Final Competition Hackathon : Artificial Intelligence

Team : CG\_TEAM\_JVQRE

Problem Description:

In this hackathon, your mission, if you choose to accept is to solve a very critical problem in the overall Smart Parking solution - "**Identify available parking spots from parking lot camera images**".

The main usage of this project is to develop a Smart Parking system that typically obtains information about available parking spaces in a particular geographic area and process is real-time to place vehicles at available parking spots.

The main advantage of implementing this project is by deployed as a system, smart parking thus reduces car emissions in urban centers by reducing the need for people to needlessly circle city blocks searching for parking. It also permits cities to carefully manage their parking supply and finally, it reduces the daily stress associated with parking woes.

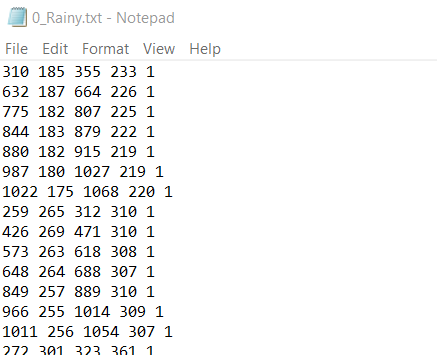
Apart from that if we can apply this model to a cloud based system, we can provide the user with the real time details of the available parking lots near his destination. As we are able to get the details from every parking lot camera, we can use them in terms of national security. Instead of checking for a suspect manually we can ask this system to search for all the vehicles with the given features.

About Dataset:

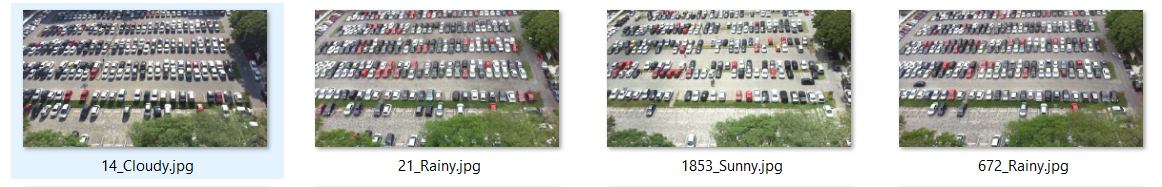
There are quite a lot standard dataset for CAR parking images, oldest dataset called PK LOT but a new dataset was released in 2016 called PUCPR Parking and I preferred this dataset over PK LOT.

Pontifical Catholic University of Parana+ Dataset (PUCPR+) - contains information of the 16,456 cars. Images - 10th-floor-view images (.\*jpg) of parking lot from a building in PUCPR. And annotations - text files (\*.txt) with the label of cars per line, and file name is corresponding to the image file.

Dataset contains 2 folders, one folder contains 125 images of car park area at different period and corresponding annotation details for each image present in ‘Annotation’ folder. E.g. of one image file is Xmin Ymin Xmax Ymax ClassId



Images are taken at during different weather conditions, the example is as follows,



Data Pre-Processing:

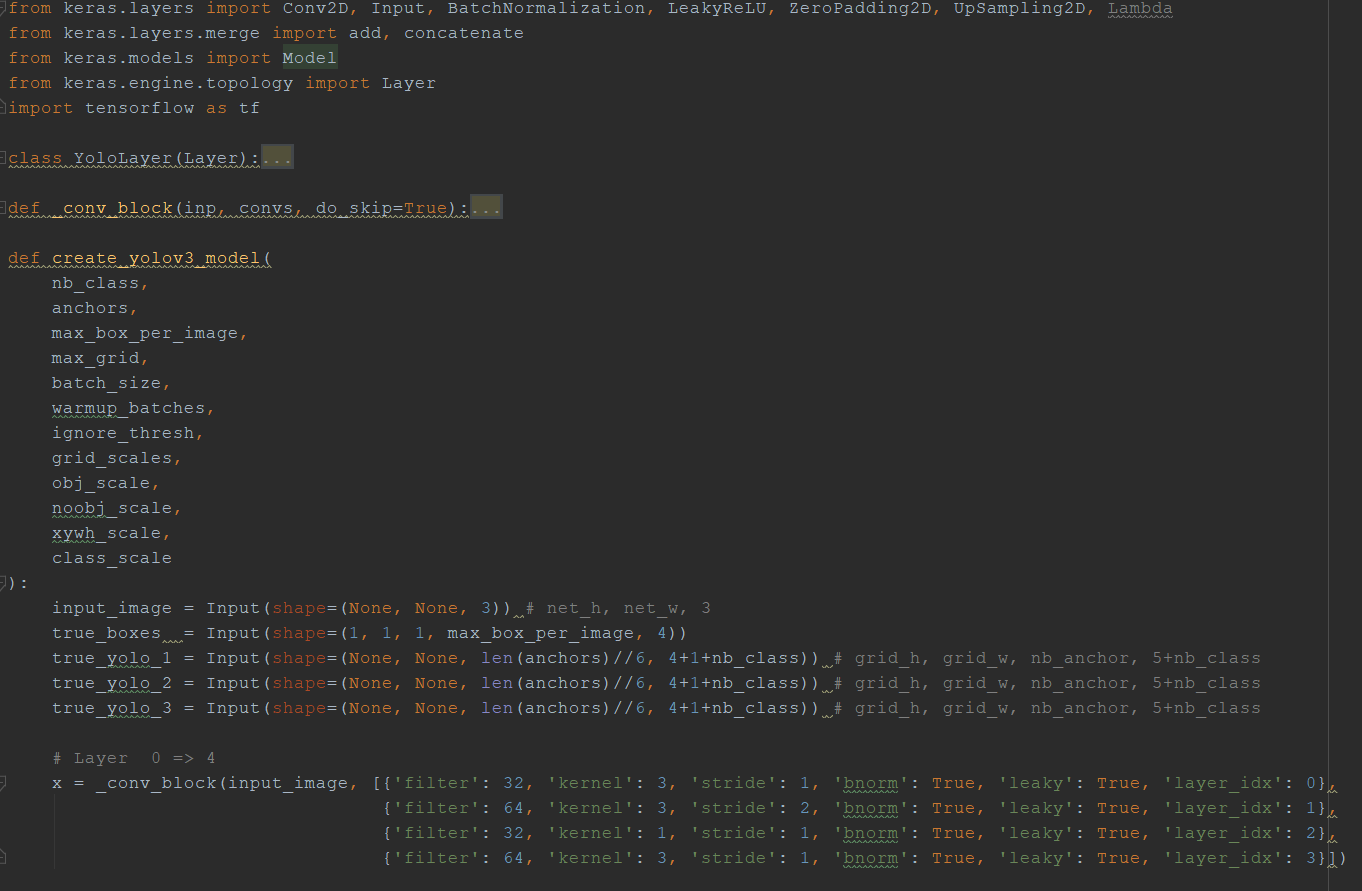
Since the given images are 1280\*720 a mask is applied to get the patches of the individual parking lot. These images are converted to a 448 \* 448 pixels image and its purpose is explained in the next phase. After getting the image we use various pre-processing available from the kears.preprocessing like ImageDataGenerator to do operations like, Rotation, width\_shift, height\_shift, shearing, zooming, horizontal\_flip, vertical\_flip etc., All these techniques are performed to make the model understand the features even if there is a change in the orientation of image during the testing phase.

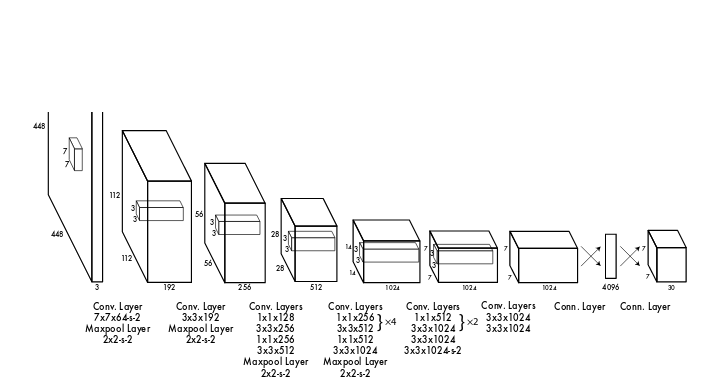
I have split trained and validation data set in 80:20 to verify my model. So this is the end of the data pre-processing phase next follows the model selection and training.

Model Decision and Training:

As this is problem of image classification hence CNN model need to be used. As there is lot of image processing needed hence I have selected Yolo v3 (You Only Look Once: Unified, Real-Time Object Detection) model which is been proven by researcher and Deep learning community a reasonable good model. It helps in providing image coordinates, label and score so that it’s easier to filter boxes from a given image. Also there are pre-trained weights(h5) already present so that processing on given dataset becomes easier and save time training the model.

Code snippet:



Yolo v3 Architecture

The above diagram is self-explanatory where multiple CNN layers sequentially added to reduce to 7 \* 7 \* 1024 layer. How actually YOLO works, now I will explain.

The following steps followed by YOLO for detecting objects in a given image.

1. YOLO first takes an input image
2. The framework then divides the input image into grids (say a 3 X 3 grid):
3. Image classification and localization are applied on each grid. YOLO then predicts the bounding boxes and their corresponding class probabilities for objects.
4. We will run both forward and backward propagation to train our model. During the testing phase, we pass an image to the model and run forward propagation until we get an output y.

Predict Car park Empty and Parked Slot using trained Model:

1] Code Snippet for NMS algorithm technique:

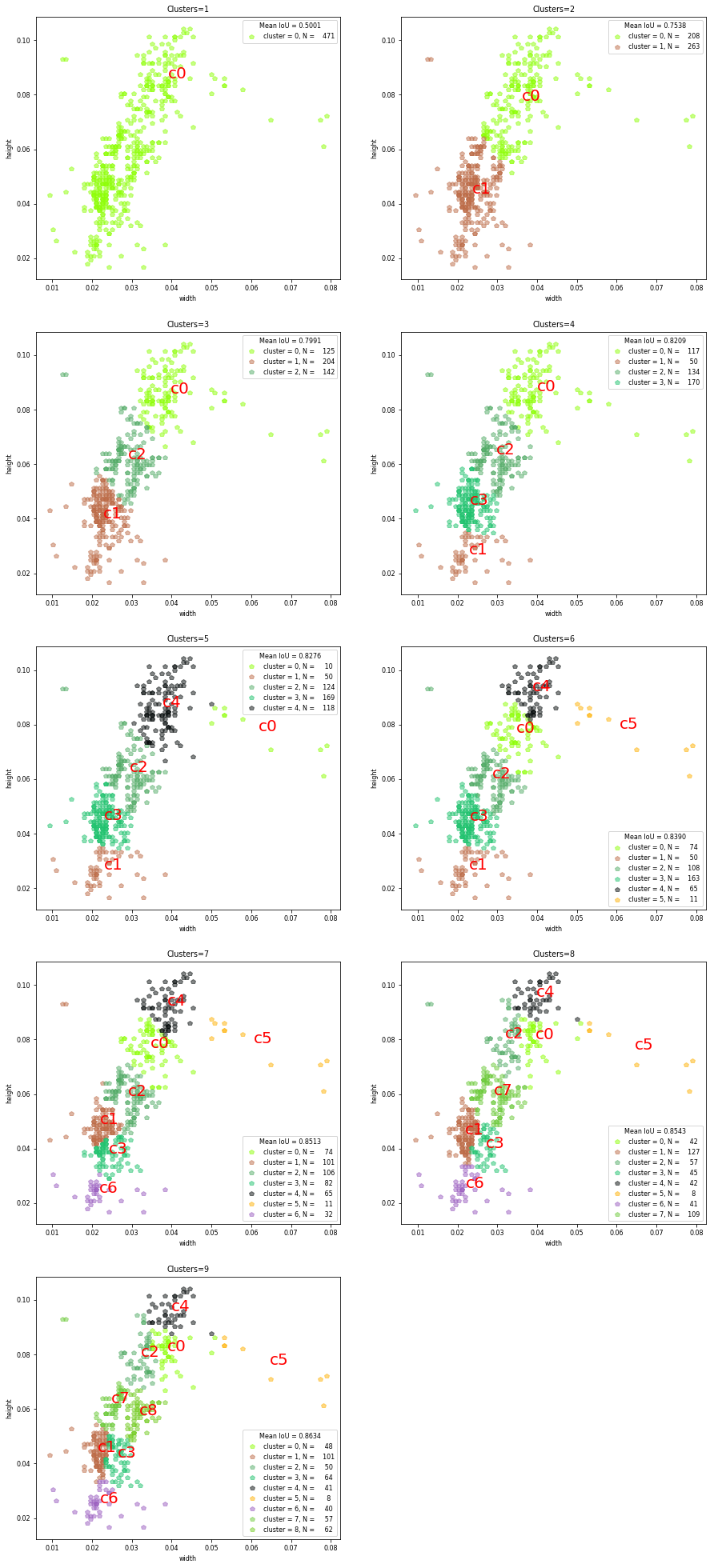
To find correct box from multiple boxes for a given object we used Non-Max suppression (NMS) algorithm technique:

1. Discard all the boxes having probabilities less than or equal to a pre-defined threshold (say, 0.5)
2. For the remaining boxes:
   1. Pick the box with the highest probability and take that as the output prediction
   2. Discard any other box which has IoU greater than the threshold with the output box from the above step
3. Repeat step 2 until all the boxes are either taken as the output prediction or discarded

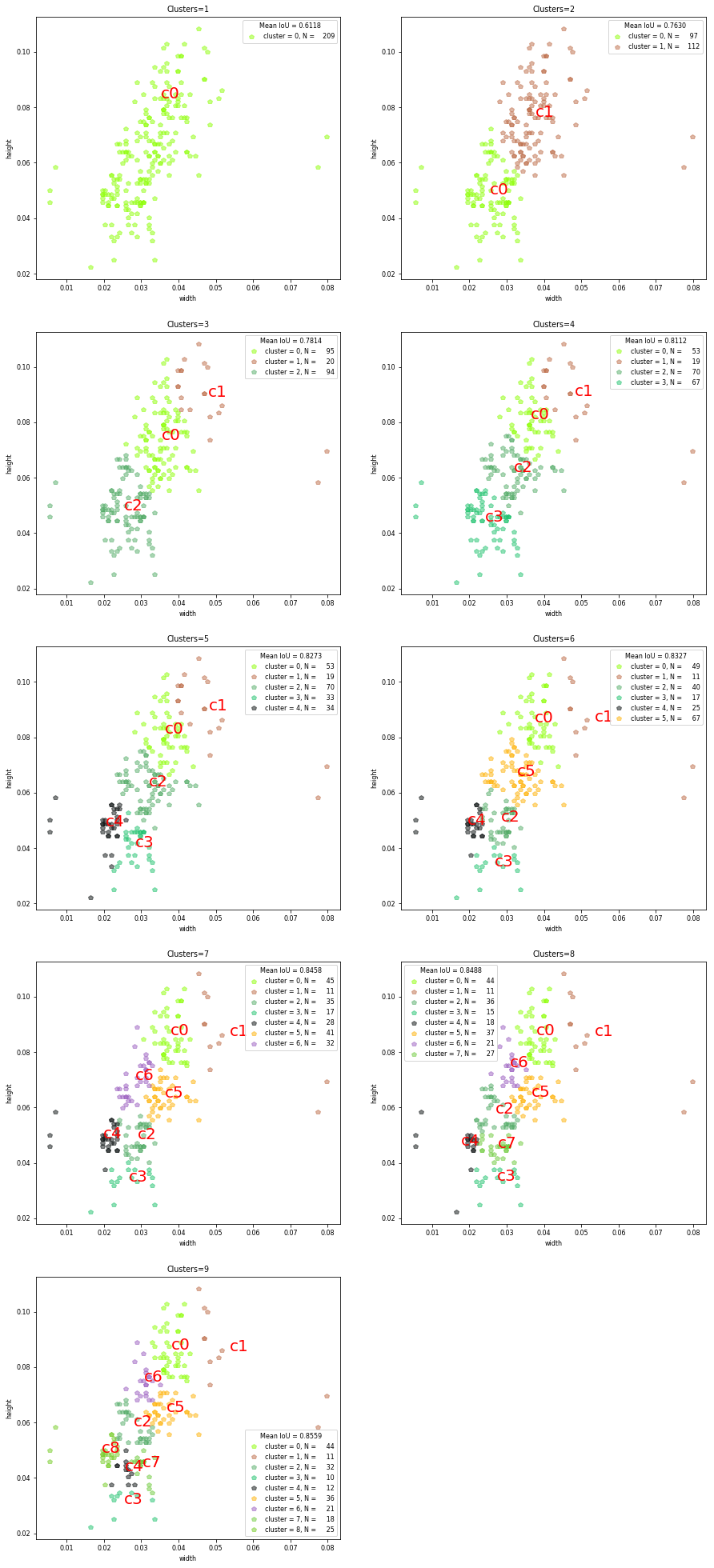
Normalised Fully Carpark and Inference Image Analysis

|  |  |
| --- | --- |
| C:\Users\tejas magia\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\1E70ED27.tmp | C:\Users\tejas magia\AppData\Local\Microsoft\Windows\INetCache\Content.MSO\F7496CF1.tmp |

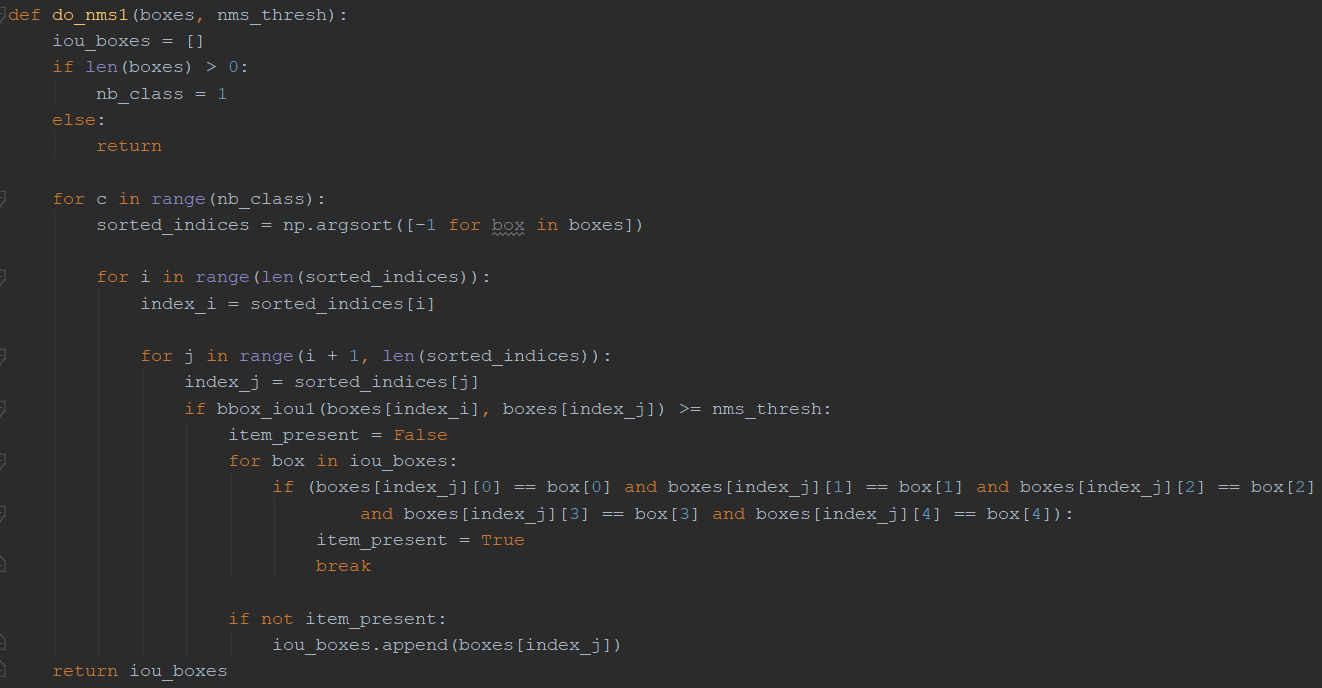
Fully CarPark IOU Clusters from 1 to 8 using K-Means

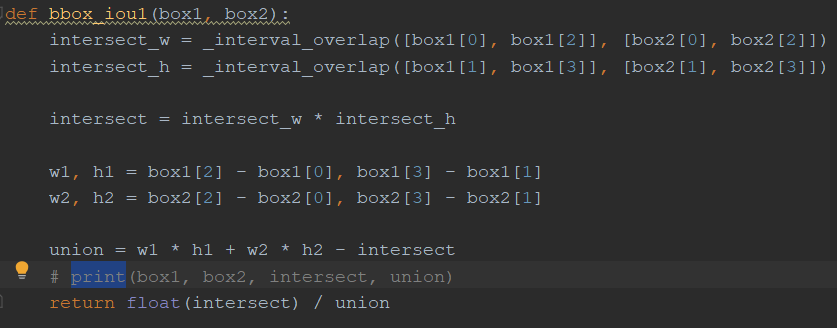


Predicted (Inference) Image IOU clusters using K-Means



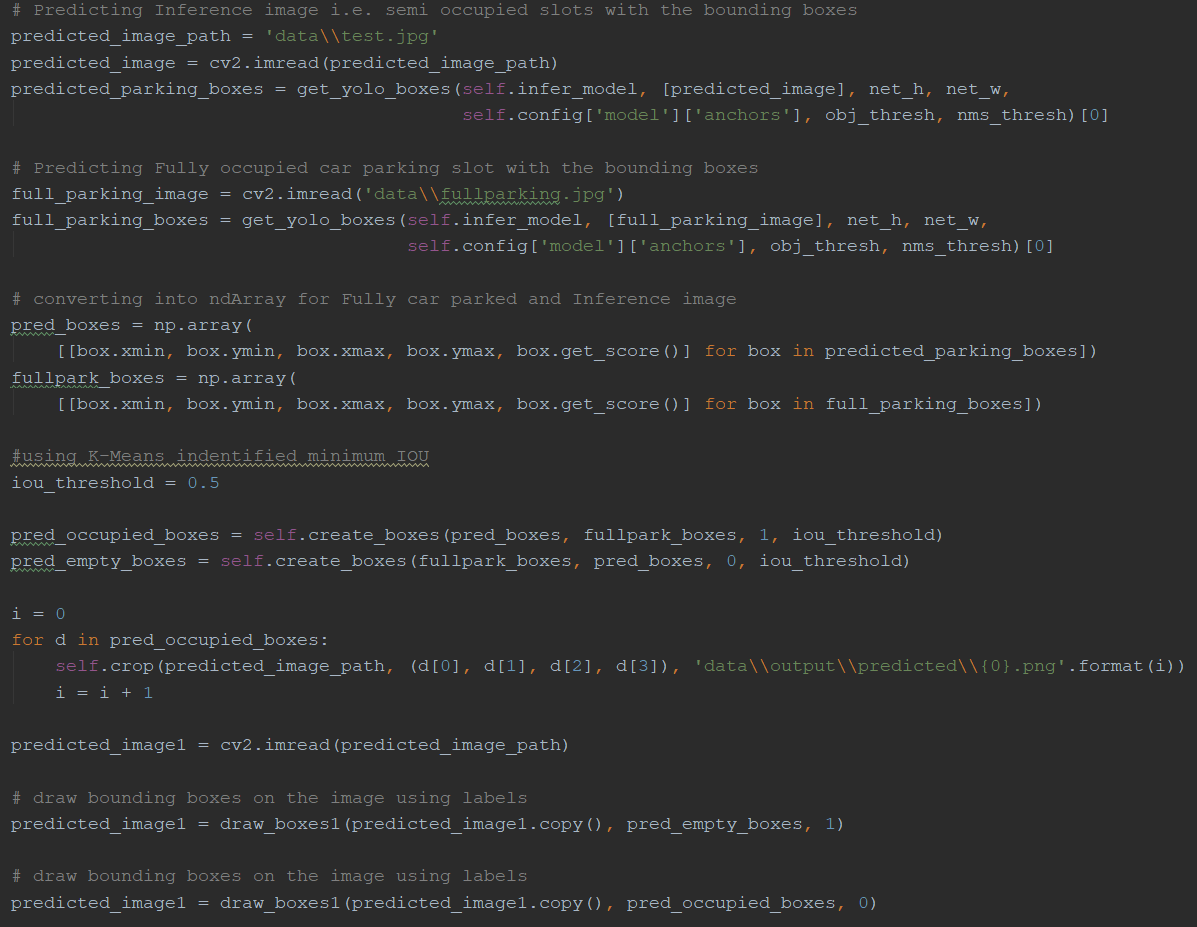
Using K-Means clustering for fully Carparked image IOU threshold identified as 0.5 and for Inference image IOU threshold was 0.6. This threshold values helps in reducing multiple overlapping images.





2] Code snippet for final Predicted Image Output:

* Trained my model using Cross entropy classifier and Sigmoid as the activation
* Using trained model collected all the slots for Fully car parked image and similarly predict occupied car slot for interference
* Using IOU (intersection over union) and NMS(non maximum suppression) able to find empty slots for inference image
* Min and max values of X,Y helps in finding the centroid
* Also, width and height helps in identify Pose of car parked



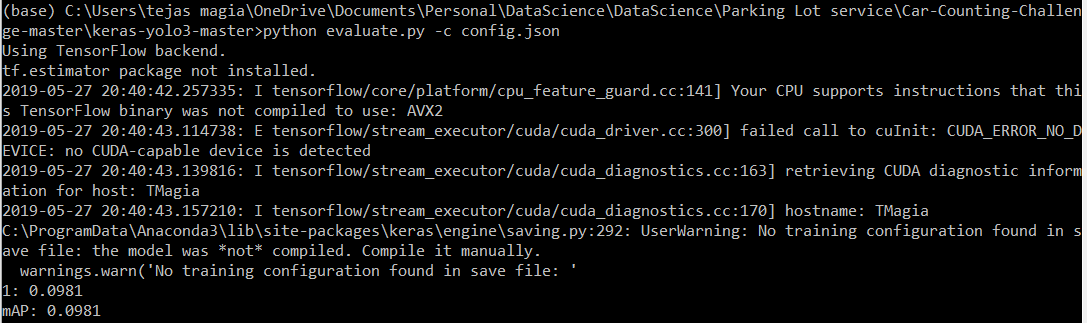
* The above unseen image passed to trained model where it shows Red colour boxes as car park slot is occupied and Orange colour for free parking slot.

|  |
| --- |
| Before |
|  |
| After |
|  |

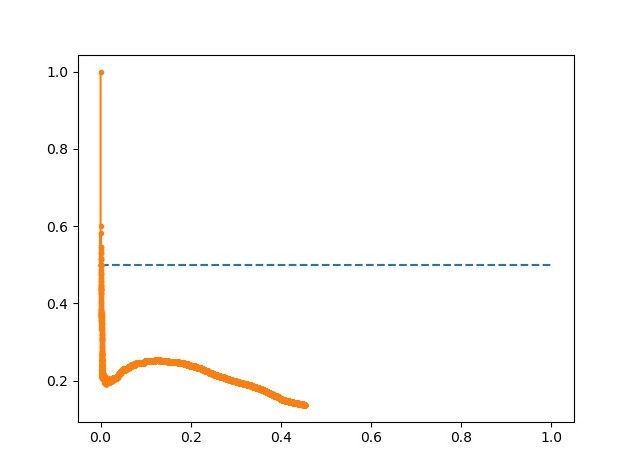
As the objects are small and model is not fully trained hence some places prediction is not correct. But if we able to trained the model for more time then we can able to achieve 100% accuracy for empty and parked car slot easily.

3] mean Average Precision(mAP) and IOC curve:

mAP score for Car label:



Precision and Recall Curve (IOC curve):



How to run the Submitted code:

**Training**

**1. Data preparation**

Download the weights and h5 dataset from <https://github.com/tejasmagia/CarParkingDetection_Contest>

Organize the dataset into 4 folders:

* train\_image\_folder <= the folder that contains the train images.
* train\_annot\_folder <= the folder that contains the train annotations in VOC format.
* valid\_image\_folder <= the folder that contains the validation images.
* valid\_annot\_folder <= the folder that contains the validation annotations in VOC format.

There is a one-to-one correspondence by file name between images and annotations. If the validation set is empty, the training set will be automatically split into the training set and validation set using the ratio of 0.8.

**2. Edit the configuration file**

The configuration file is a json file, which looks like this:

{  
 "model" : {  
 "min\_input\_size": 288,  
 "max\_input\_size": 448,  
 "anchors": [7,10, 9,17, 10,22, 12,32, 13,11, 14,24, 17,33, 17,41, 31,26],  
 "labels": ["1"]  
 },  
  
 "train": {  
 "train\_image\_folder": "..\\datasets\\PUCPR+\_devkit\\data\\Images\\",  
 "train\_annot\_folder": "..\\datasets\\PUCPR+\_devkit\\data\\Annotations\\",  
 "cache\_name": "car\_train.pkl",  
  
 "train\_times": 3,  
 "batch\_size": 32,  
 "learning\_rate": 1e-4,  
 "nb\_epochs": 10,  
 "warmup\_epochs": 3,  
 "ignore\_thresh": 0.6,  
 "gpus": "2,0",  
  
 "grid\_scales": [1,1,1],  
 "obj\_scale": 5,  
 "noobj\_scale": 1,  
 "xywh\_scale": 1,  
 "class\_scale": 1,  
  
 "tensorboard\_dir": "log\_car",  
 "saved\_weights\_name": "car.h5",  
 "debug": false  
 },  
  
 "valid": {  
 "valid\_image\_folder": "..\\datasets\\PUCPR+\_devkit\\data\\Images\\",  
 "valid\_annot\_folder": "..\\datasets\\PUCPR+\_devkit\\data\\Annotations\\",  
 "cache\_name": "car\_train.pkl",  
  
 "valid\_times": 1  
 }  
}

The labels setting lists the labels to be trained on. Only images, which has labels being listed, are fed to the network. The rest images are simply ignored. By this way, a car detector can easily be trained using VOC dataset by setting labels to ['1'].

Download pretrained weights for backend at:

<https://github.com/tejasmagia/CarParkingDetection_Contest>

**This weights must be put in the root folder of the repository i.e. ‘**keras-yolo3-master**’. They are the pretrained weights for the backend only and will be loaded during model creation. The code does not work without these weights.**

**3. Generate anchors for your dataset**

python gen\_anchors.py -c config.json

Copy the generated anchors printed on the terminal to the anchors setting in config.json.

**4. Start the training process**

python train.py -c config.json

By the end of this process, the code will write the weights of the best model to file best\_weights.h5 (or whatever name specified in the setting "saved\_weights\_name" in the config.json file). The training process stops when the loss on the validation set is not improved in 3 consecutive epoches.

**5. Perform detection using trained weights on image, set of images, video, or webcam**

Please download car.h5 from Github location which is trained model with accuracy of 93%.

<https://github.com/tejasmagia/CarParkingDetection_Contest/blob/master/car.h5>

python Model.py

Parameters :

'config.json','..\\data\\test.jpg','..\\data\\output\\'

It carries out detection on the image and write the image with detected bounding boxes to the same folder.

**Evaluation –** done against trained data only

python evaluate.py -c config.json

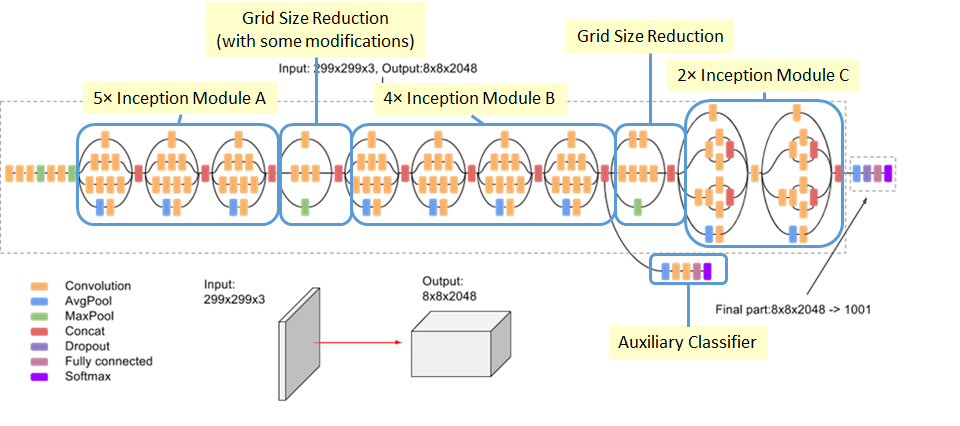
Compute the mAP performance of the model defined in saved\_weights\_name on the validation dataset defined in valid\_image\_folder and valid\_annot\_folder.

Further Improvement:

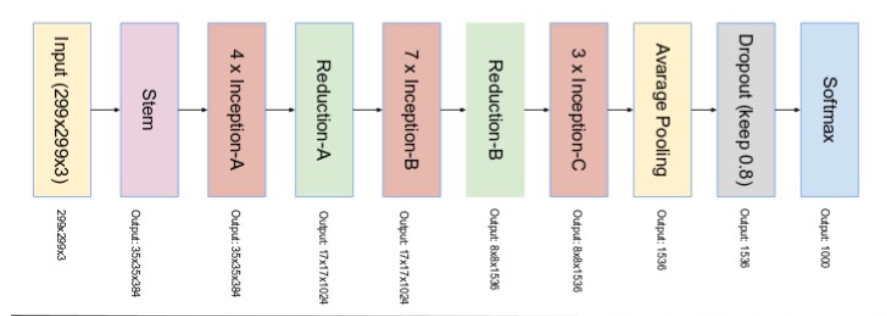
1. *Need to increase model accuracy because current model doesn’t have high accuracy.*
2. *Current model is only trained on one Car parking area which means if car parking areas increase then the model trained and accuracy may not be guaranteed hence change model from Yolo v3 to Inception-ResNet v4 & Inception v3 model for better performance and accuracy,*

Inception v3:

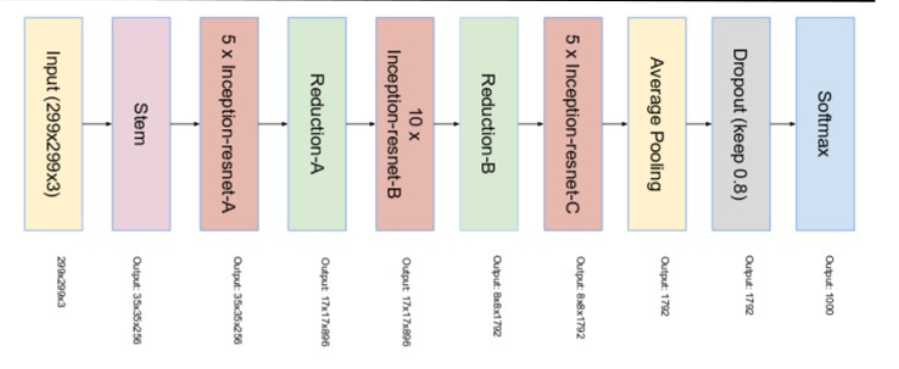
Inception Net v3 used RMSProp Optimizer, Factorized 7x7 convolutions, BatchNorm in the Auxillary Classifiers, and Label Smoothing (A type of regularizing component added to the loss formula that prevents the network from becoming too confident about a class. Prevents over fitting)



Inception v4:



*Inception-Resnet:*



1. *Need to implement same model for Webcam and video files*
2. *Implement Website/mobile/web service app which can show current parking status i.e. total number of empty and car parked count. Integration with Google map to provide this as a service would be faster and reasonable option from implementation point of view.*
3. *If car park area has multiple cameras then we can implement car park number plate detection and mapped exact location of each car parked*

Conclusion: I hereby conclude my report and I would like to thank JIO for giving such a real-world problem and helping me to gain intuition on how to apply theoretical knowledge to the practical world. I would like to thank Techgig as well for hosting such a beautiful contest.

Thank you

Tejas Magia

Team CG\_TEAM\_JVQRE

**Resources**

[1] Yolo v3 research paper : <https://arxiv.org/pdf/1804.02767.pdf>

[2] Training and detecting Objects using Yolo 3: <https://github.com/experiencor/keras-yolo3>

[3] Darkflow TFNet github : <https://github.com/thtrieu/darkflow>

[4] Darkflow TfNet Github example : <https://github.com/PrinceOfDorne/Recognition-of-Empty-Parking-Space>

[5] A simple guide to the versions of the inception network : <https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202>

[6] DataSet PUCPR : <https://lafi.github.io/LPN/>

[7] <https://www.analyticsvidhya.com/blog/2018/12/practical-guide-object-detection-yolo-framewor-python/>

[8] YOLO3 (Detection, Training, and Evaluation): <https://github.com/experiencor/keras-yolo3>