Automatic Vehicle License Plate Detection Using Deep Learning

SIADS 699 MILESTONE PROJECT

Submitted by

Shahbaz Masood, Marshall Nicholas

Mentor

Michelle LeBlanc

Project

<Github repo link>

Contents

Introduction **2**

Use Cases 3

Data Source **3**

Approach **4**

Data Preparation 4

Bounding Box Regression 5

A) Simple CNN 6

B) VGG 16 7

C) Inception ResNet - Fully trained 10

D) YOLO V5 13

OCR **15**

Conclusion **16**

Contribution **16**

References **17**

Annexure **18**

1) VGG16 Layer Specifications 18

2) Inception Resnet Layer Specifications 19

3) Sample Predictions 20

Introduction

Across the globe, automating vehicle identification in real time has gained prominence for a variety of reasons ranging from issuing speeding tickets to theft identification. In this regard, since the turn of the century, plethora of traffic cams[1] have been placed in every nook and corner of most cities that provide us with rich high resolution data necessary for detection of vehicles at scale. While the ethics and privacy concerns of uniquely identifying vehicles are debatable, a number of use cases for vehicle identification that are non-invasive, anonymised, beneficial and consent driven do still exist. And so our focus in this project is to build a real time automatic vehicle detection system for any of the net positive use cases we will discuss in further sections.

To start off, we will see what is Automatic vehicle license plate detection (AVLPD)? AVPLD is an approach that uses image recognition algorithms to automatically identify and read the license plate number of a vehicle. Although several techniques exist, in essence, the problem can be broken down into two parts. 1) An image segmentation part that extracts the bounding box of the license plate from the image. 2) A vehicle plate number recognition part that reads the vehicle number from the extracted license plate. Our project will focus exclusively on 1) and we will use ready made OCR solutions for 2)

There have been several advancements in automatic vehicle license plate detection over the years, particularly with the development of deep learning and other machine learning techniques. One of the key challenges in license plate detection is the large variation in the appearance of license plates, including different fonts, colors, and backgrounds that vary geographically. Traditional methods, such as template matching and hand-crafted features, struggled to handle this variation in real world scenarios and often had poor accuracy. Even with advanced image processing techniques like contour extraction, the sheer diversity in image angles and resolution, has made the problem very complex.

However, with the advent of deep learning, specifically CNNs, researchers have been able to train models that can automatically learn to detect and recognize license plates from images and videos. These models are able to learn complex patterns and features from large amounts of training data, resulting in high accuracy and robustness. Additionally, deep learning models can be trained to handle different variations in license plate appearance, such as changes in lighting ,orientation and resolution, making them well-suited for practical applications. Thus the primary benefit of using deep learning for license plate detection is its ability to handle complex scenes and challenging scenarios.

Motivation

The decision to take up this problem was straightforward. Our primary motivation was to complete a deep learning project to supplement the learnings of SIADS 642 Deep Learning. We also wanted to focus on a project that dealt with images and image recognition tasks as we wanted to understand the benefits, challenges and limitations of traditional Deep Learning approaches such as CNNs. Also, on a related note, we wanted to have a Convolution Neural Network Based Project in our CV as a personal goal. But, traditional image classification problem sets seemed too trivial to attempt. In this regard, we shortlisted a number of real world deep learning problems and the License Plate Detection ticked all boxes because our approach towards license plate detection is a “bounding box” approach that translates to a four dependent variable regression problem which is widely different from a standard image classification approach.

---

[1] *It is not a surprise that the global smart traffic is expected to grow at 14.6% annually due to the high demand to track vehicular movement.*

Use Cases

While vehicle identification has generally been viewed negatively as part of surveillance, a lot of common net positive use cases for license plate detection do exist such as

- Automated toll payment systems, where a camera system mounted on the toll booth reads the license plate of passing vehicles and automatically charges the toll fee to the vehicle owner's account.
- Traffic monitoring and enforcement, where cameras mounted on traffic lights or on police vehicles can detect and read license plates to identify vehicles that are speeding, running red lights, or otherwise violating traffic laws.
- Parking lot management, where cameras mounted at the entrance and exit of a parking lot can detect and read license plates to keep track of which vehicles are parked in the lot and for how long.
- Access control and security, where cameras mounted at the entrance of a secure facility or gated community can detect and read license plates to verify that only authorized vehicles are allowed entry.
- Stolen vehicle recovery, where law enforcement agencies can use license plate detection to quickly identify and locate vehicles that have been reported stolen.

The key is to ensure the above use cases are implemented in a non-invasive, anonymised, consent based manner

Data Source

The dataset of 433 vehicle image samples was sourced from
https://www.kaggle.com/datasets/andrewmvd/car-plate-detection

There are two directories:

- **Images**: contains 433 png images. They serve as the input variable X for our deep learning regression model.
- **Annotations**: contains the 4 bounding box coordinates(x1, x2, y1 ,y2) of all 433 images necessary to identify the license plate. They serve as the target variable Y for our deep learning regression model.

Approach

Fig 1 shows the approach taken for AVLPD implementation. We have broken the implementation into 3 blocks

1) Data Preparation:- involves preprocessing of image and extracting annotations
2) Bounding Box Regression:- to predict the coordinates of the license plate in the image using deep learning
3) Final Product :- uses OCR solution and integration of model in real time video. The final product will be a scope extension on this project

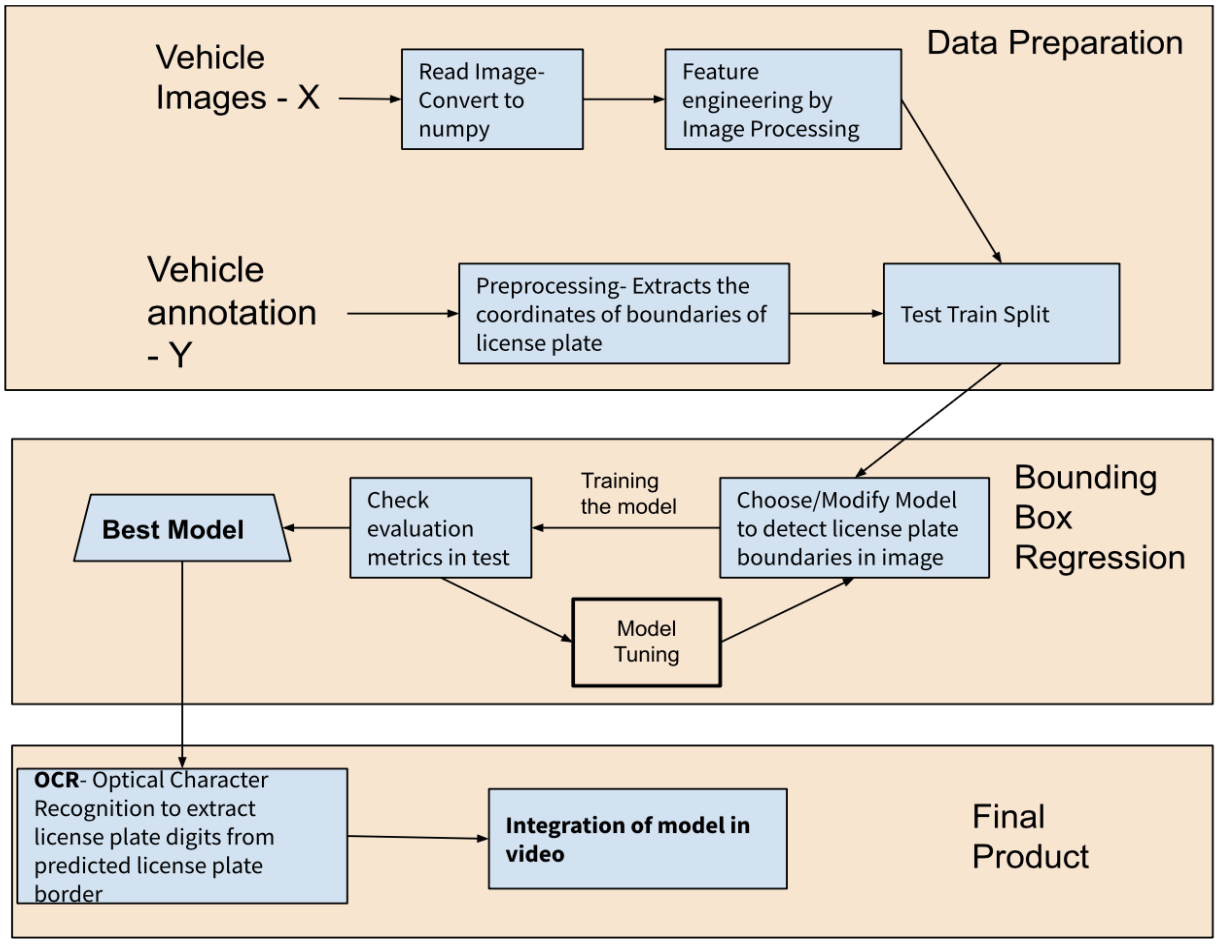


Fig 1- Block diagram for AVLPD

Data Preparation

For the images ,we applied the following two preprocessing steps and extract the necessary features for the deep learning model

1) **Image Sharpening**:- For the baseline CNN model , we applied standard image sharpening technique using a kernel filter. We experimented with other image preprocessing techniques such as contour extraction, median blurring and reorienting the image based on a predefined angle. Out of them sharpening the image proved to cause a discernible improvement in evaluation metric. For non CNN models we did not have to deal with any image enhancement techniques.
2) **Image Resizing and normalization**:-. The preprocessed photos were rescaled to a consistent 224*224 dimensions because the CNN based Deep Learning Models required standard input dimensions in 3 channels and the input images varied in size. The pixel values of the three channels (R, G, and B) were then standardized to a scale from 0 to 1. The input X thereby derived with dimensions 224*224*3 was fed to the deep learning models

For the annotations ,we applied the following cleaning and preprocessing steps

3) **Annotation extraction**:- From each individual XML file the annotations of the image plate [xmin,xmax,ymin,ymax] were extracted along with the [height,width] of the image.
4) **Annotation Rescaling**:- The boundaries of the license plate are rescaled to match the new image dimensions of 224*224. Since different images have different dimensions they are first rescaled to a standard 0 to 1 scale and then rescaled back to 224*224 dimensions. This serves as the target y for the deep learning model with dimensions 224*4

Bounding Box Regression

Objective:- To detect the four coordinates of the license plate within the image (bounding box :- xmin xmax ymin ymax)

For the bounding box regression problem four different deep learning models were attempted

A. Simple CNN Model: A Traditional Convolution Neural Network based Model
B. VGG16 Model
    a. pre-trained
    b. partially trained with transfer learning
C. Inception Resnet Model- Fully trained
D. YOLO V5 model

We will look at a few design choices common across all models

1) **Choice of Loss Function:** For both CNN and VGG16 models the Loss function was set to *Mean Squared Error.*

   The mean squared error (MSE) loss function is commonly used in convolutional neural networks (CNNs) when the goal is to predict a continuous value. This is because it penalizes large errors more heavily than small errors. This can help the CNN to learn to make more accurate predictions for the target values.

   For example, MSE loss can be used when the CNN is trained to perform regression tasks, such as predicting the depth of each pixel in a depth map, or the temperature at each location in a thermal image, etc. In our case since, identifying the continuous values of the 4 Bounding Box coordinates is somewhat similar, hence we use MSE.

2) **Choice of Optimizer:** For both CNN and VGG16 models, we used Adam Optimize for backpropagation, which is a variant of the popular stochastic gradient descent algorithm, and is known for its ability to adaptively adjust the learning rates of individual weights in a neural network based on the past gradient descent steps.
3) **Test Train Split:** For all models, we employed a standard 80:20 train test split of the images

We will look at each of of the models in detail

A) Simple CNN

In the first model we wanted to implement a simple CNN with the layer specifications shown below.

8 Layers - 3 Convolution, 2 Pooling, 1 Flattening and 2 Fully Connected Dense Output Layers. The parameter specifications are shown in Table 1

Since we wanted a naive implementation of CNN to get baseline evaluation metrics and also since the input images had low resolution and pixel ranges, we restricted to just 3 Convolution layers with 224, 64 and 64 filters respectively and all had a kernel size of 3*3. We also did not experiment with strides>1 or with padding the input.

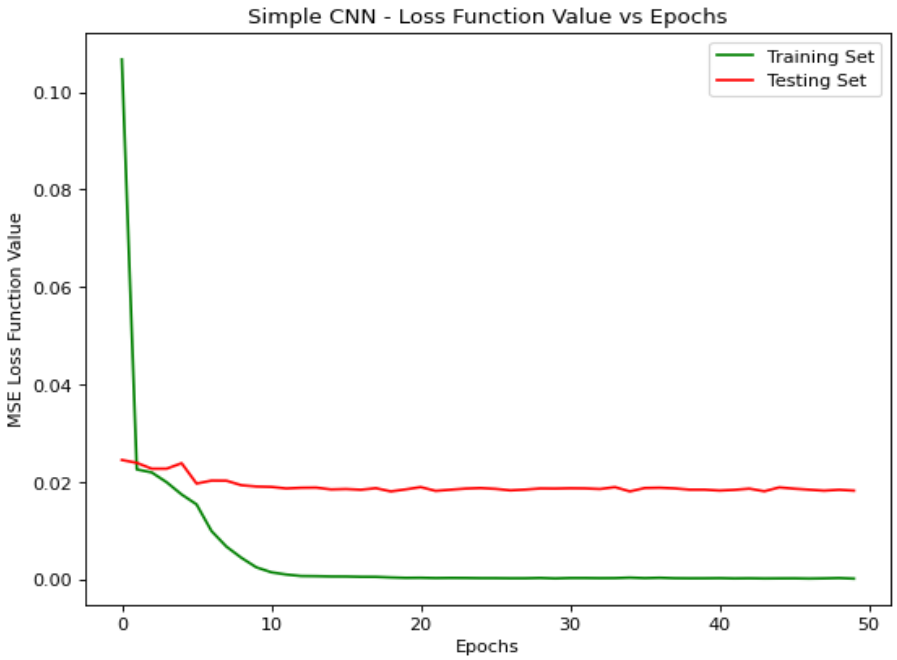
The convolution layers and one other dense layer uses *"relu activation function"* , however the final output dense Layer uses *"sigmoid activation function"* to downsample the output to 4 dimensions. The sigmoid function provides continuous output between 0 to 1 for all values of input. This is unlike a relu which zeros out negative inputs. So for a regression deep learning problem the final output layer is traditionally sigmoid as we want to retain the significance of negative values.

```
Model: "Simple CNN"
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d (Conv2D)              (None, 222, 222, 224)     6272
 max_pooling2d (MaxPooling2D)(None, 111, 111, 224)    0
 conv2d_1 (Conv2D)            (None, 109, 109, 64)     129088
 max_pooling2d_1 (MaxPooling (None, 54, 54, 64)        0
 2D)
 conv2d_2 (Conv2D)            (None, 52, 52, 64)       36928
 flatten (Flatten)            (None, 173056)           0
 dense (Dense)                (None, 128)              22151296
 dense_1 (Dense)              (None, 4)                516
Total params: 22,324,100
Trainable params: 22,324,100
Non-trainable params: 0
```

Table 1: parameter specifications for simple CNN Model

Results:

The CNN model was trained in 50 epochs using an adam optimizer that defaults to a learning rate of 0.001.



The loss function vs epoch is shown to the left in Fig 1. As we can see the training set quickly settles to the minimum Mean Squared Error by 10th epoch. Testing set settles at a faster rate. This indicates that the learning is swift and not deep. We consider this as a baseline model

Fig 1- Standard CNN - Loss Function MSE vs Epochs

For the simple CNN, the validation test results showed a Mean Absolute Error between predicted and actual values of 7.41%

B) VGG 16

VGG16 is a convolutional neural network architecture that was developed by the Visual Geometry Group at the University of Oxford. It is a popular model for image classification tasks, and has been used extensively in a variety of applications, including object recognition, scene understanding, and medical image analysis. It is known for its simplicity and effectiveness.

Please refer to Annexure 1 for layer specifications for VGG16

In our case, we will be using VGG16 for the bounding box regression problem with 2 modifications resulting in 2 different models

1) Model 1 **Pretrained VGG16**:

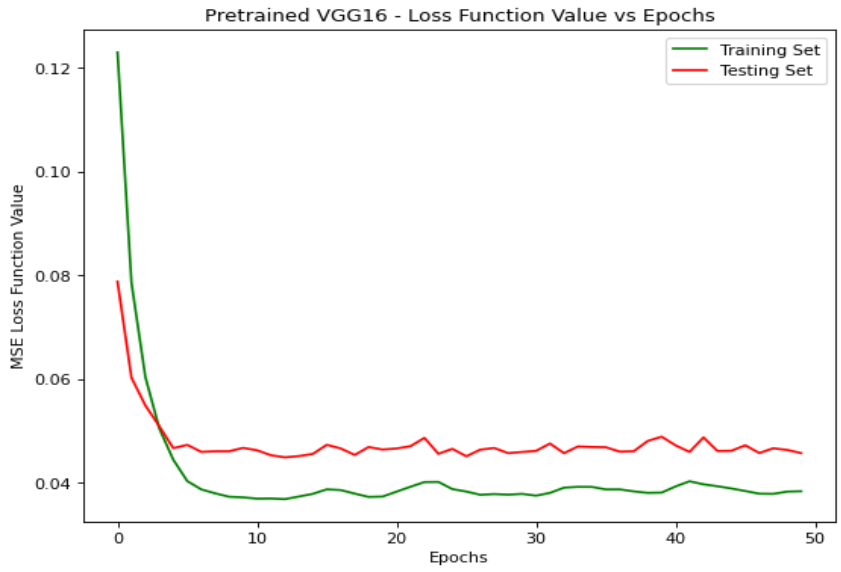
We will use the Pretrained VGG16 directly by adding just an additional fully connected output layer.

The VGG16 will itself not be trained with the training set. The flattening layer and one fully connected layer is added to reduce the model dimensions to 4 continuous variables

```
Layer (type)                 Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 7, 7, 512)         14714688
 flatten (Flatten)           (None, 25088)             0
 dense (Dense)               (None, 4)                 100356
Total params: 14,815,044
Trainable params: 100,356
Non-trainable params: 14,714,688
```

Table 2: parameter specifications for pretrained VGG16

Results:



The model was trained in 50 epochs using an adam optimizer that defaults to a learning rate of 0.001. The loss function vs epoch is shown to the left in Fig 2. Similar to standard CNN the loss function settles before 10 epochs. But unlike standard CNN both the training and testing settle above 0.04 and the gap between the curves is small, indicating possible underfitting.

Fig 2- **Pretrained VGG16** - Loss Function MSE vs Epochs

For the Pretrained VGG16, the validation test results showed a Mean Absolute Error between predicted and actual values of 15.00%

2) Model 2: **Transfer Learning based VGG16**

This is built by adding multiple fully connected output layers and retraining the last Convolution Layer in Convolutional Block 5 with vehicle image dataset using a technique called transfer learning.

Transfer learning:

What is transfer learning?

Transfer learning is a technique in machine learning where a model trained on one task is reused as the starting point for a model on a second, related task. This can be beneficial in several ways:

- It can reduce the amount of training data needed for the second task (bounding box regression), since the model has already learned relevant features on the first task (image classification).
- It can speed up training, since the model has already learned some useful representations and only needs to learn the differences between the tasks.
- It can improve performance, since the model can transfer knowledge from the first task to the second, and leverage the large amounts of data and computational resources that were used to train the original model.

In our case, since the VGG16 model is already trained on the imagenet dataset we want to preserve the rich model weights. But, equally, we also want the model to capture the intricacies of the license plate dataset. And so one convolution layer, the block_5 convolution layer 3 is retrained in this model using transfer learning. i.e all VGG16 model layers except this one are frozen and this particular layer is re-trained with the training data set.

We also add 6 additional fully connected output layers that progressively reduce the output nodes to a 4 variable continuous output.

vgg16 except block5_conv3	(None, 14, 14, 512)	12354880
block5_conv3 (Conv2D)	**(None, 14, 14, 512)**	**2359808– transfer learning**
block5_pool (MaxPooling2D)	(None, 7, 7, 512)	0
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 128)	3211392
dense_1 (Dense)	(None, 128)	16512
dense_2 (Dense)	(None, 64)	8256
dense_3 (Dense)	(None, 32)	2080
dense_4 (Dense)	(None, 16)	528
dense_5 (Dense)	(None, 4)	68

Total params: 17,953,524

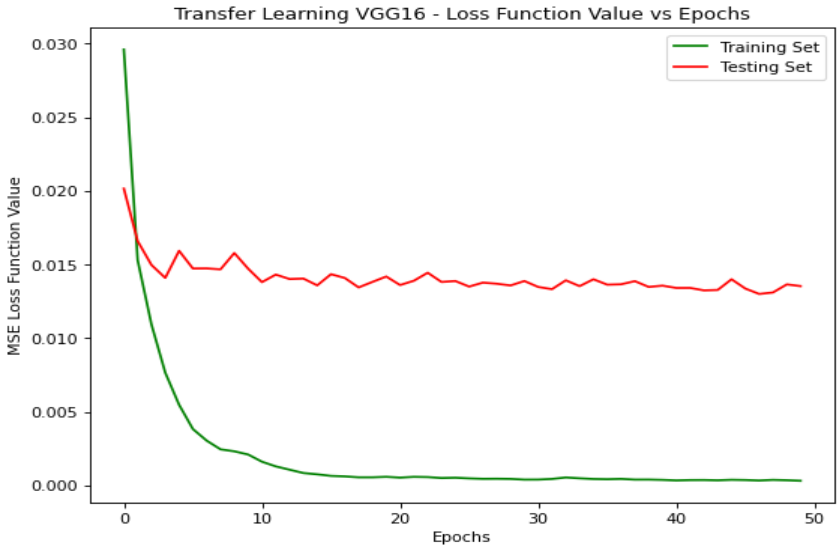
Trainable params: 5,598,644

Non-trainable params: 12,354,880

Table 3: parameter specifications for VGG16 with transfer learning

Results:

The model was trained in 50 epochs using an adam optimizer that defaults to a learning rate of 0.001.



The loss function vs epoch is shown to the left in Fig 3. Both training and testing take more than 10 epochs to converge to a min MSE. The testing MSE is significantly lower (0.015) than untuned VHH16

Fig 3- **Transfer Learning based VGG16** - Loss Function MSE vs Epochs

For the Transfer Learning based VGG16 , the validation test results showed a Mean Absolute Error between predicted and actual values of 0.44%

C) Inception ResNet - Fully trained

We next tried to use another model trained with the same ImageNet dataset that traditionally outperforms VGG16:- **Inception ResNet**

The Inception ResNet model is a variant of the Inception model, which is a convolutional neural network trained to classify images. It was developed by Google and introduced in their paper "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning" in 2016.The salient feature of this model is the use of residual connections, which are additional connections between the layers of the network that help improve the flow of information and the overall performance of the model.

Please refer to Annexure 2 for layer specifications for Inception ResNet

Unlike pretrained and transfer learning based training of VGG16, we have fully trained the Inception ResNet model from scratch to the training vehicle image set. We wanted to see if full training of the deep learning model outperforms the transfer learning based model of VGG16. But this comes at a cost of increasing model complexity which will be discussed in the Model Comparison section. We have added 6 additional fully connected output layers like before.

Layer (type)	Output Shape	Param #
inception_resnet_v2_input (InputLayer)	[(None, 224, 224, 3)]	0
inception_resnet_v2 (Functional)	(None, 5, 5, 1536)	54336736
flatten_2 (Flatten)	(None, 38400)	0
dense_12 (Dense)	(None, 128)	4915328
dense_13 (Dense)	(None, 128)	16512
dense_14 (Dense)	(None, 64)	8256
dense_15 (Dense)	(None, 32)	2080
dense_16 (Dense)	(None, 16)	528
dense_17 (Dense)	(None, 4)	68

Total params: 59,279,508

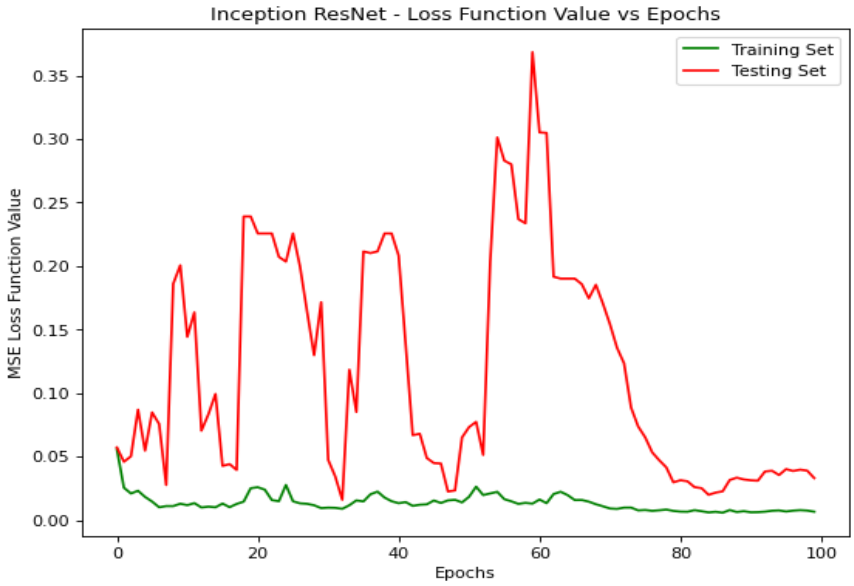
Trainable params: 59,218,964

Non-trainable params: 60,544

Table 4: parameter specifications for fully trained Inception Resnet

Results:

The model was trained in 100 epochs using an adam optimizer that defaults to a learning rate of 0.001.



The loss function vs epoch is shown to the left in Fig 4. The testing set took close to 40 epochs to settle to a minimum MSE of 0.02. It is interesting to note that that MSE hit 4 different local minima before settling to a global minima. The model was highly unstable

Fig 4- **Fully trained Inception- ResNet** - Loss Function MSE vs Epochs

For the Fully trained Inception- ResNet , the validation test results showed a Mean Absolute Error between predicted and actual values of 1.22%

Model Comparisons

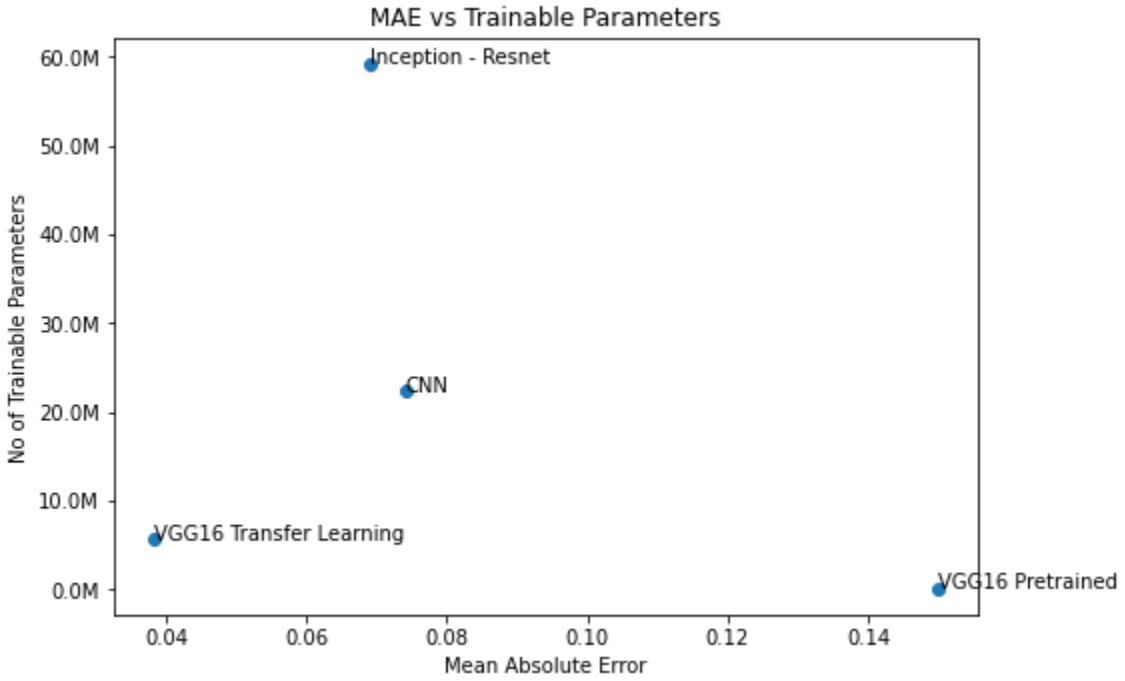


Fig 5: Comparison of Baseline Models MAE vs Trainable Parameters

In the Figure above, for all the models discussed, we have compared the Model Performance in terms of the evaluation metric Mean Absolute error with the Model Complexity in terms of # of Trainable Parameters.

We have used Mean Absolute Error as opposed to a more standard evaluation metric such as Accuracy as ours is a regression problem and not a classification problem. Other evaluation metric choices include Mean Average Precision or the Mean Square Error which also serves as our loss function for backpropagation

As we can see, the VGG16 Model with Transfer Learning gave the best performance in terms of lowest MAE to model complexity.It is interesting to note that a more advanced fully trained Inception ResNet model has slightly poorer performance with more parameters trained.

The sample predictions for all the baseline models are shown in Annexure 3

In the next section, we will see an even more advanced YOLO V5 model which uses a different approach compared to the models already discussed.

D) YOLO V5

Finally we implemented the Yolo v5 algorithm to detect the license plates along with their bounding boxes. YOLO is a fast and efficient object detection algorithm that is suitable for a wide range of applications. It is a single-stage object detection algorithm, which means that it directly predicts the bounding boxes and class probabilities for objects in an image, without using a traditional object proposal step.

YOLO works by dividing an image into a grid of cells, and each cell is responsible for predicting a certain number of bounding boxes and class probabilities. The algorithm then uses a loss function to measure

the difference between the predicted bounding boxes and the ground-truth bounding boxes, and uses this information to update the network weights and improve the accuracy of the predictions.

Yolo expects the bounding box coordinates in a different format. It requires the center of the bounding box and the width and the height of the object, while our data had the coordinates of the opposite sides of the bounding box. A function was implemented to convert those annotations into the required format as below.

Fig 6: YOLO Bounding Box

The Ultralytics implementation of Yolo was utilized where only 1 class was trained i.e. license plates. The implementation offers the leverage of multiple models and additionally the classes can be defined manually. The model itself contains 214 Layers and similar to the above, due to the small dataset size, we used pretrained weights to train the model. The model gave us an accuracy of 84% and trained in approximately 10 minutes on a Google Colab free GPU instance.

Results:

Yolo v5 has 7,022,326 trainable parameters with 214 layers In the below table we can see only 86% of those predicted as license plates were actually license plates

		Predicted as license plate
Actual	license	**0.86**
	NoObject	**0.14**

Table 4: YOLO V5 Actual vs Predicted License Plates

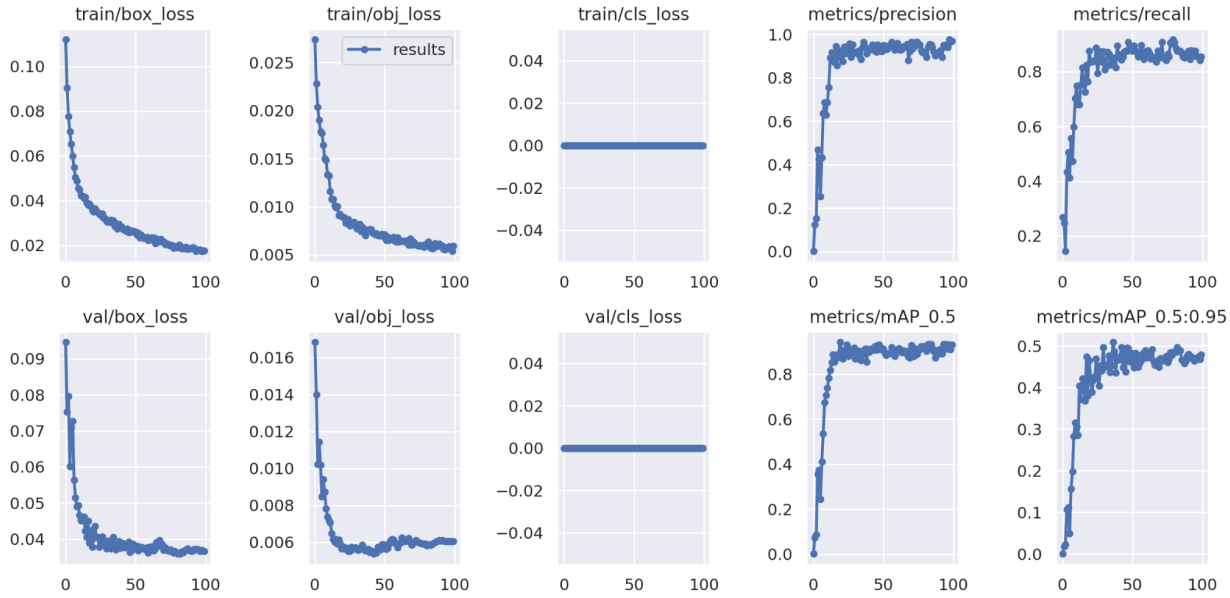


Fig 7: Performance Metrics of YOLO

The license plate detection is shown to the left

We observed that the images 138 and 261 could not be detected while the remaining were really accurate.

The Yolo model outputs the bounding box coordinates along with the probability of detection. Figure 8 shows the threshold for the detection's (confidence) relationship with the F1 Score. We observe that the highest F1 score is observed using a threshold of 0.422.

Similarly, the Precision-Recall Curve gives and AUC of 0.8975 as shown in the figure below.

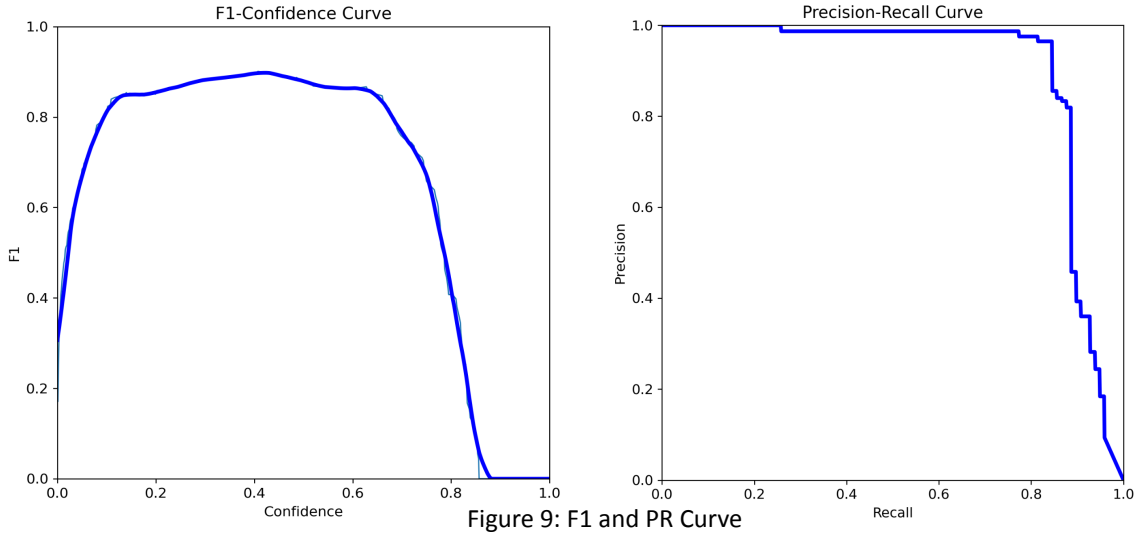


Figure 9: F1 and PR Curve

OCR

AS discussed previously, the bounding box coordinates as output by the object detection algorithm are cropped and OCR is applied onto the cropped images. For our purpose we leveraged the capabilities of the EasyOCR package.

The following are some snapshots of the text extracted.

Figure 10: Successfully extracted LP Numbers

Figure 11: Incorrectly extracted LP Numbers

In the figure 10, we see that the OCR performed well and successfully extracted all the numbers correctly. Which in figure 11, on the first image, the OCR overdid the extraction. This is probably because Yolos bounding box was too relaxed and might need to be further tuned to get better results. In the second image the red circle was missclassified as 'Q'.

The general method to evaluate OCR systems is Licehenstien distance of actual and extracted however our dataset had the limitation of not having the plate numbers with it so we cannot quantify the results of the OCR due to practical reasons.

Conclusion

Model Comparison

Overall, we trained and tested 5 different models to extract the license plates. The standard CNN model performed very poorly compared to the more advanced architectures like VGG16 and Inception ResNet. Within the advanced architectures, we experimented with different training approaches- pretrained, transfer learning and fully trained. The obvious tradeoff was that the fully trained models required more computational power and effort in training more parameters but are expected to have better accuracy and lower MSE loss. On the contrary, what we discovered was that a partial training transfer learning based approach using VGG16 outperformed the fully trained Inception ResNet model. Finally, we attempted with an even more complex model such as the YOLO V5 model. This gave the best performance with prediction accuracy of 86% and required substantially less training than the Inception Resent model

Learnings

We learnt that deep learning solutions involve a lot of complex tradeoffs that have to be optimized from the learning rate to how much training the model has to undergo. Specific to this problem, we discovered that ready made solutions like the fully trained Inception ResNet model may not be the best solution. While the YOLO V5 model gave the best performance, we believe further fine tuning the VGG16 transfer learning model would have yielded better results and this would serve as the possible scope extension for this project. We also learnt there are several edge cases that the model did not predict correctly and so it is too early to deploy any of these models in the real world without further study on the reasons for low performance in edge cases.

Future Scope

We can identify 3 avenues for future work.

1. Develop and deploy an API where images can be uploaded to extract the output.
2. Add support to overlay the predicts on a video so that detection can happen in realtime
3. Further tune Yolo and OCR to increase the detection accuracy.

Contribution

Marshall Nicholas	Shahbaz Masood
1. Project and data Ideation 2. Data Source Research and Manipulation- Extracting Annotations and Resizing Images 3.Implementing Baseline Models: Standard CNN, VGG16 and Inception ResNet 4. Final Report Writing	1. Project and data ideation 2. Data Source Research and Manipulation- Extracting Annotations and Resizing Images 3. Implementing Final Model: YOLO V5 4. OCR implementation 5. Final Report Writing

References

1. *Vehicle Number Plate Detection and Recognition Techniques: A Review:* https://www.astesj.com/publications/ASTESJ_060249.pdf

   The paper talks about how historically the License Detection problem has been approached and the pros and cons of each approach

2. *YOLO (You Only Look Once) V3*: https://arxiv.org/pdf/1804.02767.pdf

   This paper talks about a model built to predict the coordinates of the license plate in an image using 53 convolution layers. This will serve as the primary reference paper for building the model. The work is an extension of the YOLO V1 model.

3. *License Plate Detection and Recognition in Unconstrained Scenarios:* https://openaccess.thecvf.com/content_ECCV_2018/papers/Sergio_Silva_License_Plate_Detection_ECCV_2018_paper.pd

   This paper provides an alternative approach to detect license plates. Here, the license plate detection is done using Warped Planar Object Detection Network that uses the intrinsic rectangular shape of license plates.

4. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning: https://arxiv.org/abs/1602.07261

   The basics of Inception Resnet V4 model

5. Very Deep Convolutional Networks for Large-Scale Image Recognition: https://arxiv.org/abs/1409.1556

   The basics of Inception VGG16 model

6. Fine-Tuning Pre-trained Model VGG-16:

   https://towardsdatascience.com/fine-tuning-pre-trained-model-vgg-16-1277268c537f

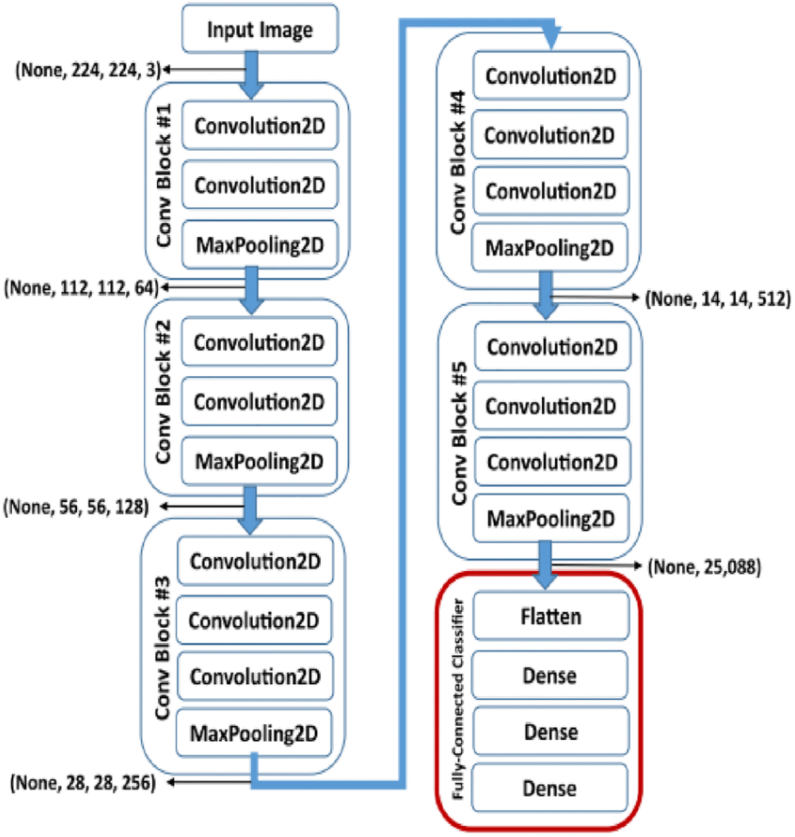
   The article explains the approach to fine tuning VGG16

Annexure

1) VGG16 Layer Specifications

VGG16's layer specifications shown below

16 layers: 13 convolutional layers and 5 MaxPooling layers in 5 Convolution Blocks and 3 fully-connected layers after a flattening layer.



The VGG16 was trained with the ImageNet dataset

ImageNet is a large dataset of labeled images that is commonly used in computer vision research. It contains over 14 million images, labeled with one of more than 20,000 different object classes.

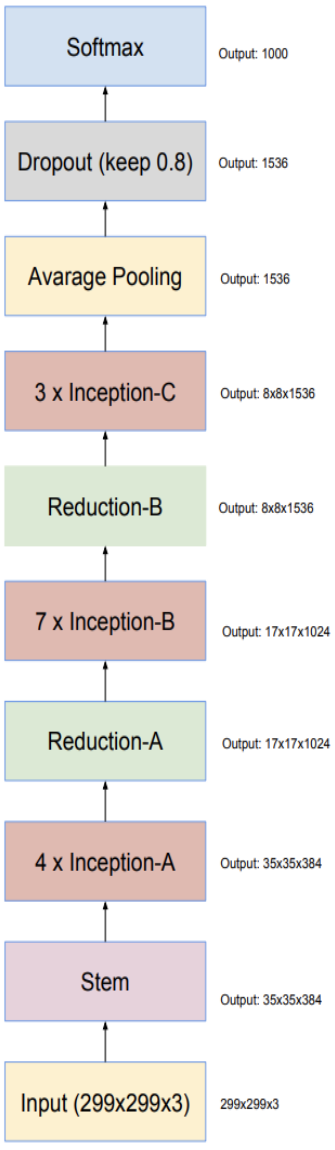
And the model weights of VGG16 are tuned to this exhaustive dataset.

Source:

https://www.researchgate.net/figure/Block-diagram-of-VGG-16-network_fig5_339851513

2) Inception Resnet Layer Specifications

Inception Resnet's layer specifications shown below



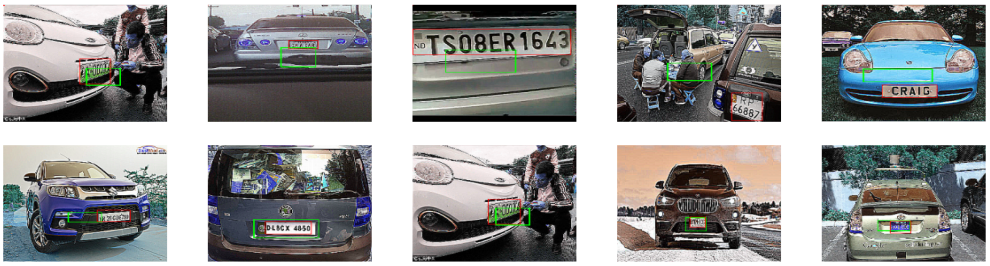
Source: [Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning]

Please refer to the above paper on more details of the design of Inception- ResNet model

3) Sample Predictions

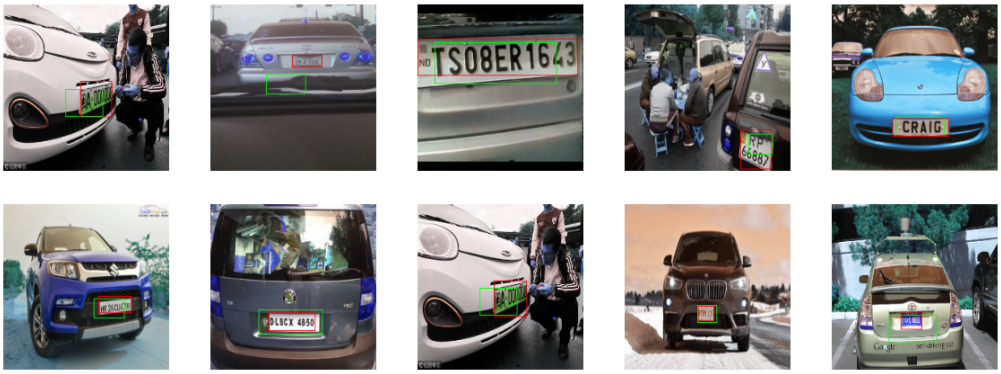
We can see the bounding box predictions made for the discarded models. The red box indicates the actual license plate and the green box represents the predicted license plate

1) SImple CNN



2) Pretrained VGG16

3) Transfer Learning VGG16



4) Inception Resnet