VISUAL RECOGNITION: MINI-PROJECT

TEAM MEMBERS:

PRATIK RATNADEEP AHIRRAO(IMT2019064)

PRATYUSH UPADHYAY(IMT2019066)

AYUSH TIWARI(IMT2019015)

Assignment 3a

Dataset provided was CIFAR10 which has 10 classes. Data was divided into 5 batches with each batch having 10,000 images.

Classification using AlexNet –

AlexNet has 5 convolution layers and 3 fully connected layers. The optimizer chosen was SGD with a momentum of 0.9–

The accuracy with AlexNet over the test dataset(5000 images) was 80.6%

The time consumed was 122 seconds for 10 epochs.

Choosing The Architecture–

- 4 convolutional layers and 3 fully connected layers Accuracy over the test dataset was 77.6% Time taken was 115 seconds for 10 epochs
- As the number of convolutional/ FC layers decreased the accuracy was either stagnant or decreased.
- Using 6 convolutional layers and 3 FC layers accuracy over the test dataset was 82.05% which was the best so far.
- The above code was run with **batch normalization** but upon running it **without batch normalization** it led to the accuracy dropping to 73%.

Now proceeding with this architecture of 6 cnv and 3 FC layers with batch normalization –

ReLu vs Tanh vs Sigmoid –

- ReLu performed the best with the best accuracy of 82.05%
- Tanh gave an accuracy of 76%
- Sigmoid did not do very well even after running for 30 epochs it gave an accuracy of 51%.

Adam vs SGD with momentum vs SGD with no momentum (6cnv and 3fc layers and 10 epochs)

- Adam optimizer-

  Takes 136 seconds

  With the best accuracy so far of 82%

- SGD without momentum-

  Takes 132 seconds

  Poor accuracy of 65.1%

- SGD with the momentum of 0.9 -

  Takes 130 seconds to execute

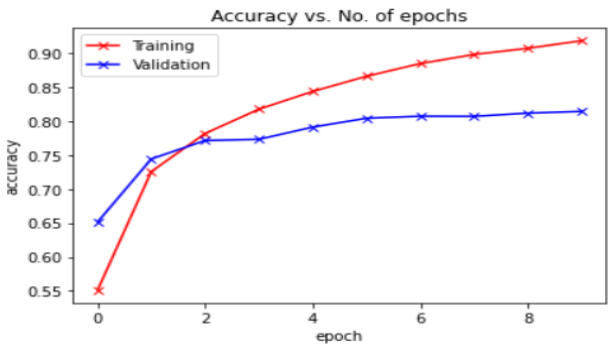
  accuracy of 79.7%

When momentum was decreasing the accuracy was also observed to be decreasing.

My recommended architecture -

6 convolutional layers 3 fully connected with Adam optimizer (also choosing relu over other activation functions and using batch normalization)- 82.05% accuracy execution time 133 seconds.

The plot of accuracy vs no of epochs in this case-



Assignment 3b

We have used the VGG16 pre-trained model as a feature extractor. It is trained on the ImageNet dataset which is the largest classification dataset. VGG16 is a convolution neural net (CNN) architecture that was used to win the Imagenet competition in 2014. It is considered to be one of the excellent vision model architecture till date. The most unique thing about VGG16 is that instead of having a large number of hyper-parameter they focused on having convolution layers of a 3x3 filter with a stride 1 and always used the same padding and maxpool layer of 2x2 filter of stride 2. It follows this arrangement of convolution and max pool layers consistently throughout the whole architecture. In the end, it has 2 FC(fully connected layers) followed by a softmax for output. The 16 in VGG16 refers to it having 16 layers that have weights. This network is a pretty large network and it has about 138 million (approx) parameters.

The features are extracted and are given as input to the different models, the accuracies on testset are recorded and are stated below for both the datasets (Cat vs Dog Classification problem) and (Horses vs Bike problem)80 % of the data is for training and the rest 20 % for testing. Here are the results

For Horse vs Bike Dataset:

Model	**Parameters**	**Accuracy**
Random Forest Classifier	n_estimators = 50, random_state = 42	100
SVM	C = 0.05 , kernel = ‘Linear’	97.33
Logistic Regression	Default parameters	94
Gaussian Naïve base	Default	100

For Dogs vs Cats Dataset(from Kaggle):

Model	**Parameters**	**Accuracy**

Random Forest Classifier	n_estimators = 100, random_state = 42	91
SVM	C = 0.5, kernel = ‘Linear’	93
SVM	C = 0.05, kernel =’Linear’	93
Logistic Regression	Solver = ‘liblinear’	94
Gaussian Naïve base	Default parameters	74

Ipython notebook for both of them are included in the zip folder.

AUTO DETECTION – 3c

Creating the Dataset:

Annotating the dataset:

- At first, we need to annotate the images of the given dataset. I have used the LabelImg annotation tool to do so.
- While annotating the images, we label the autos in each image as “Auto”.

Using Roboflow:

- We have used roboflow to store and organize the annotated dataset.
- It helps us process and augment images to help our model train faster and generalize better.
- After uploading the annotated dataset, we split the dataset into training (70%), validation (20%), and testing set (10%).
- In preprocessing part, we have enabled auto-orient and resized every image to 416 x 416.
- In the augmentation part, we can flip the images horizontal or vertical. We can also rotate the images 90° clockwise, anti-clockwise, or upside-down. We can

also increase or decrease the brightness of the image by 25%.

- We then finally generate the dataset in my roboflow account.

YOLOv5:

Training time for YOLOv5 is very less compared to that of YOLOv4.

- To start off with YOLOv5 we first clone the YOLOv5 repository and install the required dependencies.
- We have used google colab's GPU to make the execution faster.
- We then install roboflow in our notebook and import the uploaded dataset from our roboflow account.
- We also import the YAML file. It is used for configuration and sometimes it is used for data storage and debugging.
- We then train yolov5 (running *train.py* file) on our training data for 150 epochs. We have got better results with yolov5l. We are also caching images for faster training.

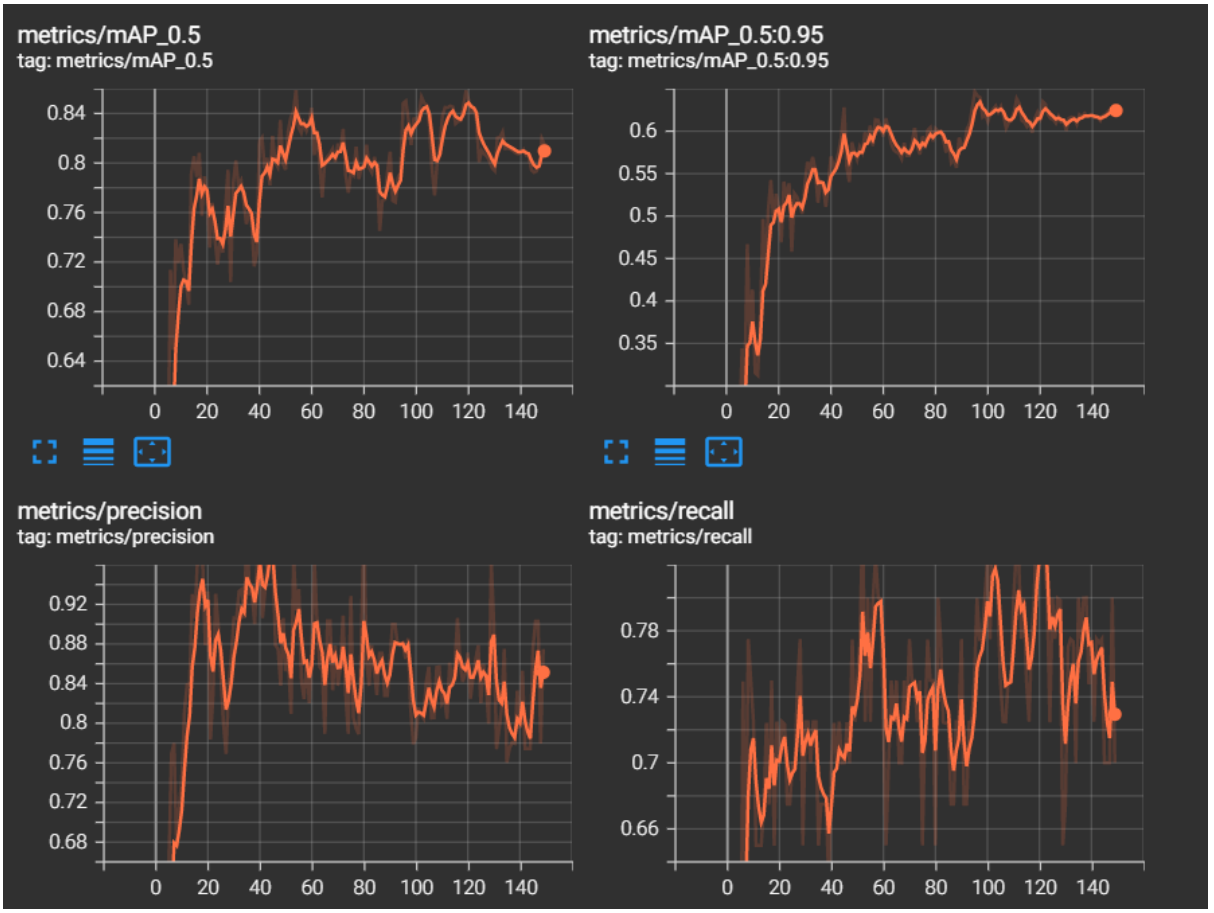
  Here is the model summary:

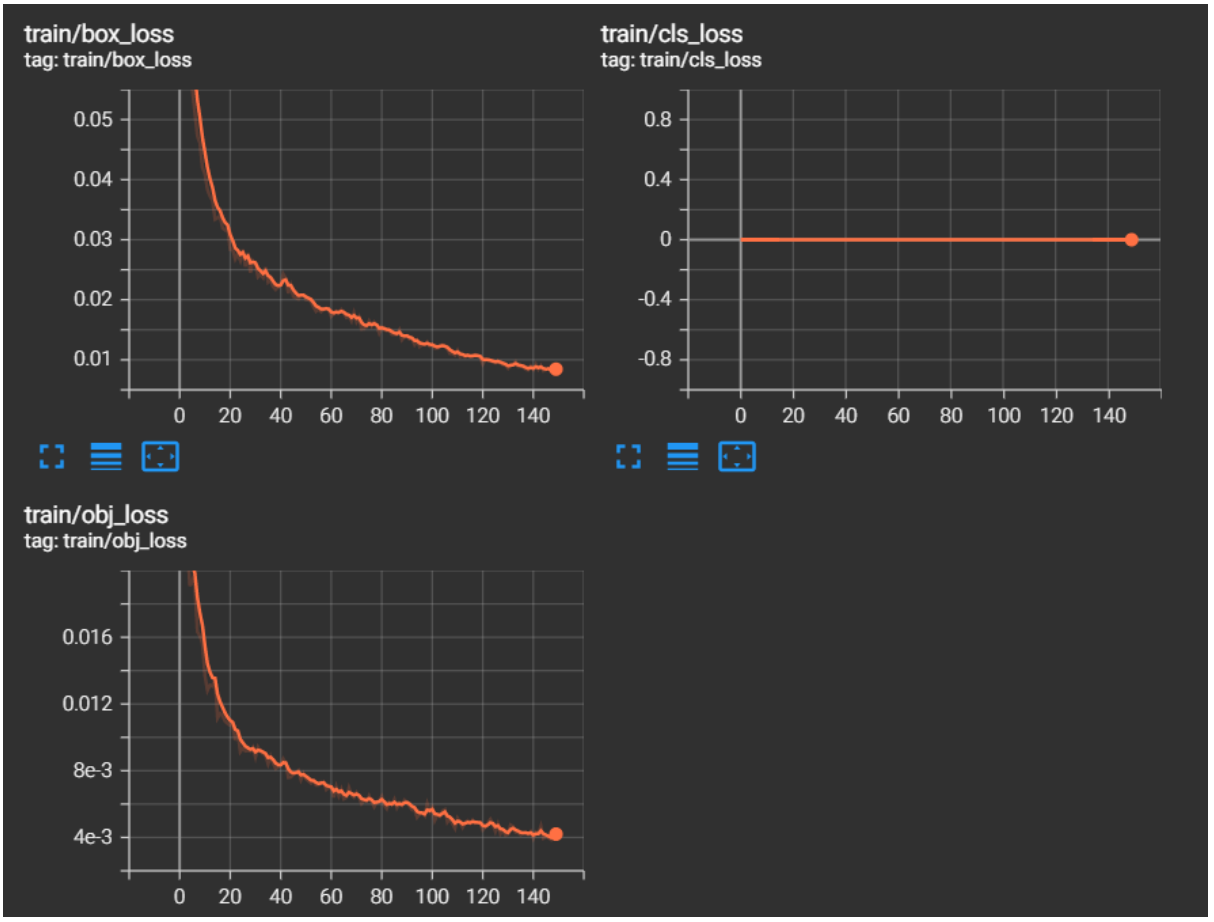
Class	Images	Labels	P	R	mAP@.5	mAP@.5:.95
All	30	40	0.909	0.75	0.849	0.647

  mAP stands for mean absolute Precision.It compares the ground-truth bounding box to the detected box and returns a score. The higher the score, the more accurate the model is in its detections.

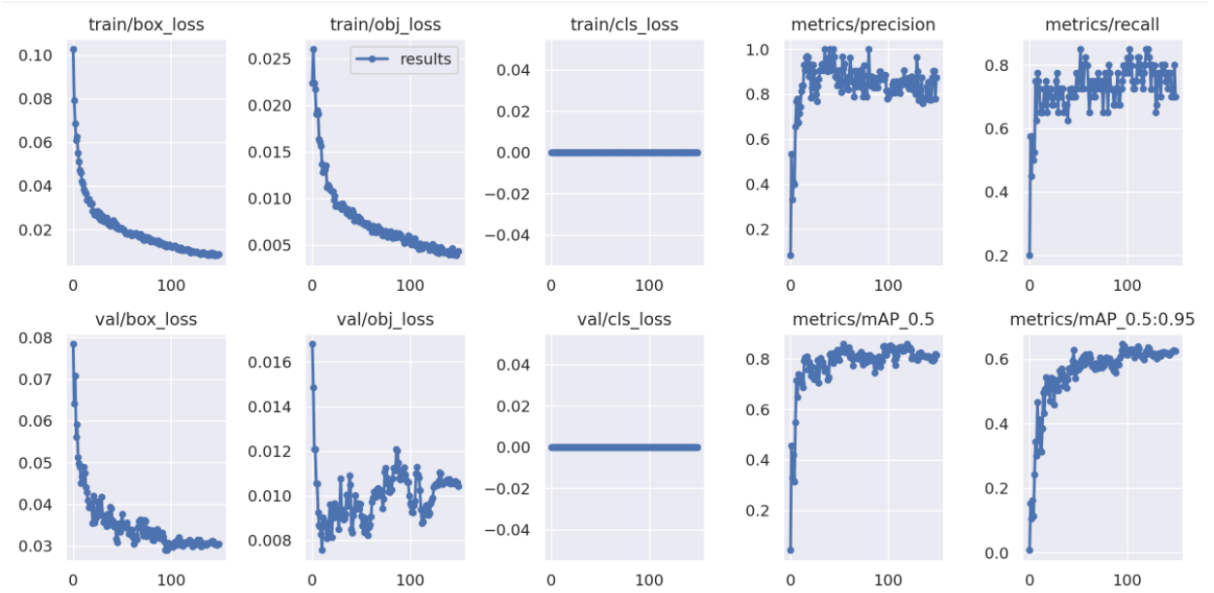
  mAP scores can change a bit if we run the notebook again.

- Now that we have completed training, we now evaluate how well the training procedure was performed by looking at validation metrics. The training script will drop tensorboard *logs* in runs.

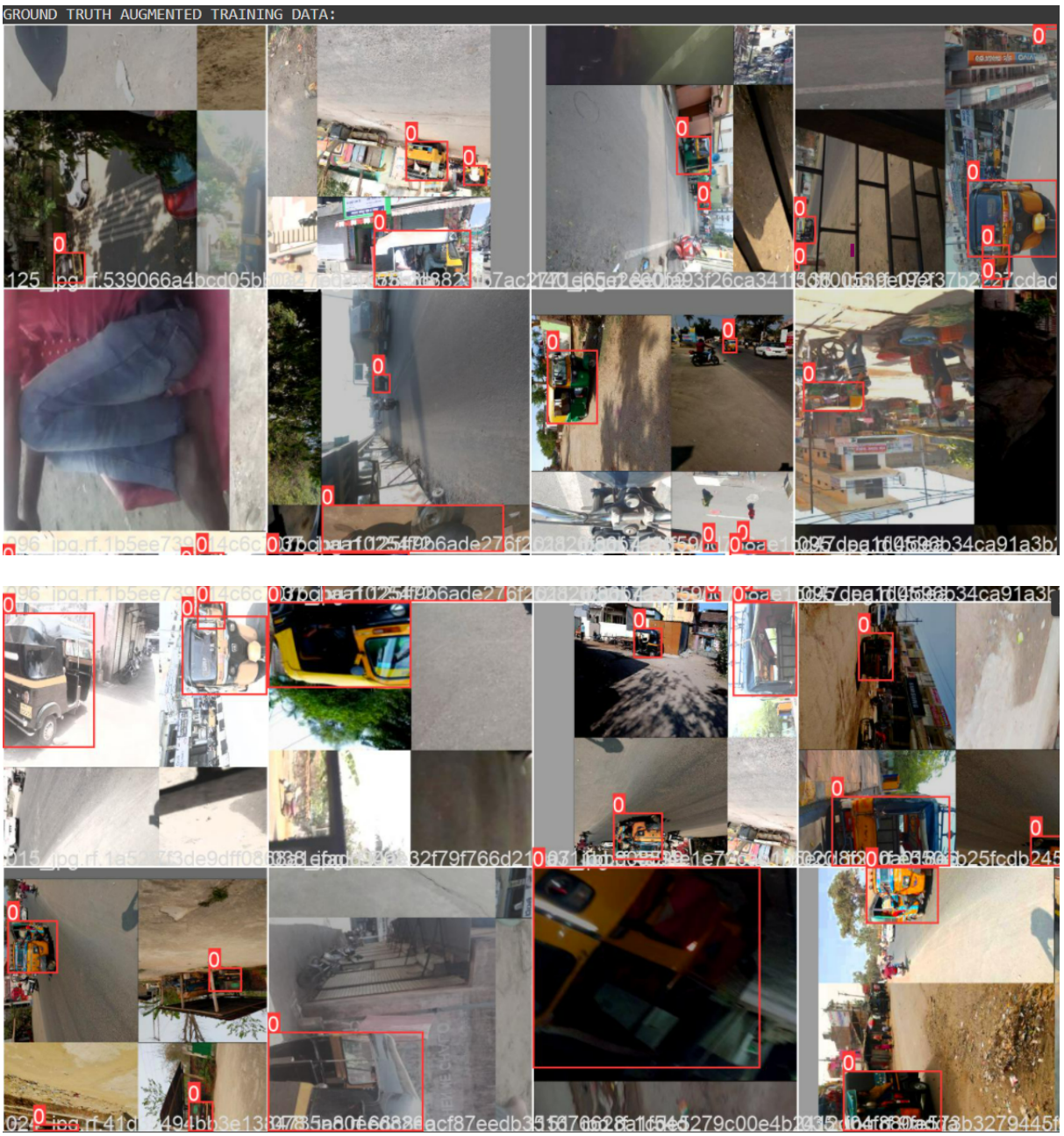

metrics/mAP_0.5
tag: metrics/mAP_0.5
0.84
0.8
0.76
0.72
0.68
0.64
0 20 40 60 80 100 120 140
metrics/mAP_0.5:0.95
tag: metrics/mAP_0.5:0.95
0.6
0.55
0.5
0.45
0.4
0.35
0 20 40 60 80 100 120 140
metrics/precision
tag: metrics/precision
0.92
0.88
0.84
0.8
0.76
0.72
0.68
0 20 40 60 80 100 120 140
metrics/recall
tag: metrics/recall
0.78
0.74
0.7
0.66
0 20 40 60 80 100 120 140



- Results can also be plotted with *utils.plot_results*. Here are the plots obtained:



- We also print the augmented training set batch.



- Now we take our trained model and make inferences on test images. After training has been completed model weights will be saved in *weights/*. For inference, we invoke those weights along with *conf* specifying model confidence, and an inference *source*. Here the *source* is the directory of test images.

  We now test our model (running *test.py* file) on our testing Images (a total of 14 images).

  Here are the images obtained:

1.

Auto has been detected correctly here with an accuracy of 0.92

2.

One auto has been detected correctly with an accuracy of 0.93 but fails to detect the other auto.

3.

Auto has been detected correctly here with an accuracy of 0.95

4.

Here the model is unable to detect the auto, also the auto is not clearly visible.

5.

Auto has been detected with an accuracy of 0.60

6.

Auto has been detected correctly here with an accuracy of 0.93. But the model also detects the car as auto with an accuracy of 0.70

7.

Auto has been detected correctly here with an accuracy of 0.93

8.

Both autos have been detected correctly here with accuracies of 0.86 and 0.94 respectively.

9.

Auto has been detected with an accuracy of 0.71

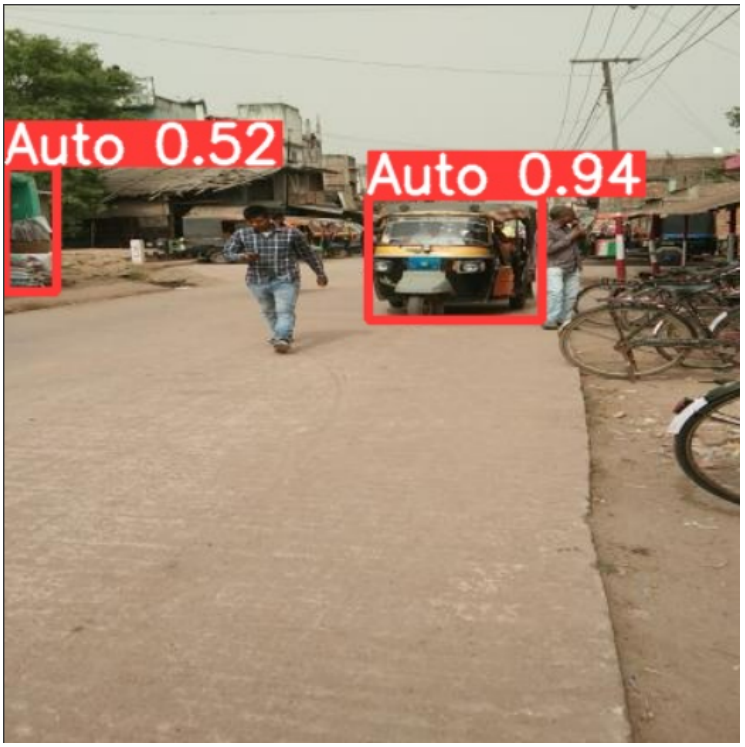
10.

Auto has been detected correctly here with an accuracy of 0.93

11.

Auto has been detected correctly here with an accuracy of 0.93

12.



Auto has been detected correctly here with an accuracy of 0.94.

It also detects some part of the image which is clearly not auto as “auto” with an accuracy of 0.52

13.

Auto has been detected correctly here with an accuracy of 0.93

14.

Auto has been detected correctly here with an accuracy of 0.85