Name(s) : Badrinarayanan Rajasekharan (br17), Rini Jasmine Gladstone (rjg7)
Kaggle Team Name: PaayumPuli

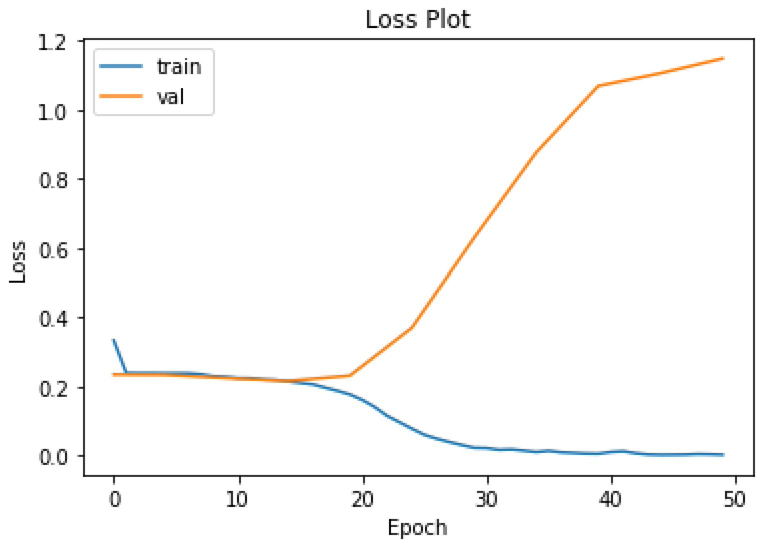
Part-1A: Pre-designed network for multi-label classification

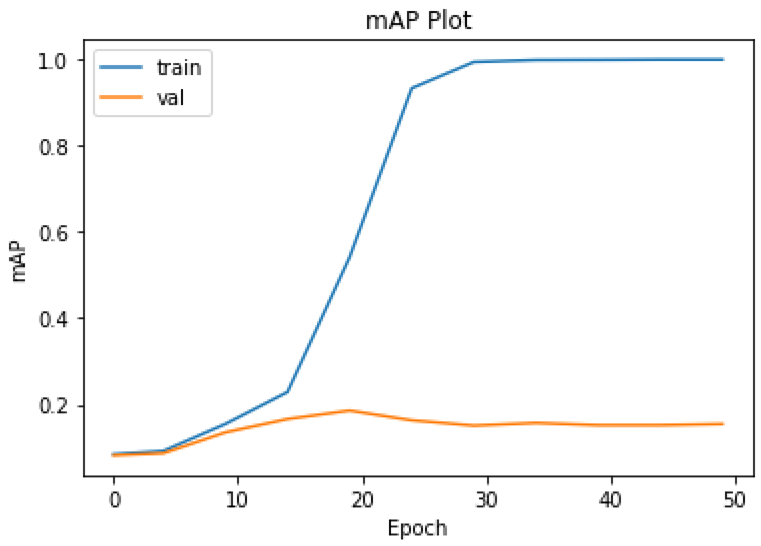
In this part, you will practice to train a neural network both by training from scratch or fine-tuning.

MP3_P1_Introduction.ipynb in your assignment3_p1_starterkit should provide you with enough instruction to start with.

We are asking you to provide the following results.

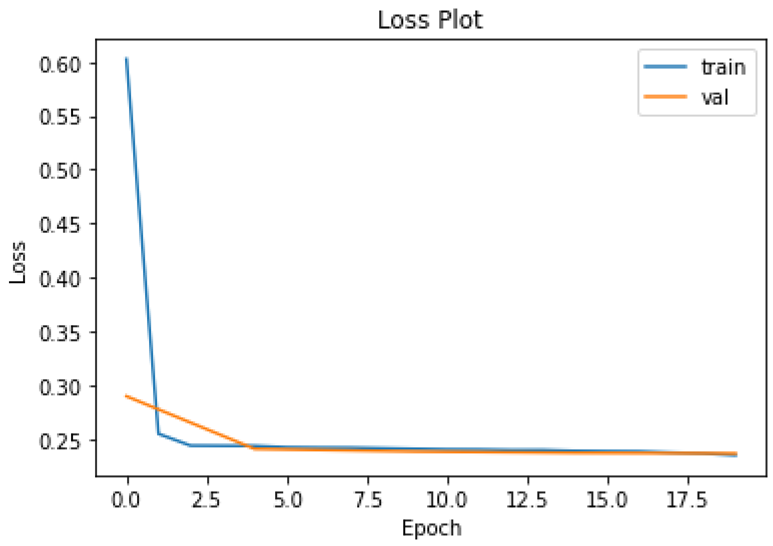
1. Simple Classifier
    a. Test mAP for simple classifier: 0.1518
    b. Visualize loss and mAP plots:

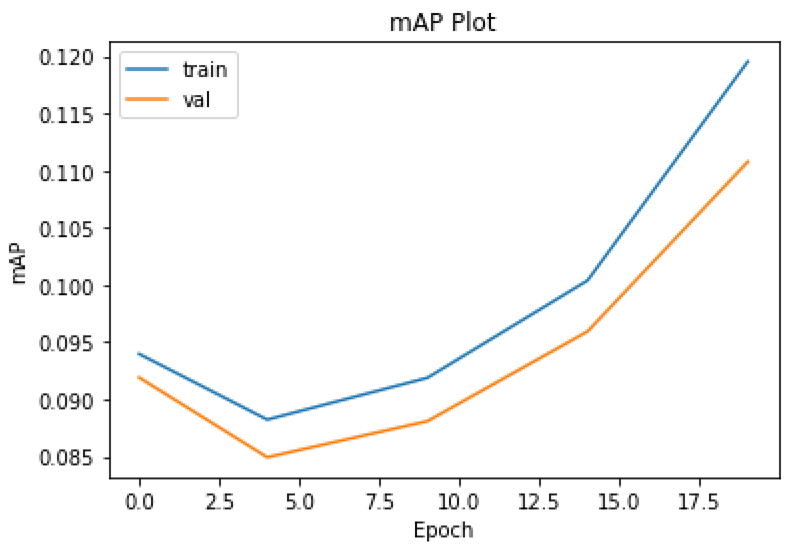




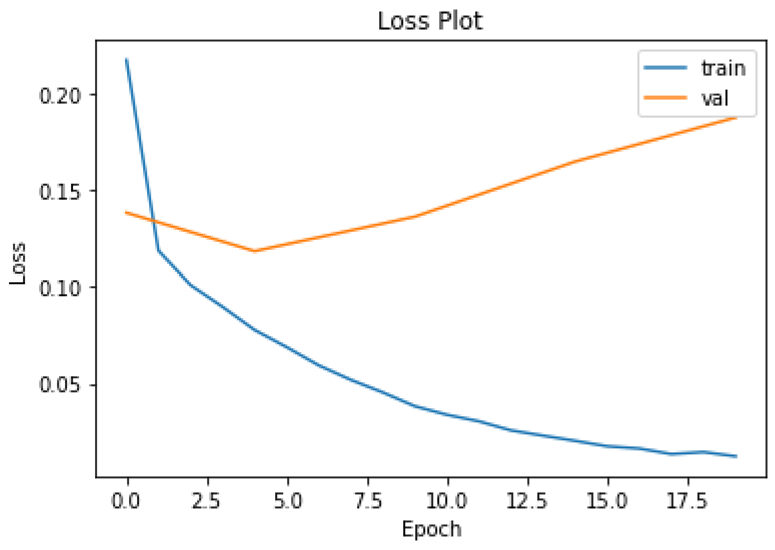
c. Provide analysis : We observe an increase in the training MAP after around 15 iterations, however, the validation MAP remains constant after that. This could be due to an overfitting of the model. The model could be improved by adding regularization parameters like weight decay (in case of learning rate) to avoid this scenario. In addition to that, adding more convolutional and fully connected layers and adding momentum to SGD or Adam optimizer could improve its performance.

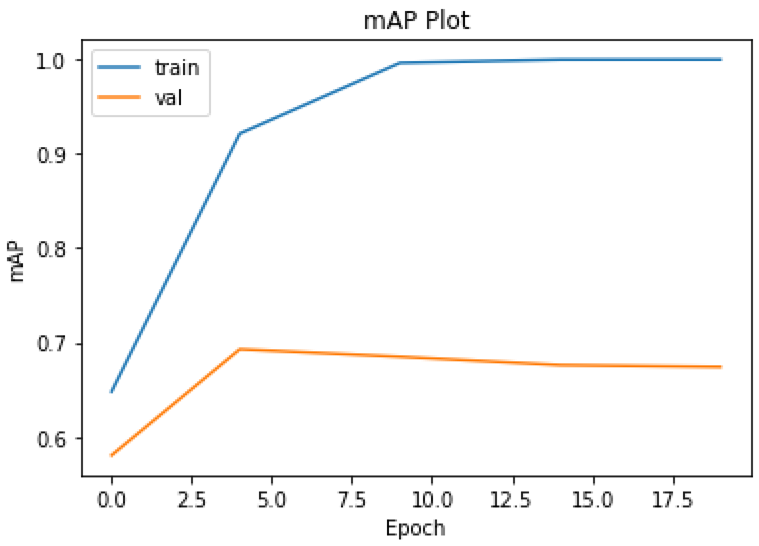
2. AlexNet from Scratch
    a. Report test mAP for alexnet: 0.1057
    b. Visualize loss and mAP plots:





3. Pretrained AlexNet
    a. Report test mAP for pretrained alexnet: 0.6905
    b. Visualize loss and mAP plots





c. Provide analysis on differences to training from scratch (at least 3 sentences):

We see a significant increase in MAP for pretrained AlexNet compared to the AlexNet trained from scratch. This is logical as pretrained Network has seen large amount of training data and has already been trained with optimal hyper parameters for large number of epochs. Since it is pretrained, it gives an mAP of 0.55 from the first epoch and increases further till ~0.7 for validation dataset. When Alexnet is trained from scratch, the model starts with an maP of 0.09 and increases only till 0.12.

Although, pretrained Alexnet gives good map, it suffers from significant overfitting as compared to the one trained from scratch

Part-1B: Self designed network for multi-label classification

MP3_P1_Develop_Classifier in your assignment3_p1_starterkit should provide you with enough instruction to start with. You upload your output of your self-designed network to kaggle.

Did you upload final CSV file on Kaggle: **Yes**

1. My best mAP on Kaggle: 0.40
2. Factors which helped improve my model
   a. Data augmentation by RandomResizedCrop and HorizontalFlip transform
   b. Adding momentum to the SGD, and Step learning rate
   c. Increasing the depth of the network by adding more convolutional and fully connected layers
   d. Adding batch normalization between every layer
   e. Adding dropout after third and fifth fully connected layers
   f. Adding residual connection from the first fully connected layer after the fourth fully connected layer
   g. Increasing the number of epochs from 20 to 150

3. Table for final architecture (replace below with your best architecture design):

Layer No.	Layer Type	Kernel size (for conv layers)	Input | Output dimension	Input | Output Channels (for conv layers)
1	conv2d	7	227 | 221	3 | 64
2	batchnorm1d	-	221 | 221	64
3	relu	-	221 | 221	-
4	maxpool2d	2	221 | 110	-
5	conv2d	7	110 | 104	64|128
6	batchnorm1d	-	104 | 104	128
7	relu	-	104 | 104	-
8	maxpool2d	2	104 | 52	-
9	conv2d	3	52 | 50	128|128
10	batchnorm1d	-	50 | 50	128
11	relu	-	50 | 50	-
12	maxpool2d	2	50 | 25	-
13	conv2d	3	25 | 23	128|256
14	batchnorm1d	-	23 | 23	256
15	relu	-	23 | 23	-
16	maxpool2d	2	23 | 11	-
17	conv2d	3	11 | 9	256|256
18	batchnorm1d	-	9 | 9	256
19	relu	-	9 | 9	-
20	maxpool2d	2	9 | 4	-

21	conv2d	1	4 | 4	256|256
22	batchnorm1d	-	4 | 4	256
23	relu	-	4 | 4	-
24	linear	-	4096 | 150	4096 | 150
25	batchnorm1d	-	150 | 150	-
26	relu	-	150 | 150	-
27	linear	-	150 | 120	150 | 120
28	batchnorm1d	-	120 | 120	-
29	relu	-	120 | 120	-
30	linear		120 | 120	120 |120
31	batchnorm1d	-	120 | 120	-
32	relu	-	120 | 120	-
33	dropout(0.2)	-	120 | 120	-
34	linear	-	120 | 150	120 | 150
35	batchnorm1d	-	150 | 150	-
36	relu	-	150 | 150	-
37	residual		150 | 150	-
38	linear	-	150 | 60	150 | 60
39	batchnorm1d	-	60 | 60	-
40	relu	-	60 | 60	-
41	dropout (0.2)	-	60 | 60	
42	linear	-	60 | 21	60 | 21

The initial network provided to you can be considered as the BaseNet. A very important part of deep learning is understanding the ablation studies of various networks. So we would like you to do a few experiments. Note, this **doesn't need to be very exhaustive** and can be in a cumulative manner in an order you might prefer. Fill in the following table :

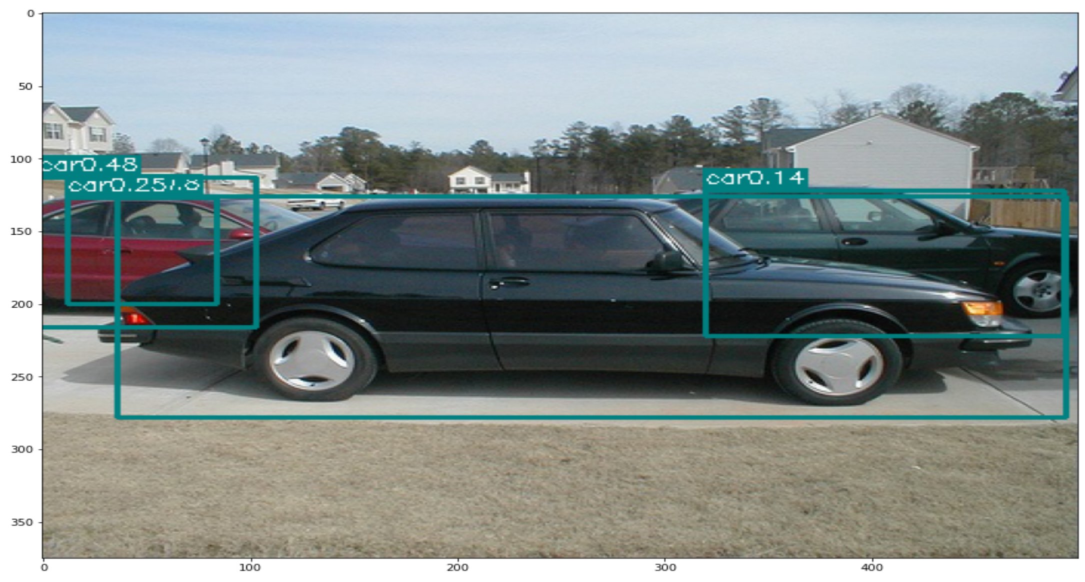
Serial #	Model architecture	Best mAP on test set
1	BaseNet	0.15
2	BaseNet + a + b	0.18
3	BaseNet + a + b + c + d	0.25
4	BaseNet + a + b + c + d + e	0.28
5	BaseNet + a + b + c + d + e + f	0.33
6	BaseNet + a + b + c + d + e + f + g	0.40

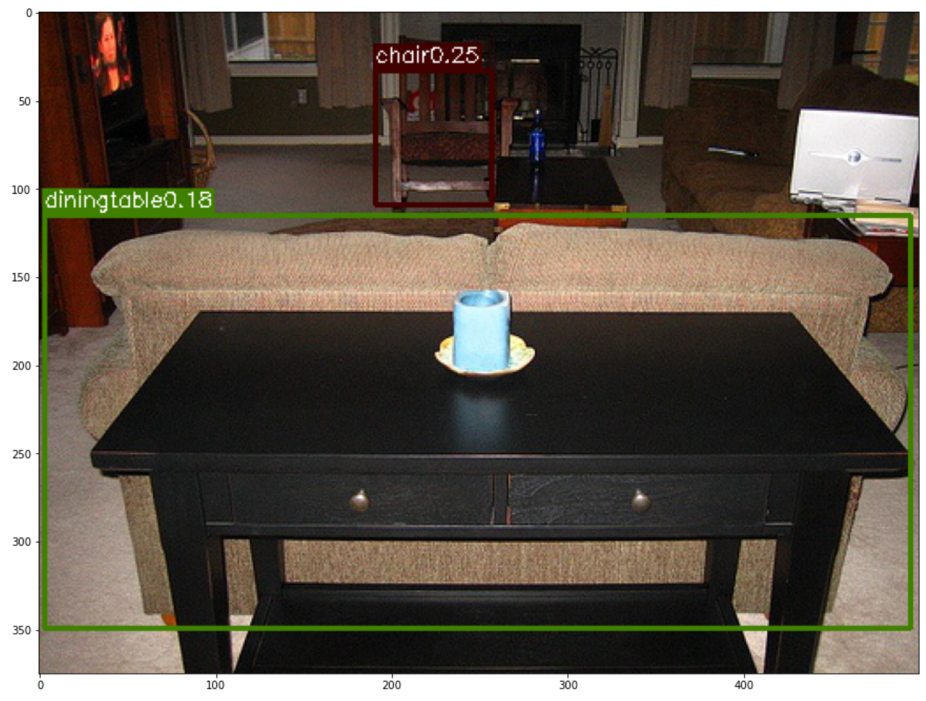
Make some analysis on why and how you think certain changes helped or didn't help:

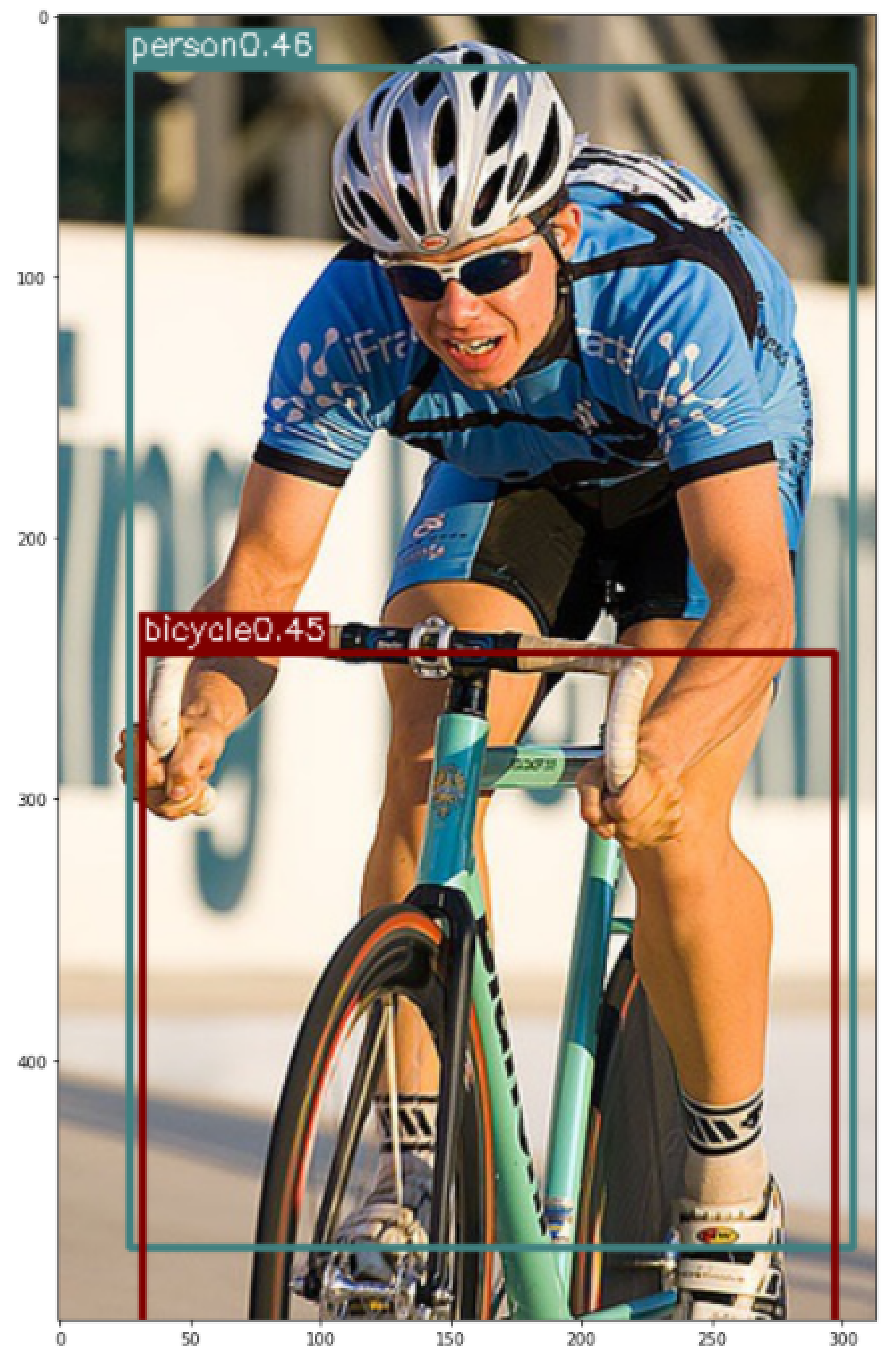
1) Adding data augmentation expanded the training set with model able to see different variations of the same picture. This helps with better object detection even if the images are distorted.
2) Adding momentum to SGD accelerated the movement of the optimizer in the relevant direction. This, along with Step learning rate significantly improved the performance of the model
3) Increase the depth of the model helped to learn the features better enabling the classification of objects much more accurately
4) Adding dropout layers prevented overfitting of the model making it more robust.
5) Adding residual network helped avoid the problem of vanishing gradients and as a results helped in training the model better
6) Adam optimizer didn't give as good results as SGD.

1) Part-2: Objection Detection by YOLO

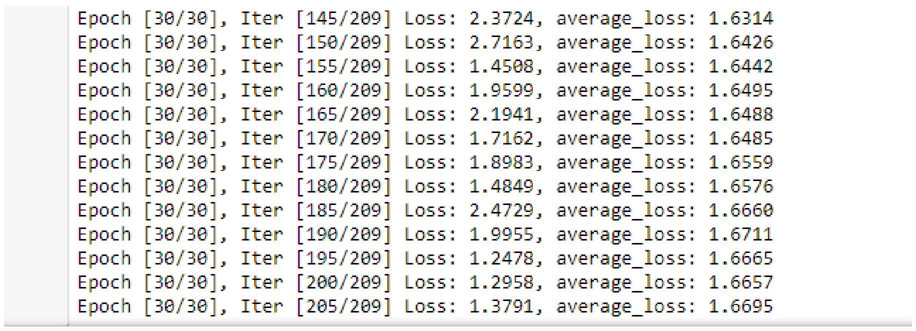
1. My best mAP value on Kaggle : 0.49974 (1-mAP)
2. Did you upload final CSV file on Kaggle: Yes
3. My final loss value : 1.6695
4. What did not work in my code(if anything):
5. Sample Images from my detector from PASCAL VOC:






person0.46
bicycle0.45
0
100
200
300
400
0
50
100
150
200
250
300

Screenshot of the last few iterations:



```
Epoch [30/30], Iter [145/209] Loss: 2.3724, average_loss: 1.6314
Epoch [30/30], Iter [150/209] Loss: 2.7163, average_loss: 1.6426
Epoch [30/30], Iter [155/209] Loss: 1.4508, average_loss: 1.6442
Epoch [30/30], Iter [160/209] Loss: 1.9599, average_loss: 1.6495
Epoch [30/30], Iter [165/209] Loss: 2.1941, average_loss: 1.6488
Epoch [30/30], Iter [170/209] Loss: 1.7162, average_loss: 1.6485
Epoch [30/30], Iter [175/209] Loss: 1.8983, average_loss: 1.6559
Epoch [30/30], Iter [180/209] Loss: 1.4849, average_loss: 1.6576
Epoch [30/30], Iter [185/209] Loss: 2.4729, average_loss: 1.6660
Epoch [30/30], Iter [190/209] Loss: 1.9955, average_loss: 1.6711
Epoch [30/30], Iter [195/209] Loss: 1.2478, average_loss: 1.6665
Epoch [30/30], Iter [200/209] Loss: 1.2958, average_loss: 1.6657
Epoch [30/30], Iter [205/209] Loss: 1.3791, average_loss: 1.6695
```