LICENSE PLATE DETECTION AND RECOGNITION

Introduction

Automatic License Plate Recognition technology has been popular due to its wide range of applications in intelligent transportation systems, traffic management, toll automation, journey time analysis, etc. Due to the variation in the background and font color, font style, size of the license plate, and non-standard characters, license plate recognition is a great challenge in many countries. To overcome such issues, this study applies a deep-learning strategy to improve license plate recognition efficiency. Several computational photography concepts were utilized along the way to make the process easier, along with Machine Learning algorithms.

Approach

A typical License Plate Detection and Recognition pipeline involves detecting vehicles in the frame/scene using an object detection deep learning model, localizing the license plate on that vehicle using a license plate detection model, and recognizing the characters on the license plate using optical character recognition. Although we didn't intend on implementing several models, over the course of solving this problem we realized that the first model didn't yield satisfactory output in specific scenarios. It became imperative that we train and implement an improved version. This following section contains approaches and results from these two models.

Inception-ResNet-V2

Fig 1: Clockwise from top-left: Results of Inception-ResNet-v2 model trained on 433 images

a. with 500 epochs
b. with 100 epochs
c. with 200 epochs
d. with 10 epochs

Inception-ResNet-v2 was the first model that was trained by our group to identify license plates and achieve desired outcomes. It is a convolutional neural network with 164 deep layers and can classify images into 1000 object categories including animals and general everyday items. The architecture of Inception-ResNet-v2 enables it to perform extraordinarily well as per the ILSVRC 2012 image classification benchmark, making it the first choice to be explored. The version pre-trained model from Tensorflow Keras applications [2]. The following procedure was followed:

- Converted each image into an array and normalized after resizing to 224 x 224.
- Normalized the labels and split data into training and testing sets using Sklearn
- Fine tuned the existing model by:
    - Instantiating a base model and loading pre-trained weights into it.
    - Creating a new model on top of the output of several layers from the base model containing 2 Dense layers with Relu activation and one output layer with sigmoid activation function to get xmin, ymin, xmax, ymax of the license plate position.

Finally, the new model was trained on the training dataset. The total number of parameters the model saw as a part of the deep layers was 74,130,404 out of which 74,069,860 were trainable parameters. We performed the

training on several epochs values- 10, 100, 200, 500 on a batch size of 10. Easyocr library [3] was used to interpret the license plate number.

The four trained models were tested to confirm the accuracy of the output, but the generated outputs either seemed to not fully capture the license plate or failed to identify the second or third license plate in the same frame, should the frame have more than one plate. Figure 1 shows the difference between model performance on 4 different values of epochs. The model performs better with increasing epoch but only up to a certain threshold. Both epochs 200 and 500 have similar performance for this input instance and surprisingly, loss for 200 epochs (0.0036) was lesser than the loss for 500 epochs (0.0040).

After discovering these discrepancies in the result, the options were to discard this model and start fresh with a new deep learning based network or continue investing in further training with more varied datasets. Since finding quality datasets had been a major challenge the first time around, starting fresh with a different model seemed to be the more sensible approach.

You Only Look Once (YOLO) Model

YOLO or You only look once, is one of the most widely used deep learning-based object detection algorithms. It divides an image into a grid system, and each grid detects objects within itself. The model based on this algorithm was implemented using Pytorch Hub [5] and required slightly different label parameters (center position of the bounding box as opposed to upper-left corner pixel indices in case of the Resnet). The labeling is then normalized between the width and height of the image, so the input label becomes [class, center_x, center_y, width, height] of the bounding box. The model was trained on 100 epochs with 157 layers, 7012,822 parameters and took 0.375hrs to fully run on Colab GPU. The output model was then exported in 'onnx' and 'torchscript' format to a local drive. The efficacy of the model was tested on a traffic video which was pulled from a website specializing in traffic flows [7]. The video was decomposed into frames and an object detection model applied on each frame to find all occurrences of the license plate. The captured plate was displayed with a bounding box around it, as an additional visual check.

Fig 2: Detection Results for YOLOv5

Figure 2 shows that license plate detection results for the YOLOv5 model are significantly better since it is able to fully capture the plate (as opposed to partial capturing for ResNet). The results also showed a marked improvement in detecting more than one license plate in a single frame.

Implementation Details

Following are some details as they relate to the technical aspects of the implementation:

- In our dataset, we had 433 images of license plates on different vehicles from different views along with corresponding XML files which had some information for license plate pixel positions in the image.
- The models were trained in Colab notebook using Colab GPU support.
- External libraries such as Numpy, Pandas, Tensorflow, Keras, Pytorch, OpenCV, Matplotlib, Pillow, Sklearn, Easyocr were installed and utilized to help with implementation.
- The notebooks are well written with comments and a README file to help execute the project.

Results

Fig 3: Comparison of Object Detection Model Results on (a) YOLOv5 vs (b) Inception Resnet v2

The results of this test are shown in Figure 3 where (a) and (b) are the results of YOLOv5 and Inception Resnet V2, respectively. It can be seen that the Resnet model only detects one occurrence of the license plate object (which is also offset from the actual area of interest) and hence fails to give good results when presented with an image containing more than one object. Some of the Optical Character Recognition results on the detected license plates are shown in Figure 4. Figure 5 contains the results of our trained model which shows the mAP increases and the loss reduces with the increase in number of epochs. The link to all our results is also compiled in a video here. As you will see in some of the images with complex number plate design, the Easyocr library fails to return the correct text of the number plate even though our object detection model performs quite well.

Fig 4: Character Recognition Using EasyOCR

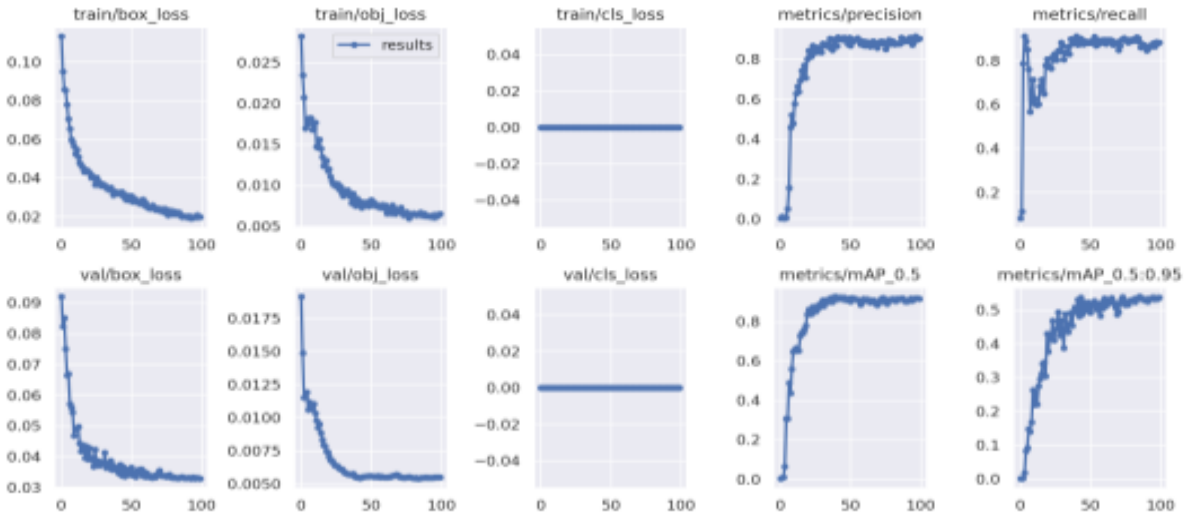

Fig 5: Results of YOLOv5 model training

Innovation and Challenges

As a part of this project, several pre-trained models were studied for object detection, and their implementations explored including Mobilenet, RCNN, Inception-Resnet, YOLO [6]. Choosing this model and researching the capabilities to understand the fit given the tasks we were trying to carry out was a big challenge, which became even more important after the first model did not return expected results.

Fig 6: Real Time Detection using YOLOv5

While implementing real time detection of license plates on a traffic video feed, we found a clever way to separate out important segments of the frames and feed that to the model for detecting objects. Since we could see that the model was trained on certain kinds of pictures of traffic including perspective views and closeups, we thought that testing the model on images with a similar setting would work better for the video feed. In a video frame of size 800 x 800, we derived the important region of the frame which had the most chances of finding information for identifying as a license plate, processed that area on the model, fitted it back to the bigger frame. The basis of deciding an important segment versus less important segment was simple. We noted the portion of the video where most number plates were falling in and cropped the same region for all the frames. This innovation worked well for the YOLOv5 model but didn't work for the Inception Resnet model as we described above.

Another challenge that the group encountered was training the model with the original labels. Since it was hard to predict the exact nature of label data that would have helped to generate accurate results, it made sense to normalize the label values between 0 and 1. A normalization and denormalization script for converting the pixel location values from a bigger range to [0,1] and back was therefore put together to accomplish the goal.
Overall, we enjoyed the learning experience of implementing this project and familiarized ourselves with many latent tweaks to tackle machine learning modeling errors that would helped save so much time if we would have known and been mindful of them.

Group Members Responsibilities

Both members contributed equally. We collaborated on pre-processing of the image data and designing and training a license plate detection Inception-Resnet-v2 model. We researched the new models after Inception's less than optimal results and curated the data and wrote code to clean and identify gaps in the input data set. Sid carried out the training for the new model while Pallavi wrote the first draft of the report which was subsequently edited and finalized together.

References

[1] https://ai.googleblog.com/2016/08/improving-inception-and-image.html
[2] https://keras.io/api/applications/inceptionresnetv2/
[3] https://pypi.org/project/easyocr/
[4] https://docs.ultralytics.com/
[5] https://github.com/ultralytics/yolov5/issues/36
[6] https://analyticsindiamag.com/top-8-algorithms-for-object-detection/
[7] https://www.pexels.com/video/traffic-flow-in-the-highway-2103099/