PGP AIML (2022-2023)

Final Report

Computer Vision – Car Object Detection

Submitted by
Manjunath Vallu
Sangameshwar Magar
Yeshwanth Chinna
Shashikant Upadhyay
Nikitha Nayak

Mentor
Abheesta Arnav

Acknowledgements

We would like to express our profound gratitude and appreciation to our mentor **Mr. Abheesta Arnav**. His suggestions are invaluable in formulating the right direction and choosing the right options for the project. Thanks to his proficiency in the field of computer vision, we got through various blockers and achieved the desired result.

Special thanks to our program manager **Ms. Keerti S Nair** for her extended support throughout the course of the project.

Our deepest thanks to the **Great Learning Platform** for providing us various source materials of AI and ML that equipped us with enough knowledge in the field of computer vision for completing this project.

Manjunath Vallu
Sangameshwar Magar
Yeshwanth Chinna
Shashikant Upadhyay
Nikitha Nayak

Table of Contents

1 Summary of problem statement, data and findings

1.1 Project Objective

The primary objective of the project is to design a **DL based car identification model**. Given an image, the model should be able to perform the following tasks:

I. **Classification** – classify the image into one of the classes of the car.

II. **Object detection** – localize the car in the image by predicting the bounding box around it.

A clickable UI to be built for uploading any image of the car, which generates an image with the above predictions. The UI would be powered up by the model in the backend.

1.2 Understanding the Input Data

- The cars dataset contains 16,185 images with 196 classes of cars.
- The data is split into 8,144 training images and 8,041 testing images. Each class has been split roughly in a 50-50 split into training and testing images.
- Each class label comprises of the level of make, model, year of the car. For example, 2012 Tesla Model S or 2012 BMW M3 coupe.
- The annotations are included for included as csv files for both train and test sets. Each annotation maps the image name to its class and bounding box coordinates.
- The bounding box coordinates follow the (xmin, ymin) and (xmax, ymax) convention. These denote the left bottom and the top right coordinates of the bounding box respectively.
- The car names csv file is also attached which can be used to map label number to the class name.

1.3 Data Exploration Summary (EDA)

- Validated if there is any missing data i.e. to verify if any of the images do not have corresponding labels and annotations and vice versa. Found no missing data.
- All the images have different heights and widths. This indicates that we've to rescale all the images as a first pre-processing step, before feeding to any model.
- All the images have 3 channels RGB. We've an option of turning them to grey scaled, if the burden of computation is too high for the model.
- All the labels are within the range of 1 to 196 indicating maximum number of classes.

- Though the 'number of images per class' are slightly varying for each class, the distribution indicated that the class imbalance is not too huge to make a model biased.
- Verified if the condition (*xmin < xmax < image_width*) and (*ymin < ymax < image_height*) are satisfied by all the images. Found an image violating this and fixed it. This makes sure our images are clean with no bounding box crossing the image boundary.
- Observed that the above condition is satisfied as a trend in the pair plot of the coordinates and the image dimensions.

(Please find the related visualizations in the **visualizations** section)

2 Overview of the final process

The first milestone involve the following stages:

1. Pre-processing for the classification model.
    a. Rescaled all the images to the same shape.
    b. Used the following augmentation techniques for the model to learn better and avoid any overfitting.
        i. Rotation
        ii. Width Shift
        iii. Height Shift
        iv. Zoom
        v. Horizontal flip
    c. Used ImageDataGenerator() to feed the images to the model on runtime.
2. Achieve benchmark results by fine tuning.
3. Evaluate the base classification model based on validation_accuracy and validation_loss.

CNN models evaluated for the classification:

- **Mobilenet-V2**
- **Resnet-50**

The second milestone involve the following stages: (Yolo V5 S)

- After initial evaluation, decide whether to stick with the base model or try other advanced techniques.
- Pre-processing for the object detection model. This model would help us in the classification as well.
    - After doing various experiments with limited number of classes, we found that yolov5 works best with more number of training images.
    - According to the official documentation, the yolov5 model works best with a data split of (train,val,test) -> (70:20:10).
    - An easy to use function has been written to combine all training and testing images together and later split into desired ratio per each class.

    - **prepare_data_yolov5(train_split=0.7, val_split=0.2, test_split=0.1, first_n_classes=196)**
    - The above function not only helped us to split the whole data but it can help us load only a limited no of classes. This was crucial for us to do estimations with a basic model with limited no of classes.
    - It also prepare data.yaml file to feed yolo model regarding class labels, names etc.
- Fine tune the model to achieve the results.
- Evaluate the final model based on recall, precision and map50 (IOU) and inference speed.

Hybrid RCNN model evaluated for the classification and object detection:

- **Yolo V5 S**

Design a basic clickable UI with the best model obtained so far.

3 Step-by-step walk through of the solution

- Initially, we worked on the basic CNN classification models to set attain a base level model. Two models were considered Mobilenet V2 and Resnet 50. Although they did not show promise in the beginning, we opted for transfer learning and froze many layers to simplify the model.
- Learnt that as the complexity of the model decreased upon freezing, the model has learnt faster and we achieved a validation accuracy of 68% on the Mobilenet V2 model.
- Considering the second milestone, the basic CNN models lack in inference speed, object detection and lighter to deploy. Considering all these departments, we've opted for the state of the arts model Yolo V5 with minimal architecture S model.
- The official recommendation is to use the default parameters and train base on the pre-trained weights from the coco image.
- Hence, when trained with all the dataset together, the model is learning but very slowly.
- Therefore, we opted to check with base model trained on only 3 classes. This showed us great result and gave us confidence to pursue the model.
- As the number of training images were less, we opted to dump all the data together and split into 70:20:10 ratios of train, val and test respectively.
- This ensured that the class with least number of images would at least get 40 images per training.
- Once the data is preprocessed, the model still took lot of time for each epoch around 5 mins. Therefore, we wanted to reduce the training time as much as possible by tweaking:
    - Image_size
    - Batch_size
    - Caching the images
    - Number of workers – (Parallel threads)

- By playing with all these parameters, we evaluated the time per epoch and RAM used as we've a limited GPU of 12GB.

	img_size	batch_size	workers	cache	epoch time in min	GPU per epoch in GB
0	640	16	8	yes	8.47	8.87
1	320	16	8	yes	1.39	2.1
2	320	32	8	yes	1.1	3.24
3	320	64	8	yes	1.03	6.84
4	320	64	8	yes	1	6.84
5	320	64	8	yes	0.59	6.84

- After playing with these parameters, we're able to reduce the per epoch time to just 59 seconds and by using just 6.84 GB of GPU RAM.
- Did not reduce the image size further in the fear of data loss. Once all the experiments are done, we considered this as the **Best configuration: --batch 64 --img-size 320 --cache --workers 32.**
- Then, we went on to train the above model with default parameters which resulted in the mAP of 96% and recall of 93% on the test images.

4 Model evaluation

- Inference speed, ease of deployment, faster to training are considered the basic eligibility criteria for selecting the model.
- After running the model for 150 epochs over 16K training images, the yolo v5 model showed great results with the mAP of 96% and recall of 93% on the test images.
- It has inference speed of 1.6ms and 2.1 ms per image on the test images.
- Considering, the light weight of the model i.e. 15.3 MB, the model has been selected the best.

5 Comparison to benchmark

The initial benchmark was laid to predict all the test images with the best accuracy and high inference speed.

However, the CNN models used for basic classification failed to reach the benchmark set. The following are the results from the first milestone:

Pre-Trained Model	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
MobilenetV2	80	**68**	0.7	1.13
Resnet50	97	**62**	0.08	1.56

The best model from the initial milestone was **Mobilenet V2** with a validation accuracy of 68%. This model is relatively heavy compared to Yolo V5 and has low inference speeds. Also it can only perform image classification. Hence, this model has been rejected and we moved forward with the state of the arts model yolov5.

The results of training Yolo V5 S are as follows:

Model	Recall	mAP50
Yolo V5 S	93	**96**

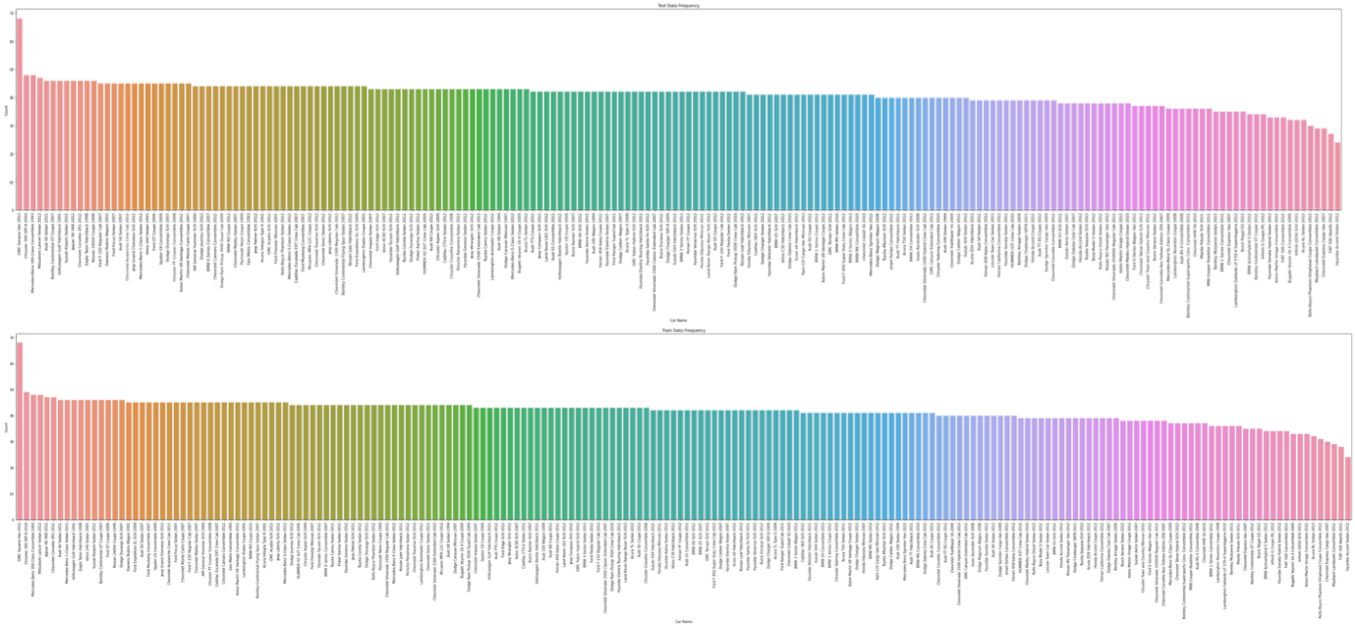
mAP 50 is nothing but the Mean Average Precision of IOU of the images over 0.5. This indicates how well the objects are being detected.

This Yolo V5 S model is very light weight about 15.3 MB and has an inference speed of 1.6ms. Hence, this model is selected as the best model we have.

This model is later used as a backend for a basic clickable UI which is built using Gradio.

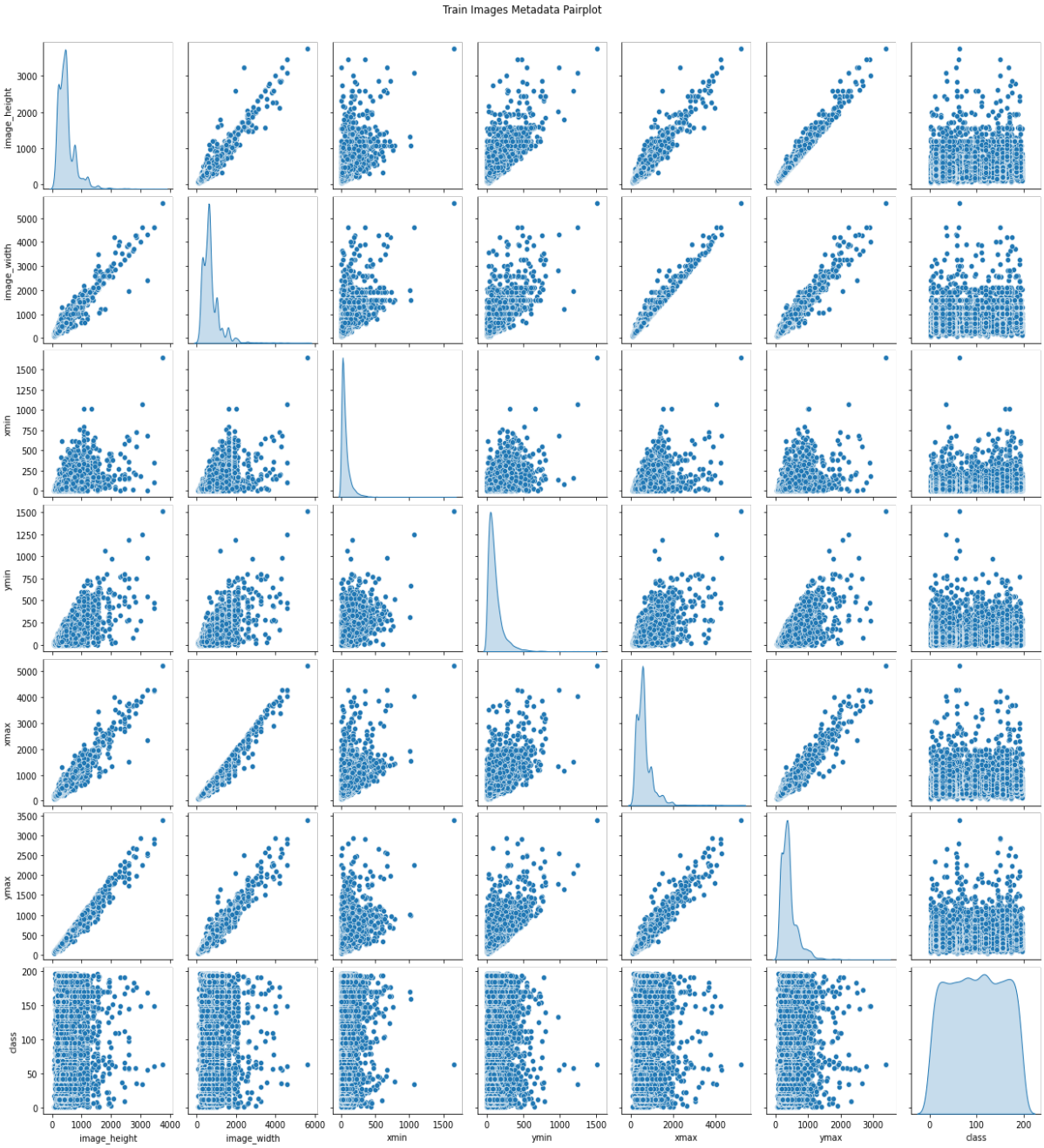
6 Visualisations

6.1 Class distribution in train and test datasets.



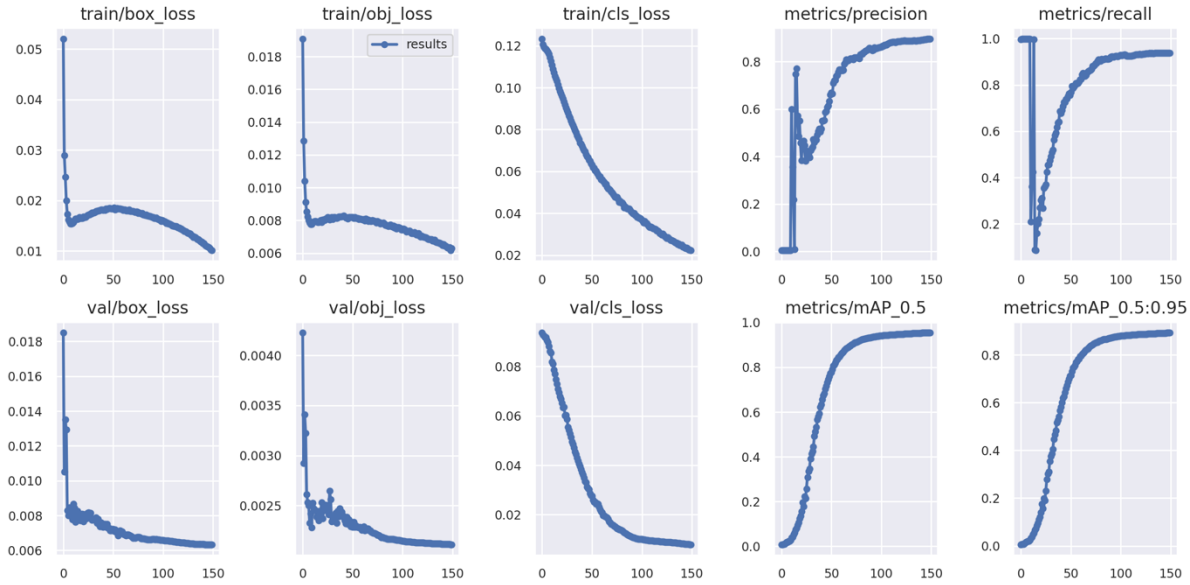
- As mentioned in the EDA summary, we can see slight class imbalance, but nit enough to make the model bias.

6.2 Metadata pair plot



* Trend between bbox coordinates and the image dimensions can be observed above.

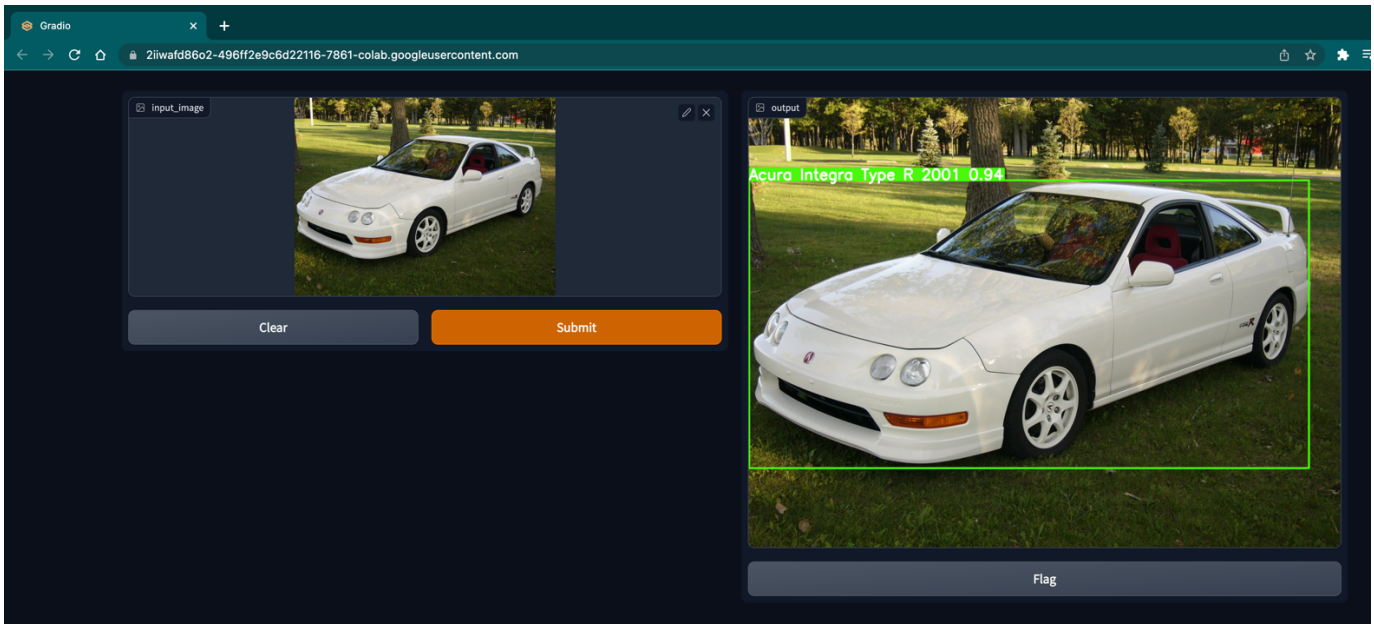
6.3 Yolov5 training metrics



- We can see the map_50 and cls_loss keep decreasing for the validation set indicating good behaviour.

6.4 Basic clickable UI for car detection using Gradio

* The following UI has been built by loading the best model and using the simple interface function of Gradio.



7 Implications

- The solution can be used for automobile surveillance.
- Capturing the images of cars parked for each
- Traffic surveillance for census capture.
- Building a car reseller catalogue.

All these above fields can be explored in more detail for the implications of the project in automobile industry

8 Limitations

- The model has to be fed with various types of images like gray scale, in different angles etc. with larger number of dataset to work in the real time world well.
- The limited number of training images made the training and validation phase difficult. All though augmentation helped, acquiring more data would help the model to be generic enough for real world applications.
- The current model deals with only one object per image, need to be trained on more objects per image.
- The project is limited to the image detection only, this can be extended to detection in the videos as inference speed is high.

9 Closing reflections and learninge

- Understanding the need of augmentation for the limited data set for validation.
- Transfer learning to save us lot of time in building the model from scratch.
- The importance of base line modelling before diving into the bigger dataset.
- Using Gradio to build a simple UI. This can be improved with Fast API as backend.

10 References

- Transfer learning – https://www.tensorflow.org/tutorials/images/transfer_learning
- Mobilenet V2 – https://keras.io/api/applications/mobilenet/
- Resnet 50 – https://keras.io/api/applications/#classify-imagenet-classes-with-resnet50
- Object detection using YOLO: https://www.mygreatlearning.com/blog/yolo-object-detection-using-opencv/
- Yolo V5 – https://github.com/ultralytics/yolov5
- Gradio for UI – https://gradio.app/image_classification_in_pytorch/