License Plate Detection

Team: Visionaries

The Problem?

- We started by discussing important milestones that were created with the use of computer vision applications.

- One that stuck out were cameras that were able to detect and identify license plates.

- There are many variations of stationary license plate detection cameras. Either in high angle areas or anchored hardware.

- The problem that we found is that there exist little to no models that could detect license plates from a dash cam point of view.

The Solution

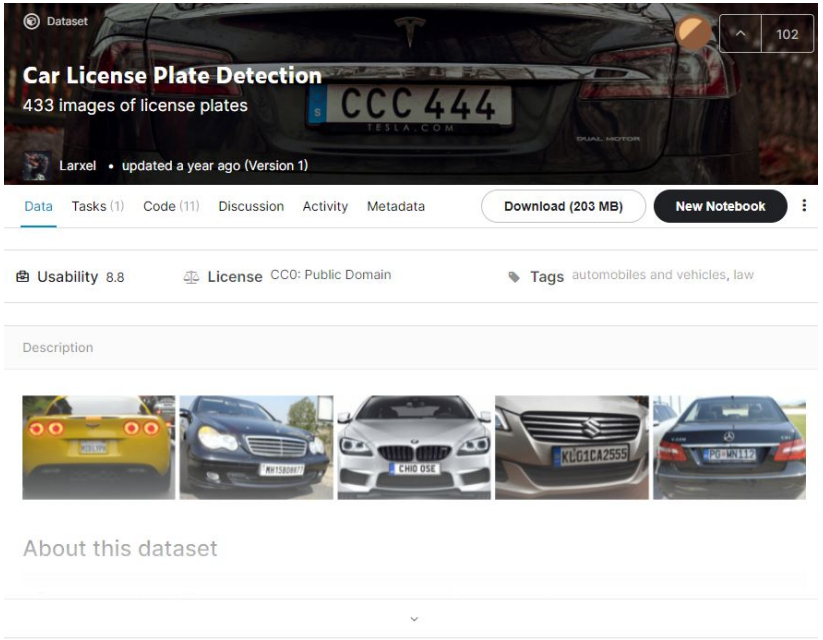
- Find data for model training consisting of a custom set of training images that follow a head-on camera angle and scenarios where the subject is in motion or strange angles representative of the regular driver experience that encompass these four scenarios:
    - Subject in motion; camera not in motion
    - Subject in motion; camera in motion
    - Subject not in motion; camera in motion
    - Subject not in motion; camera not in motion

- The goal for the training images is for our model to be able to identify a bounding box around the license plate in our angles and scenarios

- Once we have trained the model to set an adequate bounding box then we may interpret characters within the license plate

Proposed Method

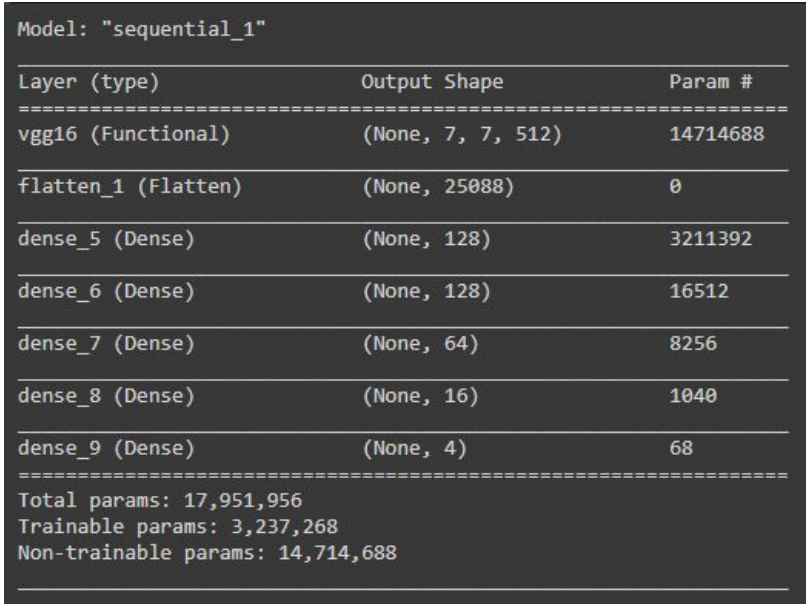
1. Collect data from our resource.
2. Model Build and Training
3. Evaluation
4. Reading the license plates



Dataset from Kaggle

Model Building and Training

- Resized all training images and bounding boxes to one size for training
- Then used a Keras model with 7 layers to train
- The VGG16 layer is an instantiation of a CNN model used in an ImageNet competition (used for object recognition)
- We also used an activation function called “swish”
- Past that we just used several densely connected layers to increase our available parameters

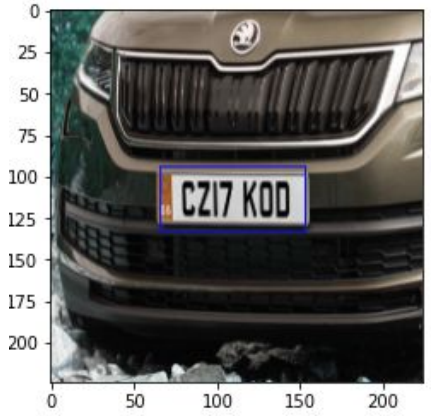


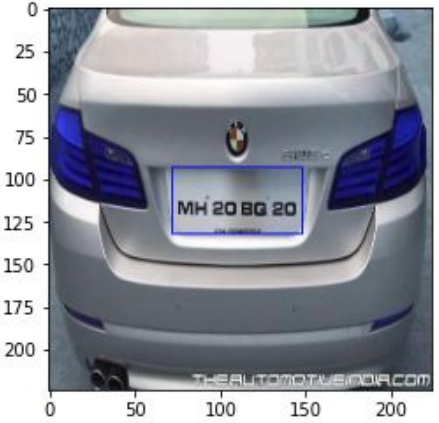
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 7, 7, 512)         14714688
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_5 (Dense)              (None, 128)               3211392
_________________________________________________________________
dense_6 (Dense)              (None, 128)               16512
_________________________________________________________________
dense_7 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_8 (Dense)              (None, 16)                1040
_________________________________________________________________
dense_9 (Dense)              (None, 4)                 68
=================================================================
Total params: 17,951,956
Trainable params: 3,237,268
Non-trainable params: 14,714,688
_________________________________________________________________
```

Results

- Predicting the location of the license plate
  - Accuracy: 86 - 94%

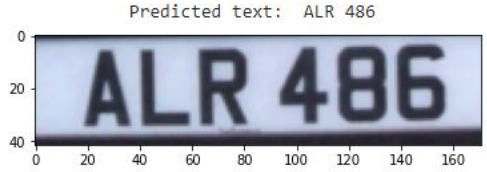
```
Epoch 44/50
11/11 [==============================] - 2s 150ms/step - loss: 0.0016 - accuracy: 0.9534 - val_loss: 0.0030 - val_accuracy: 0.8947
Epoch 45/50
11/11 [==============================] - 2s 150ms/step - loss: 0.0012 - accuracy: 0.9578 - val_loss: 0.0030 - val_accuracy: 0.9211
Epoch 46/50
11/11 [==============================] - 2s 149ms/step - loss: 0.0013 - accuracy: 0.9642 - val_loss: 0.0029 - val_accuracy: 0.9474
Epoch 47/50
11/11 [==============================] - 2s 149ms/step - loss: 0.0013 - accuracy: 0.9437 - val_loss: 0.0030 - val_accuracy: 0.9474
Epoch 48/50
11/11 [==============================] - 2s 149ms/step - loss: 7.9841e-04 - accuracy: 0.9250 - val_loss: 0.0029 - val_accuracy: 0.9474
Epoch 49/50
11/11 [==============================] - 2s 148ms/step - loss: 0.0011 - accuracy: 0.9566 - val_loss: 0.0031 - val_accuracy: 0.9211
Epoch 50/50
11/11 [==============================] - 2s 147ms/step - loss: 6.4383e-04 - accuracy: 0.9532 - val_loss: 0.0032 - val_accuracy: 0.8684
```

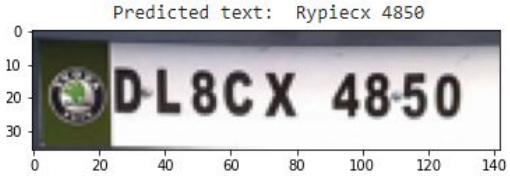


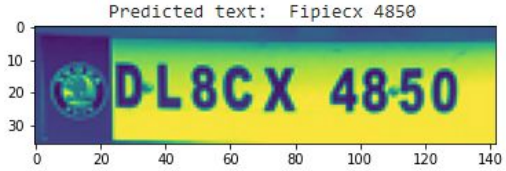


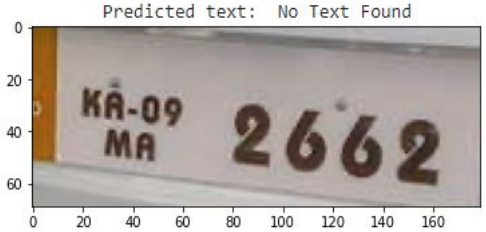
Optical character recognition:

- We used `pytesseract` to do OCR.
- Wrong OCR output for some images.









Future Work

- Find better data
- Improve optical character recognition (OCR) / Find better tool to perform OCR
- Improve our model to get the better accuracy
- Speed Cameras, Traffic Light Cameras, Police Work

Contributions:

- Amanda Cheng: Researched and managed team timeline.
- Cade Mack: Wrote most of the code preparing the data and training/creating the model
- Hari Chamlagai: Data collection, slides and GitHub repo
- AJ Daodu: Project ideas, Research, Finding data
- Haroon Kaid: Worked with project deliverables and data collection
- Aryani Patel: Research, helped with implementing OCR, presentation slides.

Members

Amanda Cheng

Cade Mack

Hari Chamlagai

AJ Daodu

Haroon Kaid

Aryani Patel

THANK YOU!