CS7015 : Programming Assignment-2

March 31, 2019

Checklist :

☑ checked.

☐ unchecked.

☒ not done.

The task you need to ensure before submission.

- ☑ We have read all the instruction carefully and followed them to our best ability.
- ☑ We have written the name, roll no in report.
- ☑ Run sanity_check.sh.
- ☑ We will be submitting only single submission on behalf of our team.
- ☑ We have not included unnecessary text, pages, logos in the assignment.
- ☑ We have not used any high level APIs(Keras, Estimators for e.g.).
- ☑ We have not copied anything for this assignment.

Team Mate 1 : Madhura Pande
Roll No : CS17S031
Team Mate 2 : Aakriti Budhraja
Roll No : CS18S009

Report:

1 Configuration and training details of the best performing model

1. We used 6 convolution layers, 2 fully connected layers (with 256 neurons each) and one output layer(with 20 neurons).

2. We used Batch Normalization on each convolution layer and Dropout on both the fully connected layers.

2 Loss vs iterations curves

We noticed, as the number of training epochs increased the training loss went on decreasing whereas the validation loss became more or less constant.

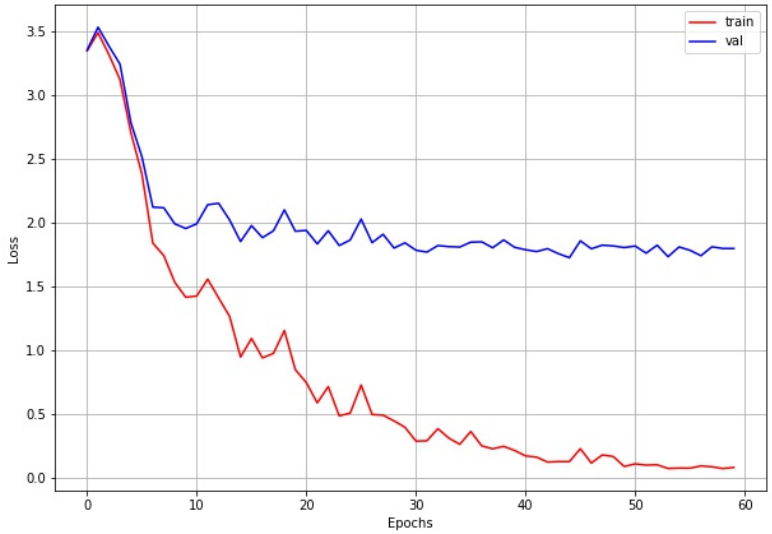

Figure 1: Train and Validation loss v/s iterations

3 Performance of best model on test data

We observed **56.357%** accuracy on the test data, as shown on Kaggle's public leaderboard.

4 Hyperparameter Tuning

4.1 Learning rate

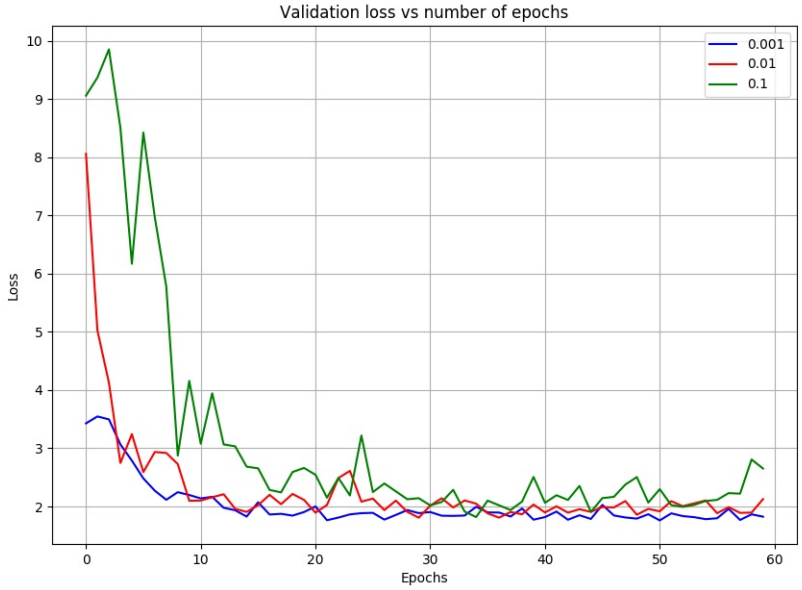

Figure 2: Comparison of different learning rates

4.2 Batch size

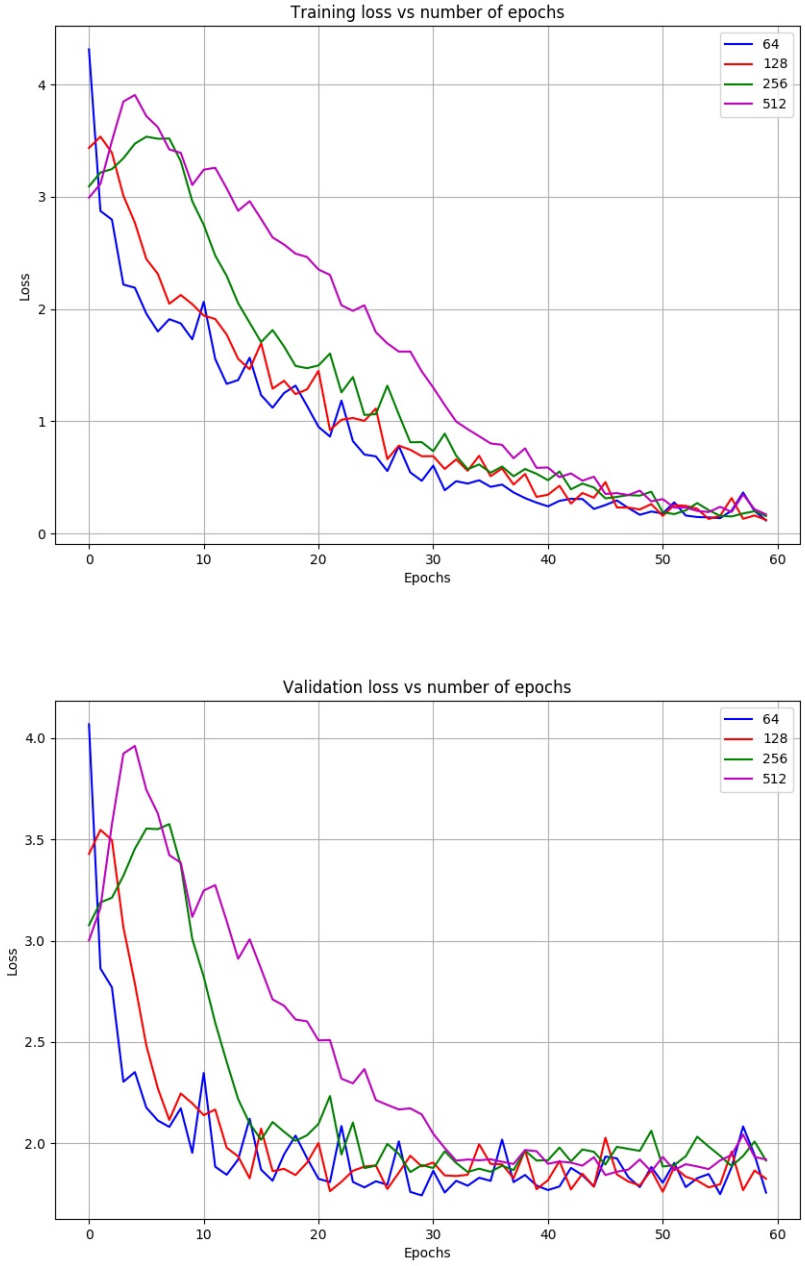

Figure 3: Comparison of different batch sizes

4.3 Initialization

We observed that Xavier initialization performed better than He.

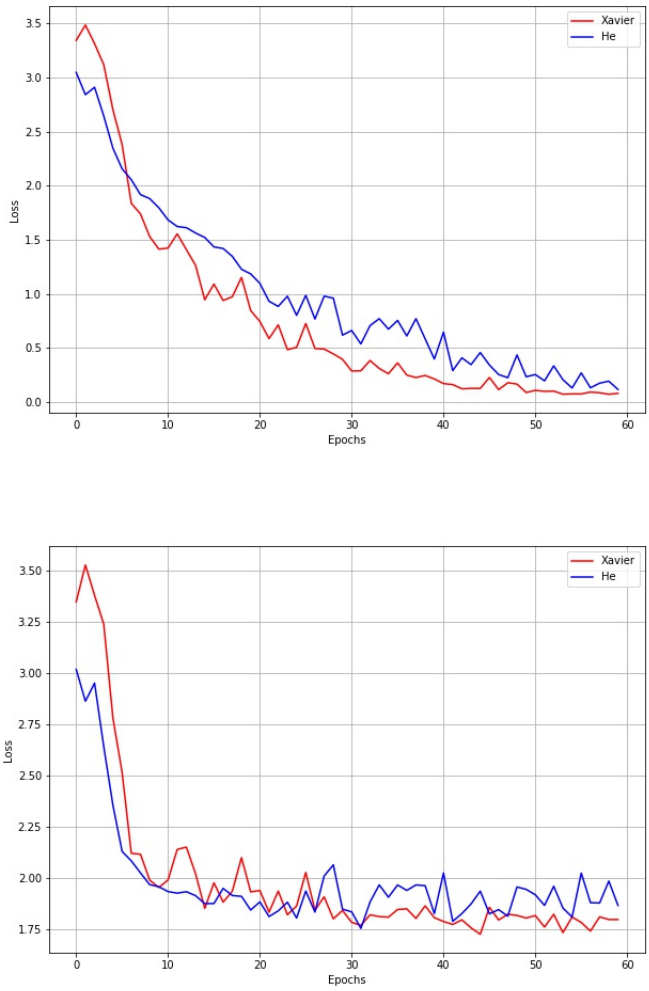

Figure 4: Train Loss(Top) and Val Loss(Bottom) for various init types

5 Parameter settings for the best model

Parameter settings that gave the best results are :
Learning rate, $\eta = 0.001$

Dropout keep probability (in Fully Connected layer) = 0.4
Number of epochs = 80
Batch size = 128
Batch Normalization on all convolution layers, Dropout on FC1

6 Dimensions of input and output at each layer

Layer	**Input size**	**Output size**
Conv1	64 x 64 x 3	62 x 62 x 32
Conv2	62 x 62 x 32	60 x 60 x 32
Pool1	60 x 60 x 32	30 x 30 x 32
Conv3	30 x 30 x 32	30 x 30 x 64
Conv4	30 x 30 x 64	30 x 30 x 64
Pool2	30 x 30 x 64	15 x 15 x 64
Conv5	15 x 15 x 64	15 x 15 x 64
Conv6	15 x 15 x 64	13 x 13 x 128
Pool3	13 x 13 x 128	7 x 7 x 128
FC1	6272	256
Output	256	20

7 Number of parameters in the network

Layer	**Parameters**	**Total**
Conv1	(5*5*3)*32	2400
Conv2	(5*5*32)*32	25600
Pool1	0	0
Conv3	(3*3*32)*64	18432
Conv4	(3*3*64)*64	36864
Pool2	0	0
Conv5	(3*3*64)*64	36864
Conv6	(3*3*64*128)	73728
Pool3	0	0
FC1	6272*256	1605632
Output	1605632 * 20	32112640

Total parameters : 33,912,160 $\approx$ 34M parameters

Layer	No. of Neurons	Total
Input layer	64 * 64 * 3	12288
Conv1	62 * 62 * 32	123008
Conv2	60 * 60 * 32	115200
Conv3	30 * 30 * 64	57600
Conv4	30 * 30 * 64	57600
Conv5	15* 15 * 64	14400
Conv6	13 * 13 * 128	21632
FC1	256	256
Output layer	20	20

8 Number of neurons in the network

If we assume one single neuron to absorb the complete depth, then the calculations would be following:

Layer	No. of Neurons	Total
Input layer	64 * 64	4096
Conv1	62 * 62	3844
Conv2	60 * 60	3600
Conv3	30 * 30	900
Conv4	30 * 30	900
Conv5	15* 15	225
Conv6	13 * 13	169
FC1	256	256
Output layer	20	20

9 Effect of using Batch Normalization

We observed great boost in accuracy after we applied batch normalization to all the convolution layers of the CNN. The increase was around 5%, solely due to Batch Normalization (in all layers). Also, we observed better performance of the model as it helped in overcoming vanishing gradient problem.

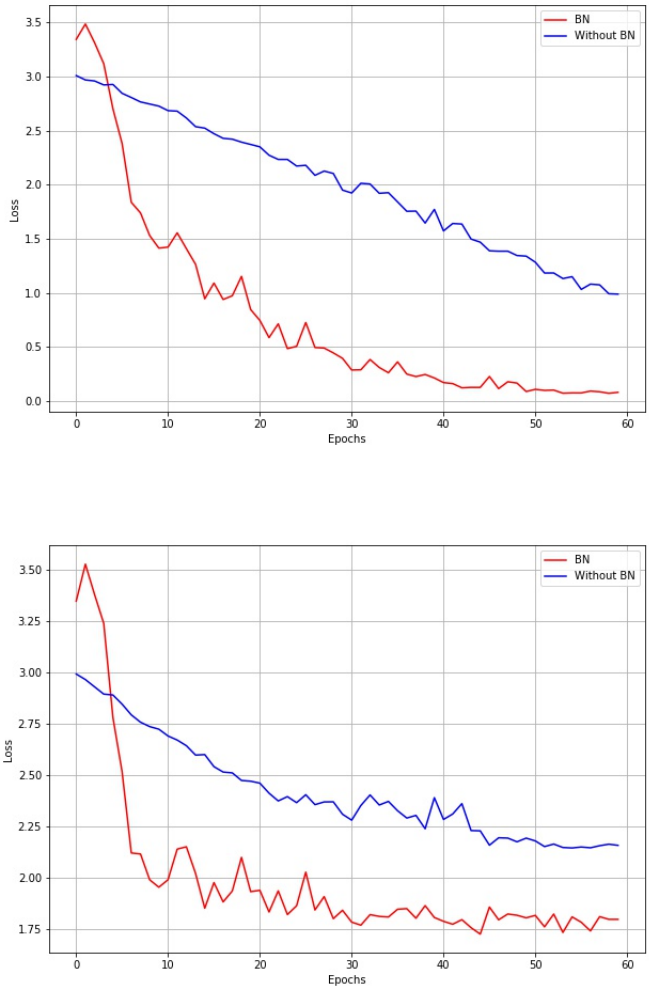

Figure 5: Effect of Batch Normalization. (Top) Training loss (Bottom) Validation loss

10 Plot of Convolution layer-1 filters

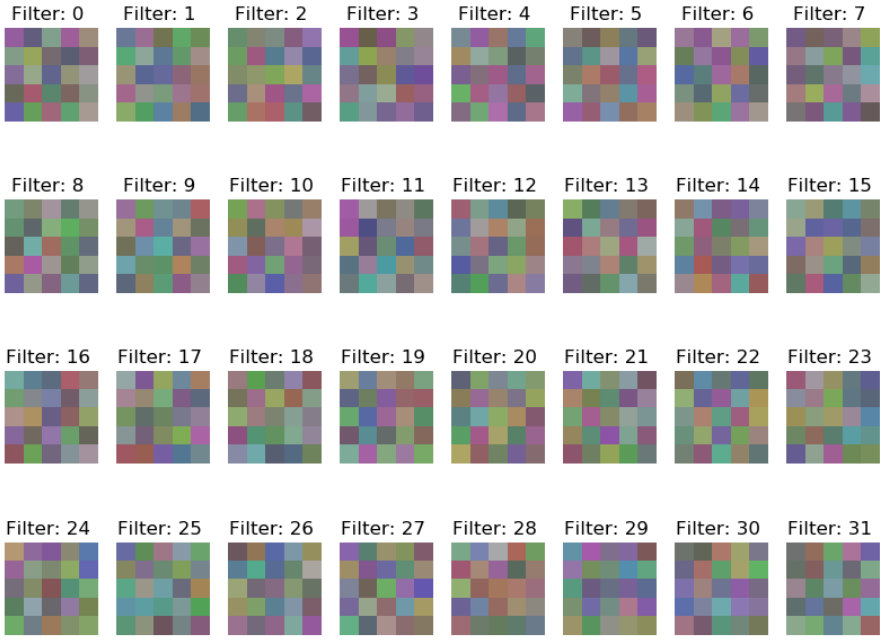

Figure 6: Plot of filters of Conv Layer1

11 Guided Back Propagation

We used Guided-ReLU to get the gradients. Guided ReLU avoids passing back a negative gradient, so we are able to observe interesting patterns which a particular set of neurons detect in the image. We observed 10 neurons and got the outputs.

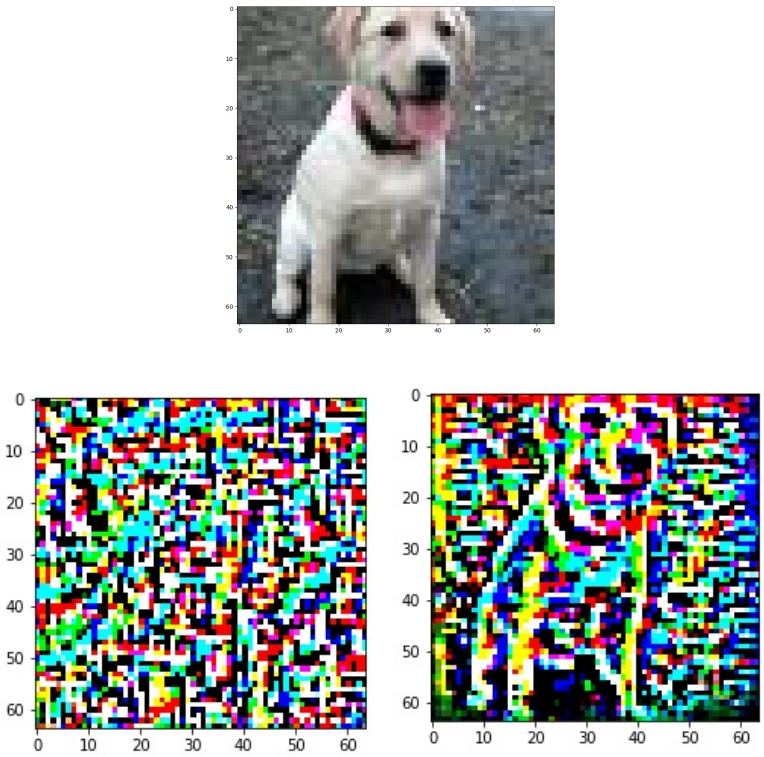

Figure 7: Original image(Top), Normal gradient(Bottom Left), Gradient via Guided Backpropagation(Bottom right)

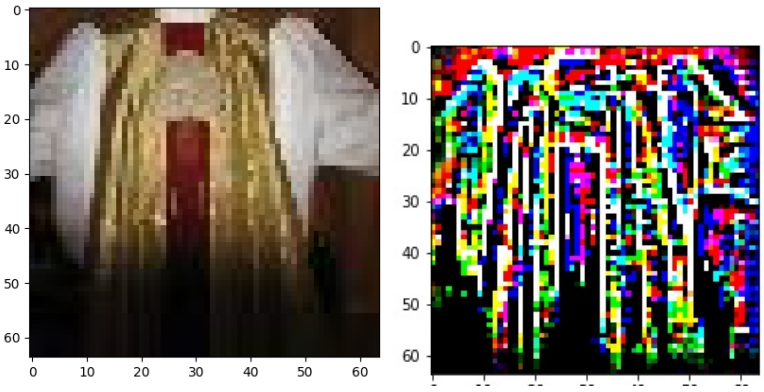

Figure 8: Original image(Left), Gradient via Guided Backpropagation(Right)

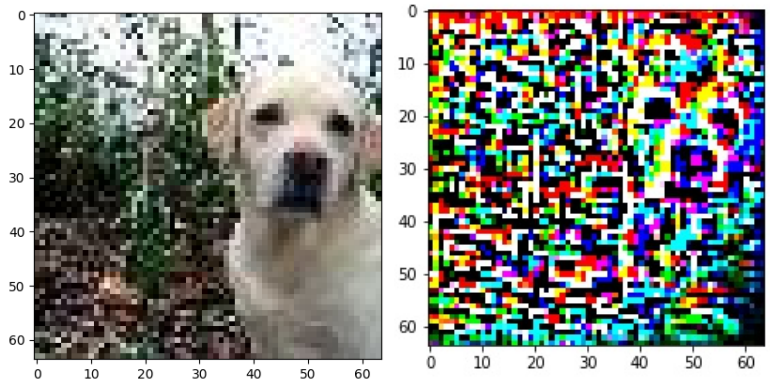

Figure 9: Original image(Left), Gradient via Guided Backpropagation(Right)

12 Network Fooling

We performed n-pixel attack to fool the network, where we introduced random Gaussian noise in n pixels of the image. We started from 1 pixel and went on till adding noise in all pixels of the image, i.e. 4096 pixels. Below are the examples of 2 such images :

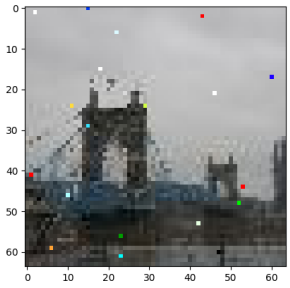

Figure 10: n = 20 pixels

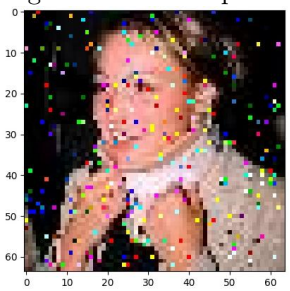

Figure 11: n = 300 pixels

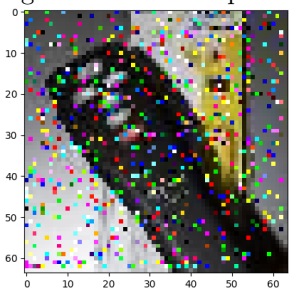

Figure 12: n = 800 pixels

Below is the graph of the decrease in accuracy on validation dataset as the number of attacked pixels are increased.

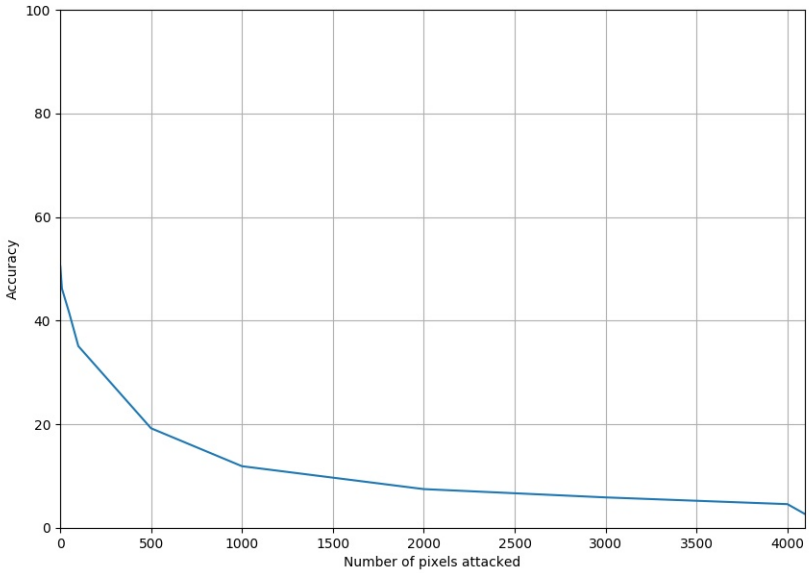

Figure 13: Accuracy vs number of pixels attacked

13 Data Augmentation

We have applied the following types of data augmentation :
1. Left-right (horizontal) flipping
2. Upside-down (vertical) flipping
3. Adding noise to data and training on noisy data
4. Image rotation 90 degree

Below are the examples of how the augmented data looks like :

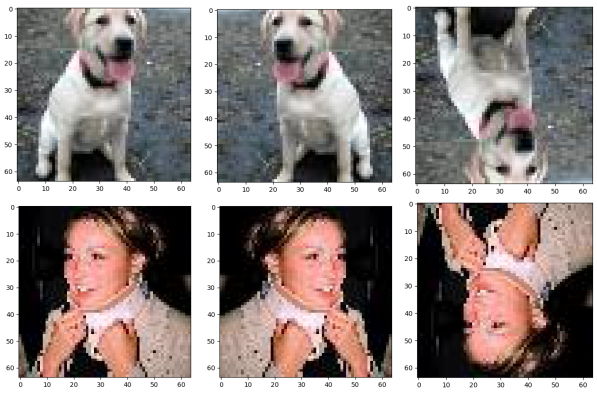

Figure 14: Original data and augmented data

Table 1: Effect of Data Augmentation

Augmentation	**Accuracy**
ON	56.32%
OFF	46.29%

14 Early stopping

We used early stopping with a patience of 5 to stop the training when the val loss starts to increase. We stopped the training around epoch 60 (though the train loss was still decreasing) and got improved accuracy.

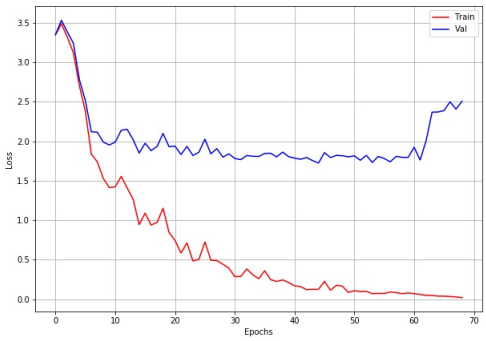

Figure 15: Early Stopping

15 Additional Creative Ideas

15.1 Augmentation Related Ideas

15.1.1 Adding types of Noise to Images

We added the following types of noise to our images:
1. Gaussian Noise
2. Salt and Pepper Noise
Adding Noise to train images makes it more robust to variations in inputs and helped us improve the accuracy.

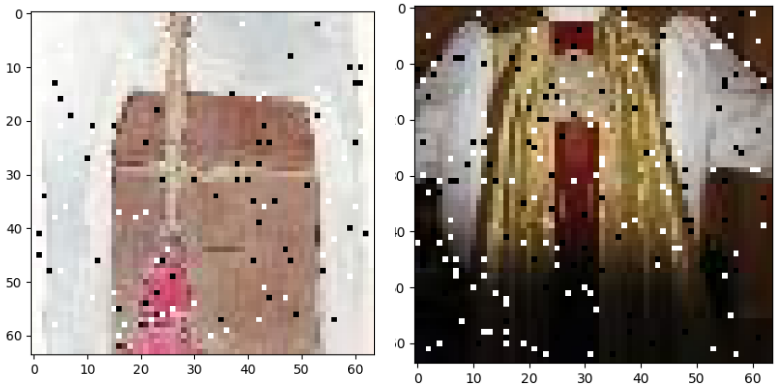

Figure 16: Salt and Pepper noise added to images

15.1.2 Greying Images

In classification task, we require the network to learn features (like edges, lines etc) which do not require colour. Hence, we tried converting image to grayscale and train the network with hope to perform better. We noticed it took less iterations to give similar results (as with color images) and slightly better performance in some runs (empirical observation).

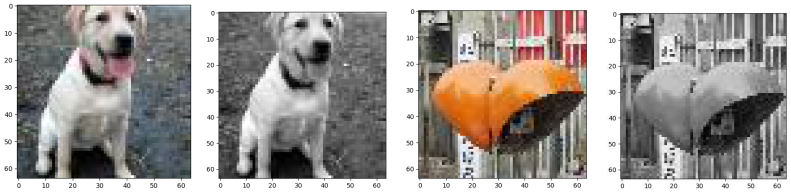

Figure 17: Colored and corresponding grayed image

15.1.3 Using Cropped Images

We carefully analyzed the images of the training set, and tried to crop few images so that the identifying features are highlighted better. Since, the given dataset had most images where the primary object to recognize was large enough, this did not help much to perform better.

15.1.4 Rotating Images through different angles

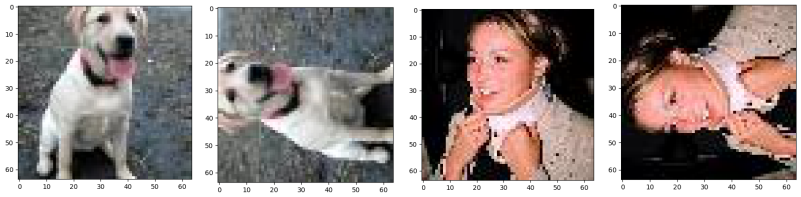

Figure 18: Rotating through different angles

15.2 Network Related Ideas

1. We applied Batch Normalization to all convolution layers and observed a multifold increase in accuracy.
2. We used 2 fully connected layers, with Dropout parameter fine-tuned for best performance.

15.3 Fooling Related Ideas

In Network Fooling, we also tried to give images to the network where noise was added proportional to the intensity of the pixel values of the image. This gave the effect of darkening the image, thereby reducing the clarity of the image and making it a bit difficult for the network to do the prediction. On the validation set, we observed a dip in accuracy from the original value of 53.6% to 24% on the modified images. Below is an example of one such image :

Figure 19: Original image(left) and Modified image(right)