Name(s): Theodoros Mamalis
Kaggle Team Name: PeacockNavigator

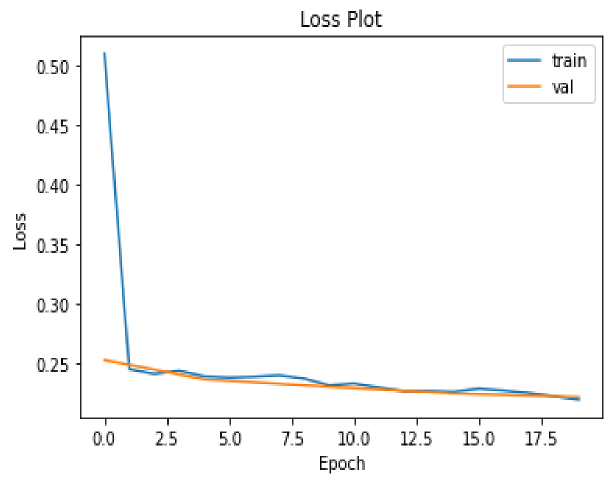
Part-1A: Pre-designed network for multi-label classification

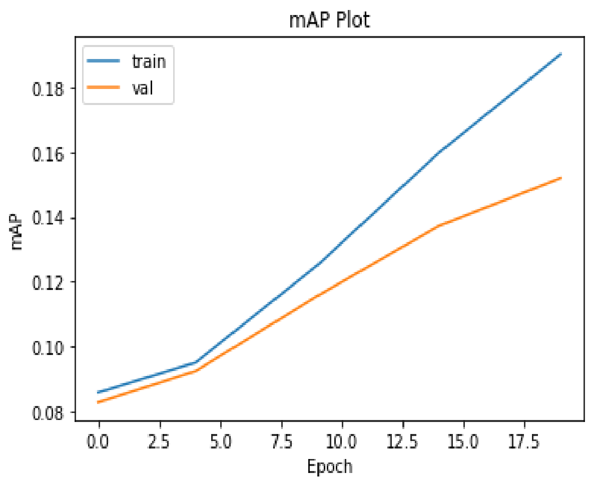
In this part, you will practice to train a neural network both by training from scratch or fine-tuning.

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.

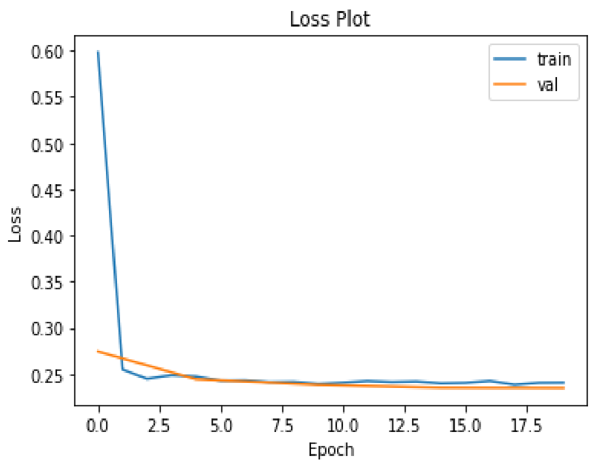
1. Simple Classifier
   a. Report test mAP for simple classifier: 0.1475
   b. Visualize loss and mAP plots:

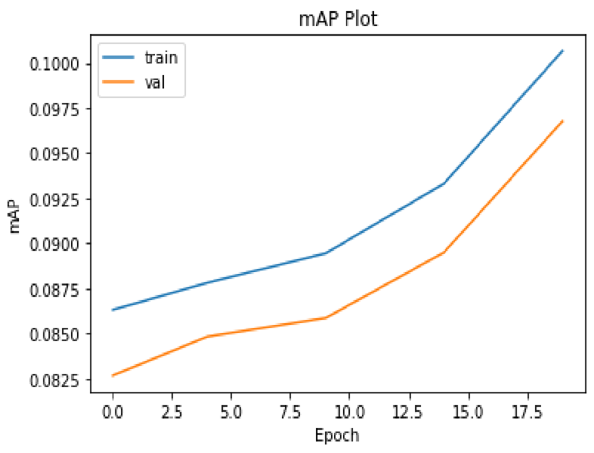




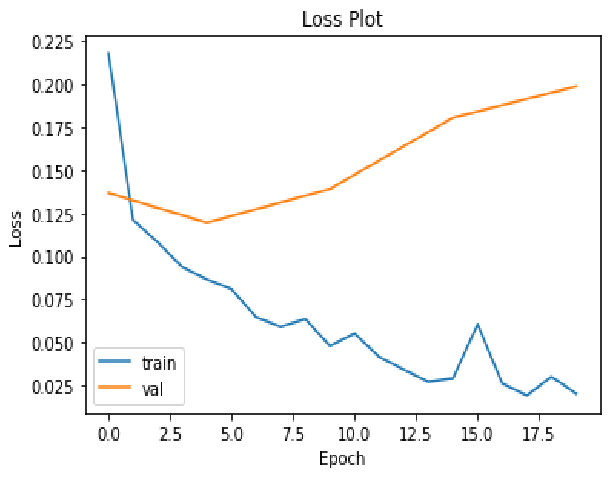
   c. Provide analysis (at least 3 sentences):
   The training loss is closely followed by the validation loss. Since the validation loss has not started going up this means that there is no overfitting. However, given that the the mAP train plot starts getting a lot ahead of the validation mAP plot, this could mean that the model is close to starting to overfit. Judging from part 1B, the reason for this behaviour could be the model architecture, and the optimization method used (SGD instead of ADAM).

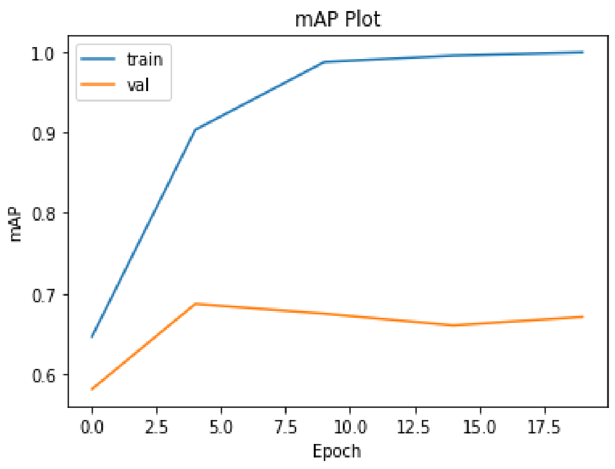
2. AlexNet from Scratch
   a. Report test mAP for alexnet: 0.0928
   b. Visualize loss and mAP plots:





3. Pretrained AlexNet
    a. Report test mAP for pretrained alexnet: 0.6821
    b. Visualize loss and mAP plots





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

It can be seen that the training loss of the pretrained net is somewhat jaggy when compared to the trained-from-scratch model, which means that it could benefit from small learning rates; this could be because the pretrained model has already been trained for a number of epochs. Moreover, since the training loss goes down but the validation loss increases after around 4 epochs, the pretrained model has started overfitting. This can be also be seen from the mAP plot, where the train mAP reaches a score close to 1 but the validation loss decreases from epoch 4 and onwards. On the contrary, when training from scratch, both the loss and mAP plots look good since validation loss decreases with training loss, and validation mAP follows the train mAP. This is because the pretrained Alexnet already knows how to recognize the features contained in the PASCAL VOC dataset (maybe because the PASCAL VOC classes and the ImageNet classes have similar features as far as the algorithm is concerned), and training it even further makes it overfit to less generalizable features (noise) to reduce the training loss. The fact that it can already recognize some relevant features, is the reason why it gives better mAP scores than when training it from scratch. On the other hand, when training Alexnet from scratch, the network has still features to learn and constantly improves on learning and predicting them without having to learn noise in order to decrease the train loss.

<u>Part-1B: Self designed network for multi-label classification</u>

<u>MP3_P1_Develop_Classifier</u> 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: 0.61209
2. Factors which helped improve my model
   a. Data augmentation by transforms.CenterCrop(227) (resize and crop the image), transforms.RandomResizedCrop(227) (resize and crop a random part of the image), transforms.RandomHorizontalFlip(p=0.5) (a picture is flipped with 0.5 probability)
   b. Adding batch normalization to every layer besides the last
   c. Using ADAM instead of SGD.
   d. Having conv2d (2-d convolutions) connections with increasing channel sizes
   e. Removing two of the three fully connected layers increased speed and did not affect accuracy
   f. Using max pooling before batch norm and the nonlinearity improved speed marginally.
   g. Normalizing the data.
   h. Using stride of 2.

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

   All below are calculated for stride=2:

Layer No.	Layer Type	Kernel size (for conv layers)	Input | Output dimension	Input | Output Channels (for conv layers)
1	conv2d	3	227x227 |113x113	3|64
2	BatchNorm2d	-	113x113|113x113	-
3	relu	-	113x113|113x113	-
4	conv2d	3	113x113|56x56	64|128
5	BatchNorm2d	-	56x56|56x56	-
6	relu	-	56x56|56x56	-
7	conv2d	3	56x56|27x27	128|256
8	maxpool2d	2	27x27|13x13	-
9	BatchNorm2d	-	13x13|13x13	-
10	relu	-	13x13|13x13	-
11	conv2d	3	13x13|6x6	256|512
12	BatchNorm2d	-	6x6|6x6	-
13	relu	-	6x6|6x6	-
14	conv2d	3	6x6|2x2	512 |512
15	BatchNorm2d	-	2x2|2x2	-

16	relu	-	2x2|2x2	-
17	Linear	-	2048|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	0.1604
2	BaseNet + ADAM instead of SGD	0.1969
3	BaseNet + ADAM + removing last two fully connected layers	0.2146
4	BaseNet + ADAM + removing last two fully connected layers + BatchNorm2d	0.2309
5	BaseNet + ADAM + removing last two fully connected layers + BatchNorm2d + going from smaller-channels-bigger-maps (as in BaseNet) to deeper-channels-smaller-maps and stride=2 (as in the network described in the final architecture table)	0.32

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

(All mAP's above are for 20 epochs)

Replacing ADAM with SGD helped increase mAP by almost 4% because ADAM adapts the stepsize, and apparently this helped with the minimization of the complex-shaped loss function. Removing the two final fully connected layers helped maybe because previously, a lot of spatial information of the image was lost in the process. Including batch norm layers helped a bit, possibly because the batch norm layers are like standardizing the distribution of their layers yielding better statistical and minimization properties.

Going for a deeper-channels-smaller-maps increased the effective receptive field of the final convolution output, passing more informative results to the fully-connected layer. Also, setting stride 2 helped reduce the total parameters before the fully connected layer by 2089 times (from 5752832 with stride=1 to 2408 with stride=2). For these combined reasons, the optimization problem was more tractable and the algorithm yielded better results than the four previous architectures. After 5. , stride=1 was tried and but training was 1.5 times slower (0.01s per batch of 10 for stride=1, in contrast with 0.65s per batch of 10 for stride=2). Also the accuracy was many percentages higher when using stride =2 than when using stride =1 for the same architecture (0.11, 0.1093 for epochs 1 and 0.18, 0.23 for epochs 2 for stride=1 and =2 respectively).

What did not help much was adding weight decay and changing the learning rate, but that could be because of using ADAM.