CS6910 : Deep Learning (for Computer Vision)

Irfaan Arif
ME18B048

Assignment 1

All experiments and results below are on the Dataset 5, mini-ImageNet with 33 classes

1 Part-A

This part was to train a network for image classification on the given dataset and experiment with the following network parameters

1. Number of Convolutional (conv) layers
2. Number of fully connected (fc) layers
3. The number of filters in different layers
4. Maxpooling layers
5. Training time (number of epochs)
6. Stride in different layers

The models were trained on a AWS instance (g4dn.xlarge) with **batch size 4** and for **100 epochs** with a learning rate**(lr) of 0.001** with SGD, and the remaining parameters same as the one in the boilerplate code.

The model with the best validation accuracy among all the epochs was taken as the result for each experiment.

1.1 Number of Convolutional (conv) layers

The boiler plate model was used as the baseline, and different models were trained by removing upto 2 conv layers or by adding upto 2 more conv layers.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 80, 80]           4,864
         MaxPool2d-2          [-1, 64, 40, 40]               0
            Conv2d-3         [-1, 128, 36, 36]         204,928
         MaxPool2d-4         [-1, 128, 18, 18]               0
            Conv2d-5         [-1, 256, 14, 14]         819,456
         MaxPool2d-6           [-1, 256, 7, 7]               0
            Linear-7                 [-1, 256]          65,792
            Linear-8                 [-1, 128]          32,896
            Linear-9                  [-1, 33]           4,257
================================================================
Total params: 1,132,193
Trainable params: 1,132,193
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.08
Forward/backward pass size (MB): 5.97
Params size (MB): 4.32
Estimated Total Size (MB): 10.37
----------------------------------------------------------------
```

Figure 1: The baseline model with 3 convolutional layers

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 80, 80]           4,864
         MaxPool2d-2          [-1, 64, 40, 40]               0
            Linear-3                [-1, 256]          16,640
            Linear-4                [-1, 128]          32,896
            Linear-5                 [-1, 33]           4,257
================================================================
Total params: 58,657
Trainable params: 58,657
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.08
Forward/backward pass size (MB): 3.91
Params size (MB): 0.22
Estimated Total Size (MB): 4.21
----------------------------------------------------------------
```

Figure 2: Model with 1 conv layer

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 80, 80]           4,864
         MaxPool2d-2          [-1, 64, 40, 40]               0
            Conv2d-3         [-1, 128, 36, 36]         204,928
         MaxPool2d-4         [-1, 128, 18, 18]               0
            Conv2d-5         [-1, 256, 14, 14]         819,456
         MaxPool2d-6           [-1, 256, 7, 7]               0
            Conv2d-7           [-1, 256, 5, 5]         590,080
         MaxPool2d-8           [-1, 256, 2, 2]               0
            Conv2d-9           [-1, 512, 2, 2]       1,180,160
           Linear-10                [-1, 256]         131,328
           Linear-11                [-1, 128]          32,896
           Linear-12                 [-1, 33]           4,257
================================================================
Total params: 2,967,969
Trainable params: 2,967,969
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.08
Forward/backward pass size (MB): 6.04
Params size (MB): 11.32
Estimated Total Size (MB): 17.44
----------------------------------------------------------------
```

Figure 3: Model with 5 conv layers

The baseline model as seen in Figure 1 has 3 conv layers so we have a model with 1 conv layer at the lower end (Figure 2) and with 5 total conv layers at the other end (Figure 3). The numbers of output filters of some models were also adjusted to fit in with the fc layers. Also after every convolution layer, we are also doing maxpooling with a (2X2) kernel and stride 2.

% Accuracy	Train	Val	Test	No: of parameters
1 conv	45.2	39.6	37.8	58657
2 conv	74.6	52.3	50.7	279969
3 conv(baseline)	94.1	59.6	58.5	1132193
4 conv	92.0	56.7	56.4	1722273
5 conv	93.2	56.7	57.1	2967969

Table 1: Accuracy of models with different number of conv layers

As can be seen from Table 1, in general as you increase the number of convolutional layers, we get better perfomance in train, val and test. However after a while you reach a point of diminishing returns wherein adding more conv layers may even slightly hurt the performance and result in very long training times as the number of parameters also increases.

Especially for the model with only 1 conv layer, we can understand that the model is under-fitting, and our model is not large enough to learn a good representation (refer Figure 4).

However for the models with 4 and 5 conv layers, we can see that these models quickly overfit and get high training accuracy and low validation accuracy, this is because there are a large number of parameters and we only have a limited dataset to work with. Transfer learning or forms of regularization like dropout could help alleviate this problem as well as also getting more training data (refer Figure 5).

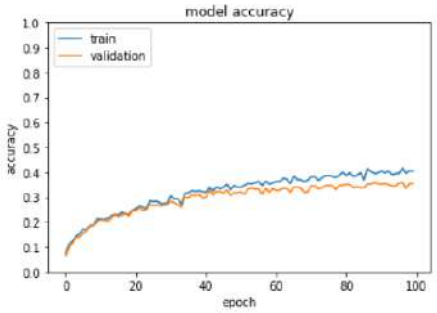

Figure 4: Model with 1 conv layer

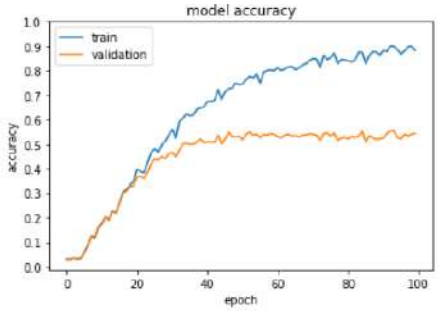

Figure 5: Model with 5 conv layers

1.2 Number of fully connected (fc) layers

Once again the boiler plate model was used as the baseline, and different models were trained by removing upto 1 fully connected layer or by adding upto 2 extra fully connected layers.

Since the baseline model had 3 fc layers (including the final classification layer) we have models with 2 fc layers to ones with 5 fc layers in total. As the number of fully connected layers are increased, the number of features in the linear layers were also appropriately increased.

% Accuracy	Train	Val	Test	No: of parameters
2 fc	93.1	59.2	57.4	1066401
3 fc(baseline)	91.5	59.6	57.9	1132193
4 fc	93.4	59.4	59.5	1197985
5 fc	89.8	58.5	57.8	1395105

Table 2: Accuracy of models with different number of fully connected layers

As can be seen from Table 2, there is not much variation when we change the number of fully connected layers. This could be due to the universal approximation theorem, in which the function mapping can be learned even if we only have 2 fc layers in feed forward style of sufficient width.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 80, 80]           4,864
         MaxPool2d-2          [-1, 64, 40, 40]               0
            Conv2d-3         [-1, 128, 36, 36]         204,928
         MaxPool2d-4         [-1, 128, 18, 18]               0
            Conv2d-5         [-1, 256, 14, 14]         819,456
         MaxPool2d-6           [-1, 256, 7, 7]               0
            Linear-7                 [-1, 256]          65,792
            Linear-8                 [-1, 256]          65,792
            Linear-9                 [-1, 128]          32,896
           Linear-10                  [-1, 33]           4,257
================================================================
Total params: 1,197,985
Trainable params: 1,197,985
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.08
Forward/backward pass size (MB): 5.97
Params size (MB): 4.57
Estimated Total Size (MB): 10.62
----------------------------------------------------------------
```

Figure 6: The best model while changing FC layers

But more the width and number, a better approximation we can usually achieve, however since we have a limited dataset for this problem it is usually better to pick a medium sized network to minimize over-fitting. Experimentally 4 fc layers (refer 6 for the network) seems to work best for this particular problem, however the differences seem to be within the margin for error due to random weight initialization.

1.3 Number of filters in different layers

For this part, 3 models were tested which had the general structure as the model in the boilerplate model. However the numbers of filters in both the convolutional and fully connected layers are vastly different in each model.

Results obtained are shown in Table 3. We can also roughly call Net1 as a small model, Net2 as a medium sized model and Net3 as a large model.

% Accuracy	Train	Val	Test	No: of parameters
Net1	52.4	44.2	42.7	75347
Net2	81.3	53.2	51.6	682913
Net3	94.8	54.0	51.4	4967329

Table 3: Accuracy of models with different number of filters in layers

As expected the small model seems to be under-fitting whereas the large model seems to definitely be over-fitting on the training dataset

```
Net1(
  (conv1): Conv2d(3, 8, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(8, 16, kernel_size=(5, 5), stride=(1, 1))
  (conv3): Conv2d(16, 32, kernel_size=(5, 5), stride=(1, 1))
  (conv4): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=64, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=33, bias=True)
  (fc3): Linear(in_features=33, out_features=33, bias=True)
)
Number of params :  75347
--------------------------------------------------------------------------------
```

Figure 7: Net1 - Small model

```
Net2(
  (conv1): Conv2d(3, 32, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(32, 64, kernel_size=(5, 5), stride=(1, 1))
  (conv3): Conv2d(64, 128, kernel_size=(5, 5), stride=(1, 1))
  (conv4): Conv2d(128, 128, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=128, out_features=64, bias=True)
  (fc2): Linear(in_features=64, out_features=64, bias=True)
  (fc3): Linear(in_features=64, out_features=33, bias=True)
)
Number of params :  682913
--------------------------------------------------------------------------------
```

Figure 8: Net2 - Medium sized model

```
Net3(
  (conv1): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (conv2): Conv2d(64, 128, kernel_size=(5, 5), stride=(1, 1))
  (conv3): Conv2d(128, 256, kernel_size=(5, 5), stride=(1, 1))
  (conv4): Conv2d(256, 512, kernel_size=(5, 5), stride=(1, 1))
  (fc1): Linear(in_features=512, out_features=1024, bias=True)
  (fc2): Linear(in_features=1024, out_features=128, bias=True)
  (fc3): Linear(in_features=128, out_features=33, bias=True)
)
Number of params :  4967329
--------------------------------------------------------------------------------
```

Figure 9: Net3 - Large model

Another experiment that was run was to change the kernel size of the convolution layers, in general (3X3) and (5X5) gave almost same results for a majority of the different network architectures, while (7X7) and higher generally did worse. In general keeping the first convolutional layer kernel as larger seemed beneficial.

1.4 Maxpooling layer

For this part, 3 models were tested which various different structure/architecture for the model. The parameters and general structure of the model are derived from the basecase in the boilerplate code.
Results obtained are shown in Table 4.

- **Net1** : Without any maxpooling layer, instead using conv layers with a stride of 2 (ref Figure: 10)
- **Net2** : Base case, maxpooling after every conv layer (ref Figure: 11)
- **Net3** : Maxpooling after every 2 conv layers (ref Figure: 12)

% Accuracy	Train	Val	Test	No: of parameters
Net1	85.3	56.7	55.1	1132193
Net2	91.5	59.6	57.9	1132193
Net3	90.2	63.1	62.4	4475041

Table 4: Accuracy of the 3 different models

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 40, 40]           4,864
            Conv2d-2         [-1, 128, 18, 18]         204,928
            Conv2d-3           [-1, 256, 7, 7]         819,456
            Linear-4                 [-1, 256]          65,792
            Linear-5                 [-1, 128]          32,896
            Linear-6                  [-1, 33]           4,257
================================================================
Total params: 1,132,193
Trainable params: 1,132,193
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.08
Forward/backward pass size (MB): 1.20
Params size (MB): 4.32
Estimated Total Size (MB): 5.60
----------------------------------------------------------------
```

Figure 10: Net1 - No maxpool

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 80, 80]           4,864
         MaxPool2d-2          [-1, 64, 40, 40]               0
            Conv2d-3         [-1, 128, 36, 36]         204,928
         MaxPool2d-4         [-1, 128, 18, 18]               0
            Conv2d-5         [-1, 256, 14, 14]         819,456
         MaxPool2d-6           [-1, 256, 7, 7]               0
            Linear-7                 [-1, 256]          65,792
            Linear-8                 [-1, 128]          32,896
            Linear-9                  [-1, 33]           4,257
================================================================
Total params: 1,132,193
Trainable params: 1,132,193
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.08
Forward/backward pass size (MB): 5.97
Params size (MB): 4.32
Estimated Total Size (MB): 10.37
----------------------------------------------------------------
```

Figure 11: Net2 - Maxpool after every conv

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 80, 80]           4,864
            Conv2d-2         [-1, 128, 76, 76]         204,928
         MaxPool2d-3         [-1, 128, 38, 38]               0
            Conv2d-4         [-1, 256, 34, 34]         819,456
            Conv2d-5         [-1, 512, 30, 30]       3,277,312
         MaxPool2d-6         [-1, 512, 15, 15]               0
            Linear-7                [-1, 256]         131,328
            Linear-8                [-1, 128]          32,896
            Linear-9                 [-1, 33]           4,257
================================================================
Total params: 4,475,041
Trainable params: 4,475,041
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.08
Forward/backward pass size (MB): 16.83
Params size (MB): 17.07
Estimated Total Size (MB): 33.98
----------------------------------------------------------------
```

Figure 12: Net3 - Maxpool after every other conv

Net3 with maxpooling after every two convolutional layers is observed to be the clear winner here achieving a good test accuracy of 62% although it a slightly large network.

Also changing the stride and kernel size of maxpooling layer produced inconsistent results, there was no general trend observed, it seemed to also depend on what the architecture was like and the parameters of the different layers.

1.5 Training time (number of epochs)

Note: The results for this section related to training time based on number of epochs was run on my local machine with a Nvidia Geforce 1060 GPU with an Intel i7-8750H CPU. Torch version was 1.6.0 with cuda 10.1, the process was profiled using the Nvidia Nsight Systems application.

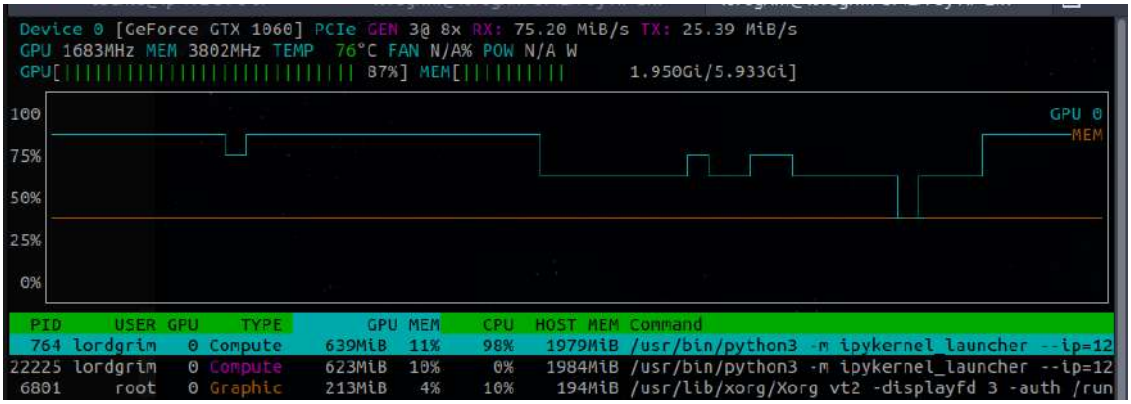

Figure 13: GPU utilization

The GPU was being utilized almost throughout the time, peaking at 87% utilization (Figure: 13), GPU memory was not used much, therefore batch size could have been increased to get better training times.

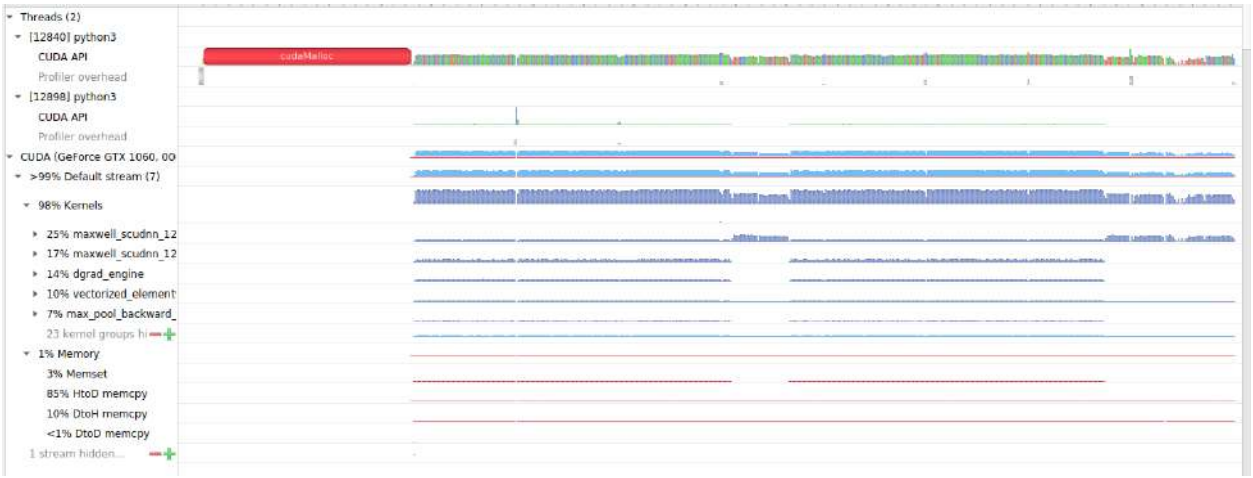

Figure 14: Nsight profiling

There also does not seem to be any other major bottleneck in the system based on the profiling results (ref Figure: 14)

Training time is as such linear with the number of epochs, on average 1 epoch took 35.63 seconds with a standard deviation of 0.165 (ref Figure: 15)

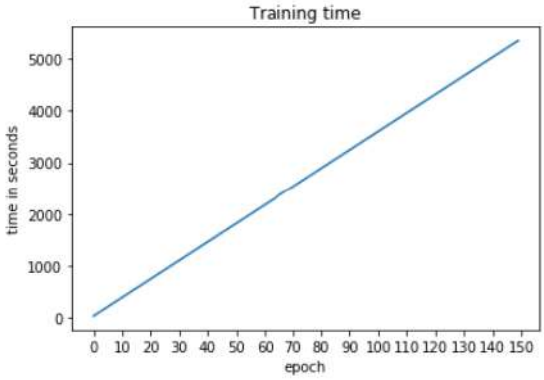

Figure 15: Training time

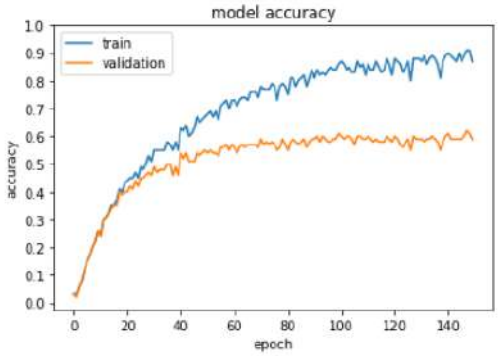

Figure 16: Accuracy's for 150 epochs

However the validation loss plateaus at around around 100 epochs, and we can say to have completed training there, since going further would just mean overfitting on the train dataset (ref Figure: 16)

% Epoch number	Train	Val
1	3	3
25	43.2	41.6
50	69.7	53.9
75	77.9	55.3
100	86.5	57.3
125	90.1	58.5
150	92.1	59.8

Table 5: Accuracy's(in %) vs number of epochs

1.6 Stride in different layers

For stride in different layers, experiments were run mainly of changing the strides parameter in the convolutional layers, 3 cases were taken

- **Net1** : Base case, as in the boilerplate code, with stride=1
- **Net2** : Setting stride=2, some maxpool layers were removed to accommodate this change
- **Net3** : Conv layers with stride=3, architecture was made different from the base case

Results are shown in Table 6

% Accuracy	Train	Val	Test
Net1	91.5	59.6	57.9
Net2	85.3	56.7	55.1
Net3	82.1	48.3	47.6

Table 6: Accuracy of the 3 different models

The results of variation of stride in the maxpooling layer has also been included in the Maxpooling layer subsection.

1.7 Results

After all the experiments and combining the best results of all the above, our final model achieved an accuracy of 63% on the test dataset and its architecture is as shown in Figure 17

One of the models in the maxpooling experiment (Net3, ref Fig: 12) has similar test accuracy, but it has around 4 times more parameters.

This model will be used in the next part of the assignment, the class-wise accuracy is also shown in Table 7.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 64, 82, 82]           1,792
            Conv2d-2          [-1, 64, 80, 80]          36,928
         MaxPool2d-3          [-1, 64, 39, 39]               0
            Conv2d-4         [-1, 128, 37, 37]          73,856
            Conv2d-5         [-1, 128, 35, 35]         147,584
         MaxPool2d-6         [-1, 128, 17, 17]               0
            Conv2d-7         [-1, 256, 15, 15]         295,168
            Conv2d-8         [-1, 256, 13, 13]         590,080
         MaxPool2d-9           [-1, 256, 6, 6]               0
           Linear-10                 [-1, 256]          65,792
           Linear-11                 [-1, 256]          65,792
           Linear-12                  [-1, 33]           8,481
================================================================
Total params: 1,285,473
Trainable params: 1,285,473
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.08
Forward/backward pass size (MB): 10.81
Params size (MB): 4.90
Estimated Total Size (MB): 15.79
----------------------------------------------------------------
```

Figure 17: Best model (63% test accuracy)

	Class name	% Accuracy
0	african hunting dog	77
1	ant	33
2	ashcan	31
3	black footed ferret	67
4	bookshop	37
5	carousel	49
6	catamaran	74
7	cocktail shaker	47
8	combination lock	38
9	consomme	78
10	coral reef	73
11	dalmatian	80
12	dishrag	66
13	fire screen	58
14	goose	58
15	green mamba	70
16	king crab	50
17	ladybug	73
18	lion	82
19	lipstick	46
20	miniature poodle	59
21	orange	86
22	organ	70
23	parallel bars	37
24	photocopier	81
25	rhinoceros beetle	78
26	slot	81
27	snorkel	69
28	spider web	57
29	toucan	74
30	triceratops	56
31	unicycle	56
32	vase	31

Table 7: Class-wise accuracy

2 Part B

We are going to be using the best model from Part A for all the remaining experiments (ref Fig 17)

2.1 Occlusion sensitivity experiment

For the occlusion sensitivity experiment, the results are based on using an (11X11) gray window being used to replace the content. The RGB value of the gray used to replace the content is taken as (211,211,211).

If the class becomes different from the original prediction, then the confidence will be set as 0, no matter what the output of running the modified image through the model. Also padding is not taken and at the edges a full (11X11) window will not be occluded/greyed out. The models outputs were softmaxed to get the probabilities.

The results obtained have been represented in the Figure 18.
Key takeaways based on the observations would be:

- In general there are specific features that stand out and are learnt by the model such as an animals eyes, nose, beak,etc.
- For the classes that the model performed badly on, interesting it seems to be because the model is focusing/learning features of the background, hence hurting generalization.
- Edges of the class objects are a prominent feature and the confidence array seems to also be an outline of the object for most well performing classes.
- Classes with accuracy around or more than 80% did not seem to be affected a lot by this experiment, as well as objets that cover a large part of the image, as expected from common intuition. As expected classes like the ladybug covering only small parts of the image, have sharp confidence decreases around the object position (ref Figure: 19).

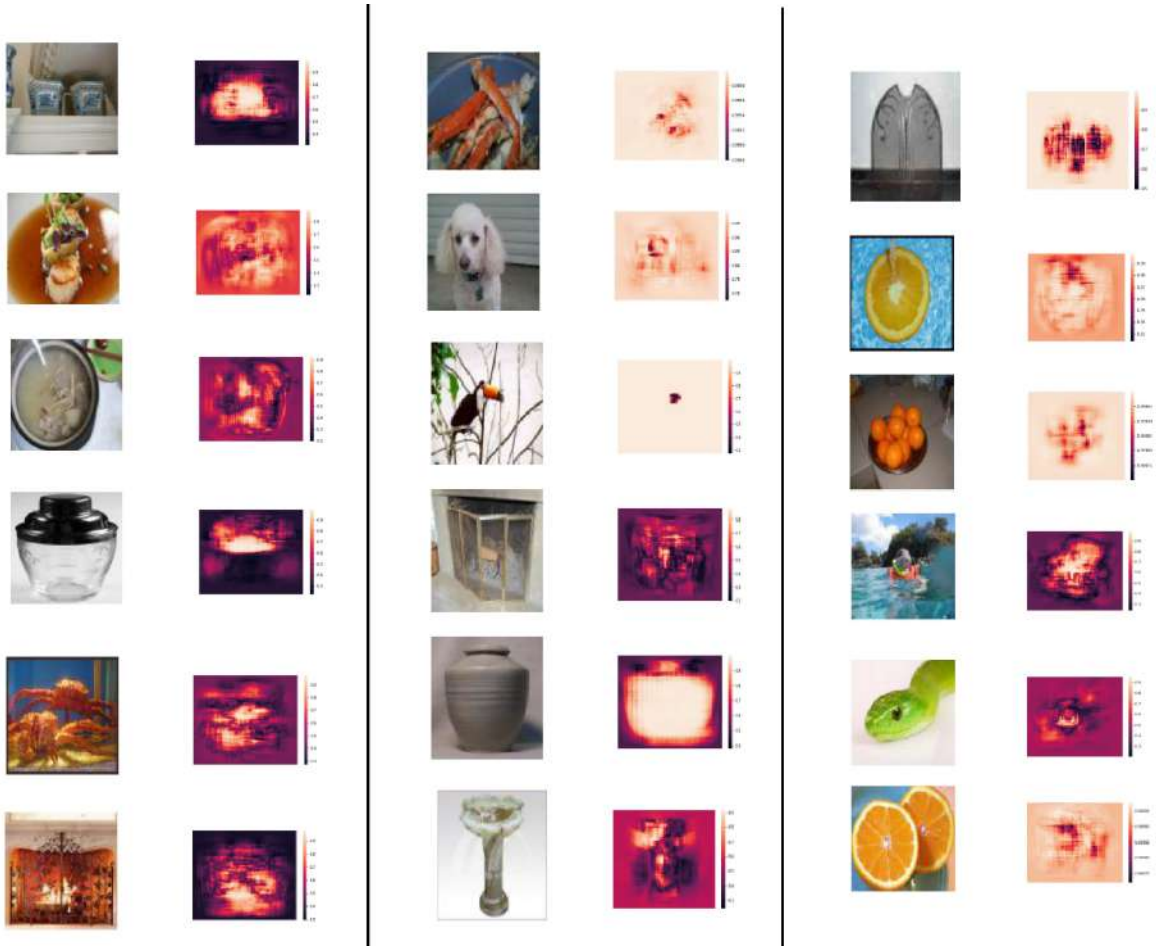

Figure 18: Occlusion sensitivity experiment Results

Figure 19: Sharp confidence decreases on small sized class objects

2.2 Filter Analysis

2.2.1 Filter Identification

2 filters each from the first 5 convolutional layers of our best model are taken for this part.

If we say that the N'th convolution layer in our model has n filters. We can choose any 2 random k'th filters in this conv layer for our filter analysis. This convolution layer has a receptive field depending on the number of preceding conv, max pool layers, and their kernel sizes.

For B test images of shape [B,3,H,W] (H=image height, W=image width), the response of the N'th conv layer is [B,n,h,w] (h=feature map height, w=feature map width).

The response of k'th filter = output[:, k, :, :]. It has a size [B,h,w]. That is each image has a response of size [h,w] and each response at (i, j) for this array corresponds to a particular patch in the image (depending on the receptive field of the layer).

For for each image, we have hxw responses for each image. From this we now find the top-5 responses among responses from all the images. Here a response is defined as the value at (i,j) instead of something like taking norm of the entire activation. The receptive field and patch is calculated for each one of these top 5 and displayed.

Note : For all the figures below the above part shows the original image from which the image patch at the bottom is extracted from and the images are ordered from left to right in decreasing order of responses.

- **Conv layer 1**
  Conv layer 1 has 64 filters with a (3X3) kernel. The output after this layer is [64,82,82] and the receptive field is (3X3). The top 5 images corresponding to the maximum response are shown in Fig 20, 21 and 22.

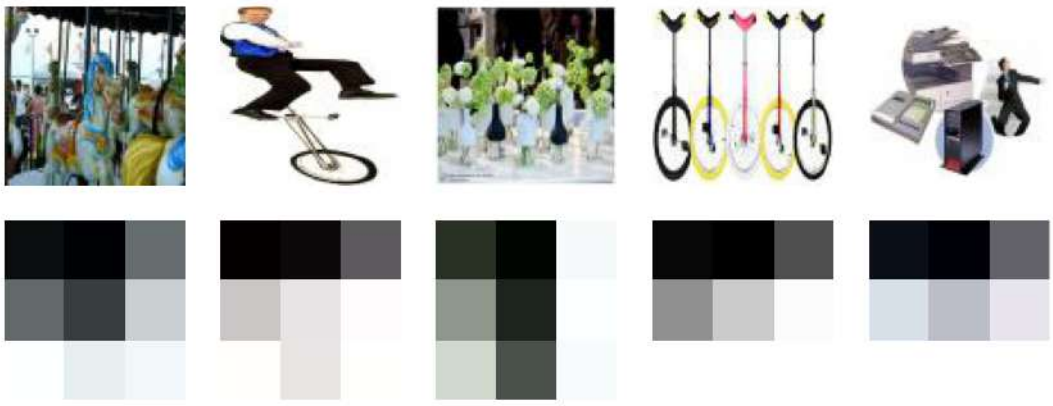

Figure 20: Layer1 - Filter 10

Figure 21: Layer1 - Filter 17

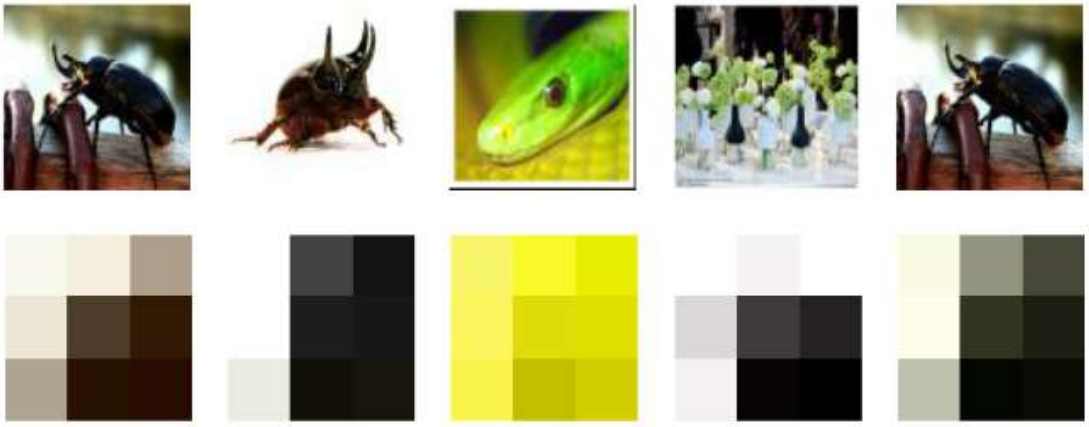

Figure 22: Layer1 - Filter 31

- **Conv layer 2**
  Conv layer 2 also has 64 filters with a (3X3) kernel. The output after this layer is [64,80,80] and the receptive field for this layer is (5X5). The top 5 images corresponding to the maximum response are shown in Fig 23 and 24.

Figure 23: Layer2 - Filter 19

Figure 24: Layer2 - Filter 3

- **Conv layer 3**
  The third convolutional layer has 128 filters with a (3X3) kernel once again. The output after this layer is [128,37,37] and the receptive field for this layer is (11X11). The top 5 images corresponding to the maximum response are shown in Fig 25 and 22.

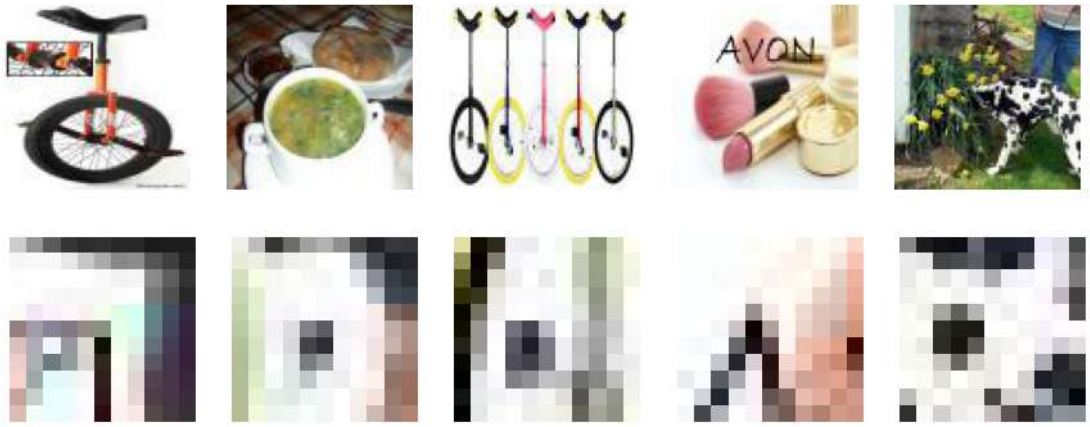

Figure 25: Layer3 - Filter 7

Figure 26: Layer3 - Filter 36

- **Conv layer 4**
  The fourth convolutional layer also has 128 filters with a (3X3) kernel once again. The output after this layer is [128,35,35] and the receptive field for this layer is (15X15). The top 5 images corresponding to the maximum response are shown in Fig 27 and 28.

Figure 27: Layer4 - Filter 2

Figure 28: Layer4 - Filter 4

There are quite some repeated imgs here but it because there are multiple lipsticks in the same image and the particular filter in Layer 5 which is filter 4 got activated at multiple (i,j) in the image.

- **Conv layer 5**
  The fourth convolutional layer has 256 filters with a (3X3) kernel. The output after this layer is [256,15,15] and the receptive field for this layer is (25X25). The top 5 images corresponding to the maximum response are shown in Fig 29, 30 and 31.

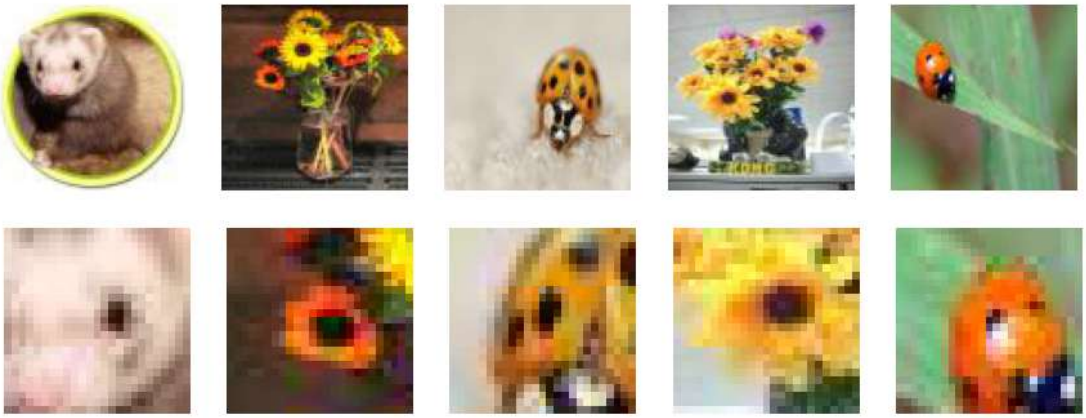

Figure 29: Layer5 - Filter 36

Figure 30: Layer5 - Filter 201

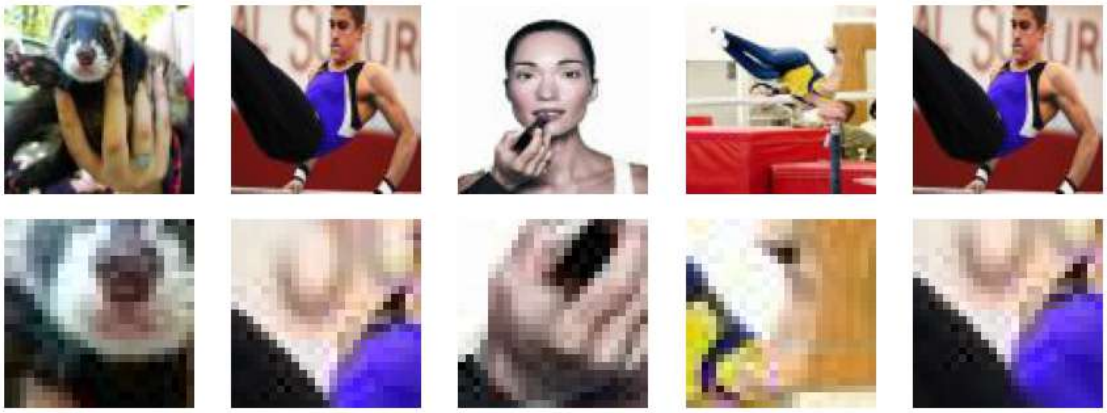

Figure 31: Layer5 - Filter 96

2.2.2 Filter Modification

For this part we will be individually turning off 10 filters from the above subsection (2 from each layer), i.e by making their weights zero. We are also only going to consider misses as those which were classified correctly before but start to mis-classify now.
For each of the figures below, the top images are the mis-classifications int the first filter and the bottom images are the mis-classifications in the second filter.(random 10)

- **Conv layer 1**

  Filter 10 and Filter 31

Figure 32: Mis-classifications in layer 1

Filter tuned off	Initial accuracy	Number of misses	New accuracy
Filter 10	64.0	50	62.5
Filter 31	64.0	12	63.7

Table 8: Conv layer 1 filters turned off individually

- **Conv layer 2**

  Filter 19 and Filter 3

Figure 33: Mis-classifications in layer 2

Filter tuned off	Initial accuracy	Number of misses	New accuracy
Filter 19	64.0	112	60.6
Filter 3	64.0	21	63.4

Table 9: Conv layer 2 filters turned off individually

- **Conv layer 3**

  Filter 7 and Filter 36

Figure 34: Mis-classifications in layer 3

Filter tuned off	Initial accuracy	Number of misses	New accuracy
Filter 7	64.0	27	63.2
Filter 36	64.0	30	63.1

Table 10: Conv layer 3 filters turned off individually

- **Conv layer 4**

  Filter 2 and Filter 4

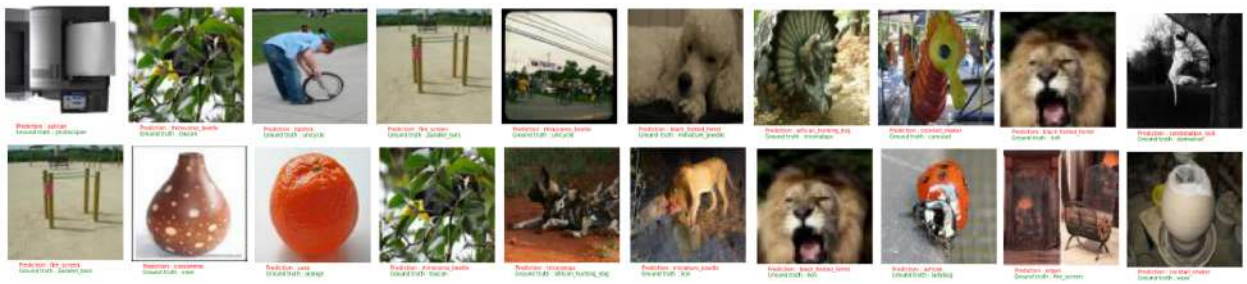

Figure 35: Mis-classifications in layer 4

Filter tuned off	Initial accuracy	Number of misses	New accuracy
Filter 2	64.0	16	63.5
Filter 4	64.0	19	63.4

Table 11: Conv layer 4 filters turned off individually

- **Conv layer 5**

  Filter 201 and Filter 96

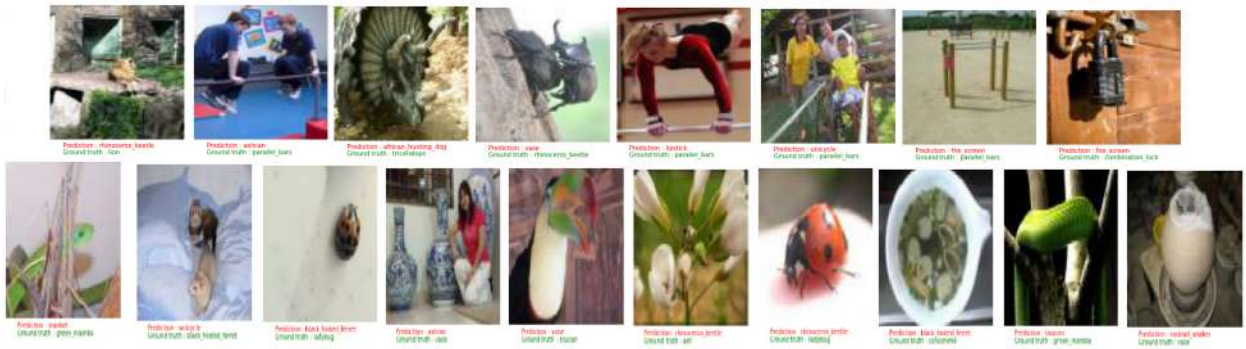

Figure 36: Mis-classifications in layer 5

Filter tuned off	Initial accuracy	Number of misses	New accuracy
Filter 201	64.0	16	63.5
Filter 96	64.0	8	63.8

Table 12: Conv layer 5 filters turned off individually

2.2.3 Summary and Conclusions

From the filter identification experiment we can observe that the model at the earlier layers tends to focus on simple features like edges, corners , places where there are sharp gradient changes etc. However as you go into the deeper layer more information is being dimension-ally reduced down and represent in a lower form. The features here can be more general like eyes, body parts, shapes, etc. which are more complex and not easily representable.

Switching off the filters also helps to further understand and infer what the model has really learnt and what features it is focussing on.