CS7015: Deep Learning (for Computer Vision)

Name: Anoubhav Agarwaal

Roll no: BE16B002

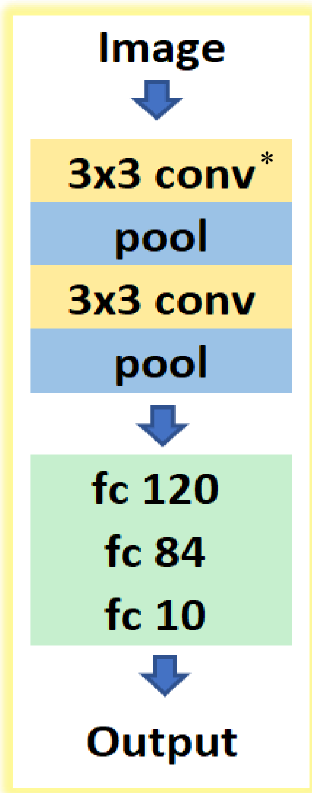
Part – 1

The following changes were made to the boilerplate code parameters: - The batch size (earlier 4) and learning rate (earlier 0.001) was increased to speed up the training process for each epoch. We settled at a batch size of 128 and learning rate of 0.01. This was determined by running the boilerplate model on all permutations of batch size: [128, 256, 512] and lr: [0.01, 0.001]. The table summarizes the results:

lr	batch_size	loss	train_accuracy	test_accuracy
0.001	128	2.112819	0.215678	0.22864
	256	2.288286	0.155022	0.15799
	512	2.306807	0.115798	0.11689
0.010	128	1.442024	0.478140	0.48958
	256	1.560710	0.435316	0.45656
	512	1.971460	0.264850	0.28071

Network 1

The first network architecture is same as the boiler plate code.



We define the following new parameters:

- **l1_oc:** the number of output channels in the first convolutional layer.
- **filter_multiplier:** the multiplicative increase in the number of filters for every convolutional layer compared to the preceding conv layer.
- **kernel_size:** the filter size for each convolutional layer.
- **stride:** the stride of each convolutional filter.

Parameters tested:

- Number of epochs = 30
- learning rate = 0.01; batch size = 128;
- l1_oc = [6, 12, 18].
- Filter_multiplier = [1.5, 2]
- Kernel_size = [3, 5]
- Stride = [1, 2]

Note: l1_oc and filter_multiplier parameters together dictate the **width** of the convolutional network.
*Kernel sizes of both 3x3 and 5x5 were tried. Only 3x3 is shown in the architecture diagram.

Analysis of Net 1 results

1) **Best network configuration**: Test accuracy of **67.04%** using **173k parameters**.

L1_oc = 18 and filter_multiplier = 2. The widest possible network with the most parameters achieved the best test accuracy.

Kernel size of 3 and stride 1. These are usually the values observed for a convolutional filter (e.g., in the case of VGG net) and as expected gave the best results in net 1.

run	epoch	loss	train_accuracy	epoch duration	run duration	test_accuracy	total parameters	l1_oc	filter_multiplier	kernel_size	stride
21	12	0.70264	0.75468	11.333719	137.952505	0.6704	173026	18	2.0	3	1

2) **Width of the network**: We clearly observe that with the increase in the width of the network (and subsequently the number of parameters) the mean accuracy for both test and train set increases. Also, the loss is lower. Filter_multiplier of 1.5 and 2 give quite similar results.

l1_oc	filter_multiplier	loss	train_accuracy	test_accuracy
6	1.5	1.394118	0.490225	0.458456
	2.0	1.343711	0.508717	0.470668
12	1.5	1.186520	0.570227	0.518204
	2.0	1.168696	0.577110	0.518260
18	1.5	1.078876	0.609680	0.539293
	2.0	1.054219	0.618167	0.544705

3) **Convolutional filter parameters**: Stride of 2 performed significantly worse (by mean accuracy) than stride of 1 and will not be considered in the next network. Kernel size of 3 and stride of 1 gave the best accuracy. However, this configuration was only slightly better than kernel size 5 and stride 1. Also, when the stride is 2, the kernel size of 5 gives almost 10% higher train and test accuracies compared to kernel size of 3.

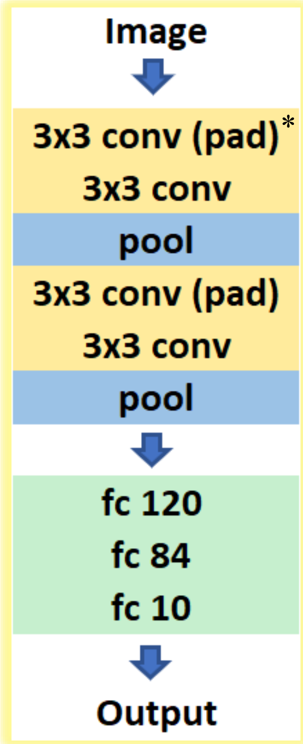
kernel_size	stride	loss	train_accuracy	test_accuracy
3	1	0.889921	0.681618	0.585004
	2	1.621003	0.403468	0.392318
5	1	0.968880	0.653581	0.565333
	2	1.337623	0.510751	0.490402

kernel_size	train_accuracy	test_accuracy
3	0.542543	0.488661
5	0.582166	0.527868

Summary & Net 2 parameter configurations

- Stride of only 1 will be considered to reduce the parameter space.
- Kernel size of 3, 5, and 7 will be explored due to interesting results in Net 1.
- Due to strong trend in the increase in accuracy with an increase in l1_oc (i.e., a proxy for the width and number of parameters of network), higher l1_oc of 18, 32, and 64 will be considered.
- Only filter_multiplier of 2 will be considered to reduce the parameter space and also due to similar performance as 1.5. We picked 2, as the train accuracy needs to rise further, and thus, we require more parameters for the next model, which will be achieved with higher filter_multiplier.

Network 2



The network architecture is almost the same as Net 1 but with the addition of two additional convolutional layers with padding of 1. This was done to achieve a higher train and test accuracies.

Parameters tested:

- Number of epochs = 30
- learning rate = 0.01; batch size = 128;
- l1_oc = [18, 32, 64].
- Filter_multiplier = [2]
- Kernel_size = [3, 5, 7]
- Stride = [1]

We added two convolutional layers with same padding to observe the effects of only increasing the no. of convolutional layers without decreasing the size of image. Hence, this should isolate the performance gained by just 2 adding conv layers.

Analysis of Net 2 results

1) **Best network configuration:** Test accuracy of **75.03%** using **4,870,654** parameters. 28th Epoch.

L1_oc = 64 and filter_multiplier = 2. The widest possible network with the most parameters achieved the best test accuracy. Kernel size of 5 gave the best accuracy.

We define overfit% = 100*(train_accuracy – test_accuracy). The overfit% for the best network configuration is extremely high of 24.5%

	run	epoch	loss	train_accuracy	epoch duration	run duration	test_accuracy	total parameters	l1_oc	kernel_size	overfit%
237	8	28	0.016232	0.99482	71.089209	2006.254373	0.7503	4870654	64	5	24.452

2) **Epochs:** We observe that with the increase in the number of epochs, the loss decreases, the training and testing accuracy increase. The disparity between the train and test accuracy in Net 2 indicates significant overfitting on the training set, leading to poor generalization on the test set. The overfit% increases with the increase in the number of epochs.

epoch	**loss**	**train_accuracy**	**test_accuracy**	**overfit%**
1	2.21909	0.158178	0.254411	0
5	1.18948	0.576427	0.584967	0.177556
9	0.719404	0.748044	0.661889	8.61556
13	0.431949	0.848084	0.663333	18.4751
17	0.234653	0.918202	0.658233	25.9969
21	0.135134	0.953464	0.670867	28.2598
25	0.0896555	0.969569	0.682633	28.6936
29	0.0579632	0.980733	0.686378	29.4356

3) **Kernel_size and L1_oc:** In the first figure, it's evident that with the increase in l1_oc, there is an increase in the mean train and test accuracies. Also, overfit% is only slightly more. Hence, higher l1_oc and thus, a wider CNN base with a greater number of parameters is desirable. In the second figure, the kernel size of 3 performs slightly better than 5(like last time) and significantly better than 7.

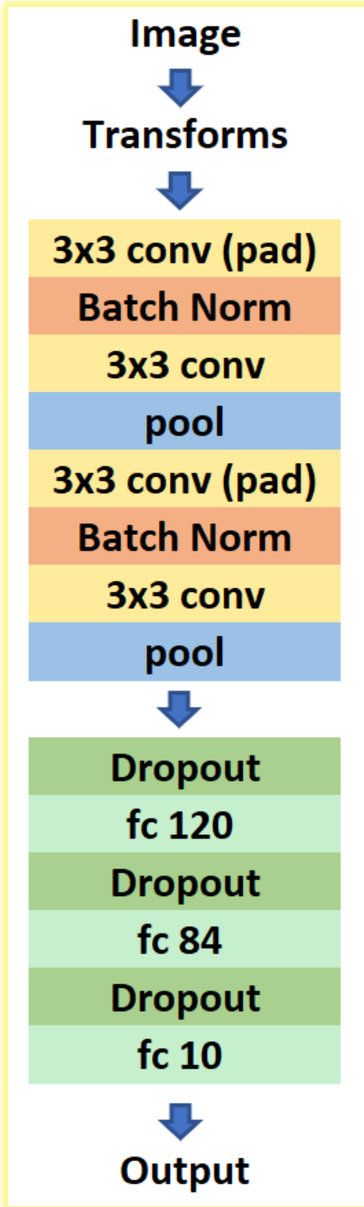
l1_oc	loss	train_accuracy	test_accuracy	overfit%	total parameters
18	0.664360	0.762174	0.596192	17.314067	654346
32	0.530685	0.809776	0.632157	18.578000	1675198
64	0.442187	0.841359	0.663361	18.666822	5719038

kernel_size	loss	train_accuracy	test_accuracy	overfit%	total parameters
3	0.495241	0.822516	0.649938	18.083244	2011954
5	0.571284	0.794897	0.642742	16.026756	2248274
7	0.570707	0.795896	0.599030	20.448889	3788354

Summary & Net 3 parameter configurations

- In Net 2, we faced the considerable issue of <u>overfitting</u> with the best network configuration (based on test accuracy) yielding <u>24% lesser test accuracy</u> compared to the training accuracy.
- Thus, with the addition of two padded convolutional, the max training accuracy reaches almost 100%. However, <u>only an 8% gain in test accuracy</u> was observed compared to Net 1. Also, the number of parameters went from 173K to about 4.8 million. Thus, in conclusion, the training accuracy and number of parameters significantly went up. But, the gains in test accuracy was less.
- To combat overfitting, following strategies will be employed:
    - Addition of **dropout layers** in the fully connected network. We will permute dropout probability of <u>0.1, 0.3 and 0.5</u>.
    - Addition of **batch normalization** layers right after the padded convolution layers.
    - **Image transformations** such as random cropping 32x32 (after padding of 4), random horizontal flips, color jittering, i.e., randomly changing the brightness, contrast, and saturation.
- Even higher value of l1_oc can be tried, due to promising results in Net 1 and Net 2 on increasing it. We will consider <u>l1_oc of 32, 64, and 96</u>.
- In Net 2, the high accuracies were obtained in the later epochs, for the following networks higher no. of epochs will be tested to attain the best accuracy.
- As the networks are getting deeper, the total training time is increasing. Exploration of the Net 1 parameter space took 131 mins, and for Net 2 it took 175 mins. As we are increasing the number of epochs, the number of permutations of parameters will be decreased.
- We <u>only consider kernel_size of 3</u> from now on. As it performed better than kernels of size 5 and 7 in the previous networks.

Network 3



The network architecture involves the addition of batch normalization and dropout layers to the convolutional base and fully connected network, respectively.

Parameters tested:

- Number of epochs = **60**
- learning rate = 0.01; batch size = 128;
- l1_oc = [32, 64, **96**].
- Filter_multiplier = [2]
- Kernel_size = [**3**]
- Stride = [1]
- **Dropout rate = [0.1, 0.3, 0.5]**

Default parameters of the batch norm layer are used. Also, random image transformations are applied to the images fed to the network as a method to augment the training dataset.

Analysis of Net 3 results

1) **Best network configuration:** Test accuracy of **87.5%** using **6,819,454** parameters. 42nd Epoch.

L1_oc = 96 and filter_multiplier = 2. The widest possible network with the most parameters achieved the best test accuracy.

The overfit% for the best network configuration is 5.98%, which is significantly lower than the results obtained in Net 2 of 24.5%. Thus, our strategies for tackling overfitting have worked to a great extent.

The test accuracy has seen a significant boost from 75% to 87.5%, and the number of parameters has only increased by 2 million.

run	epoch	loss	train_accuracy	epoch duration	run duration	test_accuracy	total parameters	l1_oc	dropout_rate	overfit%
7	42	0.189188	0.9348	109.561	4596.02	0.875	6819454	96	0.1	5.98

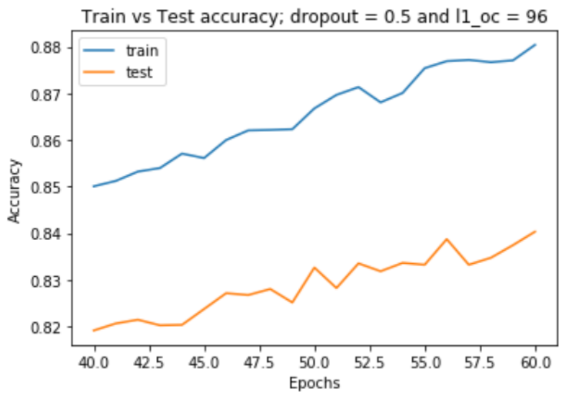
2) **Dropout rate**: With the increase in the dropout probability, the overfit% decreases. However, there is also a drop-in train (nearly 10%) and test accuracy (nearly 7%). From this figure, it is quite evident that the p parameter in dropout plays a significant role in finding the best model.

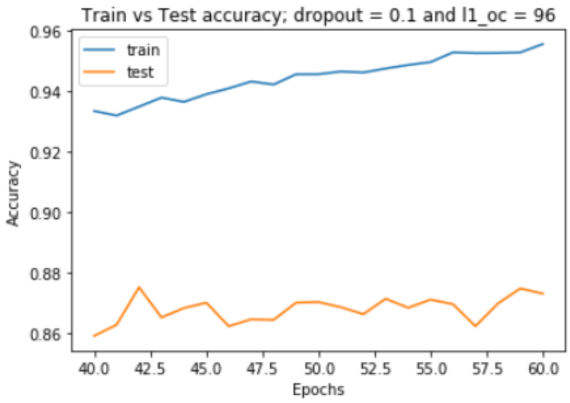
	loss	train_accuracy	test_accuracy	overfit%
dropout_rate				
0.1	0.374372	0.870216	0.827451	4.597644
0.3	0.506537	0.828132	0.799759	3.182089
0.5	0.685374	0.772994	0.757331	1.919489

3) **l1_oc:** With an increase in the number of convolutional filters in the first layer, the test accuracy increases only marginally. Compared to Net 1 and Net 2, l1_oc does not seem to have a drastic effect on the test_accuracy in the case of Net 3 (only about 2%) whereas the number of parameters is more than 4 times. The apparent gains from increasing the width of the network have finally diminished.

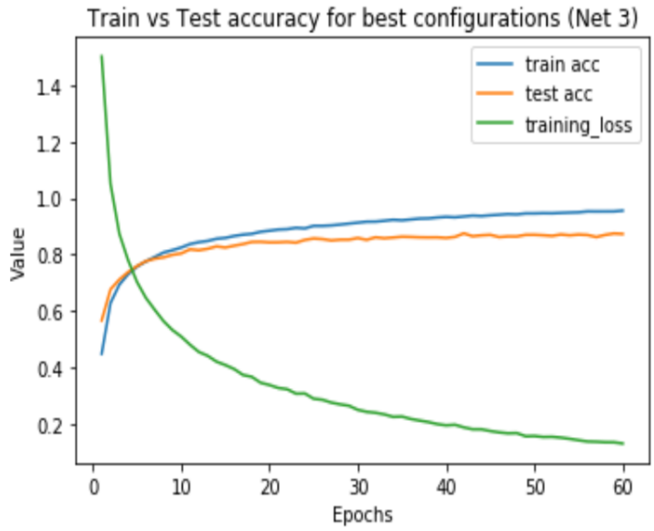
l1_oc	train_accuracy	test_accuracy	overfit%	total parameters
32	0.804032	0.780493	2.683900	1506430
64	0.828343	0.798076	3.380889	3775870
96	0.838967	0.805972	3.634433	6819454

4) **epochs**: In Fig 1, both train and test accuracy have a strong upward trend. It is clear from this that 60 epochs are not enough to fully train the network (when dropout rate is 0.5). Thus, p = 0.5(left) has lower accuracy compared to p = 0.1(right). However, the mean overfit% of p = 0.5 (1.9%) is significantly lesser than p = 0.1 (4.6%).





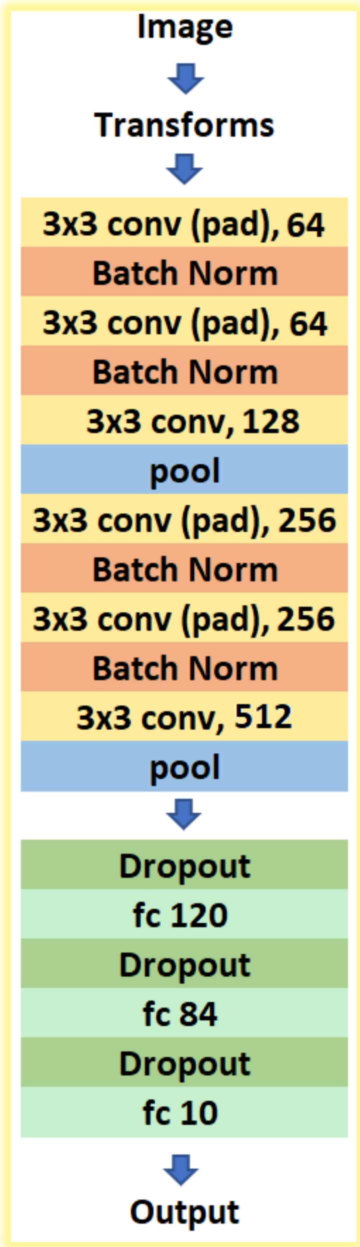
Summary & Net 4 parameter configurations:



- Net 3, overcame the major shortcoming of overfitting as seen in Net 2. It achieved a test accuracy of 87.5% with 6.8 million parameters, which was significantly higher than both Net 1 and Net 2.
- The idea behind Network 4 is to explore the depth aspect of the convolutional neural network. Whether we can achieve similar results as Net 3 using a greater number of hidden layers but with fewer/same number of parameters.
- Thus, to make the network sleeker and deeper we only consider l1_oc of 64 (instead of 96 as seen in Net 3) and add two more convolutional layers.
- A Dropout rate of only 0.3 and 0.5 is considered. Even though p = 0.1 gave the best results in Net 3, we are not going to use it due to (i) high overfitting, (ii) reduce the parameter search space.
- The number of epochs is increased from 60 to 100 to ensure better 'convergence' in test accuracy plot (i.e., to observe a plateau in the test accuracy vs. epochs plot)

Network 4

The network architecture involves the addition of a pair of convolutions (padded) + batch normalization layers to Net 3. Also, the number of filters is not doubled after **every** convolutional layer (unlike Net 2 and 3 having a filter_multiplier of 2). This is done to explore the effects of depth instead of the width of the CNN.



Parameters tested:

- Number of epochs = **100**
- learning rate = 0.01; batch size = 128;
- l1_oc = [**64**].
- Filter_multiplier = [2]
- Kernel_size = [3]
- Stride = [1]
- **Dropout rate = [0.3, 0.5]**

We use default parameters of the batch normalization layer. Also, random image transformations are applied to the images fed to the network as a method to augment the training dataset. (Same as Net 3)

Analysis of Net 4 results

1) **Best network configuration:** Test accuracy of **88.74%** using **4,403,518** parameters. 91st Epoch. Dropout = 0.3

The overfit% for the best network configuration is 8%. Which is higher than the results obtained in Net 3 of 6%.

The test accuracy is slightly better by 1.2 %, and the number of parameters has only decreased by 2.4 million.

Thus, we were able to achieve slightly better performance compared to Net 3 by using a deeper and sleeker network with almost 35% fewer trainable parameters.

run	epoch	loss	train_accuracy	epoch duration	run duration	test_accuracy	total parameters	l1_oc	dropout_rate	overfit%
1	91	0.0983406	0.96714	84.4099	8562.45	0.8874	4403518	64	0.3	7.974

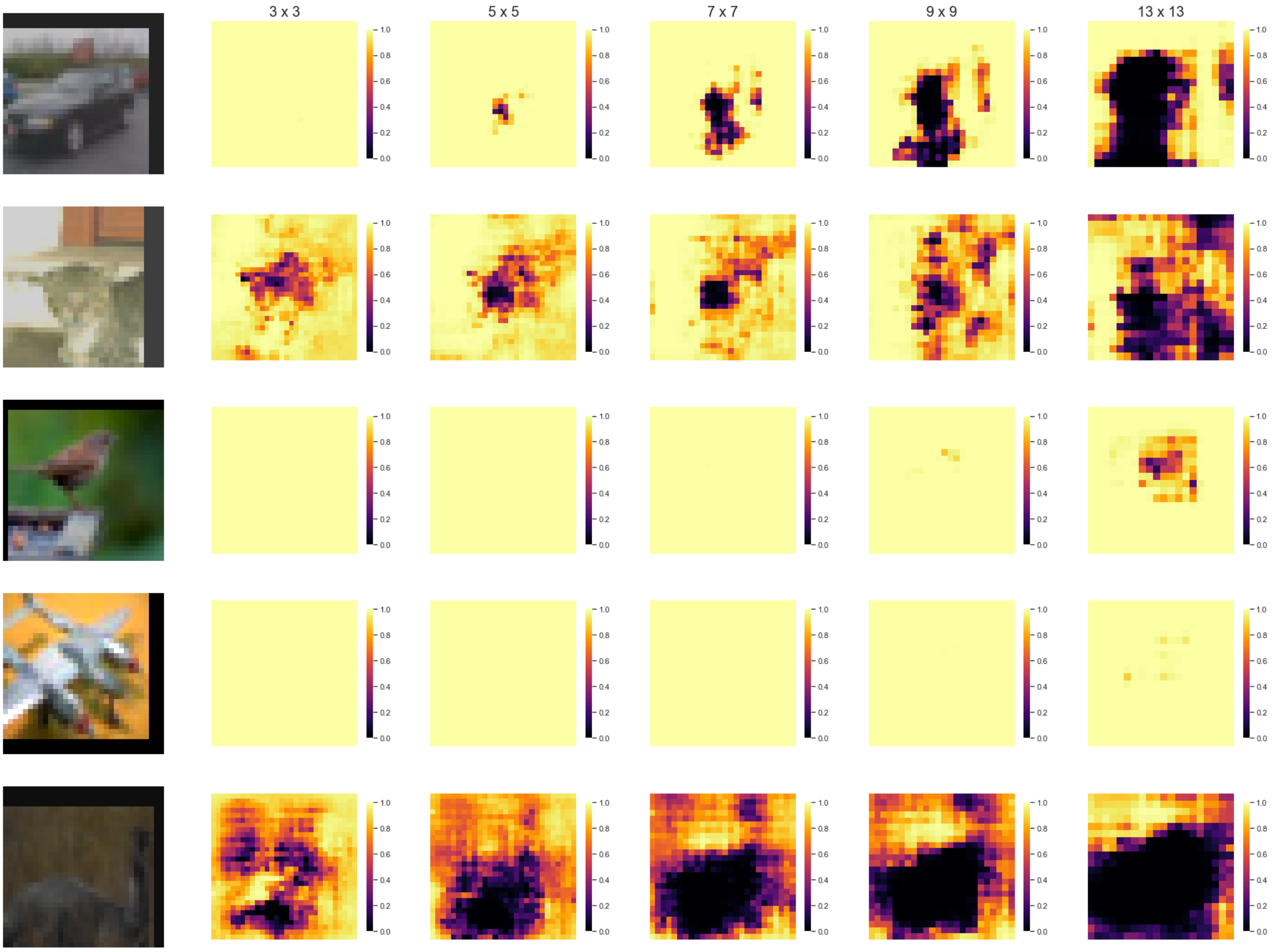
2) A **Dropout rate** of 0.3 performed much better (about 3% higher test accuracy) compared to a dropout rate of 0.5. The model was significantly overfitting. At no epoch did the model with a dropout rate of 0.5 cross an accuracy higher than 88%.

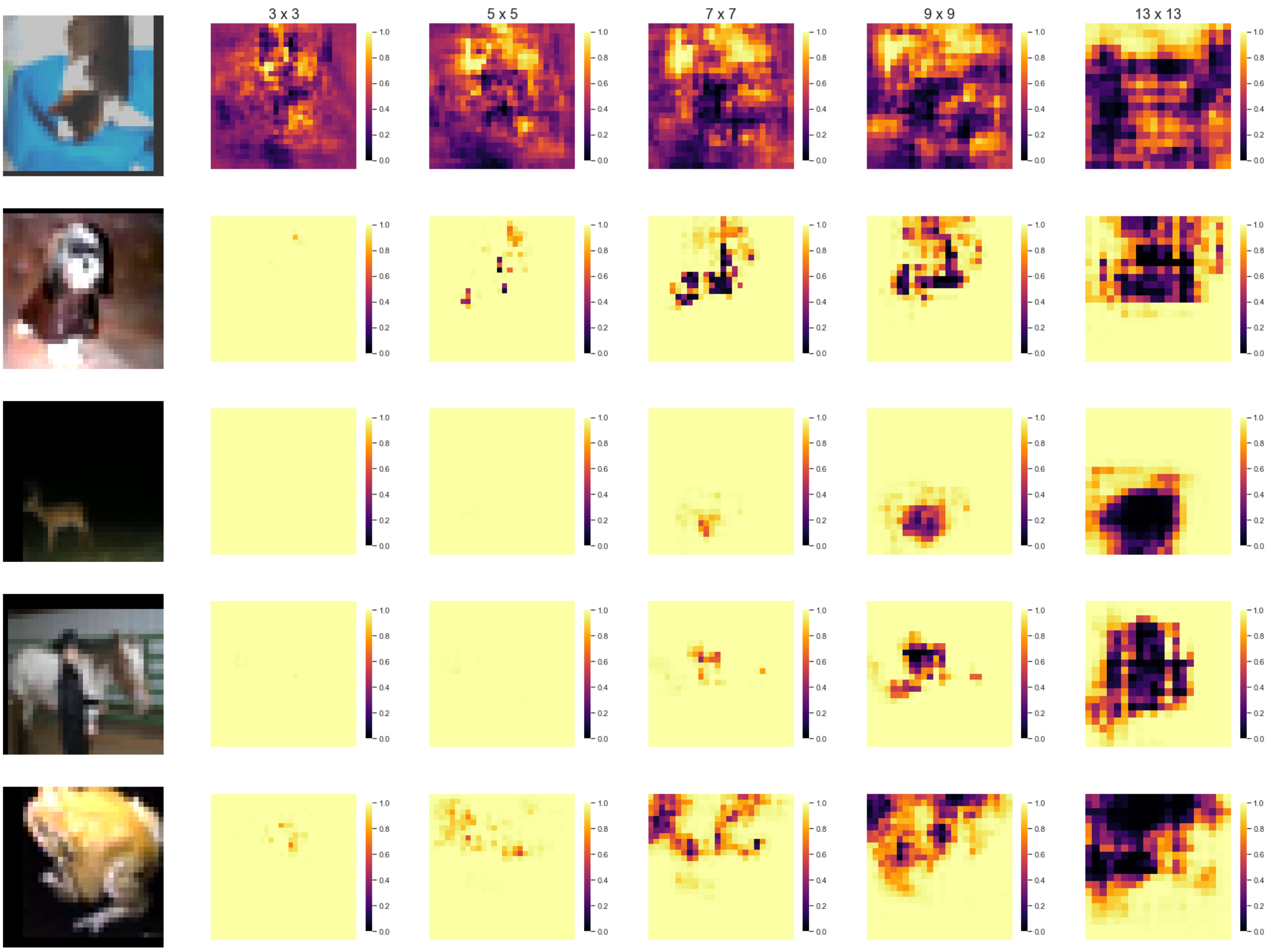
dropout_rate	loss	train_accuracy	test_accuracy	overfit%
0.3	0.282095	0.904788	0.851131	5.57996
0.5	0.403003	0.869141	0.826185	4.52628

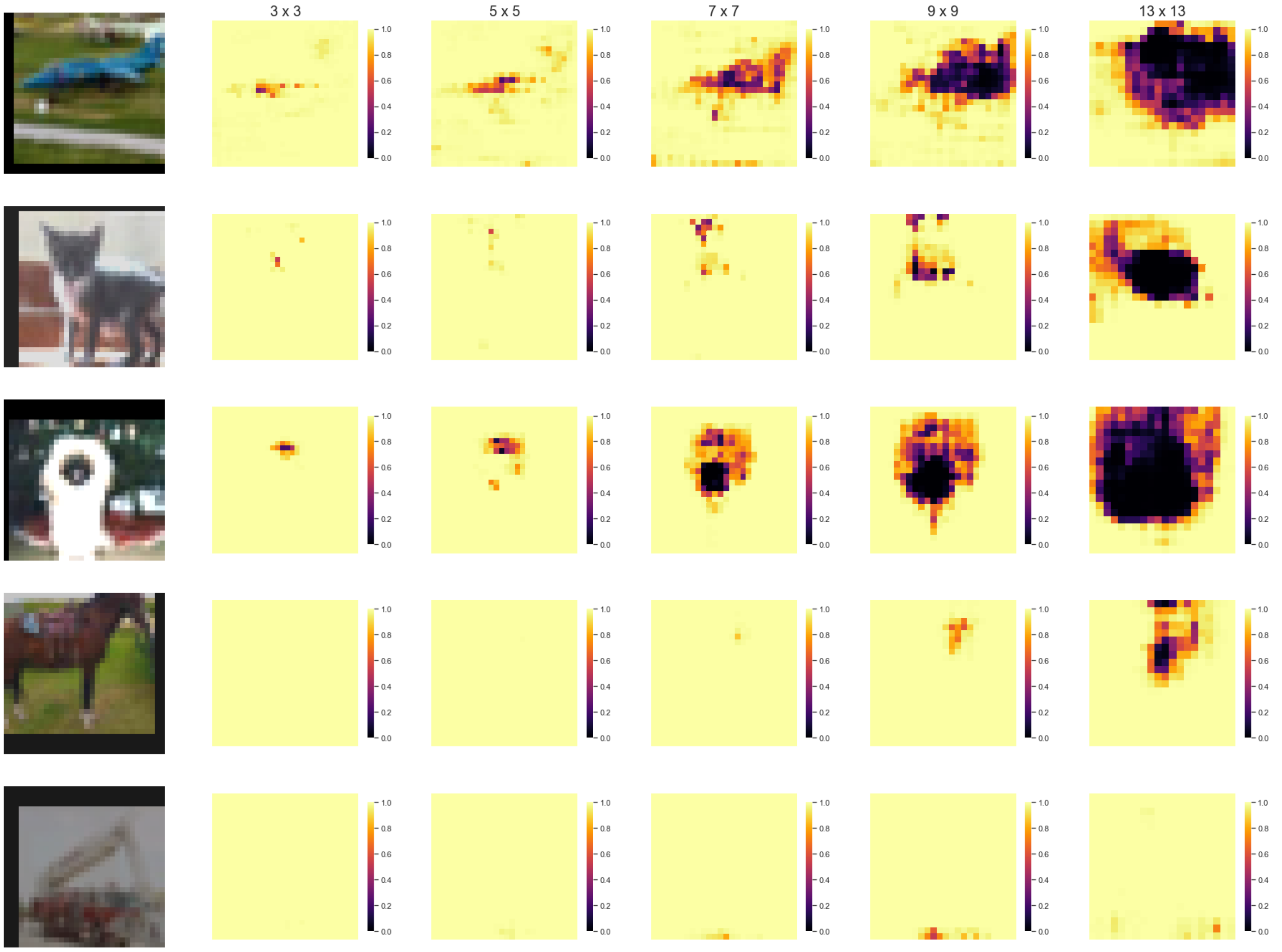
The mean overfit% observed for all epochs having a test accuracy score > 88% was 8.13%. Thus, there is a scope of improving the model even further, possibly by applying heavier augmentation, test time augmentation, and different dropout rates.

Note: We use **the best configurations of Net 4** for Part-2. We set the network in evaluation mode. This is necessary while using dropout as during training some neuron activations are set to 0 whereas in evaluation mode it considers all of them while making an inference. Thus, the test accuracy is now 90%.

Part – 2.1: Occlusion sensitivity analysis

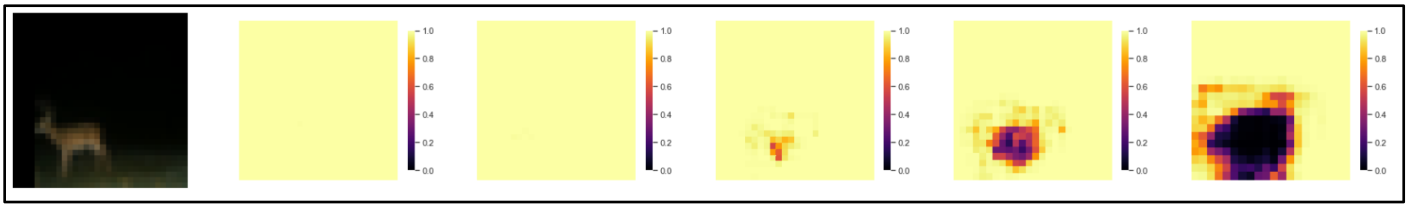




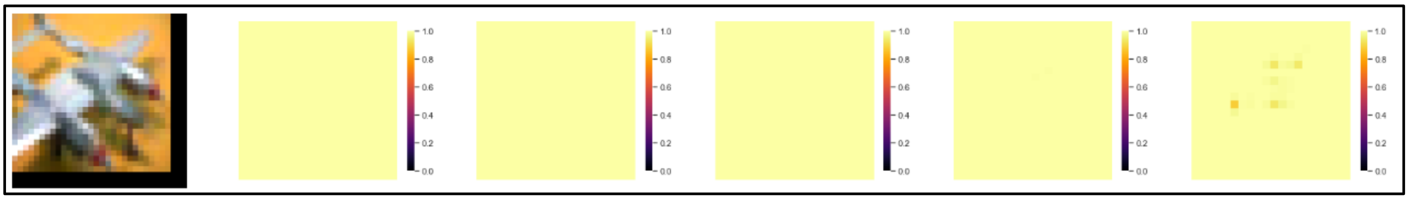


<u>General Observations:</u>

- With the increase in the occlusion window size (i.e., KxK), we observe an overall decrease in confidence, i.e., the pixel-wise sum of the confidence map. This intuitively makes sense as on increasing the size of the gray patch used to occlude an object; it becomes more challenging to identify.
- When the crucial locations (which help in identifying the true class) of an image are occluded, there is a drastic drop in the probability of the true class. (Refer to deer image below) When the gray patch is present at the bottom left corner, i.e., where the deer is present, the probability of the deer class quickly drops to 0 with the increase in the patch size.



- In a loose sense, we can localize the object by measuring wherein the image do we see a drastic drop in confidence of the true class on covering it.
- In some images, we observed that occlusion did not result in a significant drop in probability. E.g., In the case of the plane below, I hypothesize that replacing the (mostly) gray pixels of the plane with a gray patch does not affect the probability significantly. Also, vehicles, in general, have a higher number of uniquely defining characteristics present in dispersed locations in the image compared to the animals.



Note: Net 4 takes random 32x32 crops from a 40x40 padded image before running inference. Due to which we see black horizontal and vertical bars in the image. No further padding was performed on the input 32x32 images to generate the confidence maps. Thus, using a KxK occlusion window generates a (32-K+1, 32-K+1) sized confidence map.

Part – 2.2.2 Filter Modification

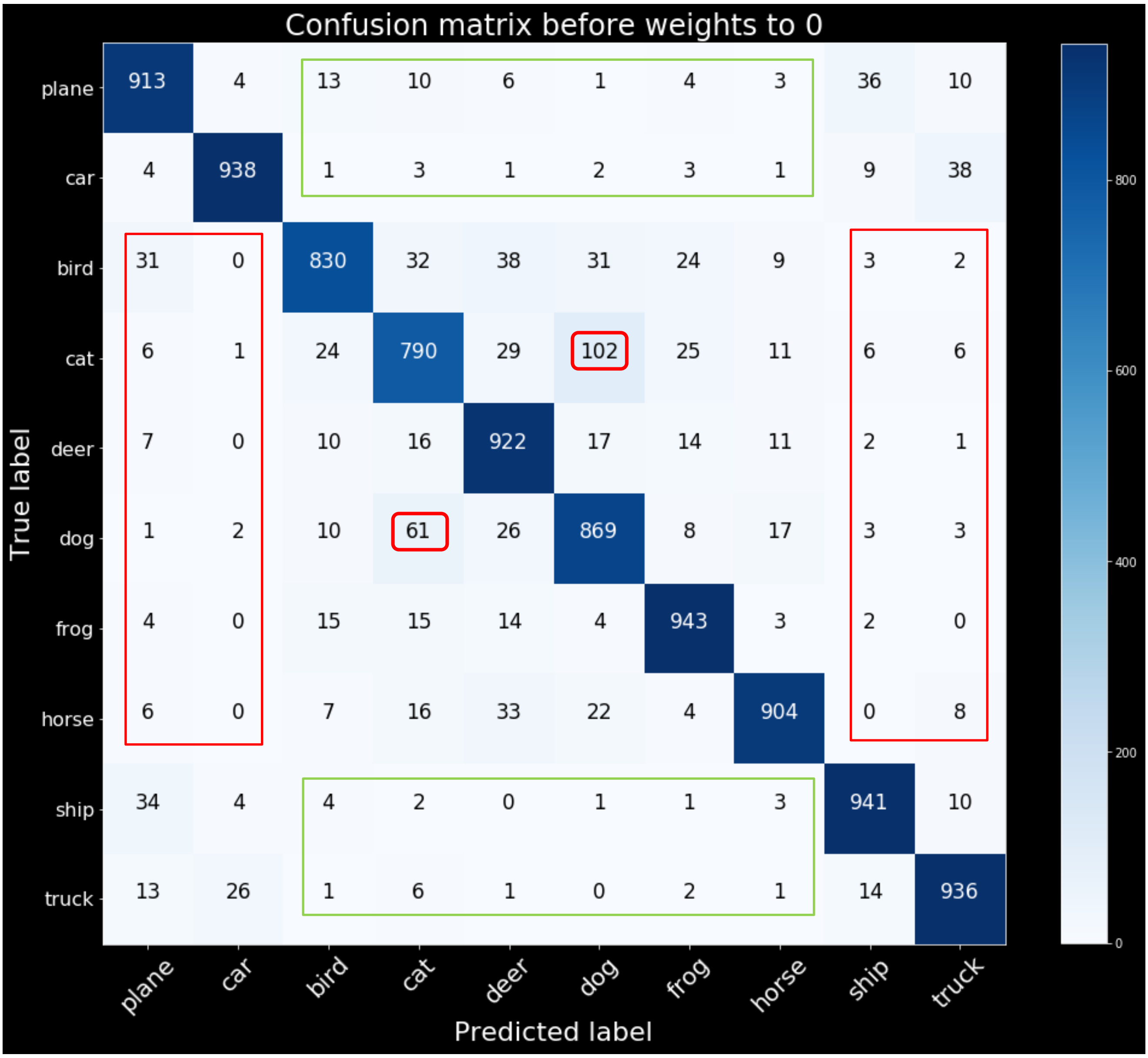
On setting the weights of the 10 filters (first two filters in the first five convolution layers of Net 4) to zero. We observe the following differences (summarized in Table 1):

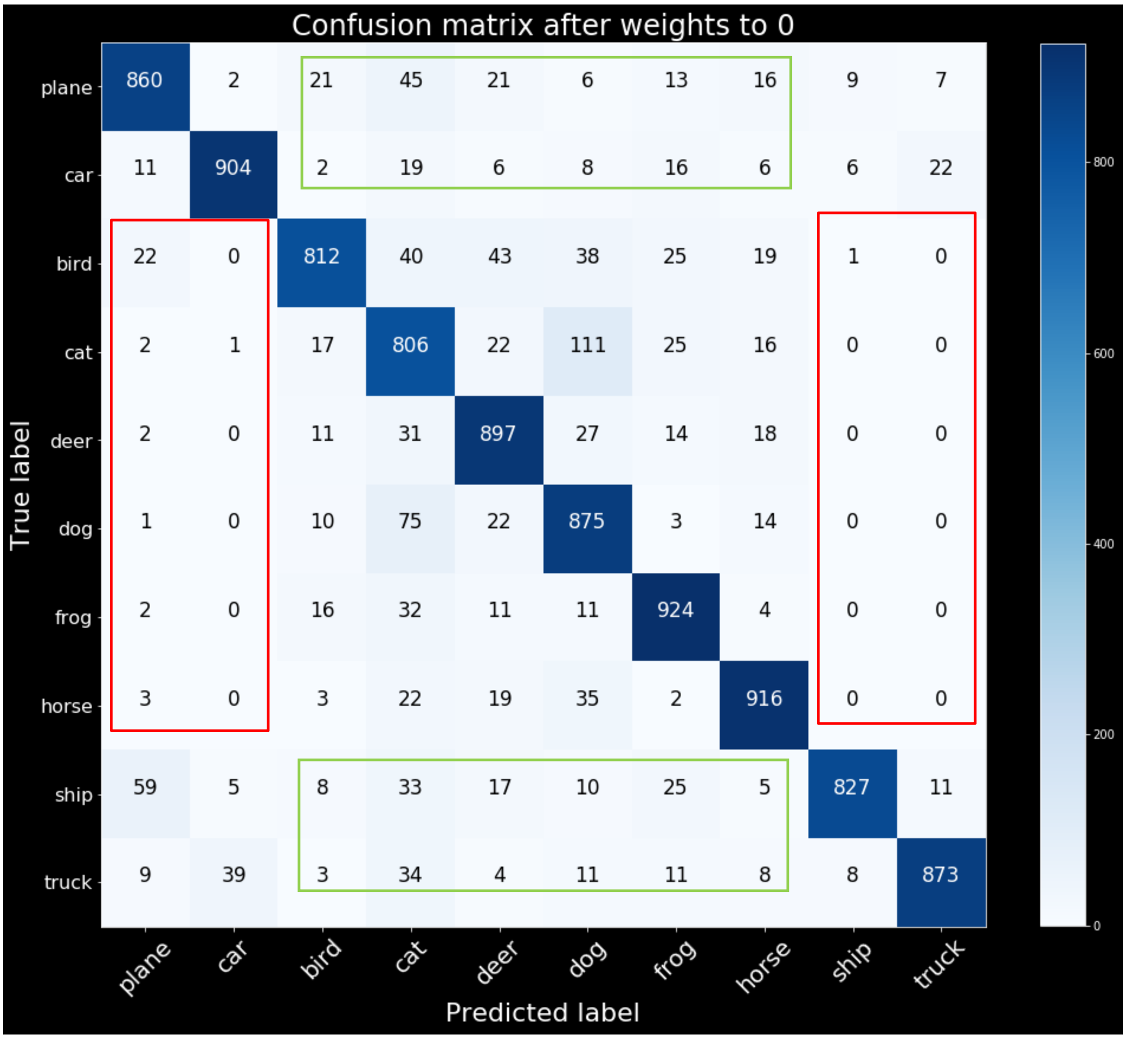
- There is a **3% drop** in overall test accuracy.
- Surprising, there is a **12% and 6% increase** in test accuracy for the classes **cat and bird**, respectively.
- There is also a significant drop in the accuracy for **transportation-related classes**: 4% for planes, **19% for ships** and 5% for trucks.
- The car and horse classes were unaffected in terms of overall accuracy.

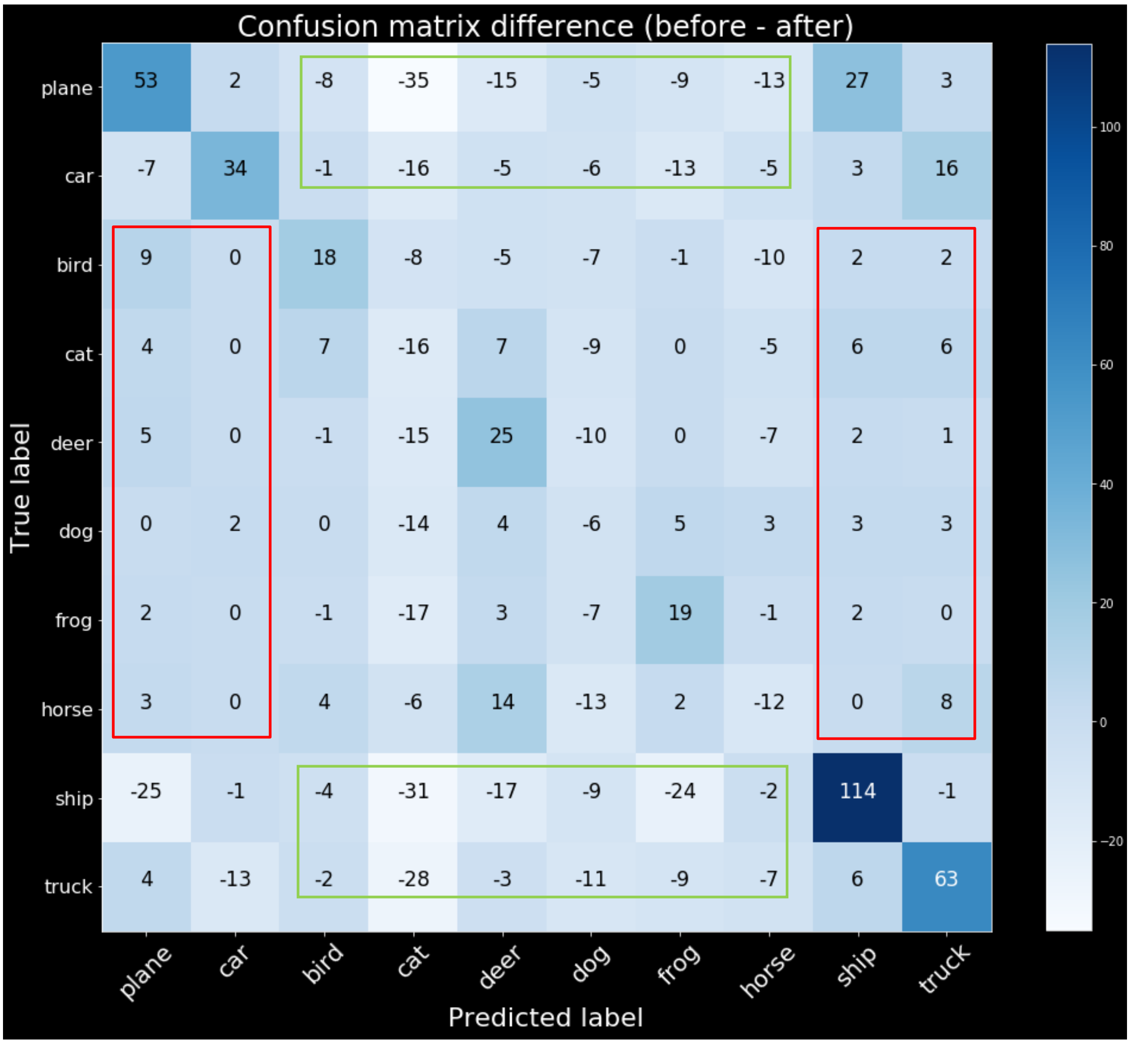
	Accuracy %		
Class	**Before**	**After**	**Difference**
plane	86	82	4
car	96	96	0
bird	75	81	-6
cat	70	82	**-12**
deer	96	92	4
dog	84	81	3
frog	86	88	-2
horse	88	88	0
ship	100	81	**19**
truck	92	87	5
Overall	89.86	86.94	2.92

Table 1: Class-wise accuracy during filter modification

Confusion Matrices (Before, After and Difference)

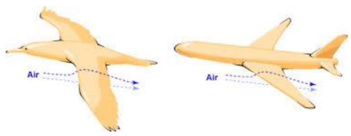



Confusion matrix after weights to 0
True label
Predicted label
plane car bird cat deer dog frog horse ship truck
plane 860 2 21 45 21 6 13 16 9 7
car 11 904 2 19 6 8 16 6 6 22
bird 22 0 812 40 43 38 25 19 1 0
cat 2 1 17 806 22 111 25 16 0 0
deer 2 0 11 31 897 27 14 18 0 0
dog 1 0 10 75 22 875 3 14 0 0
frog 2 0 16 32 11 11 924 4 0 0
horse 3 0 3 22 19 35 2 916 0 0
ship 59 5 8 33 17 10 25 5 827 11
truck 9 39 3 34 4 11 11 8 8 873
800
600
400
200
0


Confusion matrix difference (before - after)
True label
Predicted label
plane car bird cat deer dog frog horse ship truck
plane 53 2 -8 -35 -15 -5 -9 -13 27 3
car -7 34 -1 -16 -5 -6 -13 -5 3 16
bird 9 0 18 -8 -5 -7 -1 -10 2 2
cat 4 0 7 -16 7 -9 0 -5 6 6
deer 5 0 -1 -15 25 -10 0 -7 2 1
dog 0 2 0 -14 4 -6 5 3 3 3
frog 2 0 -1 -17 3 -7 19 -1 2 0
horse 3 0 4 -6 14 -13 2 -12 0 8
ship -25 -1 -4 -31 -17 -9 -24 -2 114 -1
truck 4 -13 -2 -28 -3 -11 -9 -7 6 63
100
80
60
40
20
0
−20

Observations from confusion matrices

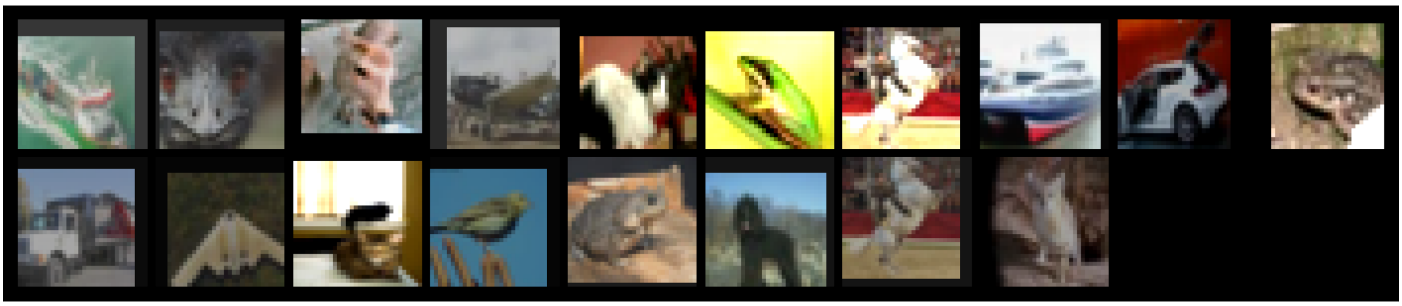
- Before weights were set to zero (Refer 1st confusion matrix):
  - ➢ The low relative accuracy of the cat class (only 70% compared to the overall test accuracy of 90%) is explained due to the high number of misclassifications (102) of cats as dogs.
  - ➢ Similarly, the number of dogs misclassified as cats are also high (61). Hence, trained **Net 4** finds it difficult to differentiate cats and dogs.
  - ➢ Animals, in general, are being misclassified mostly for other animals and not for vehicles (Refer red boxes). Vehicles, in general, are being misclassified mostly for other vehicles and not for animals (Refer green boxes). We observe that the values in these boxes are small. Also, the remaining values, i.e., not in the box, are much higher.
  - ➢ The only high value in the boxes are for the misclassification of planes as birds and vice-versa. This is fascinating, as our model finds it more challenging to differentiate planes and birds, compared to any other animal. Planes were invented to mimic flight in birds and have structural similarity to that of birds.



  - ➢ Much further analysis can be done by visualizing the adversarial examples. E.g., the case when the model misclassifies say a cat for a car (only happened once in the entire test set).
- After weights were set to zero (Refer 2nd confusion matrix):
  - ➢ The misclassification of animals for vehicles dropped even further as can be seen by the sparsity of the red boxes in the second confusion matrix.
  - ➢ However, the misclassification of vehicles for animals increased as can be seen by the high values in the green boxes.
  - ➢ The above two points are seen in the drop-in classification accuracy (Refer table 1) for ships (19%), trucks (5%), and planes (4%) and increase in classification accuracy for cats (12%) and birds (6%).
  - ➢ Differentiation between cats and dogs by the model has worsened after modification.
- The difference in accuracy before and after modification (Refer 3rd confusion matrix):
  - ➢ 114 images of ships, 63 images of trucks, 53 images of planes and 34 images of cars are now misclassified after modification.
  - ➢ Many vehicles are now classified as animals after modification. This can be seen by the large number of negative values in the green boxes.

These drastic changes in the results were obtained by modifying **only ten filters out of the 1280 3x3 filters** in Net 4. The importance of modifying the earlier layer filters compared to the later ones can also be explored.

Some images which were initially classified correctly but are now misclassified after filter modification (picked from the first 256 images of the test data loader, only 18 met the criterion)

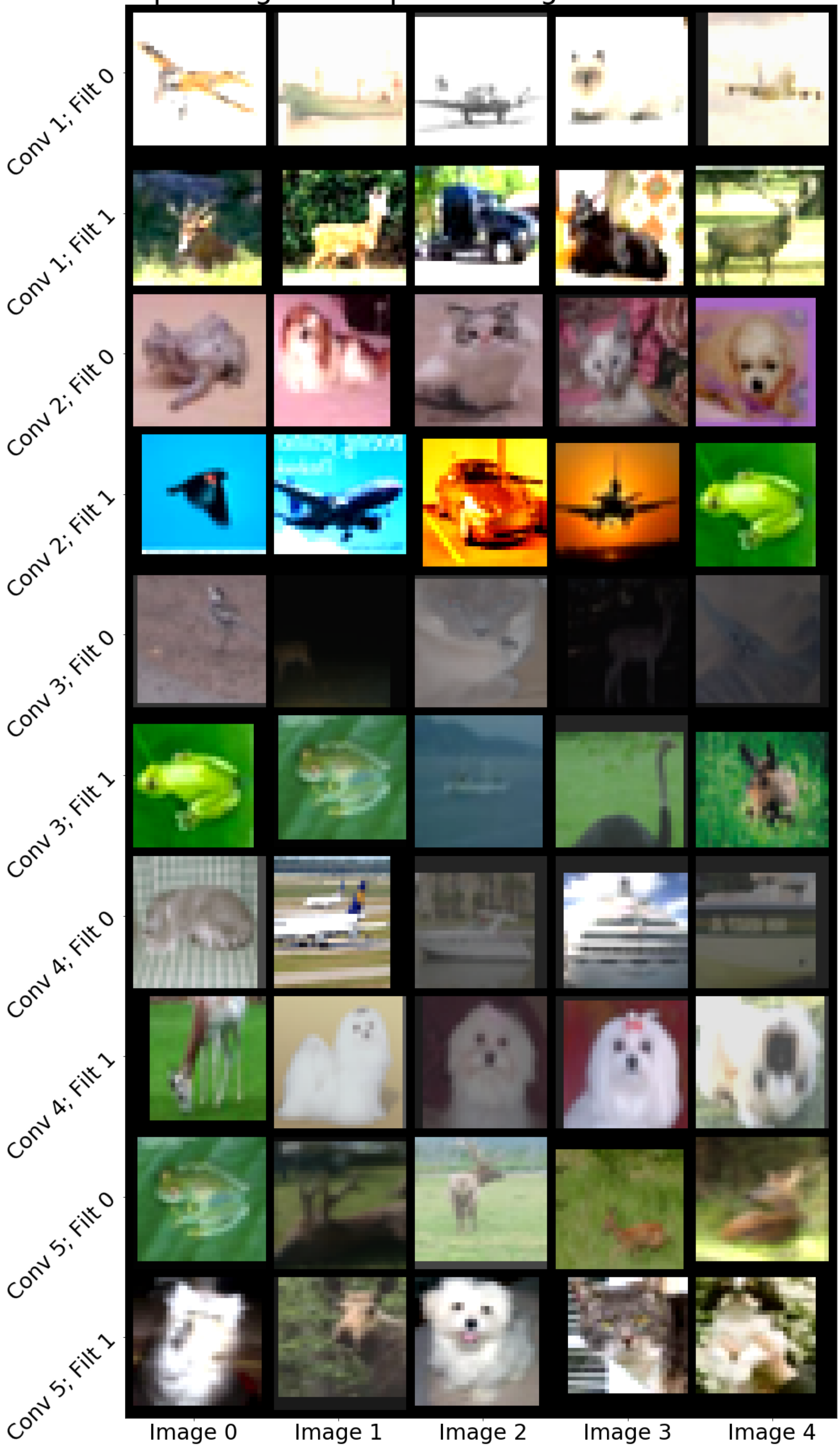


Part – 2.2.1: Filter Identification

The first two filters from each of the first five convolutional layers were chosen arbitrarily for this analysis. The code is flexible enough to pick any two filters from each of the first five conv layers.

The response of the filter was measured by taking the **pixel-wise sum of the feature map** obtained after convolving the input with the filter followed by the **batch normalization layer**. (Here, input to the next layer is from the preceding layer's output).

Top-20 highest response images for chosen filters



Filter type	plane	car	bird	cat	deer	dog	frog	horse	ship	truck	max_class
Conv 1; Filt 0	6	1	2	3	2	1	0	1	3	1	plane
Conv 1; Filt 1	1	2	0	5	7	0	1	2	0	2	deer
Conv 2; Filt 0	0	0	1	9	1	5	4	0	0	0	cat
Conv 2; Filt 1	2	1	3	3	1	2	3	0	3	2	bird
Conv 3; Filt 0	3	0	3	6	5	2	0	0	0	1	cat
Conv 3; Filt 1	4	0	3	1	4	2	3	0	3	0	plane
Conv 4; Filt 0	5	1	1	1	1	0	0	2	9	0	ship
Conv 4; Filt 1	0	0	4	2	2	10	1	1	0	0	dog
Conv 5; Filt 0	0	0	5	0	10	1	2	2	0	0	deer
Conv 5; Filt 1	0	1	3	10	1	4	1	0	0	0	cat

Table 2: The table shows the frequency of occurrence of a class amongst the top 20 images (in terms of maximum response in the filter) for each chosen filter.

From **Table 3**, we see that the average filter response for the top 20 images decreases for the deeper convolutional layers.

This may be because the earlier layers pick up more simple features like edges or color gradients, which are commonly present in many images.

Whereas, the later feature maps only activate when detecting more complicated feature abstractions which build on top of the simpler features identified by the filters in the earlier layers. Thus, this table shows that the feature maps become sparser as we go deeper.

Filter type	Avg. response of T20 images	Max response
Conv 1; Filt 0	578.772	637.594
Conv 1; Filt 1	285.001	301.407
Conv 2; Filt 0	332.098	399.211
Conv 2; Filt 1	-13.1319	-12.5783
Conv 3; Filt 0	52.3936	78.2592
Conv 3; Filt 1	29.0425	89.4517
Conv 4; Filt 0	67.9336	107.099
Conv 4; Filt 1	87.874	101.344
Conv 5; Filt 0	4.01055	21.7569
Conv 5; Filt 1	11.5044	28.9369

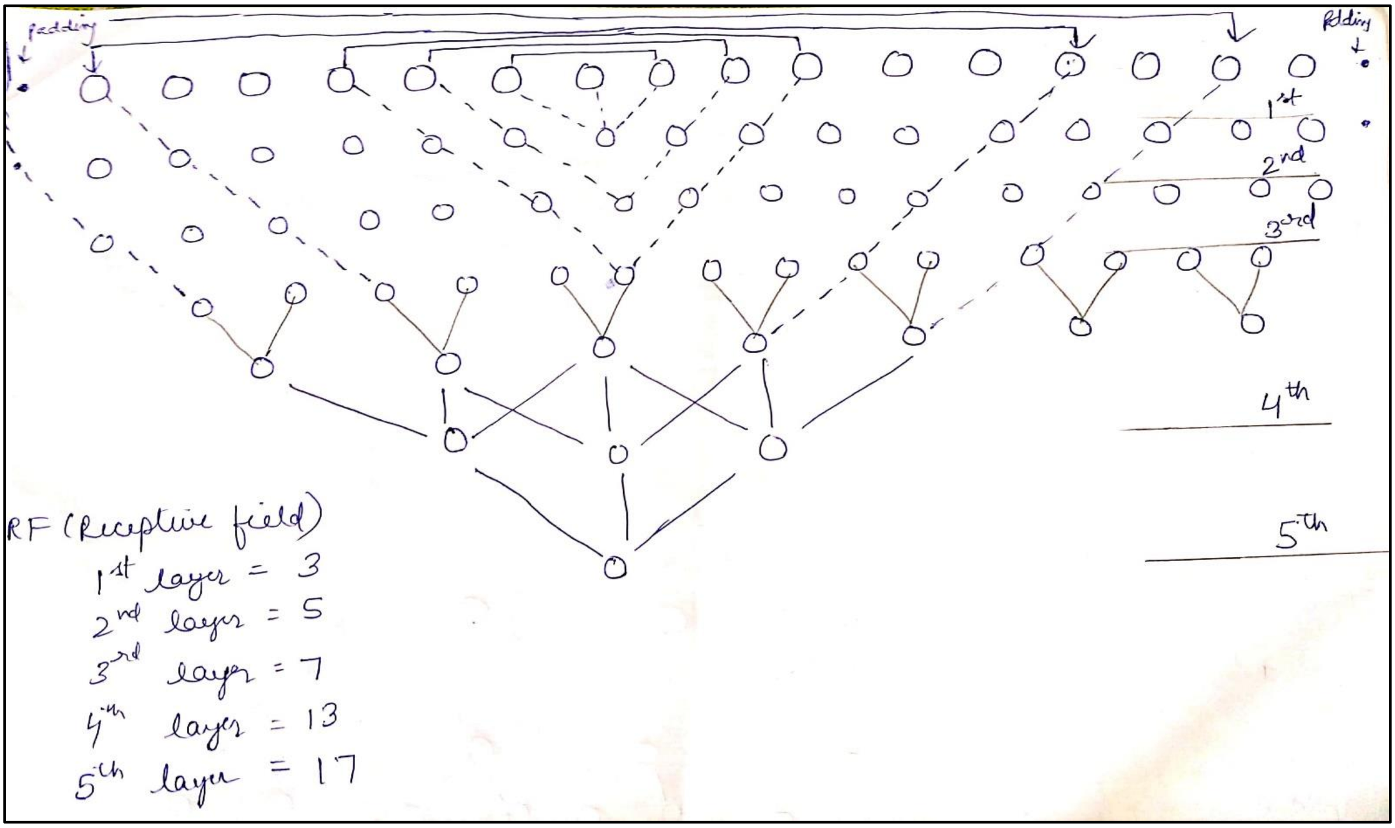
Table 3: The table shows the average filter response amongst the top 20 images for each chosen filter. This is measured by taking the average of the pixel-wise sum of the feature maps obtained after convolution + Batch normalization (but before applying ReLU non-linearity) for the Top 20 images.

In **Conv2; Filt1,** the average negative is quite peculiar. This means that for all images in the test set, the feature map is full of zeros (after passing through ReLU activation) and thus does not get activated.

This can be an instance of the **Dying ReLU problem**. This problem is aggravated when the learning rate is set too high (All my architectures were trained using 0.01 learning rate, instead of 0.001). High learning rate can lead to a large gradient flowing through a ReLU neuron. This could cause the weights to update in such a way that the neuron will never activate on any datapoint again.

Obtaining maximal response patches from the Top-K images

To obtain the maximal response patches in the image, first, we calculate the receptive field of a pixel in a feature map in different layers. As we go deeper into the network, pixels in a feature map have a higher receptive field (i.e., the region of the input space that affects a particular unit of the network). The following receptive fields were obtained for the **five layers of the Net4 convolutional base**:



Filter type	Patch size
Conv 1; Filt 0	3
Conv 1; Filt 1	3
Conv 2; Filt 0	5
Conv 2; Filt 1	5
Conv 3; Filt 0	7
Conv 3; Filt 1	7
Conv 4; Filt 0	13
Conv 4; Filt 1	13
Conv 5; Filt 0	17
Conv 5; Filt 1	17

Table 4. Shows the dimensions of the square patch for each filter based on its receptive field.

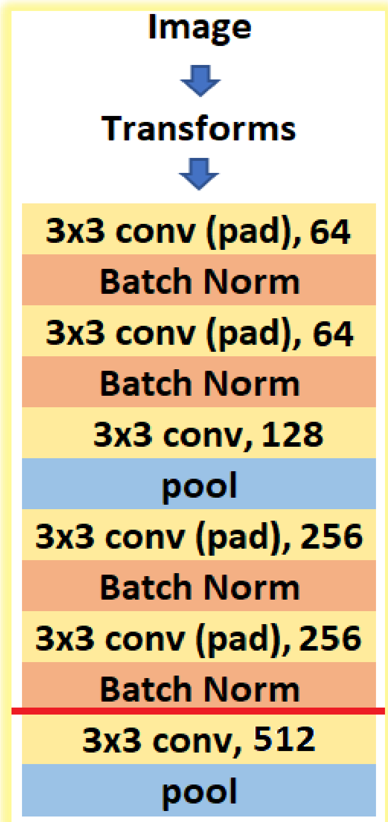
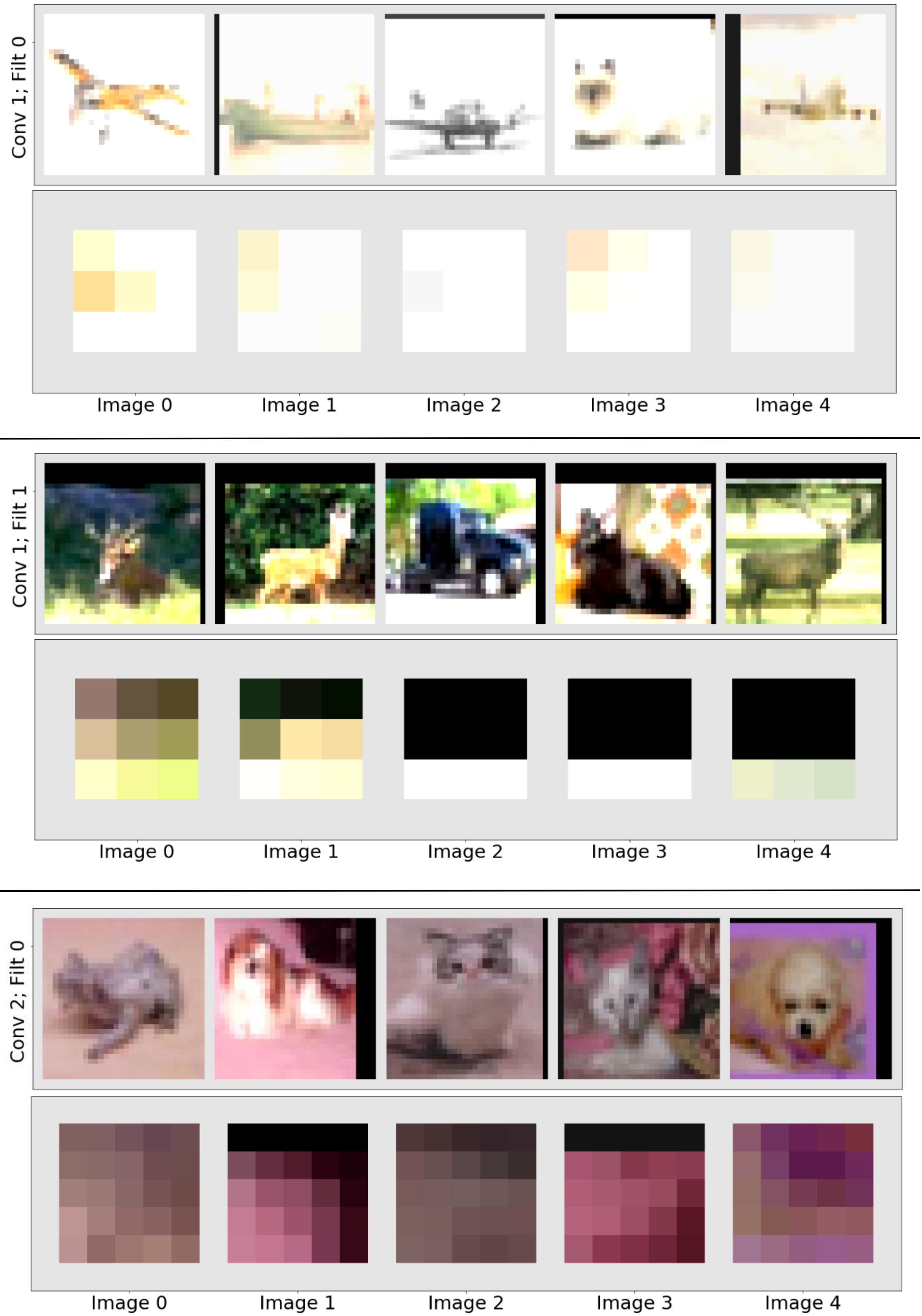
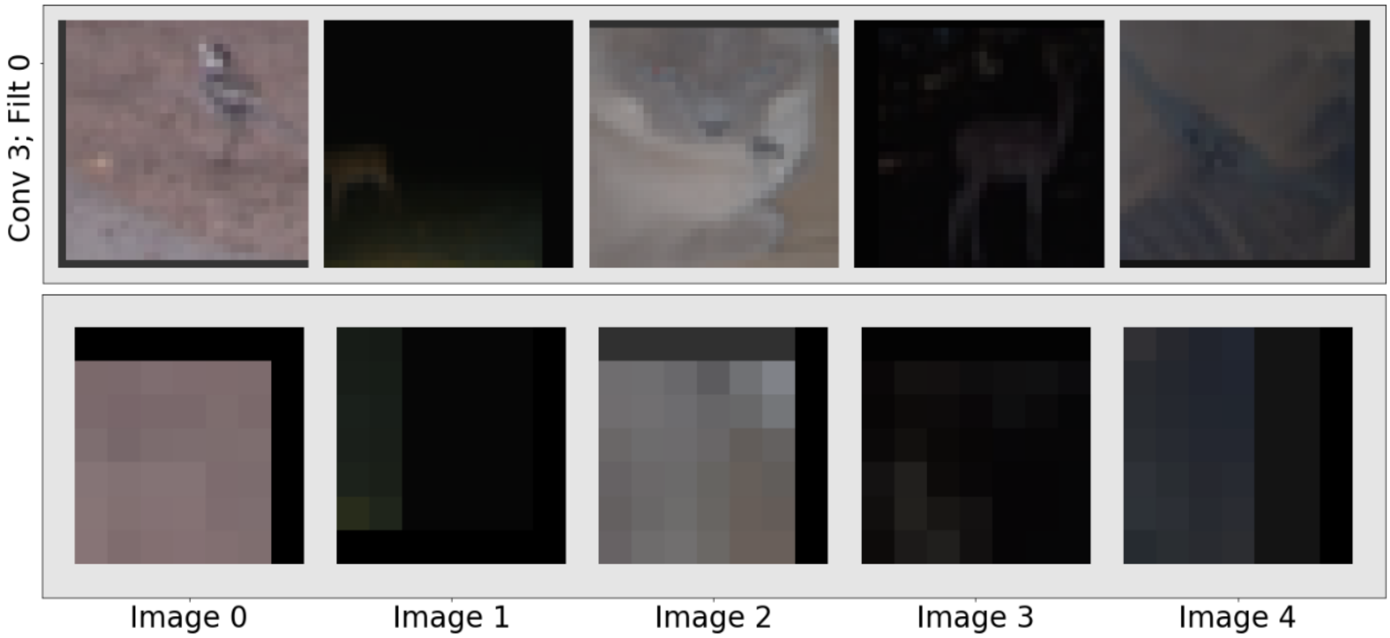
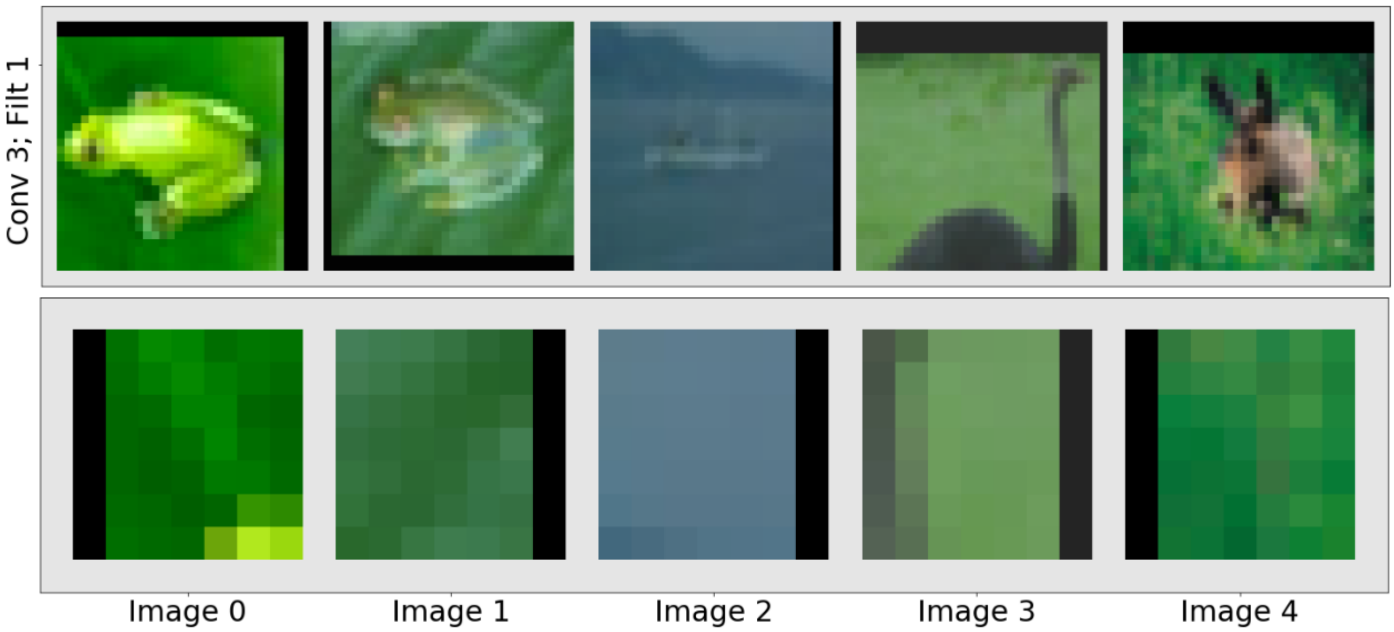


Fig. Net 4 convolutional base. We have taken filters from the first five layers.

Top 5 images based on filter response and their maximum response patches



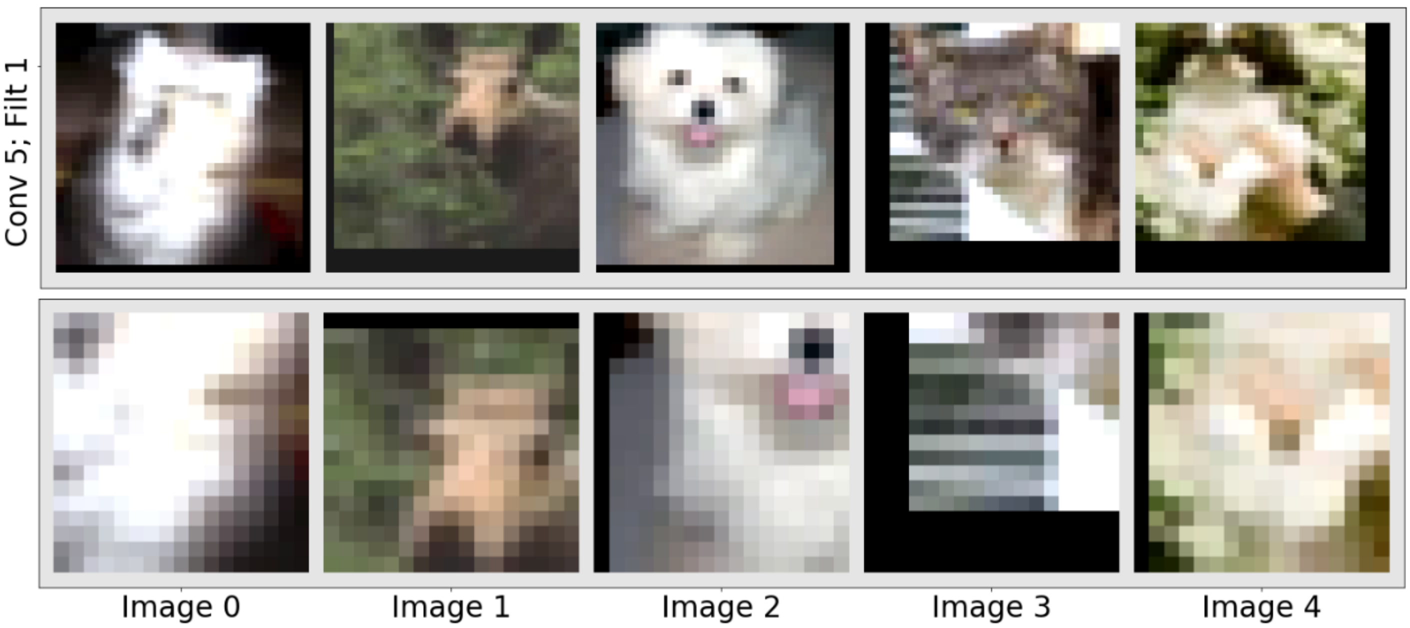

Conv 3; Filt 0
Image 0
Image 1
Image 2
Image 3
Image 4


Conv 3; Filt 1
Image 0
Image 1
Image 2
Image 3
Image 4


Conv 4; Filt 0
Image 0
Image 1
Image 2
Image 3
Image 4


Conv 4; Filt 1
Image 0
Image 1
Image 2
Image 3
Image 4


Conv 5; Filt 0
Image 0
Image 1
Image 2
Image 3
Image 4


Conv 5; Filt 1
Image 0
Image 1
Image 2
Image 3
Image 4

Note: No patches of Conv 2; Filt 1 were displayed because of the dying ReLU problem. It was observed all possible patches from its feature maps gave a negative pixel-wise sum, i.e., a negative response (pre-ReLU).

Filter Identification Analysis

- **Conv 1; Filt 0** responds to images containing bright white patches. On inspecting the filter weights, it was found that it is quite similar to a tensor full of 1's. Hence, this behaviour is justified. From table 2, we see that 6 out of the top 20 images (based on filter response) are planes. Planes are generally white.
- In **Conv 1; Filt 1**, few of the patches indicate strong response to vertical edges. Thus, both filters from the first convolution layer seem to identify simple features.
- **Conv 2; Filt 0** seems to respond to images with a maroon-brownish background, as all the top 5 images and their patches contain these colors.
- **Conv 3; Filt 0** responds to dark/ low light images.
- **Conv 3; Filt 1** responds to images containing green.
- **Conv 4; Filt 0** seems to respond to stripe-like patterns in the image. Also, 9 out of the top 20 images belong to the ship class (Table 2).
- **Conv 4; Filt 1** seems to respond strongly to images where there is a clear distinction between the background and the class object. The patches indicate that it can detect the strong horizontal color gradients. Also, 10 out of the top 20 images belong to the dog class.
- **Conv5; Filt 0** patches mostly contain greenish color. I am not able to identify other more complex features by just visualizing the patch. However, according to table 2, out of the top 20 images, Conv5; Filt 0 responds maximally to 10 deer and 5 bird images.
- According to table 2, out of the top 20 images (based on filter response) for **Conv5; Filt 1**, 10 are cats, and 4 are dogs. Thus, this filter is particularly good at recognizing these 2 classes.

Note: Only the code for part 2.2.1(Filter identification) is entirely reproducible. This is because the random seeds were set to make the entire workflow deterministic. However, the other parts of the assignment will see some variations as I was not aware of this earlier.