License Plate Detection

Ronaldo Carrazco

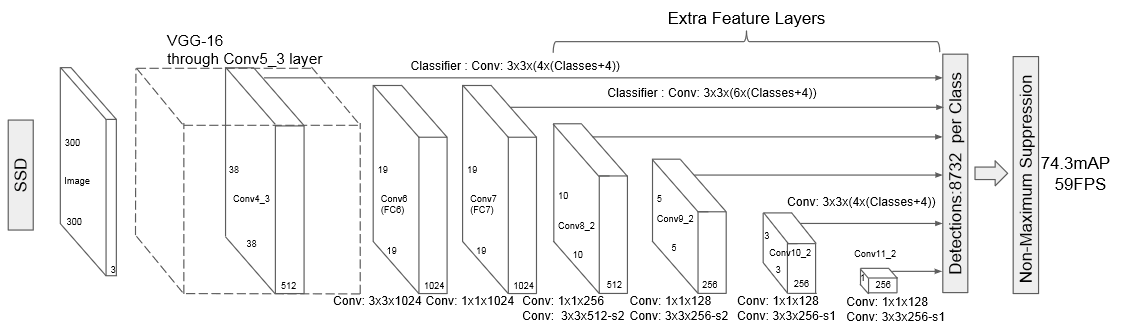
CAP 4612

## **Introduction**

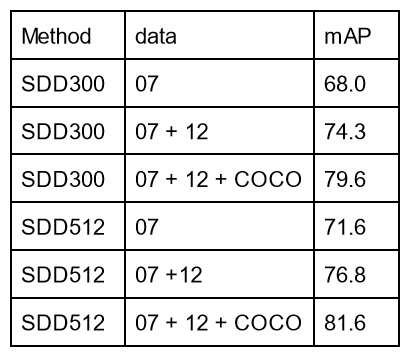
In this project, I will be creating an LPR, License Plate Detection. The sole purpose of a model like this would be to locate where a license plate is in an image. A project/ deep learning model like this is very useful in real-world scenarios. Users would be able to detect cars that are being searched for with an algorithm like this. This has also been used for years now with many cities having camera towers linked to one another. And if a certain license plate is detected it will help locate them. It can help with locating: stolen cars, kidnappings, criminals, traffic light speeding tickets, highway tolls, and much more. Additionally, models like this could be trained with other data to search for different objects. An example would be if you want to detect cars, people, animals, or anything, it all works similarly.

**Related Work**

Many papers have been made for LPRs. Each with their own variant on how to make it better; including different models, feature extraction, and OCR (optical character recognition).

In an older paper titled “SSD: Single Shot MultiBox Detector,” the authors go over their new (at the time) model for object detection. In this model, they use a model called VGG16 as the base network image classification. VGG16 is a well-known and powerful CNN, convolutional neural network. And several convolutional layers with different sizes, kerala, and strides. These extra layers will allow for the model to accurately output the coordinates of the bounding box.

With this model, the authors were able to get high precision. SSD300 and SSD512 represent the input shape. The data titled 07 and 12 represent VOC2007 and VOC2012. And if you compare the two methods with the same data the SDD300 is lower by 2-3 units. Additionally, the larger version of this model shows all the objects it was able to detect including bikes, buses, cows, people, sofas, and more. It also performed much better when compared to Fast and Faster R-CNN with the input size much larger than the SSDs at a minimum of 600.



In another paper titled “License Plate Detection and Recognition in Unconstrained Scenarios” used several models to get the license plate number. The method used in this paper was to first detect the vehicle, then license plate detection, and then OCR(optical character recognition). The average accuracy from detecting to recognition was 89.33%. For car detection, a YOLOv2 was used. For LPD a model called WPOD-NET, Warped Planar Object Detection Network, was used to verify each plate. And for OCR a modified YOLO network was used. Before the plate is sent to the OCR the license plate goes through a rectification process, allowing plates that are at different angles to be straight. The data to train the OCR was created from seven random numbers and several augmentations such as Gaussian blur, salt and pepper noise, random background, translations, and color change.

**Data**

The data used in this project was from a Kaggle data set titled “[Car License Plate Detection](https://www.kaggle.com/datasets/andrewmvd/car-plate-detection)” by Larxel. It contained 433 images of license plates, including the corresponding license plate. The images were all different sizes. The annotations were in separate .xml files for each image. With some of the most important information for this project being: xmin, ymin, xmax, and ymax. A drawback of this dataset was it contained no images with no license plate. Additionally, each image had one annotation for the bounding box, whether the image contained more than one license plate.

**Implementation**

## Libraries used

There were several libraries that were used to complete this project including openCV(cv2), Matplotlib, Numpy, XML, Pytesseract, and Pillow(PIL). The main framework to power the SSD was TensorFlow which includes keras. And two modules which include random (for sample, choice, randint) and os.

OpenCV was for reading image files, adding rectangles, and resizing images with inter\_area, an interpolation, allowing any initial size to resize to larger or smaller. Matplotlib was used to display all the images and graphs, which included a line and a histogram. Numpy was used mainly for arrays and decode tensors to allow the preprocess to be much easier. Additionally, Numpy was used to iterate over each batch in the datasets. XML simply reads XML allowing the program to get the bounding box. Pillow was used to convert and prepare an image for Pytesseract. Pytesseract was used for the OCR step returning text. It did not accept images that are in Numpy arrays which were used in the entire project, it needed pillow images.

TensorFlow was used for the deep learning and neural network in this project. The datasets were held in tensor datasets allowing the preprocessing step to go smoother than without it. Preprocessing with a single function with tensor to resize, normalize, and get annotations. Then create an SSD with TensorFlow Keras layers including the VGG16 model and 2D-Convolutional, Dense, and Flatten layers.

Pipeline

Using TensorFlow for this project allowed the pipeline to be much cleaner in code and relatively short. The first step was to create a dataset, which first contains all the file names of the image Ex. Cars1.png. The two helper functions allowed the pipeline to have the useful information it needs to train. This includes preprocess\_image and get\_normalized\_bbox. “Get\_normalized\_bbox” allows an image path to be used as an argument to search the .xml file for it and get the bounding box values. Since the bounding box was not normalized to [0,1] it was done in this step for convenience. This then returns a numpy array of [xmin, ymin, xmax, ymax]. (x,y) min corresponds to the bottom left coordinate of the bounding box and the (x,y)max is the coordinate for the top right. Then the second function preprocess\_image was used to return the image and bounding box.

So once the dataset is made with the image paths this dataset is then mapped over the preprocess\_image function to convert image\_path to numpy image and bounding box. The dataset is then shuffled though not necessary; it allows you to avoid any bias if there is no shuffle. Then the dataset is batched with eight images per batch. Now the shape of the dataset is (8,300,300,3). Once this is all done it is split 90:10 training to test.

Then the SSD300\_VGG16 is made with an input layer of (300,300). This layer will be the input layer for VGG16 with no top and all layers are set to not trainable. Having all the layers not trainable is important as it will keep all the weight from being trained on ImageNet. Add several 2D Convolutional layers based on the research paper mentioned before. The final layer is a Dense layer with 4 neurons for each coordinate. Additionally, this layer must have an activation of sigmoid as this will return a value between [0,1].

This model is then compiled with the optimizer ‘adam’ and Huber loss function which is commonly used in regression. After training to evaluate this model an IOU, intersection over union, function is used to calculate the accuracy of the predicted bounding box.

With the trained model it can now predict where the license plate is. With the bounding box, it can now crop the image to only display the license plate. This will allow the OCR to display what characters are on the screen. Then an image is converted from numpy to pillow which Pytesseract. This then outputs what characters it can detect.

Data Pre-Processing

Pre-processing in this project is short but very important. There are two areas to process the image and annotation. In this dataset, many of the images are in different sizes which for this model could not be allowed. So all the images are resized using and use a bilinear method to allow smaller and larger images to resize to the desired 300\*300 without adding margins on the side. Additionally, the RGB values are normalized from [0,255] to [0,1]. To do this you must divide all values by 255 which can be done simply with a numpy array. For annotations, all values in the bounding box must be between [0,1]. To do this divide the values of the bounding box with the corresponding dimension of the image. For example, if the image is (500,600) and the bounding box is [200,250,400,450]. You would do 500/200 = xmin, 600/250 = ymin, and so on for the other two.



A smaller portion at the end of the pipeline that is processed is to crop an image down to the predicted license plate. This image uses the values from the initial SSD model to create a new image of only the license plate. This step is important, allowing the OCR to only see the license plate and not the surrounding information such as cars, buildings or people.

## Performance measurement techniques

To evaluate this model an IOU, intersection over union, function was used. This function simply gets the intersection of the two bounding boxes, if any, and divides it over the union of the bounding boxes, if any. This will result in a value of [0,1] which can be thought about as a percentage. No accuracy or metrics were completed for the OCR as the annotations did not have the initial license plate number to check if the value was correct or incorrect. But Pytesseract is open source and based on Google’s Tesseract OCR Engine.

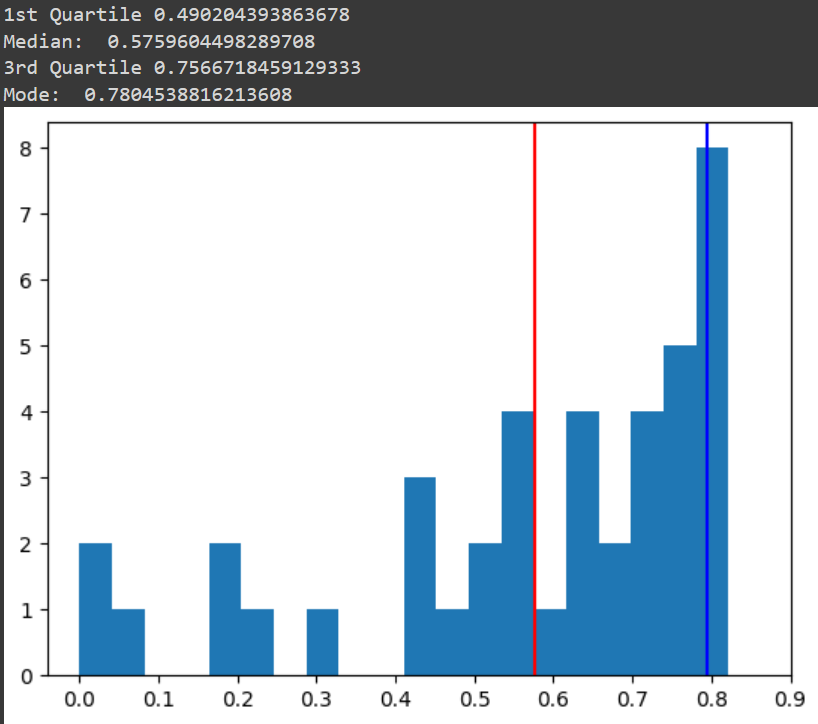
**Results**

## Finding

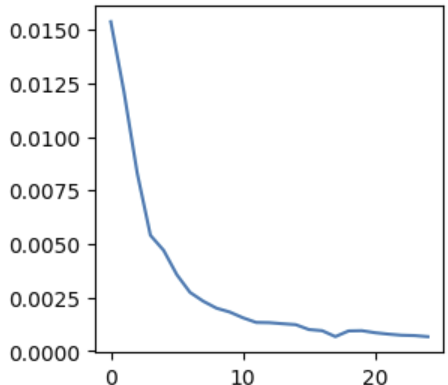
The findings for the SSD model were promising as it is clear that the model is able to detect where the license plate is. This model was also very fast when using the T4 HIgh-Ram in Google Colab. With this hardware accelerator, the prediction happens in less than a second. And as you can see below you can see that the model is pretty accurate.

As for the OCR, this was not accurate at all. Though there are no metrics to back up my claim, much of the text is not able to be detected. Though this can be due to many things such as being blurry, the license plate is not straight, the SSD not accurate enough, and much more.

Below is a histogram of IOU data on the test set. Though when training several times the vast majority of them were like this left-skewed. The X-axis are the IOU, and Y-axis are the occurrences.



Below here is the loss values when training for 25 epochs



Here is an example with both the SSD and Pytessearct. As you can see the image to the right is the input image. Once it gets past to the SSD it then returns the bounding box that can then be cropped displaying the image below. Now the image to the left gets sent to the OCR which then returns the title in this case “KL5442670”.

****

We can clearly see that this is incorrect as there is no “44” in the license plate but a “4A”. Cases like this are common with the current OCR. Many plates when given to the OCR did not return any text, though they looked similar to this case.

**Final Discussion**

## Future work

For future work, I would like to do several things. I would first start with trying a newer version of YOLO as these models are extremely accurate. I would like to test with other base models like VGG19, ResNet, Inception, and MobileNet. One may offer better detection than others. I would also try to make the model predict multiple bounding boxes and not just one. This would offer much more versatility as it can locate more license plates or objects that are trained on.

I would also try to find a larger dataset that contains more images, license plates in single images, and images without license plates.

## **References**