Convolutional Neural Networks

COL870: Assignment-1

Kshitij Alwadhi (2019EE10577)

4 September 2022

Contents

1 Training a CNN model

1.1 Parameter Tuning

For tuning the parameters, we follow a restricted grid search type of approach where we fix some parameters and vary the parameter in focus. Note that, in all the experiments, we use early stopping to avoid over-fitting. We define early stopping as that epoch when the average of the validation accuracy in the last 3 epochs is lesser than the average of the 3 epochs before them. However, we do set a hard stop at 50 epochs in the interest of time.

1.1.1 Number of Convolutional Layers

Here, we fix the fully connected layers. We have two fully connected layers having 256 nodes in each of them and finally we have another fully connected layers with 33 nodes to represent the output of the network. We vary the number of Convolutional layers from 1 to 5 and keep increasing the number of channels (filters) and progressively increase the kernel size as well. Moreover, we also have a Max pooling layer after every convolutional layer. We justify these decisions in a later part.

Number of Conv. Layers	Conv. layer parameters	Test Accuracy
1	(24,3)	15.84%
2	(24,3);(64,3)	23.57%
3	(24,3);(64,3);(128,5)	35.69%
4	(24,3);(64,3);(128,5);(256,5)	44.28%
5	(24,3);(48,3);(64,3);(128,5);(256,5)	37.51%

Table 1: Test accuracy vs Number of convolutional layers

We observe that we get the best performance when the number of convolutional layers is 4, and, the model performance decreases when we have 5 convolutional layers. Hence we set the model with 4 convolutional layers as the benchmark.

1.1.2 Number of Fully Connected layers

In this experiment, we fix the number of CNN layers to be 4 as in one of the parts of the previous subsection and vary the number of fully connected layers.

Number of Fully Connected layers	Number of nodes in each layer	Test Accuracy
1	(33)	43.36%
2	(256),(33)	43.48%
3	(256),(256),(33)	44.28%
4	(256),(256),(256),(33)	44.63%

Table 2: Test accuracy vs Number of Fully Connected layers

We observe that increasing the number of fully connected layers beyond 3 has very little impact on the test accuracy. Instead, the training time also increases significantly due to increase in parameters. Hence we set the number of fully connected layers to be 3.

1.1.3 Number of filters in different layers

We set the number of filters in our convolutional layers in such a way so that their number is increasing as we go deeper in the model which is a standard technique. We set the number of CNN layers to be 4. Here we present three variations that we take of the number of filters in the layers:

Number of filters	Test Accuracy
(24,48,64,128)	42.78%
(24,64,128,256)	44.28%
(48,64,256,256)	45.67%

Table 3: Test accuracy vs Number of filters

We observe that we get the best performance in the final setting, hence we set that as the benchmark.

1.1.4 Pooling layers

First we fix the number of CNN layers to 4 and number of Fully connected layers to be 3. Next, we perform two experiments here. We try out the following two settings:

1. CNN $\rightarrow$ Pooling $\rightarrow$ CNN $\rightarrow$ Pooling
2. CNN $\rightarrow$ CNN $\rightarrow$ Pooling

We also compare the performance of Average Pooling with Max Pooling. Following are the results we get from these experiments:

Type of Pooling	Architecture	Test Accuracy
Max	CNN ->Pooling	44.28%
Avg	CNN ->Pooling	37.15%
Max	CNN ->CNN ->Pooling	32.09%*

Table 4: Test accuracy vs Pooling

We observe that in general, max pooling significantly outperforms average pooling. Hence, the first decision that we take is to use only max pooling in our experiments. Next, we observe that having a pooling layer after each convolutional layer decreases training time significantly due to decrease in the parameters (this was so significant that we had to stop the training of the third setting in between since it was taking a lot of time). Hence, we take the decision to have a max pooling layer after every convolutional layer.

1.1.5 Kernel Size/Stride Length of Pooling Layers

Here, we vary the kernel size and the stride length of the pooling layers. We fix the type of pooling to be Max Pooling since it gave the best results. The following are the results that we get:

Kernel	Stride Length	Test Accuracy
2x2	2	44.28%
2x2	3	43.51%
3x3	2	33.91%

Table 5: Test accuracy vs Pooling Parameters

From the above experiments, we observe that having a kernel of size 2x2 significantly outperforms the setting with filter size of 3x3. Hence we use that in our experiments. Moreover, having a stride length of 2 instead of 3 or more gives better performance on the test set.

1.1.6 Training Time (Number of epochs)

Here we present the time taken per epoch to train when we vary the number of convolutional layers. We also show the number of trainable parameters to give an idea of the size of the architecture.

# of Conv. Layers	# of trainable parameters	Test Accuracy	Time Taken/epoch (s)
1	81345	15.84%	7.41
2	105473	23.57%	9.08
3	326785	35.69%	27.23
4	1179009	44.28%	84.21
5	1203249	37.51%	87.67

Table 6: Time taken/epoch vs Number of Conv. layers

1.2 Best Performing Model

Finally, the architecture that gave us the best performance on the test set had the following parameters:

1. Number of convolutional layers: 4.
2. Convolutional layer parameters: (48,3), (64,3), (256,5), (256,5).

3. Pooling used: Max Pooling (Kernel size = 2, Stride = 2).
4. Number of fully connected layers: 3
5. Number of neurons in the fully connected layers: (256, 256, 33).

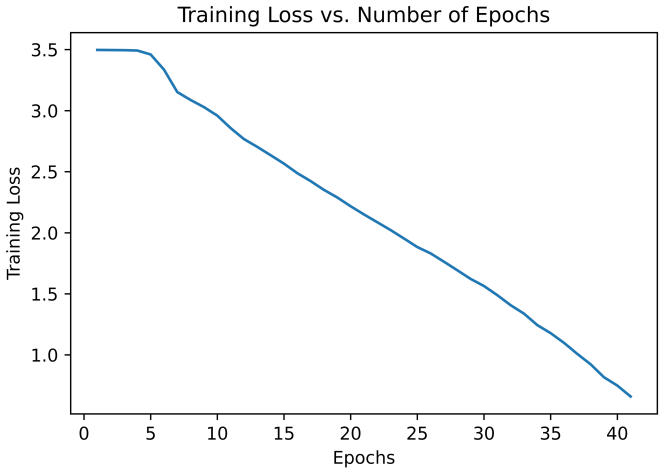

Figure 1: Training loss vs Epochs

From the above figure, we see that the training loss keeps decreasing steadily as we increase the epochs.

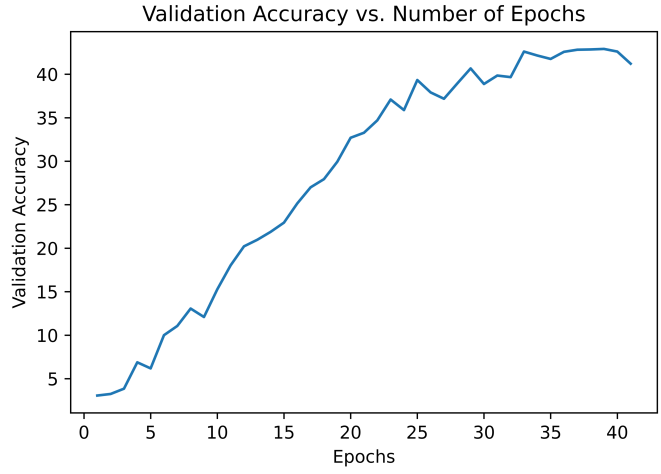

Figure 2: Validation Accuracy vs Epochs

However, the story is completely different for the case of validation accuracy. Here we see that the model starts to overfit the training data beyond the 35th epoch. Hence, we stop the training there.

1.3 Error Analysis

The test accuracy that we get from the above architecture is: **45.67%**.

The class-wise accuracy obtained is as follows:
Accuracy of Ibizan_hound : 17.857143 %
Accuracy of aircraft_carrier : 83.333333 %
Accuracy of beer_bottle : 54.166667 %
Accuracy of bolete : 41.666667 %
Accuracy of boxer : 42.857143 %
Accuracy of carton : 12.500000 %
Accuracy of dome : 62.500000 %
Accuracy of electric_guitar : 12.500000 %
Accuracy of file : 42.857143 %
Accuracy of french_bulldog : 4.166667 %
Accuracy of garbage_truck : 37.500000 %
Accuracy of golden_retriever : 50.000000 %
Accuracy of gordon_setter : 60.714286 %
Accuracy of hair_slide : 41.666667 %
Accuracy of hourglass : 41.666667 %
Accuracy of house_finch : 50.000000 %
Accuracy of komondor : 21.428571 %
Accuracy of malamute : 62.500000 %
Accuracy of meerkat : 50.000000 %
Accuracy of pencil_box : 37.500000 %
Accuracy of prayer_rug : 50.000000 %
Accuracy of reel : 37.500000 %
Accuracy of rock_beauty : 83.333333 %
Accuracy of scoreboard : 33.333333 %
Accuracy of solar_dish : 60.714286 %
Accuracy of stage : 45.833333 %
Accuracy of street_sign : 37.500000 %
Accuracy of tank : 45.833333 %
Accuracy of tile_roof : 28.571429 %
Accuracy of tobacco_shop : 37.500000 %
Accuracy of trifle : 54.166667 %
Accuracy of white_wolf : 41.666667 %
Accuracy of yawl : 57.142857 %

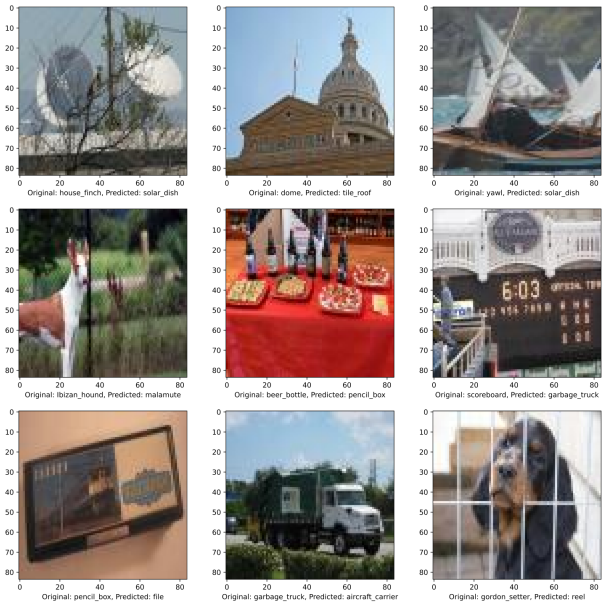

Figure 3: Misclassified Examples

From the above misclassified examples, we can see that the model gets confused in some very genuine scenarios. In the first image, we see that the image is labelled as 'house finch' however there is a solar dish in the image and the model predicts it as one. This is a fault of the image and the label. Next, we see an image of the roof of a building which is a dome, however, the model gets confused here since the image also resembles a tile roof. In the third image, the positioning of the sails of the yawl are very similar to how the solar dishes are placed in the training data and hence the model has got confused here. In the bottom left example, we see that a pencil box was predicted as a file because the image does resemble that of a file in the training data. Some of these confusions were bound to happen and would happen even in the case of a human. Hence, apart from the lack of training data and better modelling architectures, some of these mis-classifications are justified.

2 Occlusion Sensitivity Experiment

In this experiment, we consider a window of size 8x8, replace it with black pixels and slide it over the input image and get the predictions. We plot this output (confidence) as a function of position of the window and get a heatmap indicating the importance of the occluded portion.

Following are some results from the occlusion sensitivity experiment. In these set of handpicked examples, we observe that the prediction of the model does drop when we occlude an important area of the image. We justify the heatmaps on a case by case basis after the images.

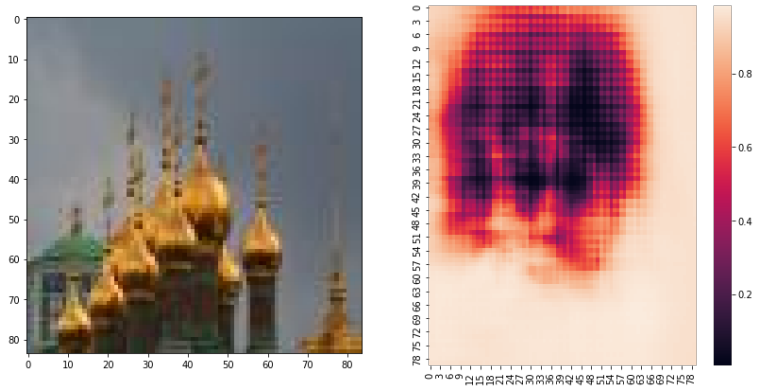

Figure 4: Image of a dome

Here, we observe that the top of the dome like structure is important for the prediction and occluding them leads to drop in probability.

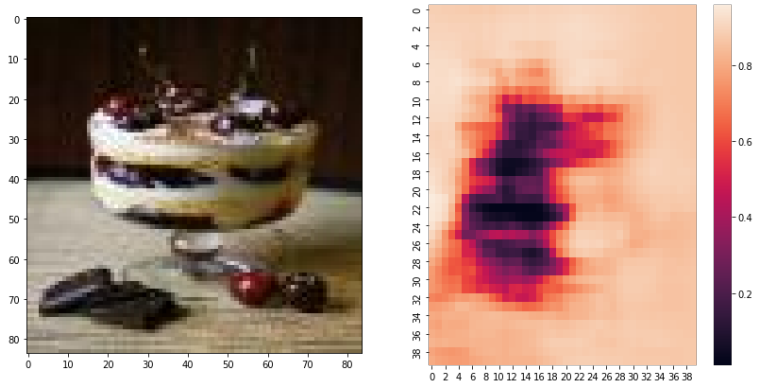

Figure 5: Image of a trifle

In the case of a trifle, it's the cream that is probably important for the prediction.

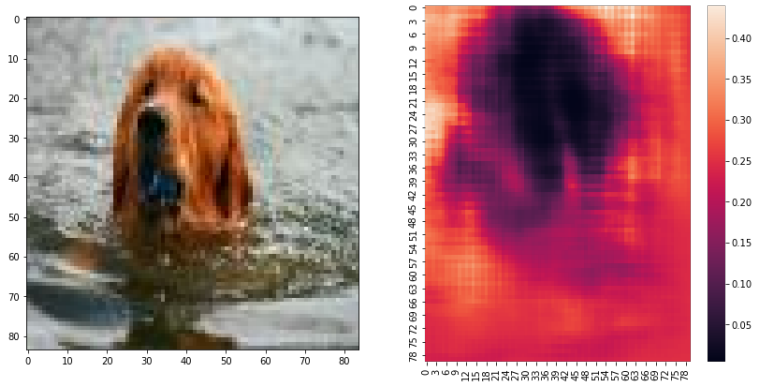

Figure 6: Image of a Golden Retriever

Here, occluding the facial features of the dog leads to drop in output probability.

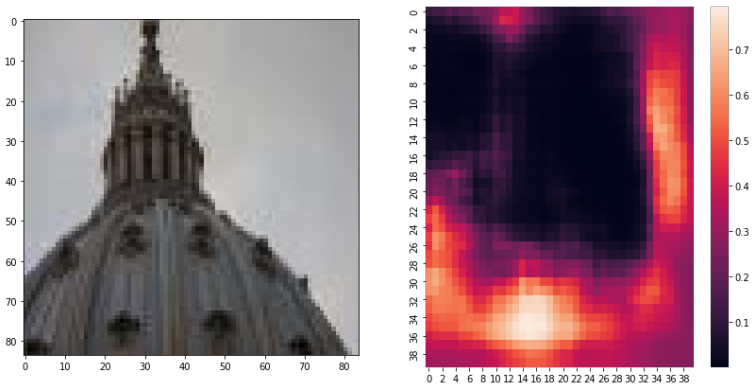

Figure 7: Image of a dome

Again, in the case of a dome, occluding the top of the dome leads to drop in probability.

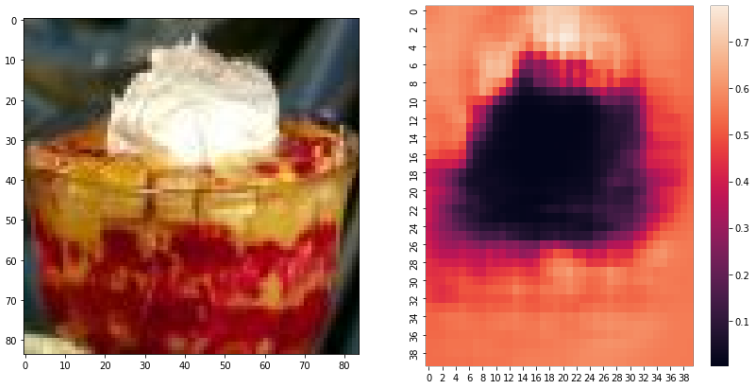

Figure 8: Image of a trifle

Again, in the case of trifle, it's the cream that heavily influences the output.

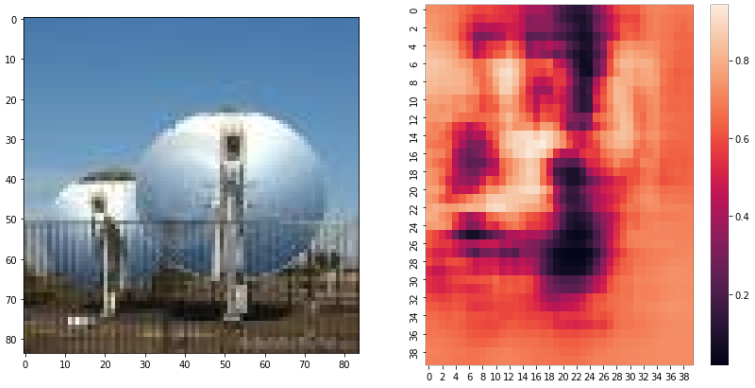

Figure 9: Image of a solar dish

In the case of solar dish, the model is probably learning that the sky is also a heavy indicator of predicting an input image as a solar dish.

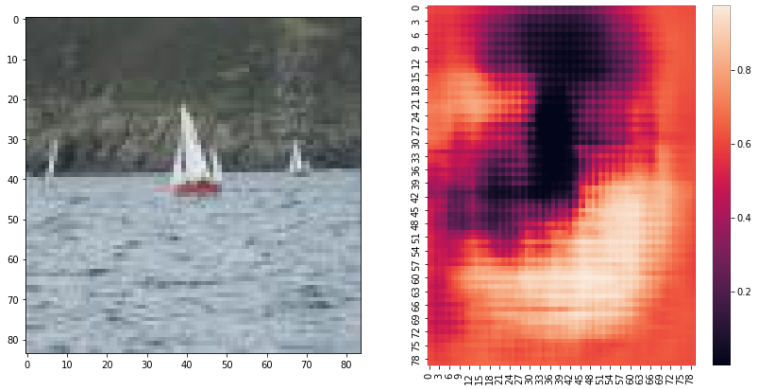

Figure 10: Image of a yawl

Here occluding the yawl leads to drop in output probability.

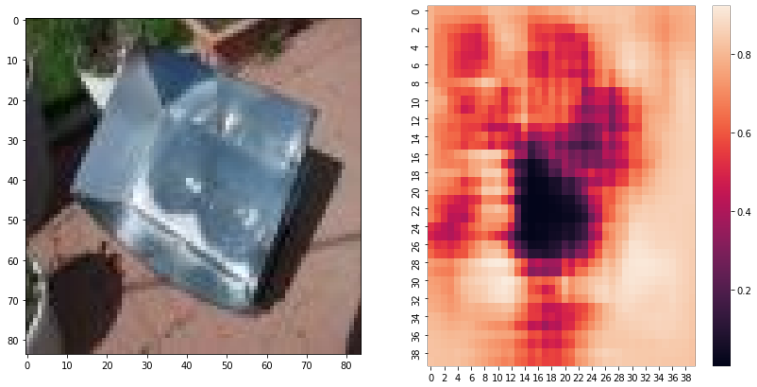

Figure 11: Image of a solar dish

Again, the output probability correctly drops when we occlude the centre of the solar dish.

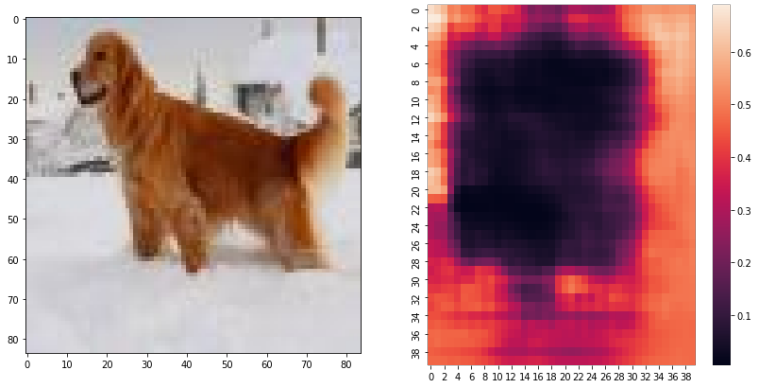

Figure 12: Image of a golden retriever

Here, we observe that the whole of the dog is infact actually leading to the prediction. This is surprising because we would expect the model to perform reasonably well even if we occlude the body of the dog.

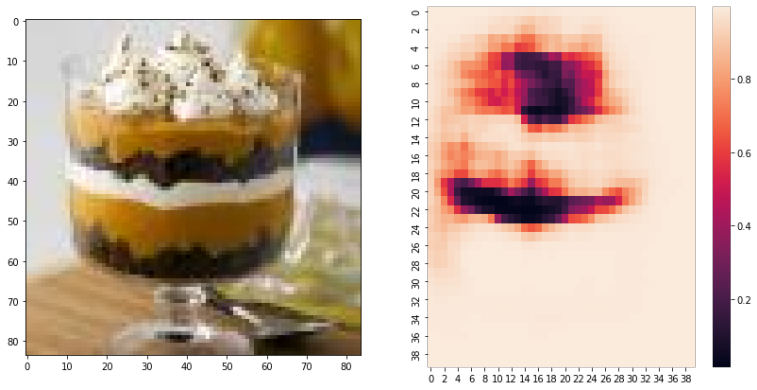

Figure 13: Image of a trifle

Finally, again in the case of a trifle, the cream is the important feature which leads to prediction as a trifle.

3 Filter Analysis

3.1 Filter Identification

3.1.1 Methodology

In this experiment, we try to identify what is being learnt by the filters of the convolutional layers. We first randomly pick some filters from our architecture (4 layers of CNN). Then for each filter, we try to see which image from the test dataset is leading to the maximum activation in that particular channel of the layer. We define the activation of channel(2D) as the L2 norm of the pixels of the output of that channel. We present these results in the next section.

3.1.2 Observations

Here are some examples from the approach described above:

(Note that the image on the left is the output of the convolutional layer (channel) and the image on the right is the original image)

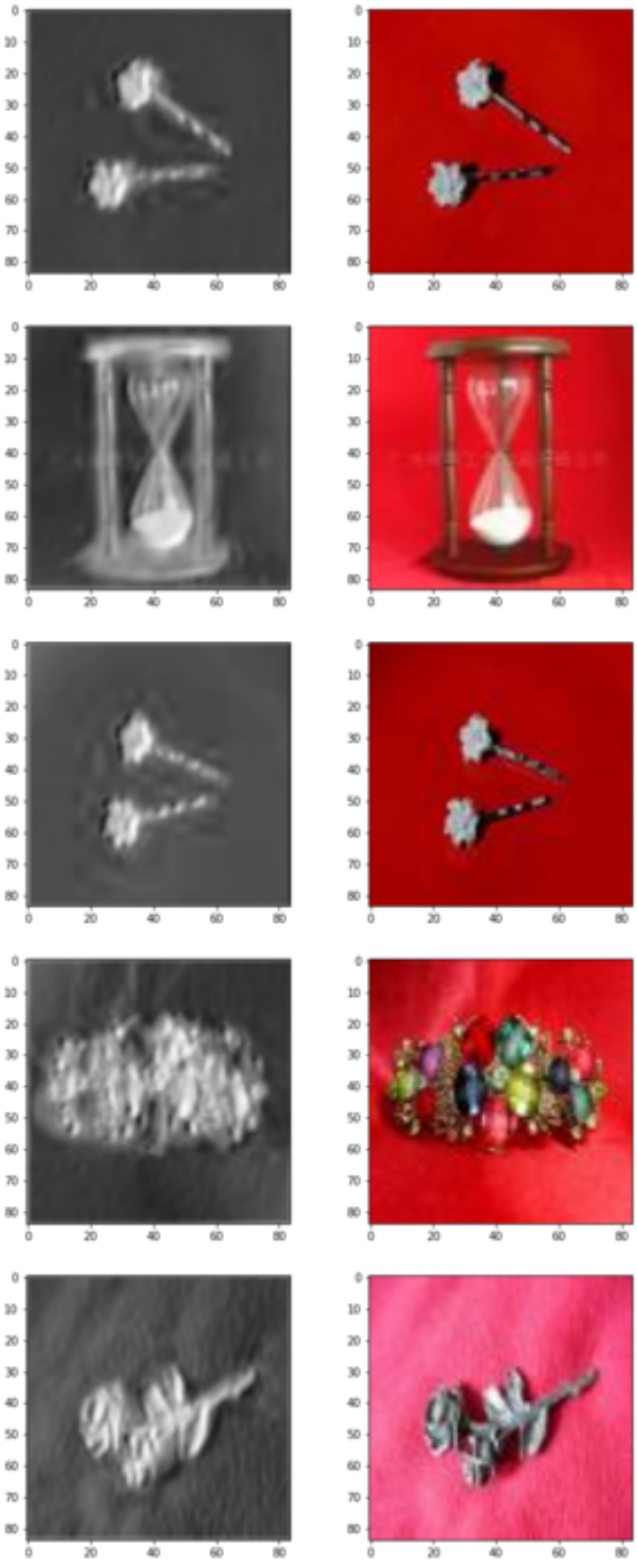

Figure 14: Layer 1 - Filter 1

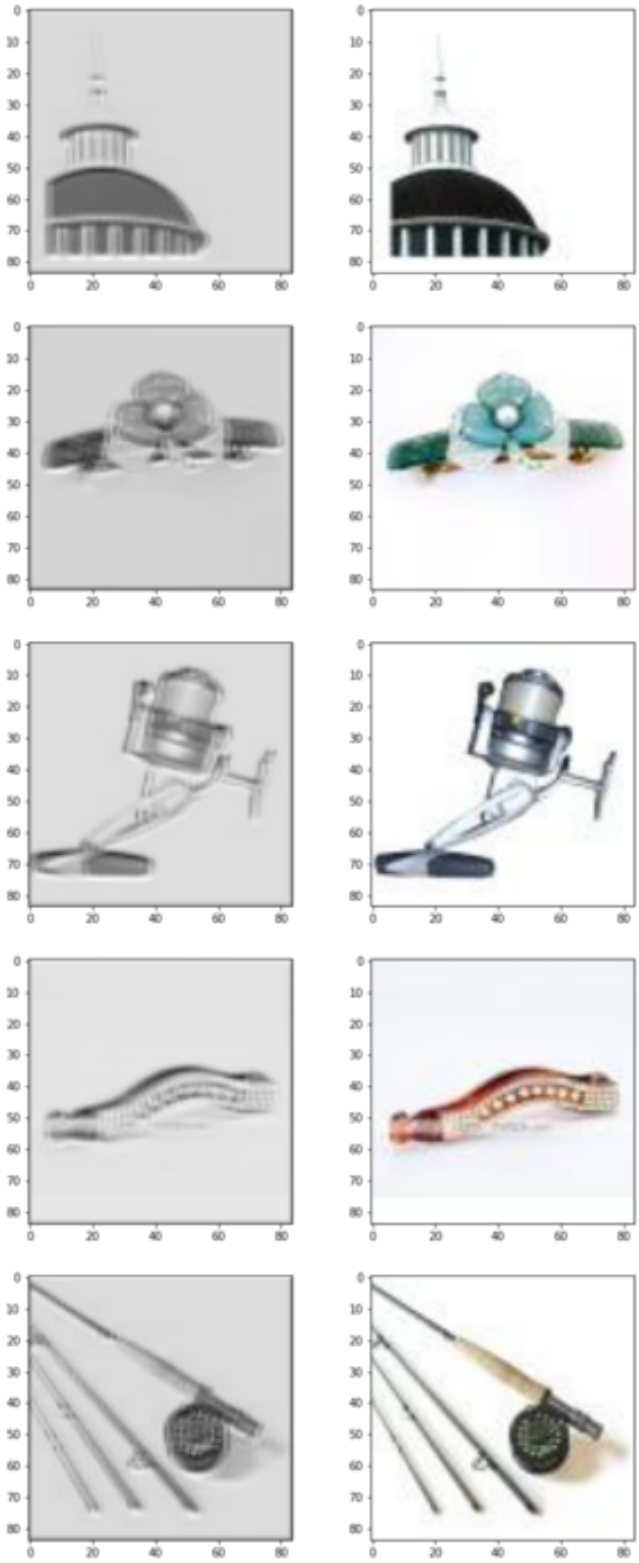

Figure 15: Layer 1 - Filter 5

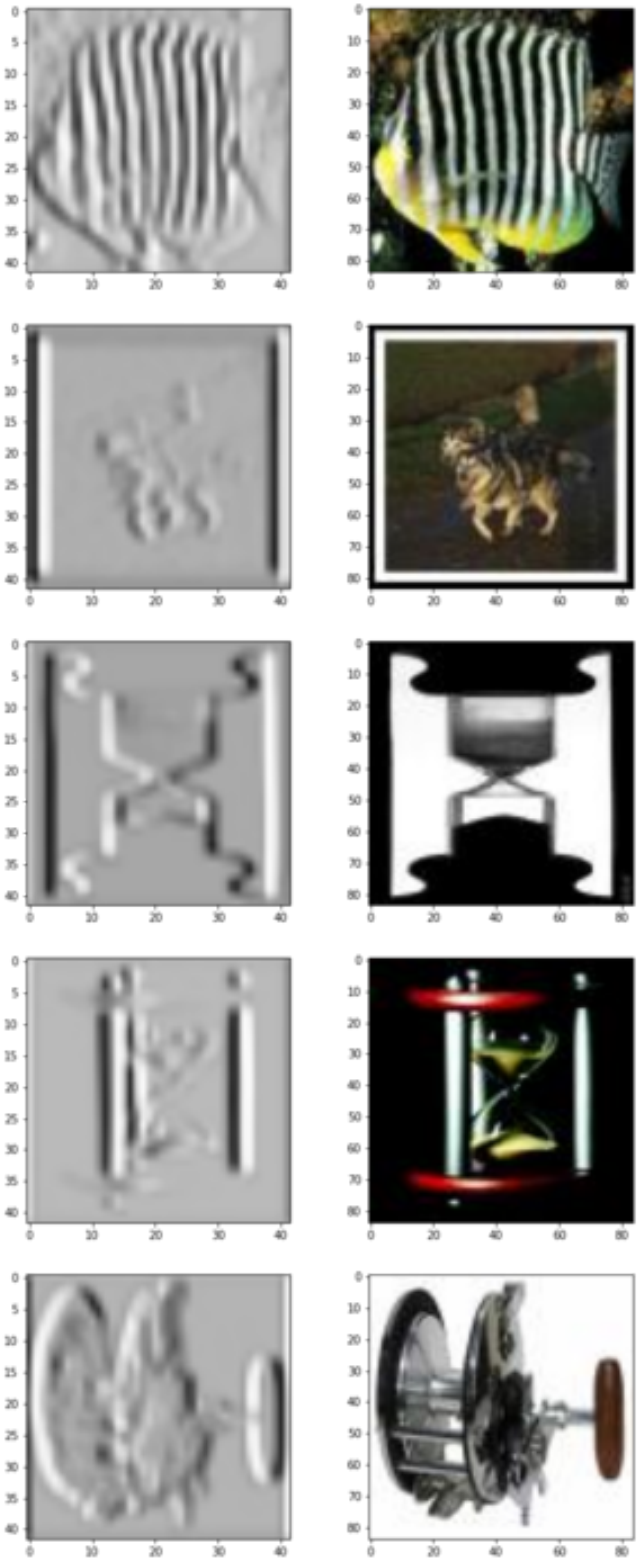

Figure 16: Layer 2 - Filter 14

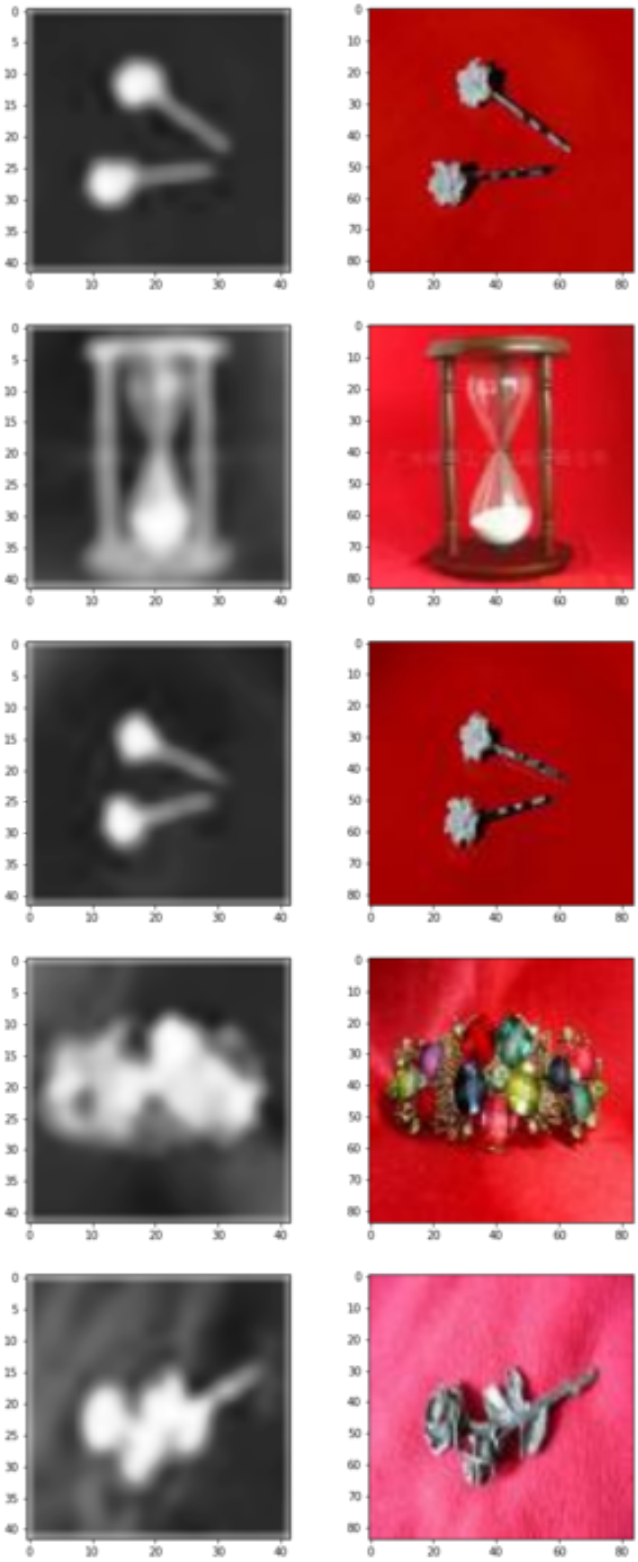

Figure 17: Layer 2 - Filter 29

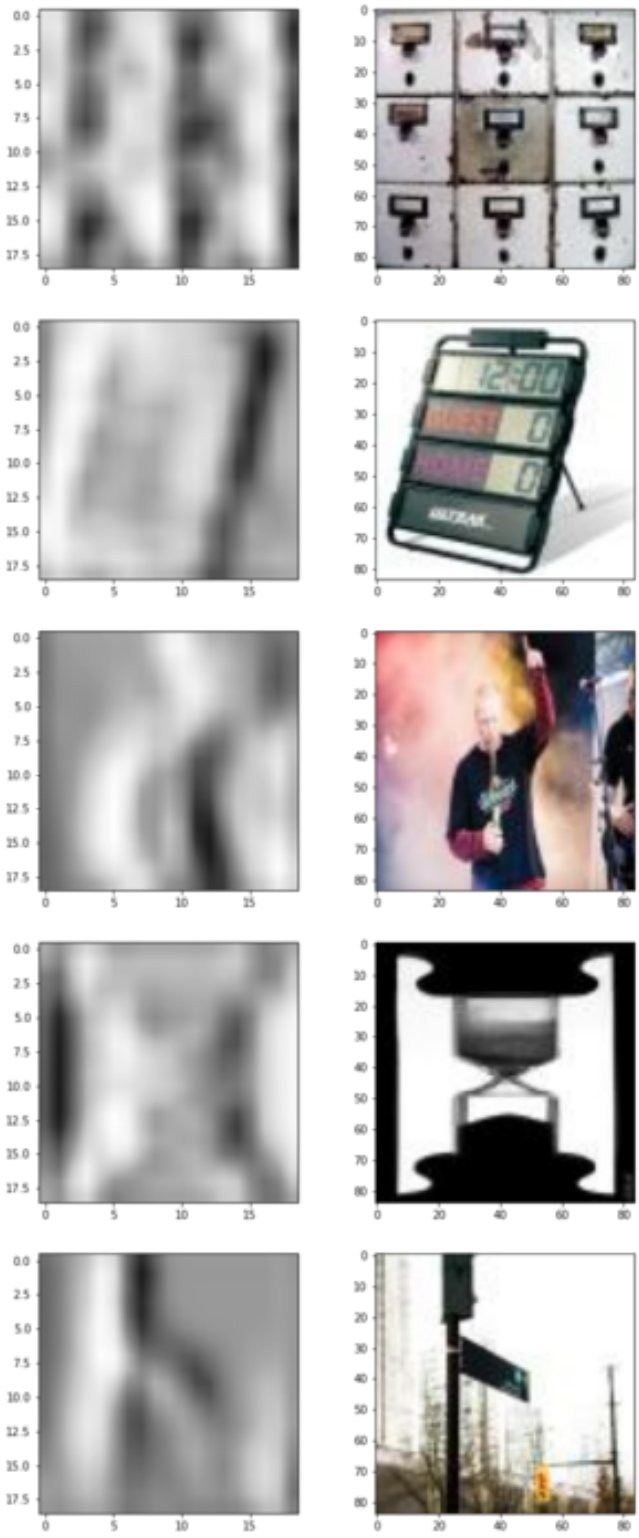

Figure 18: Layer 3 - Filter 4

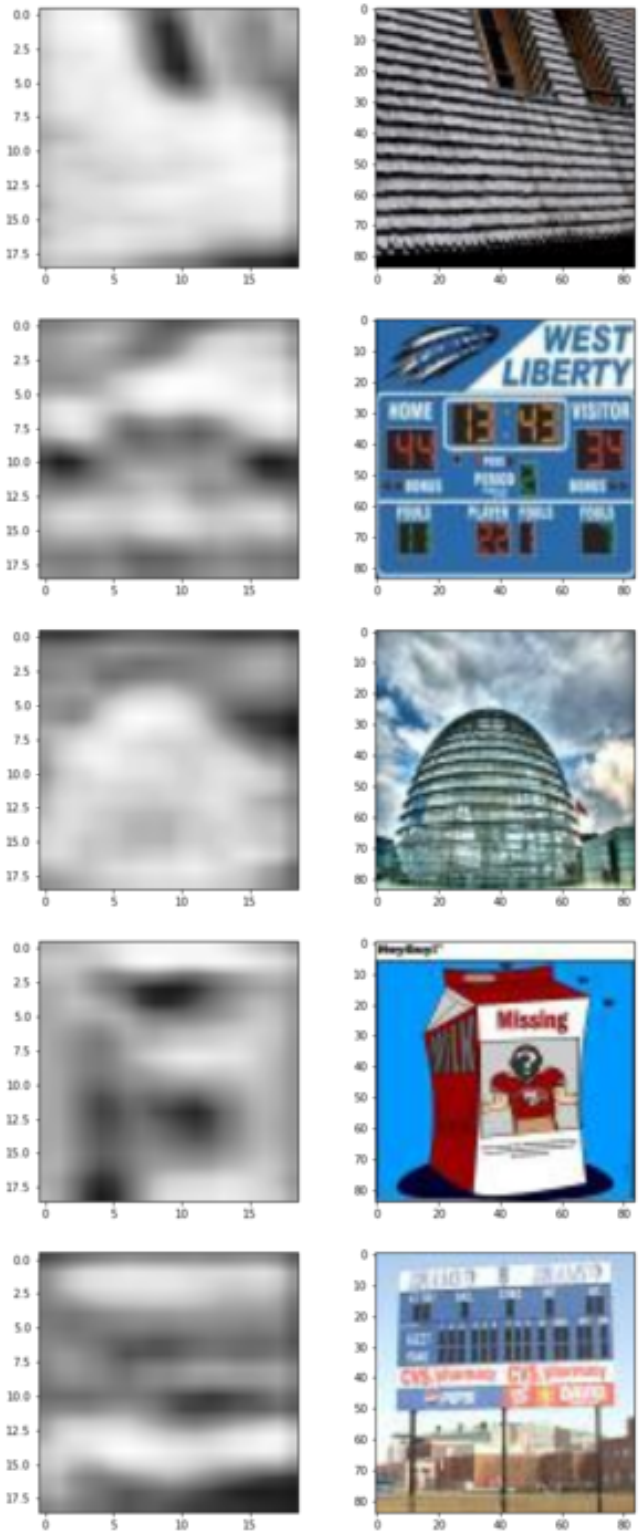

Figure 19: Layer 3 - Filter 20

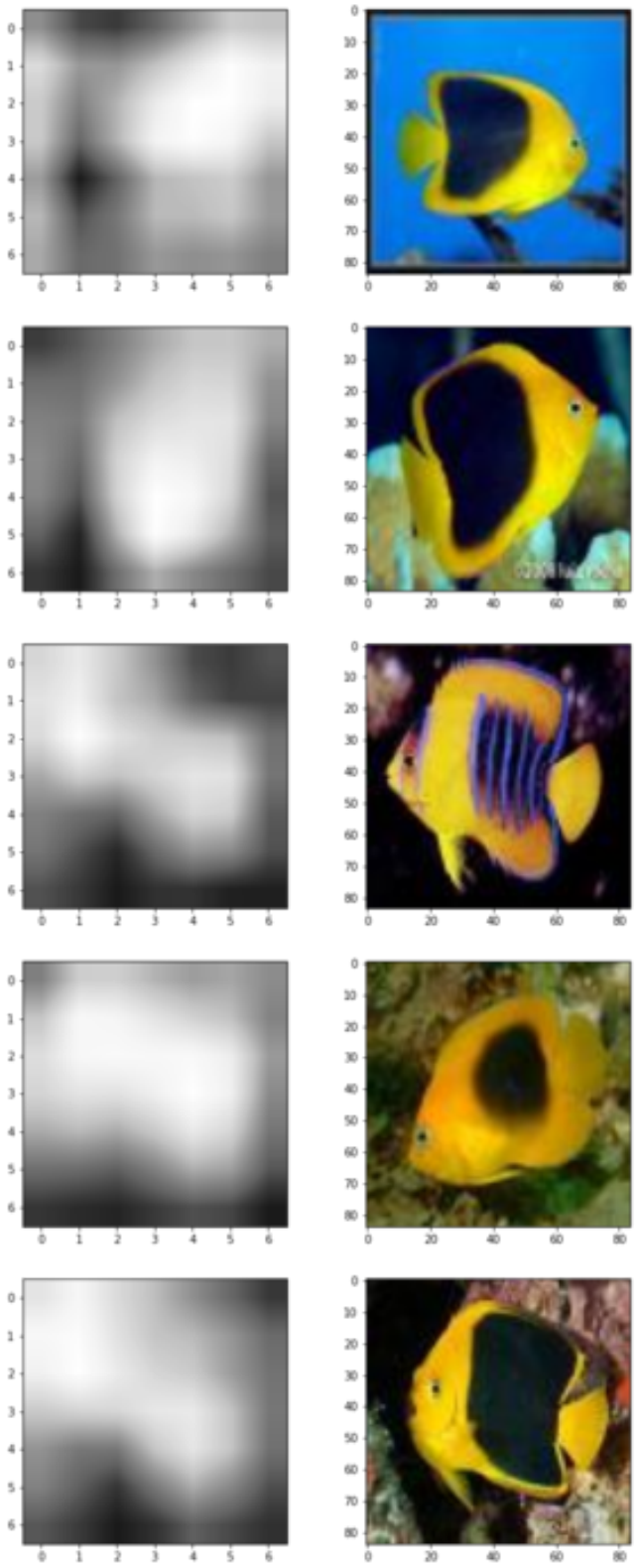

Figure 20: Layer 4 - Filter 15

Figure 21: Layer 4 - Filter 22

3.2 Filter Modification

3.2.1 Methodology

Here, we test out the performance when we turn off the above selected filters on by one as well as collectively. We show the classes for which the total accuracy on the test dataset dropped after the filter was turned off. We repeat this analysis when we turn off all the filter discussed.

3.2.2 Observations

Classes where the accuracy drops when Layer 1 - Filter 1 turned off: 'boxer, french_bulldog, gordon_setter, hair_slide, komondor, malamute, pencil_box, reel, stage, tile_roof, trifle'

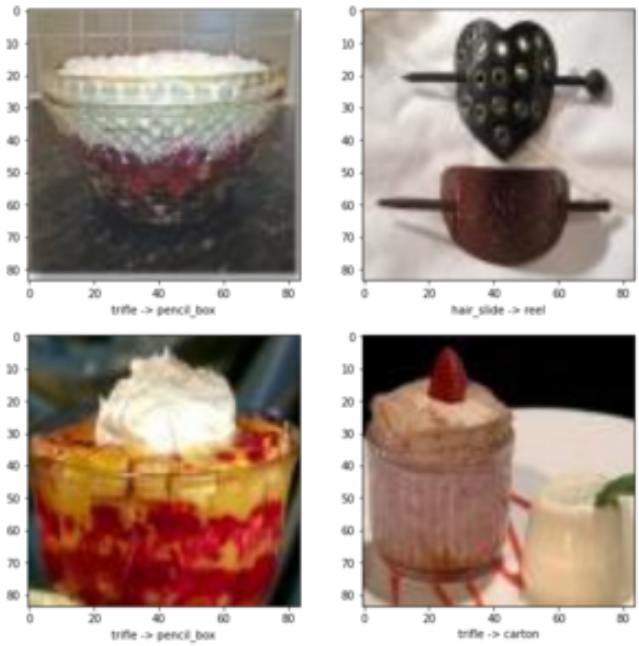

Figure 22: Misclassified examples when Layer 1 - Filter 1 turned off

Classes where the accuracy drops when Layer 1 - Filter 5 turned off: 'dome, file, scoreboard, stage'

Figure 23: Misclassified examples when Layer 1 - Filter 5 turned off

Classes where the accuracy drops when Layer 2 - Filter 14 turned off: 'boxer, hair_slide, komondor, malamute'

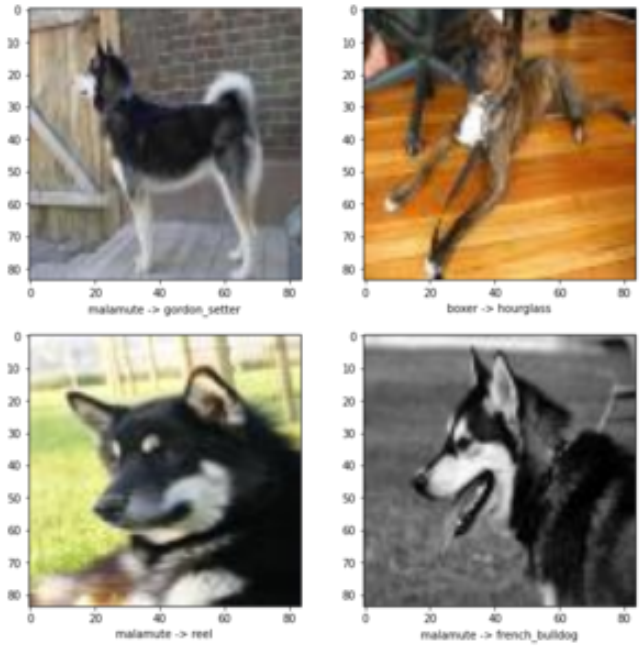

Figure 24: Misclassified examples when Layer 2 - Filter 14 turned off

Classes where the accuracy drops when Layer 2 - Filter 29 turned off: 'beer_bottle, boxer, electric_guitar, file, garbage_truck, hair_slide, komondor, pencil_box, prayer_rug, scoreboard, stage, tile_roof, trifle'

Figure 25: Misclassified examples when Layer 2 - Filter 29 turned off

Classes where the accuracy drops when Layer 3 - Filter 4 turned off: 'gordon_setter, trifle'

Figure 26: Misclassified examples when Layer 3 - Filter 4 turned off

Classes where the accuracy drops when Layer 3 - Filter 20 turned off: 'file, gordon_setter'

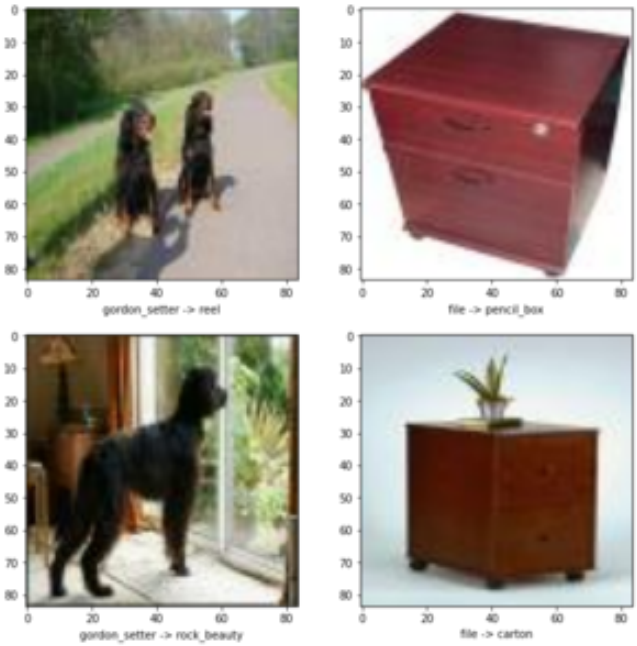

Figure 27: Misclassified examples when Layer 3 - Filter 20 turned off

Classes where the accuracy drops when Layer 4 - Filter 15 turned off: 'dome, malamute, trifle'

Figure 28: Misclassified examples when Layer 4 - Filter 15 turned off

Classes where the accuracy drops when Layer 4 - Filter 22 turned off: 'boxer, reel, stage, street_sign'

Figure 29: Misclassified examples when Layer 4 - Filter 22 turned off

Classes where the accuracy drops when all the filters above turned off: 'Ibizan_hound, beer_bottle, bolete, boxer, electric_guitar, file, garbage_truck, gordon_setter, hair_slide, house_finch, komondor, malamute, pencil_box, prayer_rug, reel, stage, tile_roof, tobacco_shop, trifle'

Figure 30: Misclassified examples when all the above filters turned off

3.3 Results

We observe that the starting layers tend to learn about the edges as is evident from the output the first four layers that we take into consideration. After this the layers start to learn more complex features as is evident from Figure 3.2.2 and 3.2.2. Particularly, in the figure 3.2.2 we see that this filter is learning about the presence of features resembling a fish.

Hence, the general hypothesis that the initial layers (with smaller kernel size) tend to learn about details like edges in the image and as we go deeper into the model, the filters start learning more complex features which the define the predicted class holds true in our experiments as well.