Phang Teng Fone 1003296
50.038 - Homework 3

Question 1

Dog.jpg

Out of the box ResNet101:

```
torch.Size([3, 224, 224])
Label: tensor(259) . Confidence Score: 64.96243286132812 %
Label: tensor(153) . Confidence Score: 18.492586135864258 %
Label: tensor(258) . Confidence Score: 3.901040554046631 %
Label: tensor(152) . Confidence Score: 3.283771514892578 %
Label: tensor(265) . Confidence Score: 1.451918601989746 %
```

Data Augmentation with 5 crops and ResNet101:

```
Averaging scores of 5 crops:
Label: tensor(259) . Confidence Score: 67.27539825439453 %
Label: tensor(153) . Confidence Score: 16.193790435791016 %
Label: tensor(258) . Confidence Score: 5.253355026245117 %
Label: tensor(265) . Confidence Score: 2.458286762237549 %
Label: tensor(152) . Confidence Score: 1.5673789978027344 %
```

Data augmentation helps us to have more data on top of the pre existing data available.

During test time, data augmentation can be used to predict on a n average of multiple views of the same input. It produces more robust results but needs to run the model n number of times which means more computational time.

During training time, data augmentation can be used to extend the training data available by creating instances which can be derived from the training data set.

Data augmentation are also noted to be an estimation of the image and might not reflect the full scale of the image during train time, as such, having an over augmented image on a single file might cause performance decrease instead of benefiting the network.

Question 2

Task 1

```
!python train.py "./flowers" --gpu
```

```
Epoch: 1/1 -  Training Loss: 3.903 -  Validation Loss: 2.646 -  Validation Accuracy: 0.517
Epoch: 1/1 -  Training Loss: 2.124 -  Validation Loss: 1.255 -  Validation Accuracy: 0.758
model: densenet169 - hidden layers: [1024] - epochs: 1 - lr: 0.001
Run time: 2.226 min
```

```
!python train.py "./flowers" --gpu --arch googlenet
```

```
Epoch: 1/1 -  Training Loss: 4.190 -  Validation Loss: 3.431 -  Validation Accuracy: 0.280
Epoch: 1/1 -  Training Loss: 2.983 -  Validation Loss: 2.121 -  Validation Accuracy: 0.583
model: googlenet - hidden layers: [1024] - epochs: 1 - lr: 0.001
Run time: 1.544 min
```

Comparing DenseNet169 to GoogleNet. Densenet 169 has a higher validation accuracy as compared to GoogleNet.

Task 2

Training Model from scratch (2.1)

```
!python train.py "./flowers" --gpu
```

```
Epoch: 1/1 -  Training Loss: 4.492 -  Validation Loss: 4.199 -  Validation Accuracy: 0.064
Epoch: 1/1 -  Training Loss: 4.105 -  Validation Loss: 3.893 -  Validation Accuracy: 0.094
model: densenet169 - hidden layers: [1024] - epochs: 1 - lr: 0.001
Run time: 1.674 min
```

Finetune model but only updating the top layers (2.2)

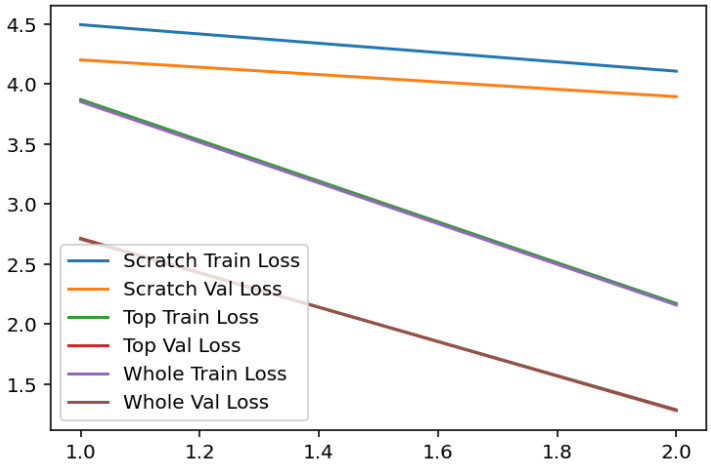
```
!python train.py "./flowers" --gpu
```

```
Epoch: 1/1 -  Training Loss: 3.869 -  Validation Loss: 2.713 -  Validation Accuracy: 0.504
Epoch: 1/1 -  Training Loss: 2.171 -  Validation Loss: 1.280 -  Validation Accuracy: 0.753
model: densenet169 - hidden layers: [1024] - epochs: 1 - lr: 0.001
Run time: 1.656 min
```

Finetuning the whole model (2.3)

```
!python train.py "./flowers" --gpu
```

```
Epoch: 1/1 -  Training Loss: 3.851 -  Validation Loss: 2.707 -  Validation Accuracy: 0.473
Epoch: 1/1 -  Training Loss: 2.158 -  Validation Loss: 1.286 -  Validation Accuracy: 0.741
model: densenet169 - hidden layers: [1024] - epochs: 1 - lr: 0.001
Run time: 1.668 min
```



The training from scratch will take a long time to converge as all weights are not loaded preemptively as compared to the rest of the methods. For the whole fine tuning, it performs better than the top layer only fine tuning as theoretically, having to fine tune the entire preloaded weights should give a better performance.

Task 3

```
!python train.py "./flowers" --gpu
```

```
Epoch: 1/1 -  Training Loss: 3.848 -  Validation Loss: 2.744 -  Validation Accuracy: 0.513
Epoch: 1/1 -  Training Loss: 2.211 -  Validation Loss: 1.308 -  Validation Accuracy: 0.749
Testing Accuracy: 0.806
model: densenet169 - hidden layers: [1024] - epochs: 1 - lr: 0.001
Run time: 1.805 min
```

Task 4

For this task, I took reference from Alex Net (https://github.com/pytorch/vision/blob/master/torchvision/models/alexnet.py) Sequential layers and atune it to this particular model, including weight initialization to obtain better results.

5 Convo Layer with 5 epochs ¶

```
: !python train.py "./flowers" --gpu --epochs 5

Epoch: 1/5 -  Training Loss: 4.619 -  Validation Loss: 4.529 -  Validation Accuracy: 0.025
Epoch: 1/5 -  Training Loss: 4.514 -  Validation Loss: 4.491 -  Validation Accuracy: 0.034
Epoch: 2/5 -  Training Loss: 4.412 -  Validation Loss: 4.189 -  Validation Accuracy: 0.036
Epoch: 2/5 -  Training Loss: 4.115 -  Validation Loss: 4.029 -  Validation Accuracy: 0.067
Epoch: 2/5 -  Training Loss: 4.005 -  Validation Loss: 3.919 -  Validation Accuracy: 0.060
Epoch: 3/5 -  Training Loss: 3.879 -  Validation Loss: 3.804 -  Validation Accuracy: 0.060
Epoch: 3/5 -  Training Loss: 3.818 -  Validation Loss: 3.802 -  Validation Accuracy: 0.088
Epoch: 4/5 -  Training Loss: 3.742 -  Validation Loss: 3.630 -  Validation Accuracy: 0.091
Epoch: 4/5 -  Training Loss: 3.671 -  Validation Loss: 3.595 -  Validation Accuracy: 0.106
Epoch: 4/5 -  Training Loss: 3.633 -  Validation Loss: 3.532 -  Validation Accuracy: 0.103
Epoch: 5/5 -  Training Loss: 3.504 -  Validation Loss: 3.462 -  Validation Accuracy: 0.120
Epoch: 5/5 -  Training Loss: 3.470 -  Validation Loss: 3.343 -  Validation Accuracy: 0.143
Testing Accuracy: 0.125
model: Custom Model - hidden layers: [1024] - epochs: 5 - lr: 0.001
Run time: 7.601 min
```

2 Convo Layer with 5 epochs

```
!python train.py "./flowers" --gpu --epochs 5

Epoch: 1/5 -  Training Loss: 4.313 -  Validation Loss: 3.927 -  Validation Accuracy: 0.073
Epoch: 1/5 -  Training Loss: 3.797 -  Validation Loss: 3.625 -  Validation Accuracy: 0.113
Epoch: 2/5 -  Training Loss: 3.584 -  Validation Loss: 3.385 -  Validation Accuracy: 0.136
Epoch: 2/5 -  Training Loss: 3.466 -  Validation Loss: 3.261 -  Validation Accuracy: 0.163
Epoch: 2/5 -  Training Loss: 3.263 -  Validation Loss: 3.126 -  Validation Accuracy: 0.182
Epoch: 3/5 -  Training Loss: 3.184 -  Validation Loss: 3.057 -  Validation Accuracy: 0.209
Epoch: 3/5 -  Training Loss: 3.059 -  Validation Loss: 2.866 -  Validation Accuracy: 0.260
Epoch: 4/5 -  Training Loss: 2.939 -  Validation Loss: 2.830 -  Validation Accuracy: 0.279
Epoch: 4/5 -  Training Loss: 2.861 -  Validation Loss: 2.591 -  Validation Accuracy: 0.310
Epoch: 4/5 -  Training Loss: 2.809 -  Validation Loss: 2.606 -  Validation Accuracy: 0.294
Epoch: 5/5 -  Training Loss: 2.652 -  Validation Loss: 2.577 -  Validation Accuracy: 0.321
Epoch: 5/5 -  Training Loss: 2.686 -  Validation Loss: 2.471 -  Validation Accuracy: 0.336
Testing Accuracy: 0.369
model: Custom Model - hidden layers: [1024] - epochs: 5 - lr: 0.001
Run time: 7.542 min
```

CNN with lesser convolutional layers performs better as the CNN layers are dependent on the number of data sets available. In our case, since the data set is small, lesser convolutionary lesser will perform better.