AUTOMATIC NUMBER PLATE RECOGNITION

This deep learning approach helps us to address the drawbacks of the previous model and also improve the accuracy of the detection and recognition of number plates.

Drawbacks of the image processing approach:

1. The plate clipping done by the model is not robust enough as it has a vulnerability to skewed images.

2. Coming to the second part of the problem which deals with segmenting that clipped number plate into individual characters, it has an inability to ignore or remove the unwanted characters that may have crept in due to the adaptive thresholding and also the number plate may contain some unwanted characters.

3. The model built using the image processing approach is sensitive to the resolution and contrast of the image(as it uses adaptive thresholding).

We can better address these issues using a deep learning approach. In this approach, the problem is broken down into 2 sub problems namely:

1. Number plate detection: This is the problem of recognising the number plates of the vehicles from an image.
2. Number plate recognition: This is the problem of recognising the characters from the extracted number plate in the first step.

Number plate detection:

We used YOLOv3 with pre-trained weights and transfer learning to train the model to detect number plates in an image. YOLO v3 uses a variant of Darknet, which originally has 53 layer network trained on Imagenet. For the task of detection, 53 more layers are stacked onto it, giving us a 106 layer fully convolutional underlying architecture for YOLO v3.

For this sub-problem, we set the number of classes as 1(only the number plate class). We trained on 1500 custom annotated images which includes vehicles from India, Dubai and Israel. These images capture various lighting and orientation conditions of the

vehicles and their number plates as well as multiple vehicles and their number plates in the same image. Below are a couple of detections done by our model:

Results:

Precision = 0.96, Recall = 0.94, F1-score = 0.95
TP = 1731, FP = 78, FN = 111
Average IoU = 74.01 %
Mean average precision (mAP) = 89.22 %
Average Detection Time: 0.0442 Seconds per image

Number plate recognition:

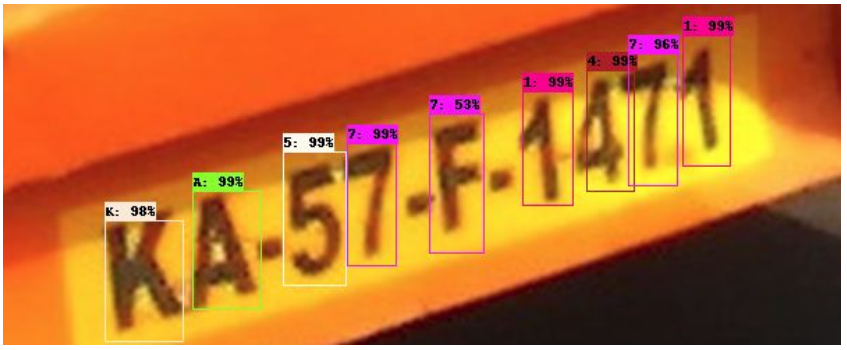
For recognition of characters in the number plates, we used two different deep learning models:

1. Faster RCNN inception v2:

We used Faster RCNN inception v2 coco with pre-trained weights and transfer learning to train the model to recognize the characters in a number plate. R-CNN is the first step for Faster R-CNN. It uses search selective to find out the regions of interests and passes them to a ConvNet. It tries to find out the areas that might be an object by combining similar pixels and textures into several rectangular boxes.

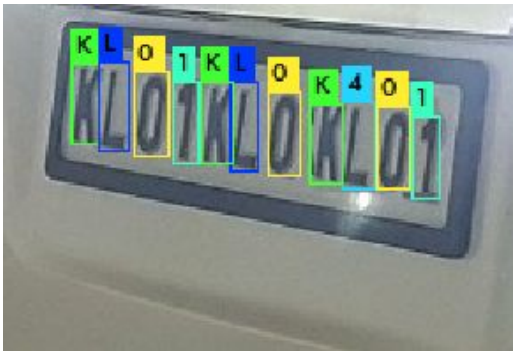
In general, bigger the model, more prone it is to overfitting. Also, increasing the number of parameters increases the computational resources needed. A solution for this, is to use to sparsely connected network architectures which replace fully connected network architectures, especially inside convolutional layers. This is the inception approach that allows to maintain the “computational budget”, while increasing the depth and width of the network.

For this sub-problem, we set the number of classes as 36. We trained on 1000 custom annotated images which includes number plate images that capture various lighting and orientation conditions.Here are two example predictions:



2. YOLO v3:

By using transfer learning and pre-trained YOLOv3 model, we trained the model for recognising the characters in the number plate.For this sub-problem, we set the number of classes as 36. We trained on 1000 custom annotated images which includes number plate images that capture various lighting and orientation conditions. Below are two example predictions:



Results:

Precision = 0.98, Recall = 0.96, F1-score = 0.97
TP = 408, FP = 10, FN = 17
Average IoU = 79.20 %
Mean average precision (mAP) = 86.65 %
Average Detection Time: 0.0444 Seconds per image

Trade Off between accuracy and time:

By using the above approach we get two pipelined versions for the Automatic Number Plate Recognition problem.The first version uses YOLOv3 for both Number plate detection and Number plate recognition and the second version uses YOLOv3 for Number plate detection and FasterRCNN inception v2 for Number plate recognition.

YOLOv3 is faster than FasterRCNN inception v2 but FasterRCNN inception v2 performs better for the detection of smaller objects in images.Now for selecting the version we need to consider the tradeoff between accuracy and time.

For accuracy sensitive applications, choosing the version which uses FasterRCNN inception v2 for Number plate recognition is a better option whereas for the real-time applications the goto version is the one which uses YOLOv3 for Number plate Recognition.

How to run our models:

Link to download the models: VG_AlexeyAB_darknet-master.zip

The following should be installed in order to run our models:

1. CUDA 10.0
2. Cudnn 7.4.2
3. Python 3.6
4. Tensorflow 1.13.1 and all of its dependencies

For version 1, follow ANPR/build/darknet/x64/YOLOv3+FasterRCNN.ipynb
For version 2, follow ANPR/build/darknet/x64/YOLOv3+YOLOv3.ipynb

References:

1. YOLOv3 models:
   https://github.com/AlexeyAB/darknet
   https://github.com/Vic-TheGreat/VG_AlexeyAB_darknet

2. YOLOv3:
   http://pjreddie.com/darknet
   https://pjreddie.com/media/files/papers/YOLOv3.pdf
3. Fast RCNN:
   https://github.com/EdjeElectronics/TensorFlow-Object-Detection-API-Tutorial-Train-Multiple-Objects-Windows-10
   https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md
4. Data sets:
   https://dataturks.com/projects/devika.mishra/Indian_Number_plates
   https://dataturks.com/projects/alex1/ALPR%20-%20Dubai
   https://dataturks.com/projects/cnaant73/Israeli%20license%20plates
   https://www.kaggle.com/thamizhsterio/indian-license-plates
5. Stack overflow and Github forums for any implementation issues.