Name(s) : Punit K. Jha
Kaggle Team Name: PunitKJha

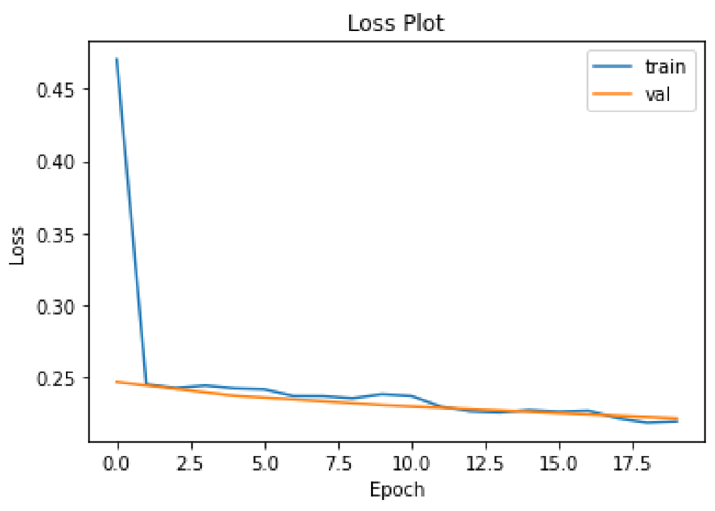
Part-1A: Pre-designed network for multi-label classification

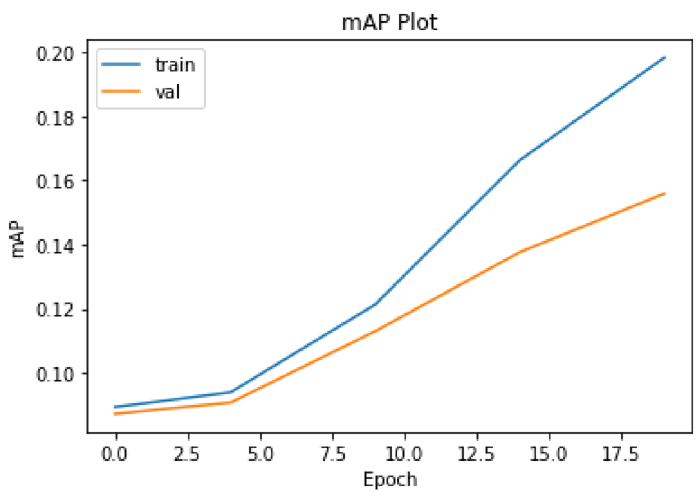
In this part, you will practice to train a neural network both by training from scratch or fine-tuning. **MP3_P1_Introduction.ipynb** in your assignment3_p1_starterkit should provide you with enough instruction to start with.
We are asking you to provide the following results.

1. **Simple Classifier**
   a. **Report test mAP for simple classifier:**
   Solution:
   - The test mAP for the simple classifier provided in the classifier.py file using the SGD with momentum method of optimization and 20 epochs was **0.1468** whereas the average loss on the test set was **0.2184.**
   - These values are for the default parameters provided in the jupyter notebook. However, I went on to fine tune the parameters with learning rates=[0.01,0.001,0.0001] and increased the number of epochs to 30 and 40. The best results were observed for learning rate of 0.01 with the number of epochs=40. The test mAP for learning rate= **0.1856** whereas the average loss on the test set was **0.216**.
   - I also observed that with the Adam optimizer with learning rate of 1e-4 along with the betas set to (0.9, 0.999), and the epsilon values=1e-08 the test mAP for the simple classifier increased to **0.2079** and the average loss is **0.2119** .

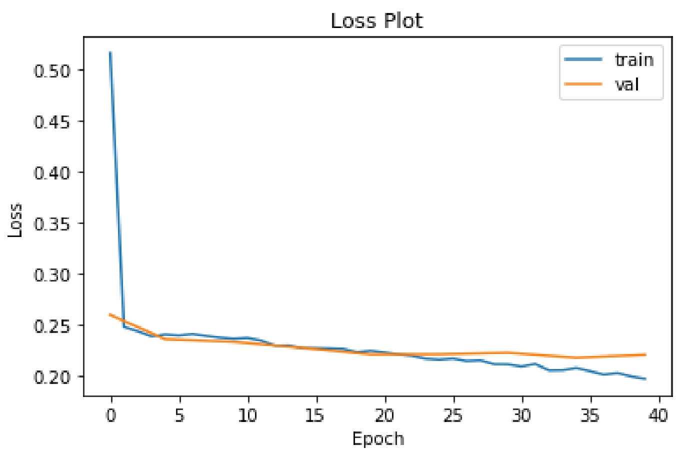
   b. **Visualize loss and mAP plots:**
   Solution:

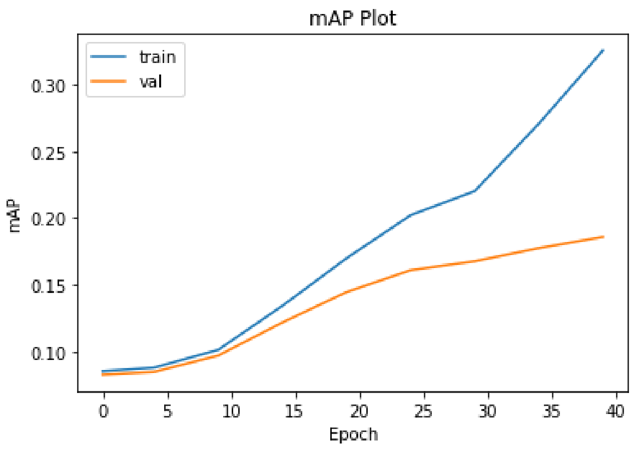
Shown below is the loss plot and the mAP plot on the PASCAL VOC 2007 dataset optimized with the SGD with momentum. **All the parameters are the default parameters originally provided.**



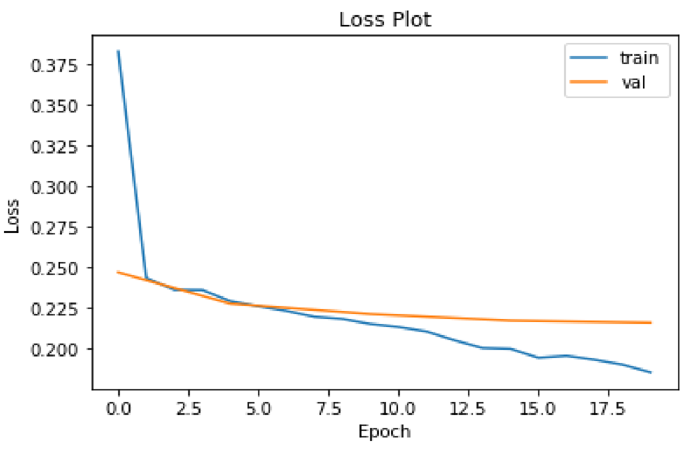


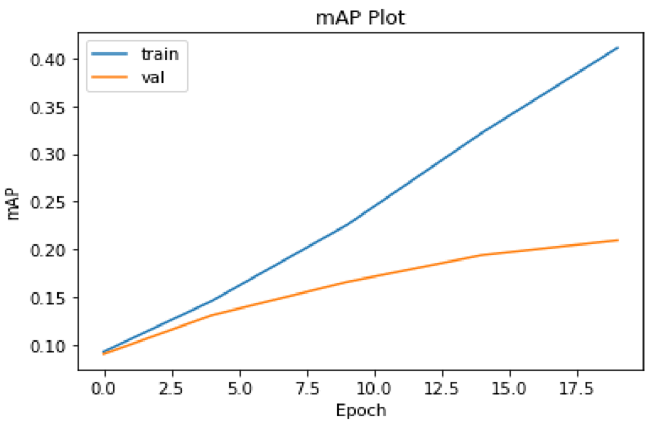
Shown below is the loss mAP plot on the PASCAL VOC 2007 dataset optimized with the **SGD with momentum. Here the learning rate is = 0.01 and the number of epochs=40**





Shown below is the loss mAP plot on the PASCAL VOC 2007 dataset optimized with the **Adam optimizer with a learning rate of 1e-4 along with the betas set to (0.9, 0.999), and the epsilon values=1e-08 with number of epochs set to 20.**





c. Provide analysis (at least 3 sentences):

Solution:

For analysis and comparison purposes I went ahead and changed the optimizer to the Adam method with learning rate of 1e-4 along with the betas set to (0.9, 0.999), and the epsilon values=1e-08 and ran it for 20 epochs. The analysis for simple classifier with default values and SGD and the finetuned for the simple classifier as provided in the classifer.py file with Adam updates are shown below.

Analysis of the default classifier with SGD method: In this case we note that the mAP is only 0.1468 which is on the lower side of the accuracy compared to what we will see later for the AlexNet model. I also saw that the validation loss does not decrease by much from the first to last epoch (it starts low and ends slightly lower) while the training loss starts from 0.48 and decreases to below 0.25 by the last epoch. I also noted that the original learning rate is set to 0.01 which on the higher side. This means that using a faster descent algorithm with a lower learning rate might be able to give us higher mAP. We also see that both the training and validation mAP keep rising all through the 20 epochs which means that doing more number of epochs might be helpful but as we see during the

runtime it is very computationally expensive. This calls for faster optimization methods like Adam which I implemented and the plots are shown above.
Analysis of the default classifier with Adam method: The Adam optimizer, as expected improved the performance over the default and fine-tuned hyperparameters (no. of epochs=40). However, the performance was still very low with an accuracy of around 0.20 (mAP).

2. AlexNet from Scratch

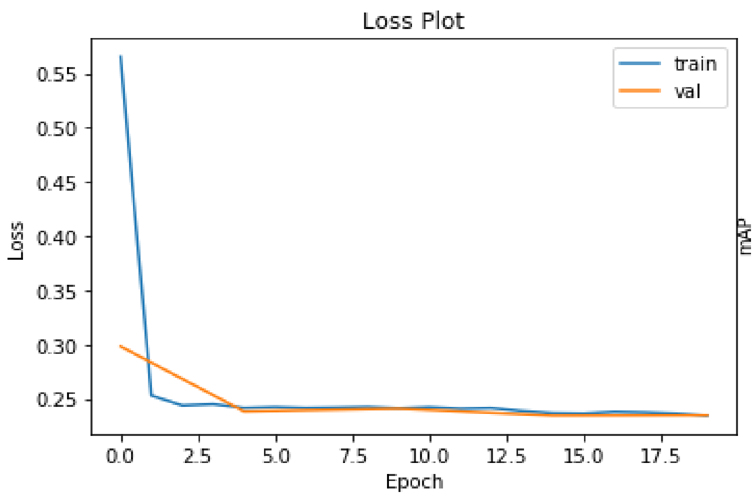
a. Report test mAP for alexnet:

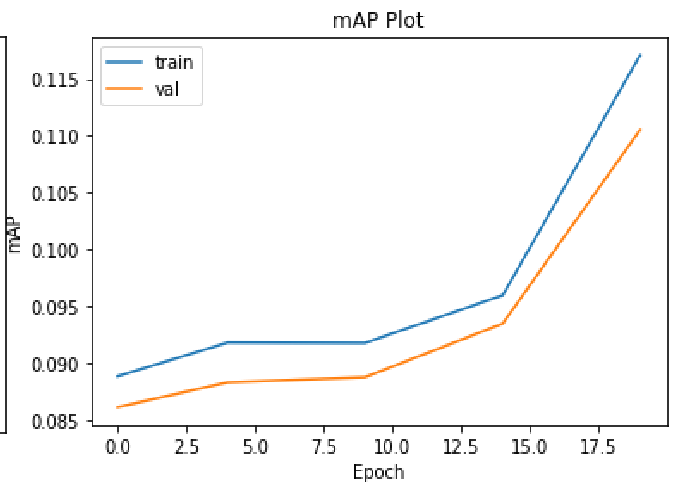
Solution:

- The mAP value for the test set from training AlexNet from scratch with the default value of learning parameter provided in the jupyter notebook was found to be **0.1057** while the average loss on the test set was **0.2313.**
- In question 1, I had shown that the best performance is obtained if the hyperparameters are finetuned and the learning rate becomes 0.01 and the number of epochs is set to 40. Using these hyperparameters for the AlexNet resulted in a mAP value for the test set to be **0.150** and the average loss on the test set was **0.2374.**
- I observed a very poor performance by training AlexNet from scratch and so I went on to implement AlexNet with an Adam optimizer with learning rate of 1e-4 along with the betas set to (0.9, 0.999), and the epsilon values=1e-08 and the performance improved and the mAP value for the test set was found to be **0.2723** while the average loss on the test set was **0.201** for 20 epochs**.**

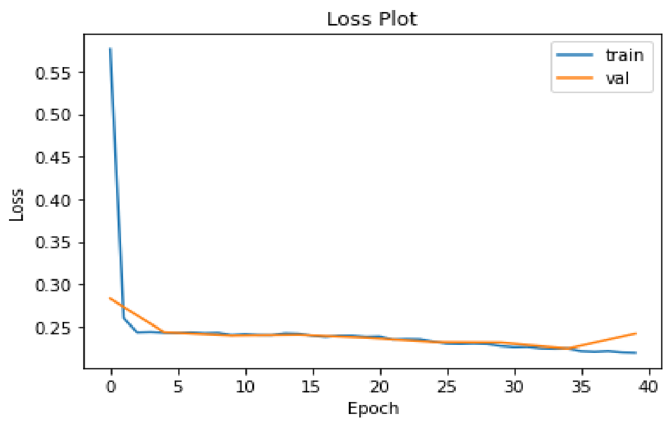
b. Visualize loss and mAP plots:

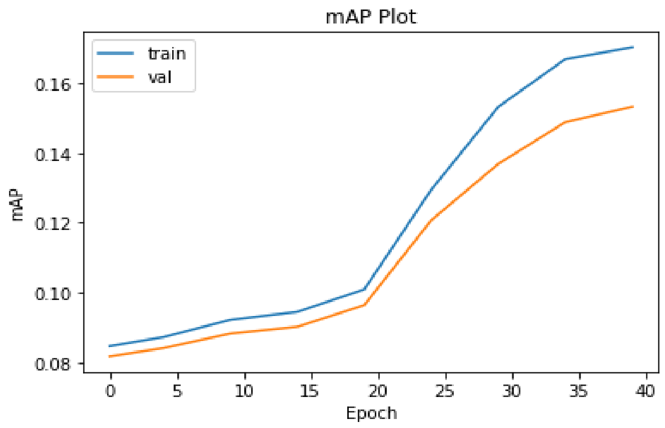
Shown below is the loss plot and the mAP plot of the train AlexNet from scratch on the PASCAL VOC 2007 dataset optimized with the SGD with momentum. **All the parameters are the default parameters originally provided.**



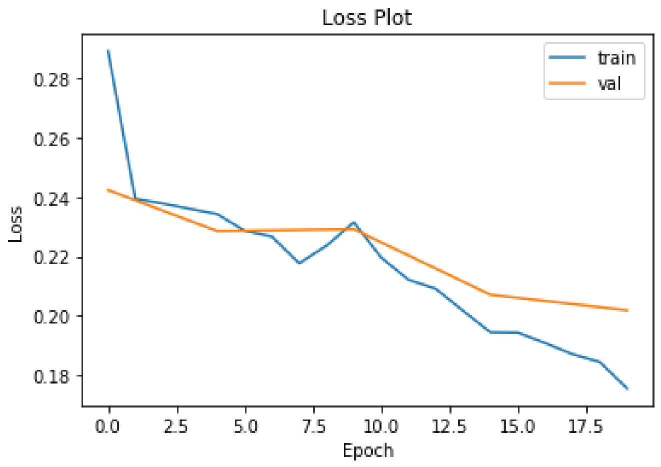


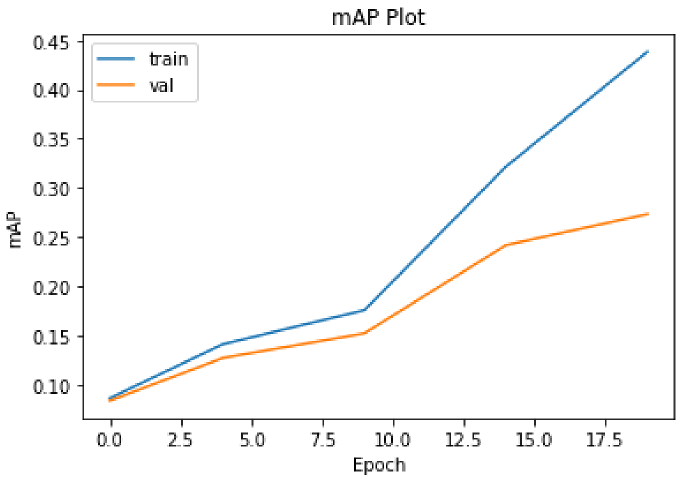
Shown below is the loss plot and the mAP plot of the train AlexNet from scratch on the PASCAL VOC 2007 dataset optimized with the SGD with momentum. **All the parameters are the optimally trained parameters obtained from experiments in question 1 (learning rate=0.01 and number of epochs=40).**



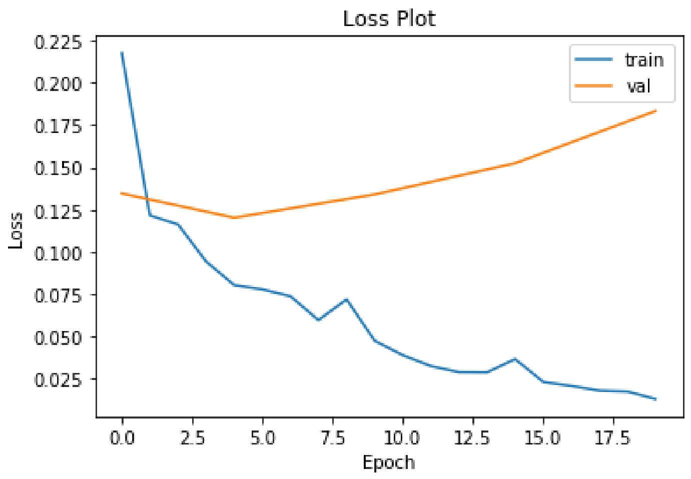


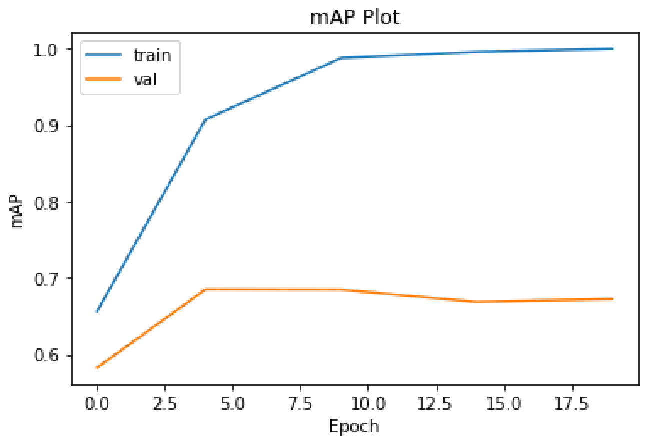
Shown below is the loss plot and the mAP plot of the train AlexNet from scratch on the PASCAL VOC 2007 dataset optimized with the SGD with momentum. The parameters used here are the for Adam updates with learning rate of 1e-4 along with the betas set to (0.9, 0.999), and the epsilon values=1e-08.





3. **Pretrained AlexNet**
   a. **Report test mAP for pretrained Alexnet:**
   Solution:
   For the pretrained AlexNet the test mAP was **0.680** while the average loss was **0.179.**
   b. **Visualize loss and mAP plots.**
   Shown below is the loss plot and the mAP plot of the pretrained AlexNet on the PASCAL VOC 2007 dataset optimized with the SGD with momentum. **All the parameters are the optimally default parameters originally provided.**





   c. **Provide analysis on differences to training from scratch (at least 3 sentences):**
   Solution:
   - It is very clear from the plots and the test mAP reports that the pretrained AlexNet is far better than the AlexNet trained from scratch. The pretrained AlexNet increases mAP by almost 3.5 times on the test set and the training accuracy is almost perfect.
   - This large difference in performance may be due to accurate "transfer learning" that is happening. As we know that the ImageNet dataset has more than 1000 classes and they also include the classes contained in the PASCAL VOC 2007 dataset so the training data set is kind of similar to the test dataset and it much larger. This improves performance.
   - We also know that the initial layers of the AlexNet learns the general features of the images like edges, curves, bends etc. which is also relevant to the PASCAL data set.

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.

Here is the detailed structure of my NN that I built on the top of the AlexNet Structure:

```
Classifier(
  (features): Sequential(
(0): Conv2d(3, 64, kernel_size=(11, 11), stride=(4, 4), padding=(2, 2))
(1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
(2): ReLU(inplace=True)
(3): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
(4): Conv2d(64, 192, kernel_size=(5, 5), stride=(1, 1), padding=(2, 2))
(5): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
(6): ReLU(inplace=True)
(7): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
(8): Conv2d(192, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(9): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
(10): ReLU(inplace=True)
(11): Conv2d(384, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(12): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
(13): ReLU(inplace=True)
(14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
(15): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
(16): ReLU(inplace=True)
(17): MaxPool2d(kernel_size=3, stride=2, padding=0, dilation=1, ceil_mode=False)
)
(avgpool): AdaptiveAvgPool2d(output_size=(6, 6))
(classifier): Sequential(
(0): Dropout(p=0.5, inplace=False)
(1): Linear(in_features=9216, out_features=4096, bias=True)
(2): ReLU(inplace=True)
(3): Dropout(p=0.5, inplace=False)
(4): Linear(in_features=4096, out_features=4096, bias=True)
(5): ReLU(inplace=True)
(6): Linear(in_features=4096, out_features=21, bias=True)
)
)
```

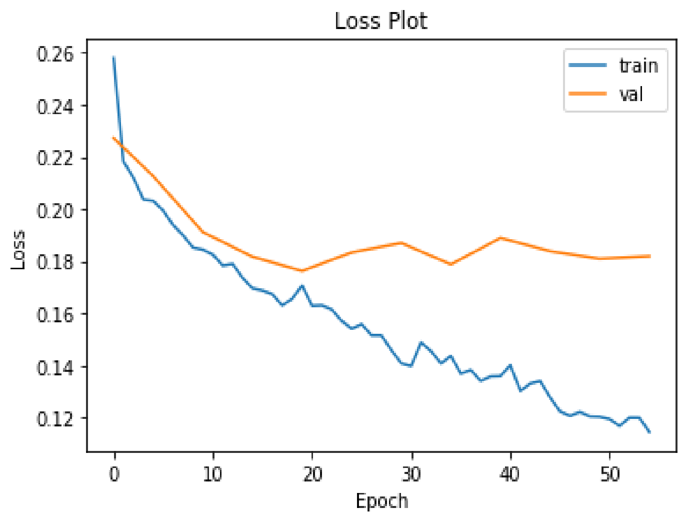
The output structure of my classifier is (where -1 stands for variable batch size):

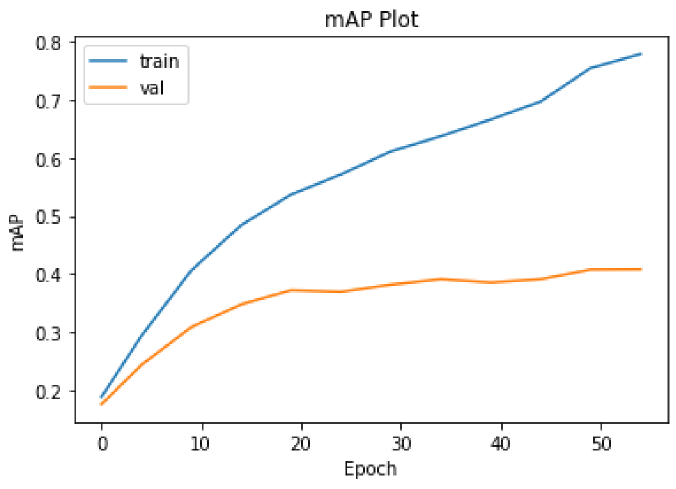
```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1           [-1, 64, 56, 56]          23,296
       BatchNorm2d-2           [-1, 64, 56, 56]             128
              ReLU-3           [-1, 64, 56, 56]               0
         MaxPool2d-4           [-1, 64, 27, 27]               0
            Conv2d-5          [-1, 192, 27, 27]         307,392
       BatchNorm2d-6          [-1, 192, 27, 27]             384
              ReLU-7          [-1, 192, 27, 27]               0
         MaxPool2d-8          [-1, 192, 13, 13]               0
            Conv2d-9          [-1, 384, 13, 13]         663,936
      BatchNorm2d-10          [-1, 384, 13, 13]             768
             ReLU-11          [-1, 384, 13, 13]               0
           Conv2d-12          [-1, 256, 13, 13]         884,992
      BatchNorm2d-13          [-1, 256, 13, 13]             512
             ReLU-14          [-1, 256, 13, 13]               0
           Conv2d-15          [-1, 256, 13, 13]         590,080
      BatchNorm2d-16          [-1, 256, 13, 13]             512
             ReLU-17          [-1, 256, 13, 13]               0
        MaxPool2d-18            [-1, 256, 6, 6]               0
AdaptiveAvgPool2d-19            [-1, 256, 6, 6]               0
          Dropout-20                 [-1, 9216]               0
           Linear-21                 [-1, 4096]      37,752,832
             ReLU-22                 [-1, 4096]               0
          Dropout-23                 [-1, 4096]               0
           Linear-24                 [-1, 4096]      16,781,312
             ReLU-25                 [-1, 4096]               0
           Linear-26                   [-1, 21]          86,037
================================================================
Total params: 57,092,181
Trainable params: 57,092,181
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.59
Forward/backward pass size (MB): 12.23
Params size (MB): 217.79
Estimated Total Size (MB): 230.61
```

Did you upload final CSV file on Kaggle: Yes/No

1. **My best mAP on Kaggle:**
   Yes, I uploaded my (1-mAP)= 0.58920 score to Kaggle. The plots below show the loss and training accuracy of my NN as a function of no. of epochs. A total of 55 epochs were used.

2. **Factors which helped improve my model**
   a. **Data augmentation (for detailed explanation of all factors please see below)**
   b. **Adding Batch Normalization**
   c. **Using Adam optimizer**
   d. **Adding more convolutional layers**
   e. **Adding 1x1 convolution layers**





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	11	227x227 | 56 x56	3 | 64
2	BatchNorm2d	-	56 x56 | 56 x56	-
3	ReLU	2	56 x 56 | 56 x56	-
4	MaxPool2d	3	27x 27 | 27x 27	
5	Conv2d	5	27x 27 | 27x 27	64 | 192
6	BatchNorm2d	2	27x 27 | 27x 27	-
7	ReLU	-	27x 27 | 27x 27	-
8	MaxPool2d	3	13x13 | 13x13	-
9	Conv2d	3	13x13 | 13x13	192| 384
10	BatchNorm2d-	5	13x13| 13x13	-
11	ReLU	-	13x13| 13x13	-
12	Conv2d	3	13x13| 13x13	384 | 256
13	BatchNorm2d	-	13x13 | 13x13	-
14	ReLU	-	13x13 | 13x13	-
15	Conv2d	3	13x13| 13x13	256 | 256
16	BatchNorm2d	-	13x13| 13x13	-
17	ReLU	-	13x13| 13x13	-
18	MaxPool2d	3	6x6| 6x6	-
19	AdaptiveAvgPool2d	-	6x6| 6x6	-
20	Dropout	-	256x6x6| 9216	-
21	Linear	-	9216| 4096	-
22	ReLU	-	4096| 4096	-
23	Dropout	-	4096| 4096	-
24	Linear	-	4096| 4096	-
25	ReLU	-	4096| 4096	-
26	Linear	-	4096| 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 :

The initial network that I used was AlexNet and its best mAP and subsequent addition along with their respective mAPs are listed below (all the experiments below were run for 20 epochs with the Adam optimizer instead of the optimum 55 epochs to save Colab time.)

Serial #	**Model architecture**	**Best mAP on test set**
1	BaseNet (SGD)	0.15
2	BaseNet (Adam)	0.27
3	BaseNet (Adam) + data agumentation	0.28
4	BaseNet + a + data agumentation Where a= one BatchNorm2d	0.30
5	BaseNet + a + b + data agumentation Where b= 2 additional BatchNorm2d	0.32
6	BaseNet + a + b+c + data agumentation Where c= 2 additional BatchNorm2d	0.36
7	BaseNet + a + b+c+d+ data agumentation Where d= two layers of Conv2d(384, 384, kernel_size=3, padding=1) followed by one layer of MaxPool2d(kernel_size=3, stride=2)	0.35
8	BaseNet + a + b+c+d+e+ data agumentation Where e= three layers of Conv2d(256, 256, kernel_size=3, padding=1) followed by one layer of MaxPool2d(kernel_size=3, stride=2)	0.32

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

a. **Data augmentation –** this is very helpful in increasing the amount of data to train the parameters. More the number of parameters the more is the data required to train them to get higher mAP on testset. I used the following inbuilt data augmentation in Pytorch

```
transforms.ColorJitter(hue=.05, saturation=.05),
transforms.RandomHorizontalFlip(),
transforms.RandomRotation(20, resample=PIL.Image.BILINEAR),
transforms.CenterCrop(227),
```

b. **Adding Batch Normalization** – I added 5 batch normalization layers in successively and saw a gradual increase in the mAP score in the test set. Batch normalization of the input layer by re-centering and re-scaling makes the NN faster and more stable.

c. **Using Adam optimizer** -- Using the Adam optimizer helped in faster convergence and achieve higher rate of learning. The algorithms leverage the power of adaptive learning rates methods to find individual learning rates for each parameter.

d. **Adding more convolutional layers –** helped in adding more number of parameters to learn. Using data augmentation (provides additional data to train) along with more convolution layers can help the model learn better. However, I also noticed that after certain threshold adding more convolution layers decreased the mAP on test set. This is because the number of parameters increased but there was not enough data to train so this resulted in under fitting.

e. **Adding 1x1 convolution layers** – as mentioned above adding convolution layers helps increase the network capacity by increasing the number of trainable parameters and are used for reducing dimensionality.