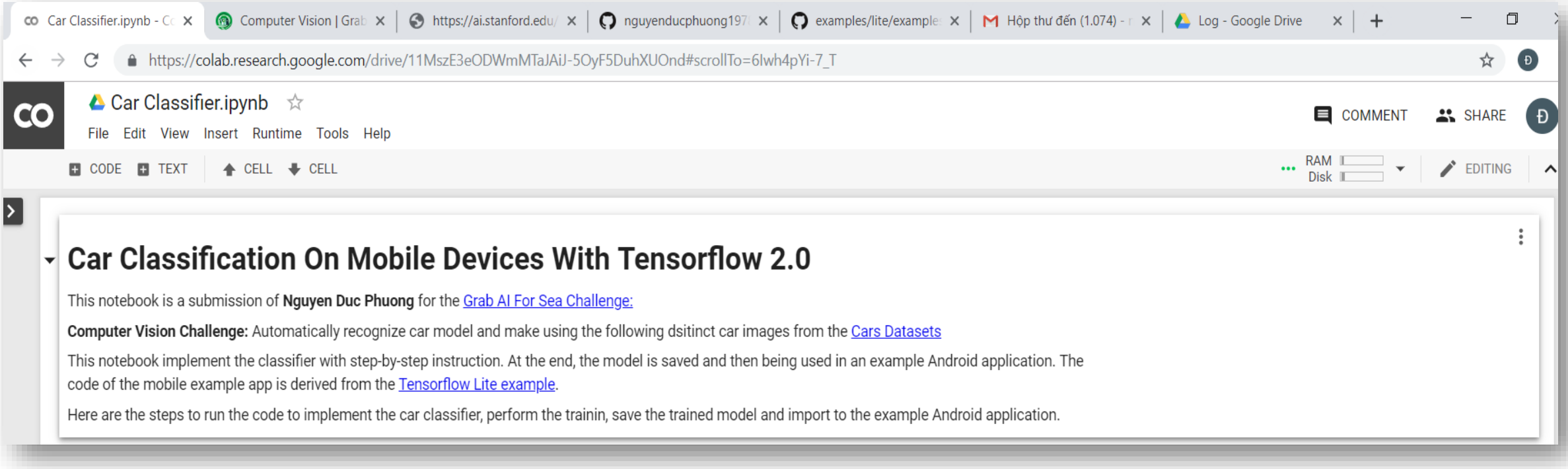
(Tested on Anroid 8.0.0 mobile phone; Camera 13MP)

C2

How to run the code

Following section is step by step instruction how to run the code (download dataset and train the model; export to Tensorflow Lite for mobile device and run the code on Android mobile phone)

CAR CLASSIFIER



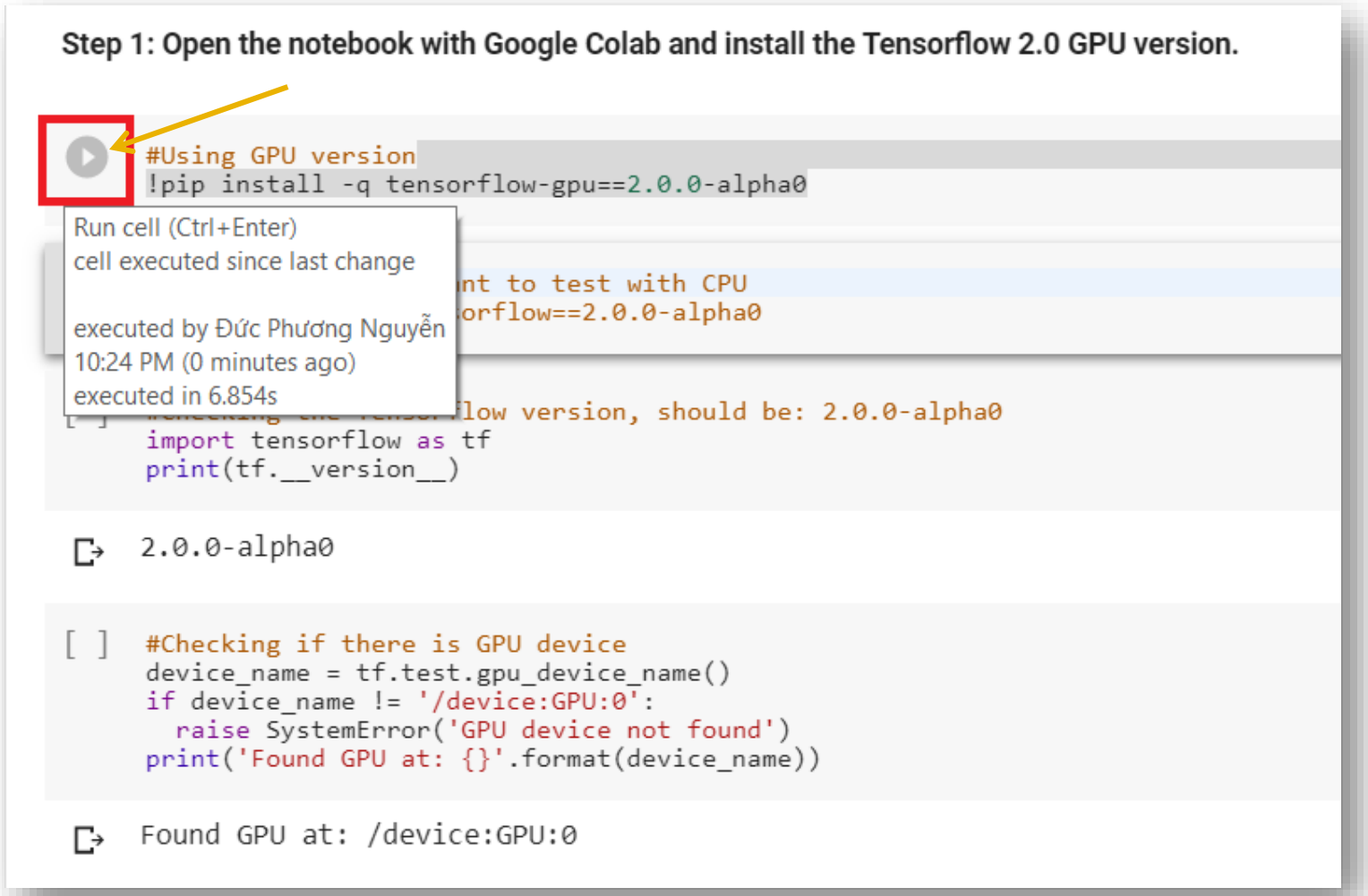
Starting point: Open python notebook with Google Colab

You only need a Google email account.

Get the notebook Car Classifier.ipynb and open it in Colab.

C2

CAR CLASSIFIER



Step 1: Follow and run all the steps listed in the notebook

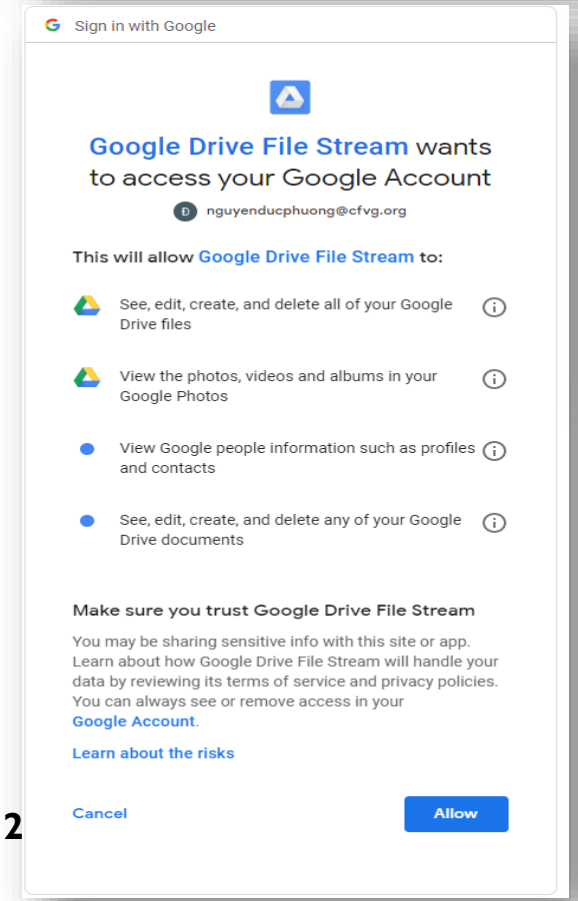
Click in the run icon of the cell to execute the python code.
Follow the instruction in the notebook

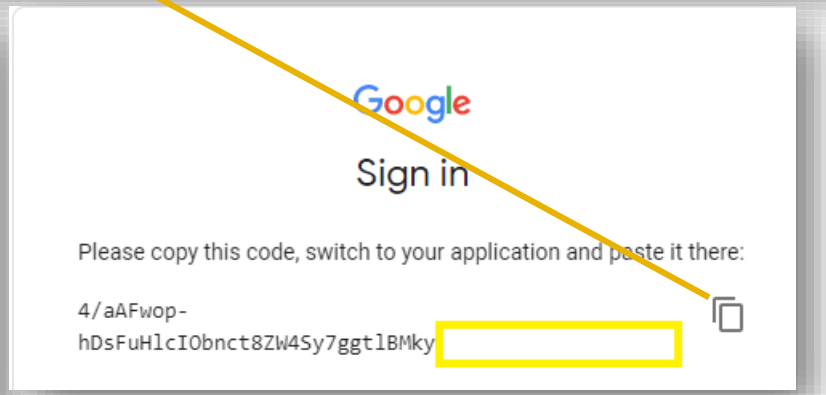
(Car Classifier.ipynb)

C2

Click on the link and login to your Google account to map the Google Drive

```
[ ]  #Map Google Drive to save the checkpoint file
     from google.colab import drive
     drive.mount('/content/drive')

     Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3Aietf%3Aw

     Enter your authorization code:
     ··········
     Mounted at /content/drive
```





Notes: Google Drive mapping

If you have Google Drive, you can save the trained model into the Drive

(Car Classifier.ipynb)

CAR CLASSIFIER

Step 1.5: Prepare the training and testing DataSet

The classifier can be implemented with base model of either Mobinet V2 or XCeption or NASNet. Update the constant USE_MODEL to switch the base model.

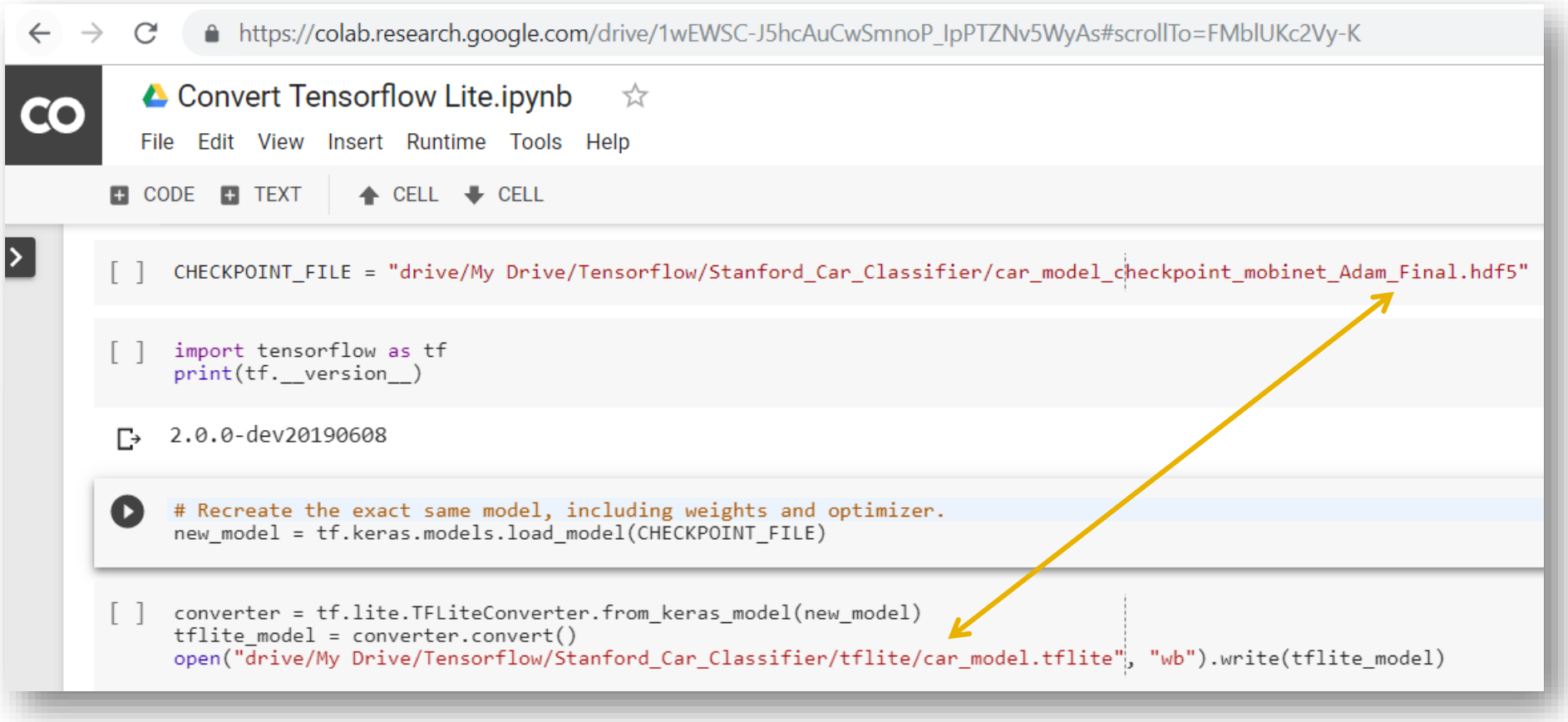
```
USE_MODEL = 'xception' # mobinet or xception or nasnet

#For MobileNet V2
if USE_MODEL == 'mobinet':
  IMG_WIDTH = 224
  IMG_HEIGHT = 224
elif USE_MODEL == 'xception':
  #For XCeption model
  IMG_WIDTH = 299
  IMG_HEIGHT = 299
elif USE_MODEL == 'nasnet':
  #For NASNet model
  IMG_WIDTH = 224 # NASNet Large = 331
  IMG_HEIGHT = 224 # NASNet Large = 331
```

Notes: Update the base model at Step 1.5 in the notebook

Change the base model by update USE_MODEL to either 'mobinet' or 'xception' or 'nasnet' or 'densenet'

(Car Classifier.ipynb)

C2



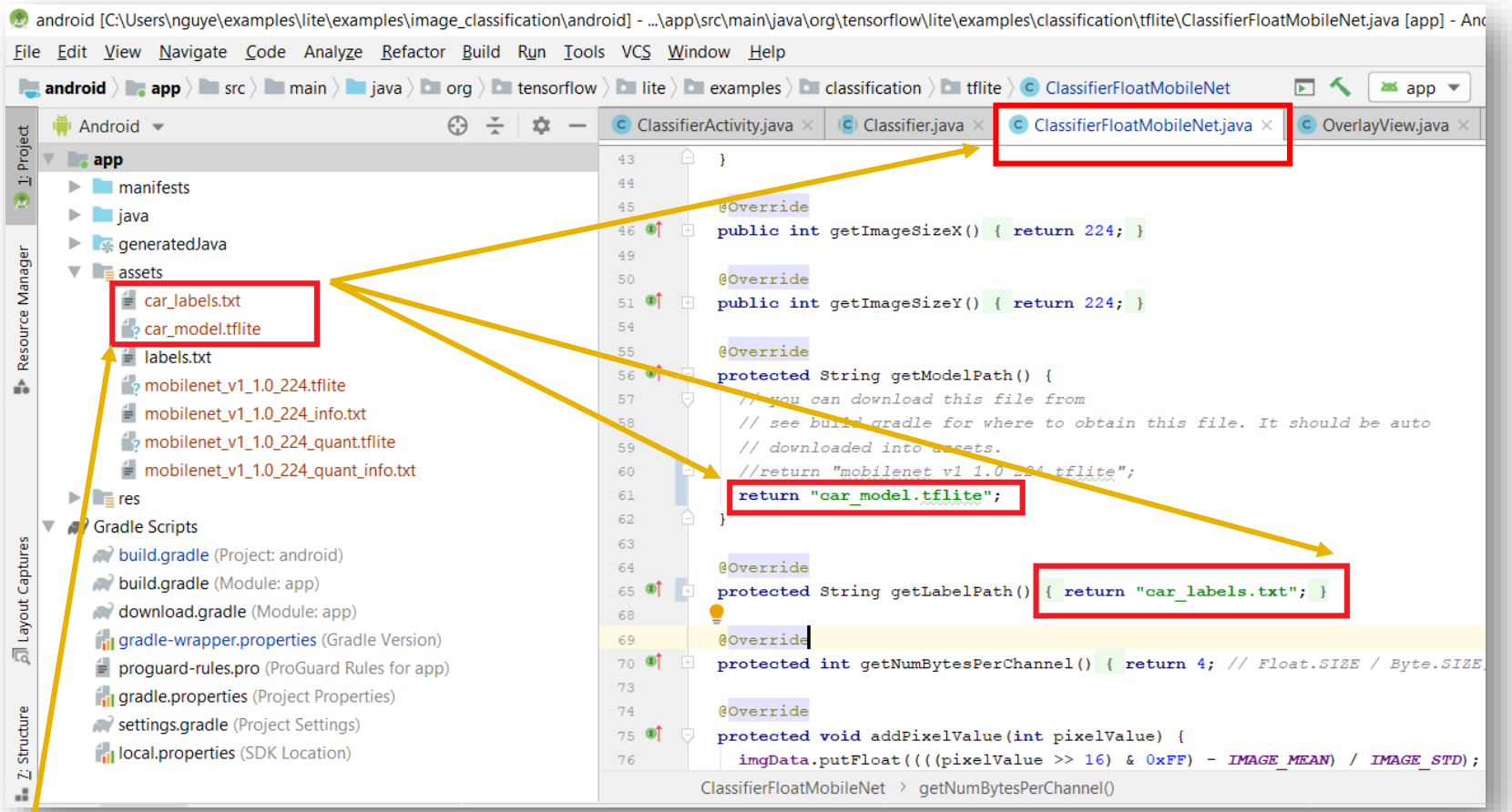
Specify the saved model (hdf5) file and the destination folder for the Tensorflow Lite (tflite) file.

Notes: Convert saved model to Tensorflow Lite for mobile

Follow the steps in the notebook to convert to Tensorflow Lite for mobile

(Convert Tensorflow Lite.ipynb)

CAR CLASSIFIER



Download the sample Android mobile application and copy the Tensorflow Lite (tflite) file and the car labels textfile into the assets folder. Update the values in the class "ClassiferFloatMobileNet". Build the mobile application and Run the mobile application on your mobile phone. Enjoy!

Notes: Copy the car model's Tensorflow Lite for mobile. Build the mobile app!

Download the example mobile application:

https://github.com/tensorflow/examples/tree/master/lite/examples/image_classification/android

C2

References

1. Source code: https://github.com/nguyenducphuong1978/CarClassifier

2. Tensorflow Lite, example mobile app: https://github.com/tensorflow/examples/tree/master/lite/examples/image_classification/android

3. Car Dataset: https://ai.stanford.edu/~jkrause/cars/car_dataset.html

4. Grab AI Challenges – Computer Vision: https://www.aiforsea.com/computer-vision

5. Tensorflow 2.0 Alpha: https://www.tensorflow.org/beta/

6. Car Classifier – Mobile App – Demo Video: https://youtu.be/MWQmZx-PZGo

PHUONG NGUYEN

+84 (0) 898-311-335

nguyenducphuong@cfvg.org

CAR CLASSIFIER

THANK YOU

https://github.com/nguyenducphuong1978/CarClassifier

C2