# cs577: Project Report

Department of Computer Science
IllinoisInstitute of Technology

November 15, 2022

**Abstract**

In this project, we'll create a very deep convolution network to increase the resolution of lower resolution image to higher resolution. We will make use of our previous knowledge of Convolution layers taught in this course to create the architecture of network mentioned in the research paper. We will train our network two with datasets with having different size images to examine the results and observe how the network performs on different size images as well how the depth of the network effects on its performance.

## 1. Problem Statement:

We will create model inspired by the architecture of Very Deep Convolution Network for Super Resolution (VDSR) to create high resolution images from low resolution images.

### Understanding the Architecture:

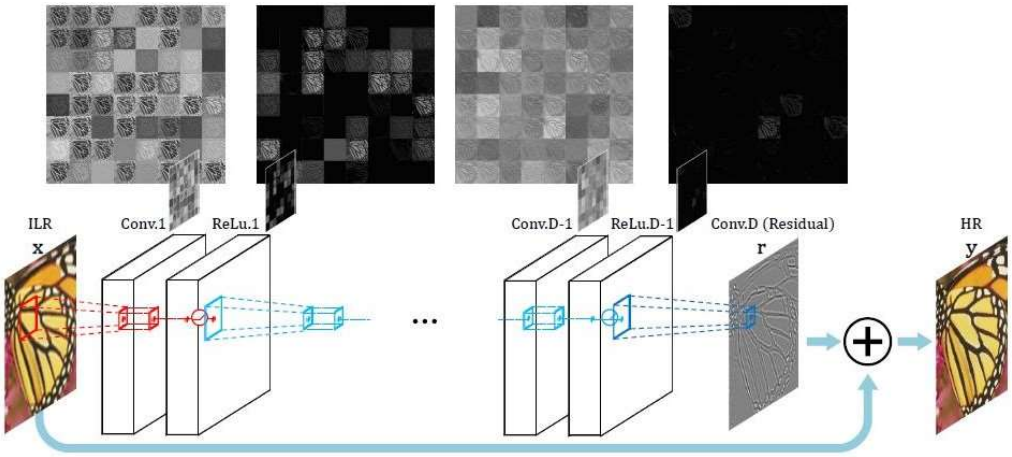



Fig.1

The VDSR network then learns the mapping between these high- and low-resolution images. This mapping is only possible as both the low- and high-resolution images share the same contents. They only differ in higher frequency details, such as amount of noise, distortion and etc.

The way this VDSR network learns to maps lower resolution images to the corresponding higher resolution images is by employing a residual learning strategy. In terms of Super Resolution, a residual image can be defined as the difference between a high-resolution image and low-resolution image which has been upscaled using bicubic interpolation. This is done so as to match the size of the reference image. A residual image holds the information related to the high-frequency (resolution) image.

To put it simply this model increases the resolution of image but as we know the resolution of an image is described in terms of PPI, PPI is Pixel Per Inch, which is the measure of pixel density. An image with higher resolution will posses a higher PPI when compared to that of a lower resolution image. The image on right can help us to understand the concept of PPI better.

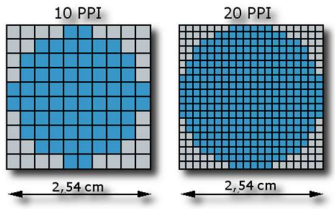



Fig.2

Despite both images having same dimensions, widths and heights, they differ in resolution. The image with more PPI has smoother edges compared to the one with less pixel per inch. We can say images with higher resolution have more information/ pixels in them. According to this, if we were to increase the resolution of pre-existing image, we need to increase its' pixel density/ PPI. This becomes the basic premise of the VDSR model.

We can infer from the architecture that the we apply Convolution layer with zero padding, followed by a Relu activation layer. The model will comprise number of consecutive Convolution layer and Relu layers for a specified depth, say 'D', the last layer i.e., Dth layer will be a 1x1 Convolution layer and have filter size of 3x3, whereas all the remaining convolution layers will have 64 filters of size 3x3. All the convolution layers have zero padding to keep the width and height of the image constant. We only increase the depth of the image as the we progress through the convolution layers.

Now for the actual working of the network we feed it an Interpolated Low-Resolution image (ILR) as input and feed the corresponding High-Resolution image as the label/ target. An Interpolated Low-Resolution image is created by upscaling a low-resolution image using bicubic interpolation. Image interpolation can be best described as trying to achieve best approximation of a pixel's color and intensity based on values of the pixels surrounding it.

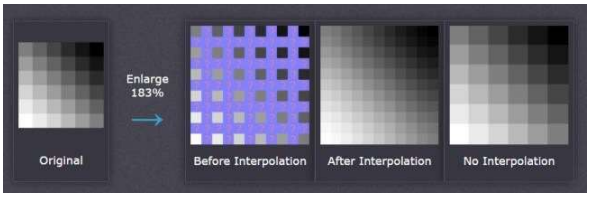


Fig. 3

In Bicubic Interpolation, a 4x4 neighbourhood of a known pixel I considered and since these pixels would be at varying distances from the unknown pixel, the pixel to the closest are given higher weights as compared to ones which are further away. Bicubic is known for producing sharper images. This is the reason for it being a standard for many editing software like Adobe Photoshop, printer drivers and in-camera interpolation.

The network learns to predict the luminance of a residual image from the luminance from the input ILR image. Luminance is channel of an image which represents the represents the brightness of each pixel through linear combination of the corresponding red, green and blue pixel for the given pixel. The VSDR network is trained only using the luminance channel of the image. For a given input image of luminance of $L_{low}$ and high-resolution image of luminance $L_{high}$, the model predicts the luminance of residual image by $L_{residual} = L_{high} - L_{low}$ from the training data.

## 2. Proposed Solution:

We will create a function which would create model while taking in depth and input dimension as parameters. We will use two datasets for this project. The first dataset of choice would be "CIFAR-10" as we are familiar with it and its easy to load and process. Even tough it has images of size 32x32x3 but we use this dataset to get ourselves familiar with working of VDSR network. We will train our model for varying depths and store the results. We will evaluate our results by calculating Peak Signal to Noise Ratio (PSNR) value between the high-resolution image and predicted images. PSNR can de defined as the ratio between maximum power of a signal and power of affecting noise which corrupts its representation.
Then we'll load our second dataset "Image Super Resolution". This dataset consists of two folders containing of low-resolution and high-resolution images. The size of these images is 256x256x3, there are a total of 855 images high as well as low resolution images.

For final evaluation we will test our models on Set5 dataset which consist of 5 images and record the results. We will create a specific model for each dataset for varying image sizes.

## 3. Implementation Detail:

The first dataset, CIFAR-10, was imported using the tf.keras.dataset and loaded using load_data () function. The next step was to pre-process the data for our requirement. Firstly, we set the ground truth which means we set the images as train label and test label. After doing so have to create the low-resolution images of these images to feed them into the network we created.

To do so we used a for loop and openCV2 to store the image in train label in a temporary variable and then resize the said variable and append this to empty variable" train data" we created to make our training data. We repeat the same process on test label to create test data. But as we resized the images using the scale we set, the hight and width of the images was altered depending upon the scale factor.

We provided the scale factor to be 2 so we the size of images in train data and test data found to be 16x16x3 which exactly the half of 32x32x3, the original image size. Now we need to upscale the images in train and test data, to do so we made use of resize_images from the keras.backend library. This can only be performed on a tensor but our images were saved in a tuple so I used the tf.convert_to_tensor to convert the tuple into tensor.

We faced one major issue here, the test data had 50,000 images of 16x16x3 which had to converted to tensor but trying to do so we got OOM error i.e., Out of Memory so we created 5 batched of training data so each batch can be converted to tensor separately and resized to be of 32x32x3. Finally, we split our training data and labels into training and validation data and labels. Now we created an instance of our model, assigned the model. Compile, set the hyperparameters and trained it on the dataset and plot the results. We implemented this to understand the working and plotting the outputs. We only implemented two models of depth 5 and 10.

For the "Image Super Resolution" dataset we already have set of folders containing the high-res and low-res images. The low-res images had a scale factor 2 which means they were reshaped to half their size and then upscaled by a factor of two create the required low-resolution image. The reason for selecting this dataset was that the data was already processed and we didn't need to process it.

We had to split the code in two parts as we faced the OOM error again when we tried to run the model for the first time. This could be due to fact that the image size was 256x256 and we didn't sufficient physical memory for the default batch size of 32. To overcome this, we implemented one code where we reduced the image size of this data set and for the other code, we used the original image size of 256x256x3 with very small batch size to train the network.

Another reason for doing so we can evaluate the performance of each model on images having different sizes and how the depth of the network effected the performance. We didn't implement Adjusted Gradient Clipping, this technique was just briefly mentioned once, just the authors of the paper even we aren't familiar with this technique and we had no knowledge about this topic.

While we trying to load CIFAR we faced the issue of OOM as we created very large tensor which could not be accommodated on the physical memory. We tried to load this data using for loop and incorporating Open CV to read the images and store them in tuple but doing so we ended up using a lot of our memory as the of every image was 256x256x3 and there were 855 images high-res images and 855 low-res images. To counter this issue, we referred to this code https://www.kaggle.com/code/harshraone/super-resolution-using-multi-scale-learning to load the data and plot the data loaded. Then we split the data by converting the data into NumPy array and then used slicing to create the required training, validation and testing dataset.

Finally, we created instances of the models with different depths, feed these models the data, one set of models was fed the reduced image dataset and other were fitted with images of size 256x256x3. We recorded our outputs and then evaluated our results to come to a conclusion.

## 4. Results:

For CIFAR-10 dataset: After pre-processing the dataset, the data and labels we prepared. Fig. 4(below)


```
After upscaling the input images:
Train data shape: (40000, 32, 32, 3)
Train label shape: (40000, 32, 32, 3)
Vali data shape: (10000, 32, 32, 3)
Vali label shape: (10000, 32, 32, 3)
Test data shape: (10000, 32, 32, 3)
Test label shape: (10000, 32, 32, 3)
```

The summary of two models of depths 5 and 10 layers. These were trained on the CIFAR dataset and we evaluated their performance by calculating PSNR values.

5 Layer Model

```
cifar_l5 model(input_dim,5)
cifar_l5.summary()
```

```
(32, 32, 3)
Model: "model"
```

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input (InputLayer) | [(None, 32, 32, 3)] | 0 | [] |
| zero_padding2d (ZeroPadding2D) | (None, 34, 34, 3) | 0 | ['input[0][0]'] |
| CONV1 (Conv2D) | (None, 32, 32, 64) | 1792 | ['zero_padding2d[0][0]'] |
| re_lu (ReLU) | (None, 32, 32, 64) | 0 | ['CONV1[0][0]'] |
| zero_padding2d_1 (ZeroPadding2D) | (None, 34, 34, 64) | 0 | ['re_lu[0][0]'] |
| CONV2 (Conv2D) | (None, 32, 32, 64) | 36928 | ['zero_padding2d_1[0][0]'] |
| re_lu_1 (ReLU) | (None, 32, 32, 64) | 0 | ['CONV2[0][0]'] |
| zero_padding2d_2 (ZeroPadding2D) | (None, 34, 34, 64) | 0 | ['re_lu_1[0][0]'] |
| CONV3 (Conv2D) | (None, 32, 32, 64) | 36928 | ['zero_padding2d_2[0][0]'] |
| re_lu_2 (ReLU) | (None, 32, 32, 64) | 0 | ['CONV3[0][0]'] |
| zero_padding2d_3 (ZeroPadding2D) | (None, 34, 34, 64) | 0 | ['re_lu_2[0][0]'] |
| CONV4 (Conv2D) | (None, 32, 32, 64) | 36928 | ['zero_padding2d_3[0][0]'] |
| re_lu_3 (ReLU) | (None, 32, 32, 64) | 0 | ['CONV4[0][0]'] |
| zero_padding2d_4 (ZeroPadding2D) | (None, 34, 34, 64) | 0 | ['re_lu_3[0][0]'] |
| CONV5 (Conv2D) | (None, 32, 32, 1) | 577 | ['zero_padding2d_4[0][0]'] |
| add (Add) | (None, 32, 32, 3) | 0 | ['input[0][0]', 'CONV5[0][0]'] |

```
Total params: 119,153
Trainable params: 119,153
Non-trainable params: 0
```

Fig. 5

10 Layer model

```
cifar_l10 = model10(input_dim)
cifar_l10.summary()
```

```
Model: "model_1"
```

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_2 (InputLayer) | [(None, 32, 32, 3)] | 0 | [] |
| conv2d_5 (Conv2D) | (None, 32, 32, 64) | 1792 | ['input_2[0][0]'] |
| re_lu_8 (ReLU) | (None, 32, 32, 64) | 0 | ['conv2d_5[0][0]'] |
| conv2d_6 (Conv2D) | (None, 32, 32, 64) | 36928 | ['re_lu_8[0][0]'] |
| re_lu_9 (ReLU) | (None, 32, 32, 64) | 0 | ['conv2d_6[0][0]'] |
| conv2d_7 (Conv2D) | (None, 32, 32, 64) | 36928 | ['re_lu_9[0][0]'] |
| re_lu_10 (ReLU) | (None, 32, 32, 64) | 0 | ['conv2d_7[0][0]'] |
| conv2d_8 (Conv2D) | (None, 32, 32, 64) | 36928 | ['re_lu_10[0][0]'] |
| re_lu_11 (ReLU) | (None, 32, 32, 64) | 0 | ['conv2d_8[0][0]'] |
| conv2d_9 (Conv2D) | (None, 32, 32, 64) | 36928 | ['re_lu_11[0][0]'] |
| re_lu_12 (ReLU) | (None, 32, 32, 64) | 0 | ['conv2d_9[0][0]'] |
| conv2d_10 (Conv2D) | (None, 32, 32, 64) | 36928 | ['re_lu_12[0][0]'] |
| re_lu_13 (ReLU) | (None, 32, 32, 64) | 0 | ['conv2d_10[0][0]'] |
| conv2d_11 (Conv2D) | (None, 32, 32, 64) | 36928 | ['re_lu_13[0][0]'] |
| re_lu_14 (ReLU) | (None, 32, 32, 64) | 0 | ['conv2d_11[0][0]'] |
| conv2d_12 (Conv2D) | (None, 32, 32, 64) | 36928 | ['re_lu_14[0][0]'] |
| re_lu_15 (ReLU) | (None, 32, 32, 64) | 0 | ['conv2d_12[0][0]'] |
| conv2d_13 (Conv2D) | (None, 32, 32, 64) | 36928 | ['re_lu_15[0][0]'] |
| re_lu_16 (ReLU) | (None, 32, 32, 64) | 0 | ['conv2d_13[0][0]'] |
| conv2d_14 (Conv2D) | (None, 32, 32, 1) | 577 | ['re_lu_16[0][0]'] |
| add_1 (Add) | (None, 32, 32, 3) | 0 | ['input_2[0][0]', 'conv2d_14[0][0]'] |

```
Total params: 297,793
Trainable params: 297,793
Non-trainable params: 0
```

Fig. 6

Both models were trained for 10 epochs with a batch size of 64, Adam as optimizer and a learning rate of 0.0001, beta_1=0.9.

These are the outputs for layer 5 Model

```
Maximum psnr value: 90.75793099593488
Minimum PSNR value: 67.10327293532522
Average PSNR value: 75.71551563253736
```

Plotting the output

```
plot_output(test_label,test,y_pred,cifar_l5_psnr)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..2
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..2
```

```
PSNR value 73.37904031693259
```


```
PSNR value 74.27174695433888
```


```
PSNR value 76.0954141151014
```

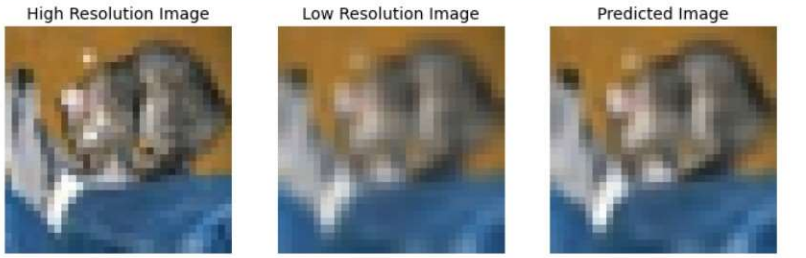
Fig. 7

These are the results from later 10 model. (Fig. 8 below)

```
Maximum psnr value: 90.77963159850285
Minimum PSNR value: 67.16190871738924
Average PSNR value: 75.88647588521926
```

```
1 plot_output(test_label,test,y_pred,cifar_l10_psnr)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..2
```




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..2
```

```
PSNR value 73.18976305329967
```


```
PSNR value 74.10565829635279
```


```
PSNR value 76.1984259454641
```

For the results we can infer both models perform very similarly This could be due the fact the image size being really small, 32x32. These models converge very fast the size of the images are smaller than the other ones.
We can say the size of the image has a huge factor in the computation speed of the network. We'll confirm this by looking at the performances of the other models.

We made 4 models to train on mage Super Resolution dataset when the sizes of image were reduced to 100x100x3.

This is the first model with depth of 5 layers.

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 100, 100, 3) 0
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 100, 100, 64) 1792        input_1[0][0]
__________________________________________________________________________________________________
re_lu (ReLU)                    (None, 100, 100, 64) 0           conv2d[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 100, 100, 64) 36928       re_lu[0][0]
__________________________________________________________________________________________________
re_lu_1 (ReLU)                  (None, 100, 100, 64) 0           conv2d_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 100, 100, 64) 36928       re_lu_1[0][0]
__________________________________________________________________________________________________
re_lu_2 (ReLU)                  (None, 100, 100, 64) 0           conv2d_2[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 100, 100, 64) 36928       re_lu_2[0][0]
__________________________________________________________________________________________________
re_lu_3 (ReLU)                  (None, 100, 100, 64) 0           conv2d_3[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 100, 100, 1)  577         re_lu_3[0][0]
__________________________________________________________________________________________________
add (Add)                       (None, 100, 100, 3)  0           input_1[0][0]
                                                                 conv2d_4[0][0]
==================================================================================================
Total params: 113,153
Trainable params: 113,153
Non-trainable params: 0
```

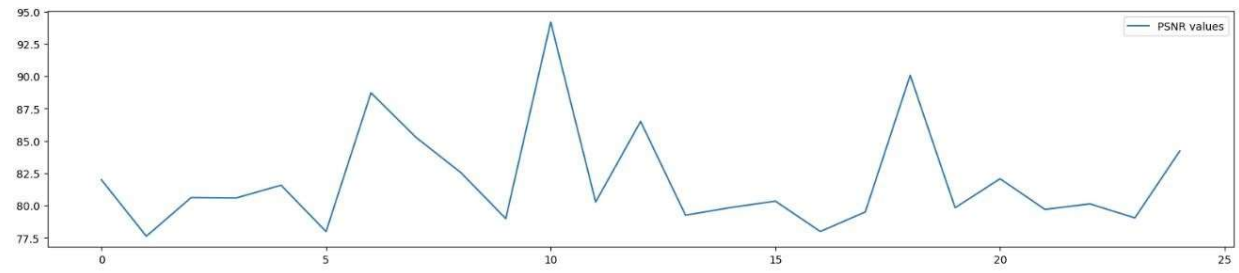
This model for trained for 10 epochs with default batch size for image sizes 100x100.

While training the first epoch took 91ms/step and rest of the epoch took 48ms/step – 49ms/step. We can predict this model was easy to train as it was not very deep and image size of was moderate.

This has a total of 113,153 trainable parameters.

Fig. 9 (left)

These are the predictions made by the model with 5 layers on 25 test images of size 100x100x3 of the Image Super resolution dataset. Fig. 10 (below)




```
Maximum psnr value: 94.19227506195163
Minimum PSNR value: 77.62178609938891
Average PSNR value: 81.95185692865257
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
PSNR value 81.98996161541376
```

These are the outputs we achieved with model. We got a maximum PSNR value of 94.19, minimum PSNR of 77.62 and an average PSNR of 81.95. We also plotted a graph to see the PSNR values of every image in the test dataset.

```
model_110 = VDSR10(inp)
model_110.summary()
```

```
Model: "model_1"
_________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=========================================================================================
input_2 (InputLayer)         [(None, 100, 100, 3) 0
_________________________________________________________________________________________
conv2d_5 (Conv2D)            (None, 100, 100, 64) 1792        input_2[0][0]
_________________________________________________________________________________________
re_lu_4 (ReLU)               (None, 100, 100, 64) 0           conv2d_5[0][0]
_________________________________________________________________________________________
conv2d_6 (Conv2D)            (None, 100, 100, 64) 36928       re_lu_4[0][0]
_________________________________________________________________________________________
re_lu_5 (ReLU)               (None, 100, 100, 64) 0           conv2d_6[0][0]
_________________________________________________________________________________________
conv2d_7 (Conv2D)            (None, 100, 100, 64) 36928       re_lu_5[0][0]
_________________________________________________________________________________________
re_lu_6 (ReLU)               (None, 100, 100, 64) 0           conv2d_7[0][0]
_________________________________________________________________________________________
conv2d_8 (Conv2D)            (None, 100, 100, 64) 36928       re_lu_6[0][0]
_________________________________________________________________________________________
re_lu_7 (ReLU)               (None, 100, 100, 64) 0           conv2d_8[0][0]
_________________________________________________________________________________________
conv2d_9 (Conv2D)            (None, 100, 100, 64) 36928       re_lu_7[0][0]
_________________________________________________________________________________________
re_lu_8 (ReLU)               (None, 100, 100, 64) 0           conv2d_9[0][0]
_________________________________________________________________________________________
conv2d_10 (Conv2D)           (None, 100, 100, 64) 36928       re_lu_8[0][0]
_________________________________________________________________________________________
re_lu_9 (ReLU)               (None, 100, 100, 64) 0           conv2d_10[0][0]
_________________________________________________________________________________________
conv2d_11 (Conv2D)           (None, 100, 100, 64) 36928       re_lu_9[0][0]
_________________________________________________________________________________________
re_lu_10 (ReLU)              (None, 100, 100, 64) 0           conv2d_11[0][0]
_________________________________________________________________________________________
conv2d_12 (Conv2D)           (None, 100, 100, 64) 36928       re_lu_10[0][0]
_________________________________________________________________________________________
re_lu_11 (ReLU)              (None, 100, 100, 64) 0           conv2d_12[0][0]
_________________________________________________________________________________________
conv2d_13 (Conv2D)           (None, 100, 100, 64) 36928       re_lu_11[0][0]
_________________________________________________________________________________________
re_lu_12 (ReLU)              (None, 100, 100, 64) 0           conv2d_13[0][0]
_________________________________________________________________________________________
conv2d_14 (Conv2D)           (None, 100, 100, 1)  577         re_lu_12[0][0]
_________________________________________________________________________________________
add_1 (Add)                  (None, 100, 100, 3)  0           input_2[0][0]
                                                              conv2d_14[0][0]
=========================================================================================
Total params: 297,793
Trainable params: 297,793
Non-trainable params: 0
```

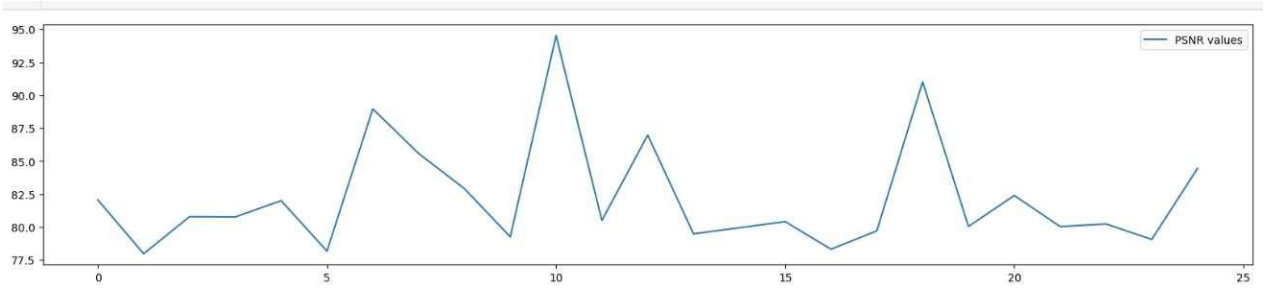
This is the second model with 10 layers. This model was trained for 10 epochs but we changed slightly changed the learning rate from past model. We set the learning rate to be 0.0005.

This model has approximately twice the number of trainable parameters when compared to the previous one. While training each epoch tool 110-115ms/step. This model was also easy to train as the image size was very large

Fig. 11 (left)

These are the predictions made by the model with 10 layers on 25 test images of size 100x100x3 of the Image Super resolution dataset.

Fig. 12 (below)




```
Maximum psnr value: 94.52585184538634
Minimum PSNR value: 77.96944144935829
Average PSNR value: 82.22284161828495
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
PSNR value 82.05784627455509
```

We can see slight improve in the Average PSNR values. This model having more layers helped the model to achieve a little higher resolution in the predicted images.

Now we will at the summary of the 15-layer model.

```
model_115=VDSR15(inp)
model_115.summary()
model_115.compile(optimizer=Adam(learning_rate=0.0007,beta_1=0.9), loss='mse', metrics=['accur
```

```
Model: "model_2"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_3 (InputLayer)            [(None, 100, 100, 3) 0
__________________________________________________________________________________________________
conv2d_15 (Conv2D)              (None, 100, 100, 64) 1792        input_3[0][0]
__________________________________________________________________________________________________
re_lu_13 (ReLU)                 (None, 100, 100, 64) 0           conv2d_15[0][0]
__________________________________________________________________________________________________
conv2d_16 (Conv2D)              (None, 100, 100, 64) 36928       re_lu_13[0][0]
__________________________________________________________________________________________________
re_lu_14 (ReLU)                 (None, 100, 100, 64) 0           conv2d_16[0][0]
__________________________________________________________________________________________________
conv2d_17 (Conv2D)              (None, 100, 100, 64) 36928       re_lu_14[0][0]
__________________________________________________________________________________________________
re_lu_15 (ReLU)                 (None, 100, 100, 64) 0           conv2d_17[0][0]
__________________________________________________________________________________________________
conv2d_18 (Conv2D)              (None, 100, 100, 64) 36928       re_lu_15[0][0]
__________________________________________________________________________________________________
re_lu_16 (ReLU)                 (None, 100, 100, 64) 0           conv2d_18[0][0]
__________________________________________________________________________________________________
conv2d_19 (Conv2D)              (None, 100, 100, 64) 36928       re_lu_16[0][0]
__________________________________________________________________________________________________
re_lu_17 (ReLU)                 (None, 100, 100, 64) 0           conv2d_19[0][0]
__________________________________________________________________________________________________
conv2d_20 (Conv2D)              (None, 100, 100, 64) 36928       re_lu_17[0][0]
__________________________________________________________________________________________________
re_lu_18 (ReLU)                 (None, 100, 100, 64) 0           conv2d_20[0][0]
__________________________________________________________________________________________________
conv2d_21 (Conv2D)              (None, 100, 100, 64) 36928       re_lu_18[0][0]
__________________________________________________________________________________________________
re_lu_19 (ReLU)                 (None, 100, 100, 64) 0           conv2d_21[0][0]
__________________________________________________________________________________________________
conv2d_22 (Conv2D)              (None, 100, 100, 64) 36928       re_lu_19[0][0]
__________________________________________________________________________________________________
re_lu_20 (ReLU)                 (None, 100, 100, 64) 0           conv2d_22[0][0]
__________________________________________________________________________________________________
conv2d_23 (Conv2D)              (None, 100, 100, 64) 36928       re_lu_20[0][0]
__________________________________________________________________________________________________
re_lu_21 (ReLU)                 (None, 100, 100, 64) 0           conv2d_23[0][0]
__________________________________________________________________________________________________
conv2d_24 (Conv2D)              (None, 100, 100, 64) 36928       re_lu_21[0][0]
__________________________________________________________________________________________________
re_lu_22 (ReLU)                 (None, 100, 100, 64) 0           conv2d_24[0][0]
__________________________________________________________________________________________________
conv2d_25 (Conv2D)              (None, 100, 100, 64) 36928       re_lu_22[0][0]
__________________________________________________________________________________________________
re_lu_23 (ReLU)                 (None, 100, 100, 64) 0           conv2d_25[0][0]
__________________________________________________________________________________________________
conv2d_26 (Conv2D)              (None, 100, 100, 64) 36928       re_lu_23[0][0]
__________________________________________________________________________________________________
re_lu_24 (ReLU)                 (None, 100, 100, 64) 0           conv2d_26[0][0]
__________________________________________________________________________________________________
conv2d_27 (Conv2D)              (None, 100, 100, 64) 36928       re_lu_24[0][0]
__________________________________________________________________________________________________
re_lu_25 (ReLU)                 (None, 100, 100, 64) 0           conv2d_27[0][0]
__________________________________________________________________________________________________
conv2d_28 (Conv2D)              (None, 100, 100, 64) 36928       re_lu_25[0][0]
__________________________________________________________________________________________________
re_lu_26 (ReLU)                 (None, 100, 100, 64) 0           conv2d_28[0][0]
__________________________________________________________________________________________________
conv2d_29 (Conv2D)              (None, 100, 100, 1)  577         re_lu_26[0][0]
__________________________________________________________________________________________________
add_2 (Add)                     (None, 100, 100, 3)  0           input_3[0][0]
                                                                 conv2d_29[0][0]
==================================================================================================
Total params: 482,433
Trainable params: 482,433
Non-trainable params: 0
__________________________________________________________________________________________________
```

For here the models started getting deeper. We had never built such deep networks before. Training this network are not a very complex for smaller sized images but for larger sized images this model would take longer time to train.

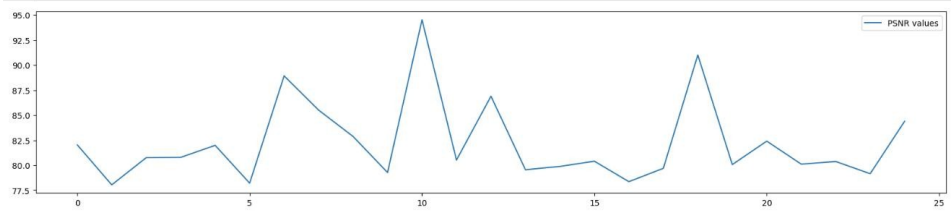
For training of this network, we specified the epoch to be 10 and increased the learning rate to 0.007.

Each epoch was completed in the range of 170-177ms/step. We kept the batch size to be the default one.

And below are the results of this model on test of Image Super Resolution dataset consisting of 25 images of size 100x100x3.
We can see similar results to that of the 10-layer model and there aren't any significant changes in the PSNR values.

Fig. 13 (left)

The average PSNR value stays relatively similar to that of 10-layer model, we observe a slight in minimum PSNR values.
Fig. 14 (below)




```
Maximum psnr value: 94.5279913729024
Minimum PSNR value: 78.04142831254168
Average PSNR value: 82.23253825927748
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
PSNR value 82.04015788866411
```

```
[8]: model_l20 = VDSR20(inp)
     model_l20.summary()
     model_l20.compile(optimizer=Adam(learning_rate=0.0009,beta_1=0.9), loss='mse', metrics=['accuracy
```

Model: "model_5"

| Layer (type) | Output Shape | Param # | Connected to |
|---|---|---|---|
| input_6 (InputLayer) | [(None, 100, 100, 3) | 0 | |
| conv2d_60 (Conv2D) | (None, 100, 100, 64) | 1792 | input_6[0][0] |
| re_lu_55 (ReLU) | (None, 100, 100, 64) | 0 | conv2d_60[0][0] |
| conv2d_61 (Conv2D) | (None, 100, 100, 64) | 36928 | re_lu_55[0][0] |
| re_lu_56 (ReLU) | (None, 100, 100, 64) | 0 | conv2d_61[0][0] |
| conv2d_62 (Conv2D) | (None, 100, 100, 64) | 36928 | re_lu_56[0][0] |
| re_lu_57 (ReLU) | (None, 100, 100, 64) | 0 | conv2d_62[0][0] |
| conv2d_63 (Conv2D) | (None, 100, 100, 64) | 36928 | re_lu_57[0][0] |
| re_lu_58 (ReLU) | (None, 100, 100, 64) | 0 | conv2d_63[0][0] |
| conv2d_64 (Conv2D) | (None, 100, 100, 64) | 36928 | re_lu_58[0][0] |
| re_lu_59 (ReLU) | (None, 100, 100, 64) | 0 | conv2d_64[0][0] |
| conv2d_65 (Conv2D) | (None, 100, 100, 64) | 36928 | re_lu_59[0][0] |
| re_lu_60 (ReLU) | (None, 100, 100, 64) | 0 | conv2d_65[0][0] |
| conv2d_66 (Conv2D) | (None, 100, 100, 64) | 36928 | re_lu_60[0][0] |
| re_lu_61 (ReLU) | (None, 100, 100, 64) | 0 | conv2d_66[0][0] |
| conv2d_67 (Conv2D) | (None, 100, 100, 64) | 36928 | re_lu_61[0][0] |
| re_lu_62 (ReLU) | (None, 100, 100, 64) | 0 | conv2d_67[0][0] |
| conv2d_68 (Conv2D) | (None, 100, 100, 64) | 36928 | re_lu_62[0][0] |
| re_lu_63 (ReLU) | (None, 100, 100, 64) | 0 | conv2d_68[0][0] |
| conv2d_69 (Conv2D) | (None, 100, 100, 64) | 36928 | re_lu_63[0][0] |
| re_lu_64 (ReLU) | (None, 100, 100, 64) | 0 | conv2d_69[0][0] |
| conv2d_70 (Conv2D) | (None, 100, 100, 64) | 36928 | re_lu_64[0][0] |
| re_lu_65 (ReLU) | (None, 100, 100, 64) | 0 | conv2d_70[0][0] |
| conv2d_71 (Conv2D) | (None, 100, 100, 64) | 36928 | re_lu_65[0][0] |
| re_lu_66 (ReLU) | (None, 100, 100, 64) | 0 | conv2d_71[0][0] |
| conv2d_72 (Conv2D) | (None, 100, 100, 64) | 36928 | re_lu_66[0][0] |
| re_lu_67 (ReLU) | (None, 100, 100, 64) | 0 | conv2d_72[0][0] |
| conv2d_73 (Conv2D) | (None, 100, 100, 64) | 36928 | re_lu_67[0][0] |
| re_lu_68 (ReLU) | (None, 100, 100, 64) | 0 | conv2d_73[0][0] |
| conv2d_74 (Conv2D) | (None, 100, 100, 64) | 36928 | re_lu_68[0][0] |
| re_lu_69 (ReLU) | (None, 100, 100, 64) | 0 | conv2d_74[0][0] |
| conv2d_75 (Conv2D) | (None, 100, 100, 64) | 36928 | re_lu_69[0][0] |
| re_lu_70 (ReLU) | (None, 100, 100, 64) | 0 | conv2d_75[0][0] |
| conv2d_76 (Conv2D) | (None, 100, 100, 64) | 36928 | re_lu_70[0][0] |
| re_lu_71 (ReLU) | (None, 100, 100, 64) | 0 | conv2d_76[0][0] |
| conv2d_77 (Conv2D) | (None, 100, 100, 64) | 36928 | re_lu_71[0][0] |
| re_lu_72 (ReLU) | (None, 100, 100, 64) | 0 | conv2d_77[0][0] |
| conv2d_78 (Conv2D) | (None, 100, 100, 64) | 36928 | re_lu_72[0][0] |
| re_lu_73 (ReLU) | (None, 100, 100, 64) | 0 | conv2d_78[0][0] |
| conv2d_79 (Conv2D) | (None, 100, 100, 1) | 577 | re_lu_73[0][0] |
| add_5 (Add) | (None, 100, 100, 3) | 0 | input_6[0][0]<br>conv2d_79[0][0] |

Total params: 667,073
Trainable params: 667,073
Non-trainable params: 0

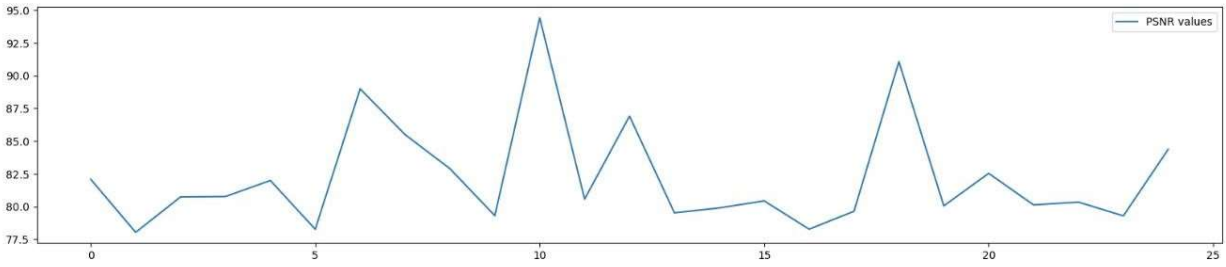
Fig. 15

This the mammoth 20-layer deep mode. The deepest model we have every created. It had around 600,000 trainable parameters. This a very deep network.

To train this model, we set the learning rate at 0.009 and trained it for 10 epochs while keeping the batch size default.

Training this network for small size image seems like an overkill but we can only say something on this after looking at the results.

Here are the results of the 20-layer network for test set of 25 images which were resized to 100x100x3.




```
Maximum psnr value: 94.41647871441019
Minimum PSNR value: 78.02908746185197
Average PSNR value: 82.24007084432304
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
PSNR value 82.08008428346795
```

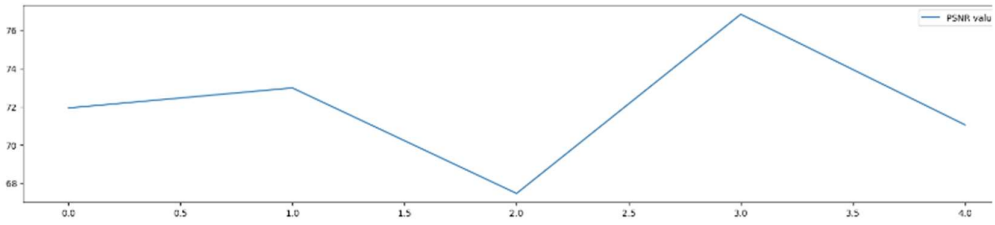
Fig. 16

The results very close to the 15-layer network and don't have significant improvement over the previous ones. We can definitely say we didn't need such a deep layer to implement super resolution on images of smaller sizes.

Now we test these models un-trained dataset, Set5, which consisted 5 RGB images, which had to pre processed to create low resolution images and resized to 100x100x3 as these models were trained on 100x100x3 sized images.

Output for 5- layer model:

```
56]: 1 plot_psnr(psnr_15)
```




```
Maximum psnr value: 76.82350719754461
Minimum PSNR value: 67.46321202290873
Average PSNR value: 72.04777329875927
```

```
57]: 1 plot_output_final(set5_out,set5_inp,pred_15,psnr_15)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

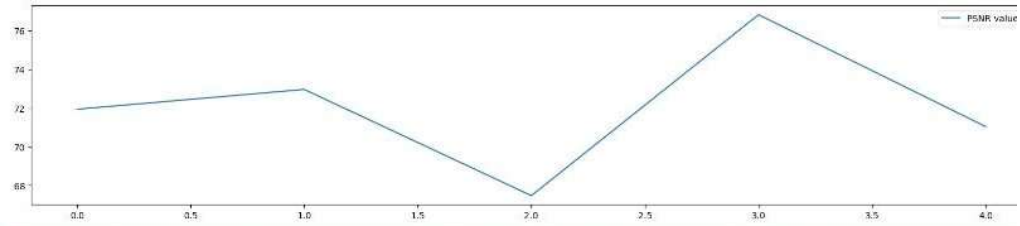
```
PSNR value 71.92662860635079
```


PSNR value 72.97550751549244


PSNR value 67.46321202290873

Fig. 17

Output for 10- layer model:




Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Maximum psnr value: 76.83099023408647
Minimum PSNR value: 67.46807878931378
Average PSNR value: 72.0485211665302


PSNR value 71.93877223976742

Fig. 18 a.

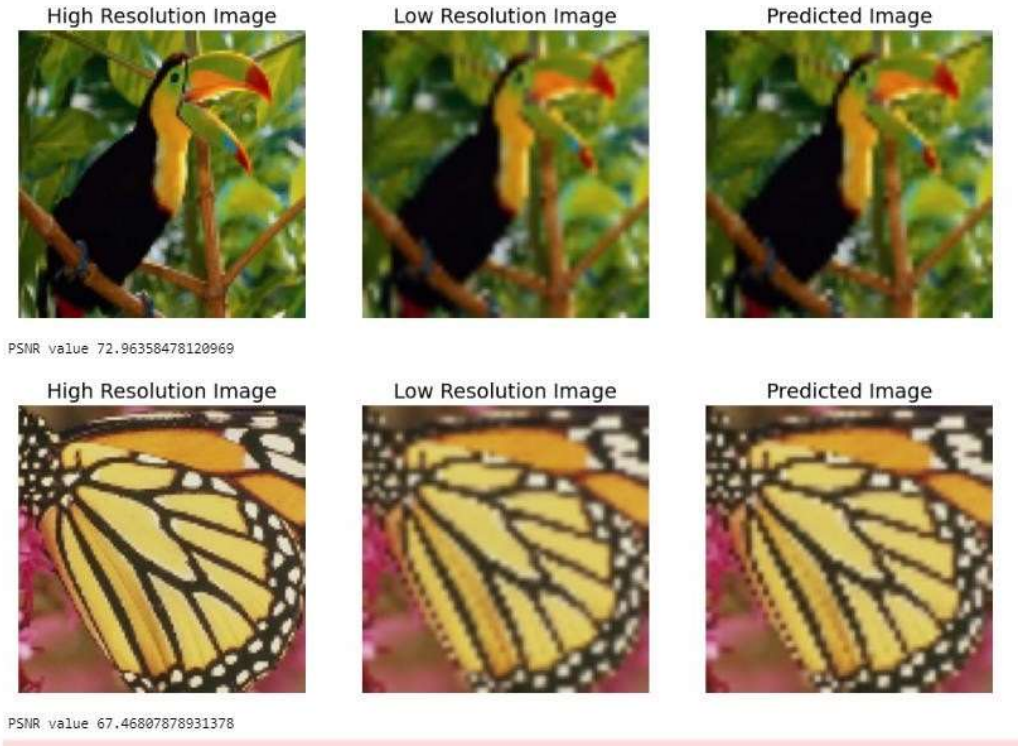
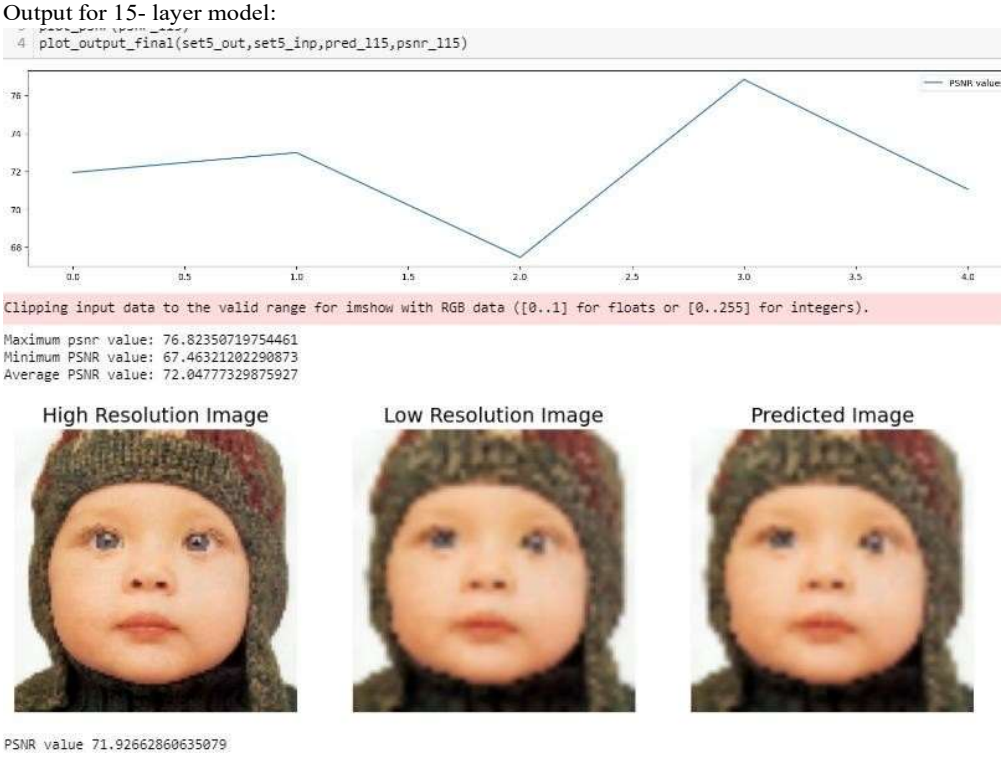



Fig. 18 b.

Output for 15- layer model:



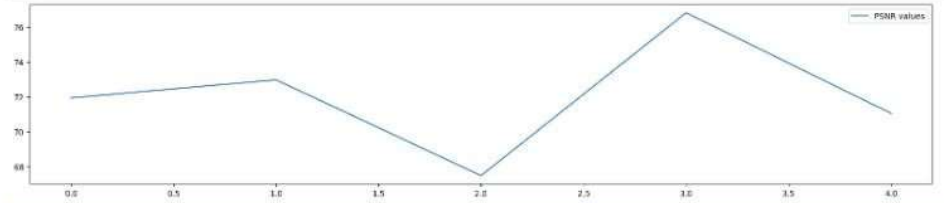

```
PSNR value 72.97550751549244
```


```
PSNR value 67.46321202290873
```

Fig. 19

Output for 20- layer model:

```
pred_l20 = model_l20.predict(set5_inp)
psnr_l20 = PSNR(set5_out,pred_l20)
plot_psnr(psnr_l20)
plot_output_final(set5_out,set5_inp,pred_l20,psnr_l20)
```




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Maximum psnr value: 76.82599349665594
Minimum PSNR value: 67.474225561178
Average PSNR value: 72.0518834899664
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
PSNR value 71.93669695780186
```

Fig. 20

All the models achieve similar results for small size images of dimensions 100x100x3.
Now we will look the model which were trained on images of size 256x256x3, observe their performance and in the end compare their results to that of these models.

```
Model: "model"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_1 (InputLayer)            [(None, 256, 256, 3) 0
__________________________________________________________________________________________________
conv2d (Conv2D)                 (None, 256, 256, 64) 1792        input_1[0][0]
__________________________________________________________________________________________________
re_lu (ReLU)                    (None, 256, 256, 64) 0           conv2d[0][0]
__________________________________________________________________________________________________
conv2d_1 (Conv2D)               (None, 256, 256, 64) 36928       re_lu[0][0]
__________________________________________________________________________________________________
re_lu_1 (ReLU)                  (None, 256, 256, 64) 0           conv2d_1[0][0]
__________________________________________________________________________________________________
conv2d_2 (Conv2D)               (None, 256, 256, 64) 36928       re_lu_1[0][0]
__________________________________________________________________________________________________
re_lu_2 (ReLU)                  (None, 256, 256, 64) 0           conv2d_2[0][0]
__________________________________________________________________________________________________
conv2d_3 (Conv2D)               (None, 256, 256, 64) 36928       re_lu_2[0][0]
__________________________________________________________________________________________________
re_lu_3 (ReLU)                  (None, 256, 256, 64) 0           conv2d_3[0][0]
__________________________________________________________________________________________________
conv2d_4 (Conv2D)               (None, 256, 256, 1)  577         re_lu_3[0][0]
__________________________________________________________________________________________________
add (Add)                       (None, 256, 256, 3)  0           input_1[0][0]
                                                                 conv2d_4[0][0]
==================================================================================================
Total params: 113,153
Trainable params: 113,153
Non-trainable params: 0
__________________________________________________________________________________________________
```

Fig. 21

The summary shows the model architecture of 5-layer model. We selected same optimizer: Adam for all of these models.
For training of this model, we set the learning rate to 0.0001 and the number of epochs to be 5 and reduced the default batch size to 5 as we were getting an OOM error as tensor of shape 32x64x256x256 was very talking up the entire remaining memory. We were able to train the same network with default batch when the input image size was 100x100x3.

While training each epoch was executed in between 58-60ms/step. Training speed was slightly slower than the model which was trained on 100x100 sized images.

```
model_l10 = VDSR10(inp)
model_l10.summary()
```

```
Model: "model_1"
_________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=================================================================================
input_2 (InputLayer)         [(None, 256, 256, 3) 0
_________________________________________________________________________________
conv2d_5 (Conv2D)            (None, 256, 256, 64) 1792        input_2[0][0]
_________________________________________________________________________________
re_lu_4 (ReLU)               (None, 256, 256, 64) 0           conv2d_5[0][0]
_________________________________________________________________________________
conv2d_6 (Conv2D)            (None, 256, 256, 64) 36928       re_lu_4[0][0]
_________________________________________________________________________________
re_lu_5 (ReLU)               (None, 256, 256, 64) 0           conv2d_6[0][0]
_________________________________________________________________________________
conv2d_7 (Conv2D)            (None, 256, 256, 64) 36928       re_lu_5[0][0]
_________________________________________________________________________________
re_lu_6 (ReLU)               (None, 256, 256, 64) 0           conv2d_7[0][0]
_________________________________________________________________________________
conv2d_8 (Conv2D)            (None, 256, 256, 64) 36928       re_lu_6[0][0]
_________________________________________________________________________________
re_lu_7 (ReLU)               (None, 256, 256, 64) 0           conv2d_8[0][0]
_________________________________________________________________________________
conv2d_9 (Conv2D)            (None, 256, 256, 64) 36928       re_lu_7[0][0]
_________________________________________________________________________________
re_lu_8 (ReLU)               (None, 256, 256, 64) 0           conv2d_9[0][0]
_________________________________________________________________________________
conv2d_10 (Conv2D)           (None, 256, 256, 64) 36928       re_lu_8[0][0]
_________________________________________________________________________________
re_lu_9 (ReLU)               (None, 256, 256, 64) 0           conv2d_10[0][0]
_________________________________________________________________________________
conv2d_11 (Conv2D)           (None, 256, 256, 64) 36928       re_lu_9[0][0]
_________________________________________________________________________________
re_lu_10 (ReLU)              (None, 256, 256, 64) 0           conv2d_11[0][0]
_________________________________________________________________________________
conv2d_12 (Conv2D)           (None, 256, 256, 64) 36928       re_lu_10[0][0]
_________________________________________________________________________________
re_lu_11 (ReLU)              (None, 256, 256, 64) 0           conv2d_12[0][0]
_________________________________________________________________________________
conv2d_13 (Conv2D)           (None, 256, 256, 64) 36928       re_lu_11[0][0]
_________________________________________________________________________________
re_lu_12 (ReLU)              (None, 256, 256, 64) 0           conv2d_13[0][0]
_________________________________________________________________________________
conv2d_14 (Conv2D)           (None, 256, 256, 1)  577         re_lu_12[0][0]
_________________________________________________________________________________
add_1 (Add)                  (None, 256, 256, 3)  0           input_2[0][0]
                                                              conv2d_14[0][0]
=================================================================================
Total params: 297,793
Trainable params: 297,793
Non-trainable params: 0
```

Fig. 22

The summary shows the model architecture of 10-layer model. We selected same optimizer
For training of this model, we set the number of epochs to be 10, batch size of 5 with a learning rate of 0.0005
While training each epoch was executed in between 136-138ms/step.

```
3 model_l15.compile(optimizer=Adam(learning_rate=0.0007,beta_1=0.9), loss='mse', m
```

```
Model: "model_2"
_________________________________________________________________________________________
Layer (type)                 Output Shape              Param #     Connected to
=========================================================================================
input_3 (InputLayer)         [(None, 256, 256, 3)      0
_________________________________________________________________________________________
conv2d_15 (Conv2D)           (None, 256, 256, 64)      1792        input_3[0][0]
_________________________________________________________________________________________
re_lu_13 (ReLU)              (None, 256, 256, 64)      0           conv2d_15[0][0]
_________________________________________________________________________________________
conv2d_16 (Conv2D)           (None, 256, 256, 64)      36928       re_lu_13[0][0]
_________________________________________________________________________________________
re_lu_14 (ReLU)              (None, 256, 256, 64)      0           conv2d_16[0][0]
_________________________________________________________________________________________
conv2d_17 (Conv2D)           (None, 256, 256, 64)      36928       re_lu_14[0][0]
_________________________________________________________________________________________
re_lu_15 (ReLU)              (None, 256, 256, 64)      0           conv2d_17[0][0]
_________________________________________________________________________________________
conv2d_18 (Conv2D)           (None, 256, 256, 64)      36928       re_lu_15[0][0]
_________________________________________________________________________________________
re_lu_16 (ReLU)              (None, 256, 256, 64)      0           conv2d_18[0][0]
_________________________________________________________________________________________
conv2d_19 (Conv2D)           (None, 256, 256, 64)      36928       re_lu_16[0][0]
_________________________________________________________________________________________
re_lu_17 (ReLU)              (None, 256, 256, 64)      0           conv2d_19[0][0]
_________________________________________________________________________________________
conv2d_20 (Conv2D)           (None, 256, 256, 64)      36928       re_lu_17[0][0]
_________________________________________________________________________________________
re_lu_18 (ReLU)              (None, 256, 256, 64)      0           conv2d_20[0][0]
_________________________________________________________________________________________
conv2d_21 (Conv2D)           (None, 256, 256, 64)      36928       re_lu_18[0][0]
_________________________________________________________________________________________
re_lu_19 (ReLU)              (None, 256, 256, 64)      0           conv2d_21[0][0]
_________________________________________________________________________________________
conv2d_22 (Conv2D)           (None, 256, 256, 64)      36928       re_lu_19[0][0]
_________________________________________________________________________________________
re_lu_20 (ReLU)              (None, 256, 256, 64)      0           conv2d_22[0][0]
_________________________________________________________________________________________
conv2d_23 (Conv2D)           (None, 256, 256, 64)      36928       re_lu_20[0][0]
_________________________________________________________________________________________
re_lu_21 (ReLU)              (None, 256, 256, 64)      0           conv2d_23[0][0]
_________________________________________________________________________________________
conv2d_24 (Conv2D)           (None, 256, 256, 64)      36928       re_lu_21[0][0]
_________________________________________________________________________________________
re_lu_22 (ReLU)              (None, 256, 256, 64)      0           conv2d_24[0][0]
_________________________________________________________________________________________
conv2d_25 (Conv2D)           (None, 256, 256, 64)      36928       re_lu_22[0][0]
_________________________________________________________________________________________
re_lu_23 (ReLU)              (None, 256, 256, 64)      0           conv2d_25[0][0]
_________________________________________________________________________________________
conv2d_26 (Conv2D)           (None, 256, 256, 64)      36928       re_lu_23[0][0]
_________________________________________________________________________________________
re_lu_24 (ReLU)              (None, 256, 256, 64)      0           conv2d_26[0][0]
_________________________________________________________________________________________
conv2d_27 (Conv2D)           (None, 256, 256, 64)      36928       re_lu_24[0][0]
_________________________________________________________________________________________
re_lu_25 (ReLU)              (None, 256, 256, 64)      0           conv2d_27[0][0]
_________________________________________________________________________________________
conv2d_28 (Conv2D)           (None, 256, 256, 64)      36928       re_lu_25[0][0]
_________________________________________________________________________________________
re_lu_26 (ReLU)              (None, 256, 256, 64)      0           conv2d_28[0][0]
_________________________________________________________________________________________
conv2d_29 (Conv2D)           (None, 256, 256, 1)       577         re_lu_26[0][0]
_________________________________________________________________________________________
add_2 (Add)                  (None, 256, 256, 3)       0           input_3[0][0]
                                                                   conv2d_29[0][0]
=========================================================================================
Total params: 482,433
Trainable params: 482,433
Non-trainable params: 0
```

Fig. 23

The summary shows the model architecture of 15-layer model. We selected same optimizer
For training of this model, we set the number of epochs to be 10, batch size of 5 with a learning rate of 0.0007
While training each epoch was executed in between 215-223ms/step.

Fig. 24 (below)

```
model_l20 = VDSR20(inp)
model_l20.summary()
model_l20.compile(optimizer=Adam(learning_rate=0.0009,beta_1=0.9), loss='mse', metrics=['accu
```

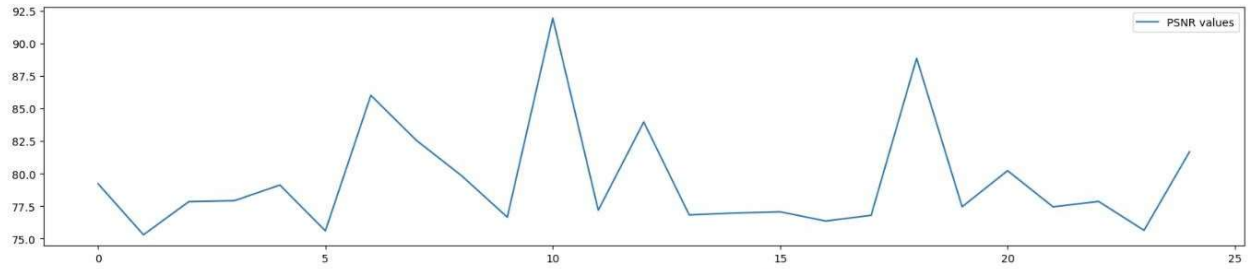
```
Model: "model_3"
__________________________________________________________________________________________________
Layer (type)                    Output Shape         Param #     Connected to
==================================================================================================
input_4 (InputLayer)            [(None, 256, 256, 3) 0
__________________________________________________________________________________________________
conv2d_30 (Conv2D)              (None, 256, 256, 64) 1792        input_4[0][0]
__________________________________________________________________________________________________
re_lu_27 (ReLU)                 (None, 256, 256, 64) 0           conv2d_30[0][0]
__________________________________________________________________________________________________
conv2d_31 (Conv2D)              (None, 256, 256, 64) 36928       re_lu_27[0][0]
__________________________________________________________________________________________________
re_lu_28 (ReLU)                 (None, 256, 256, 64) 0           conv2d_31[0][0]
__________________________________________________________________________________________________
conv2d_32 (Conv2D)              (None, 256, 256, 64) 36928       re_lu_28[0][0]
__________________________________________________________________________________________________
re_lu_29 (ReLU)                 (None, 256, 256, 64) 0           conv2d_32[0][0]
__________________________________________________________________________________________________
conv2d_33 (Conv2D)              (None, 256, 256, 64) 36928       re_lu_29[0][0]
__________________________________________________________________________________________________
re_lu_30 (ReLU)                 (None, 256, 256, 64) 0           conv2d_33[0][0]
__________________________________________________________________________________________________
conv2d_34 (Conv2D)              (None, 256, 256, 64) 36928       re_lu_30[0][0]
__________________________________________________________________________________________________
re_lu_31 (ReLU)                 (None, 256, 256, 64) 0           conv2d_34[0][0]
__________________________________________________________________________________________________
conv2d_35 (Conv2D)              (None, 256, 256, 64) 36928       re_lu_31[0][0]
__________________________________________________________________________________________________
re_lu_32 (ReLU)                 (None, 256, 256, 64) 0           conv2d_35[0][0]
__________________________________________________________________________________________________
conv2d_36 (Conv2D)              (None, 256, 256, 64) 36928       re_lu_32[0][0]
__________________________________________________________________________________________________
re_lu_33 (ReLU)                 (None, 256, 256, 64) 0           conv2d_36[0][0]
__________________________________________________________________________________________________
conv2d_37 (Conv2D)              (None, 256, 256, 64) 36928       re_lu_33[0][0]
__________________________________________________________________________________________________
re_lu_34 (ReLU)                 (None, 256, 256, 64) 0           conv2d_37[0][0]
__________________________________________________________________________________________________
conv2d_38 (Conv2D)              (None, 256, 256, 64) 36928       re_lu_34[0][0]
__________________________________________________________________________________________________
re_lu_35 (ReLU)                 (None, 256, 256, 64) 0           conv2d_38[0][0]
__________________________________________________________________________________________________
conv2d_39 (Conv2D)              (None, 256, 256, 64) 36928       re_lu_35[0][0]
__________________________________________________________________________________________________
re_lu_36 (ReLU)                 (None, 256, 256, 64) 0           conv2d_39[0][0]
__________________________________________________________________________________________________
conv2d_40 (Conv2D)              (None, 256, 256, 64) 36928       re_lu_36[0][0]
__________________________________________________________________________________________________
re_lu_37 (ReLU)                 (None, 256, 256, 64) 0           conv2d_40[0][0]
__________________________________________________________________________________________________
conv2d_41 (Conv2D)              (None, 256, 256, 64) 36928       re_lu_37[0][0]
__________________________________________________________________________________________________
re_lu_38 (ReLU)                 (None, 256, 256, 64) 0           conv2d_41[0][0]
__________________________________________________________________________________________________
conv2d_42 (Conv2D)              (None, 256, 256, 64) 36928       re_lu_38[0][0]
__________________________________________________________________________________________________
re_lu_39 (ReLU)                 (None, 256, 256, 64) 0           conv2d_42[0][0]
__________________________________________________________________________________________________
conv2d_43 (Conv2D)              (None, 256, 256, 64) 36928       re_lu_39[0][0]
__________________________________________________________________________________________________
re_lu_40 (ReLU)                 (None, 256, 256, 64) 0           conv2d_43[0][0]
__________________________________________________________________________________________________
conv2d_44 (Conv2D)              (None, 256, 256, 64) 36928       re_lu_40[0][0]
__________________________________________________________________________________________________
re_lu_41 (ReLU)                 (None, 256, 256, 64) 0           conv2d_44[0][0]
__________________________________________________________________________________________________
conv2d_45 (Conv2D)              (None, 256, 256, 64) 36928       re_lu_41[0][0]
__________________________________________________________________________________________________
re_lu_42 (ReLU)                 (None, 256, 256, 64) 0           conv2d_45[0][0]
__________________________________________________________________________________________________
conv2d_46 (Conv2D)              (None, 256, 256, 64) 36928       re_lu_42[0][0]
__________________________________________________________________________________________________
re_lu_43 (ReLU)                 (None, 256, 256, 64) 0           conv2d_46[0][0]
__________________________________________________________________________________________________
conv2d_47 (Conv2D)              (None, 256, 256, 64) 36928       re_lu_43[0][0]
__________________________________________________________________________________________________
re_lu_44 (ReLU)                 (None, 256, 256, 64) 0           conv2d_47[0][0]
__________________________________________________________________________________________________
conv2d_48 (Conv2D)              (None, 256, 256, 64) 36928       re_lu_44[0][0]
__________________________________________________________________________________________________
re_lu_45 (ReLU)                 (None, 256, 256, 64) 0           conv2d_48[0][0]
__________________________________________________________________________________________________
conv2d_49 (Conv2D)              (None, 256, 256, 1)  577         re_lu_45[0][0]
__________________________________________________________________________________________________
add_3 (Add)                     (None, 256, 256, 3)  0           input_4[0][0]
                                                                 conv2d_49[0][0]
==================================================================================================
Total params: 667,073
Trainable params: 667,073
Non-trainable params: 0
```

The summary shows the model architecture of 20-layer model. We selected same optimizer For training of this model, we set the number of epochs to be 5, batch size of 1 with a learning rate of 0.0009 While training each epoch was executed in between 67-68ms/step. Training of this model was a bit complex as we were unable to train it with smaller batch size and set the batch size to 1 and reduced the number of epochs from 10 to 5.

Training of this model requires good hardware such as a powerful core and a descent gpu to train it faster. My laptop has a CPU: 11th Gen Intel(R) Core(TM) i7-11800H @ 2.30GHz, 2304 MHz, 8 Core(s), 16 Logical Processor(s) and a GPU: NVIDIA GeForce RTX 3060 Laptop GPU

Here are the results we obtained from test sets of Image Super Resolution.

Layer 5 Model:




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
Maximum psnr value: 91.93375984978093
Minimum PSNR value: 75.30735197984922
Average PSNR value: 79.45693001774035
```


```
PSNR value 79.23257195028756
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

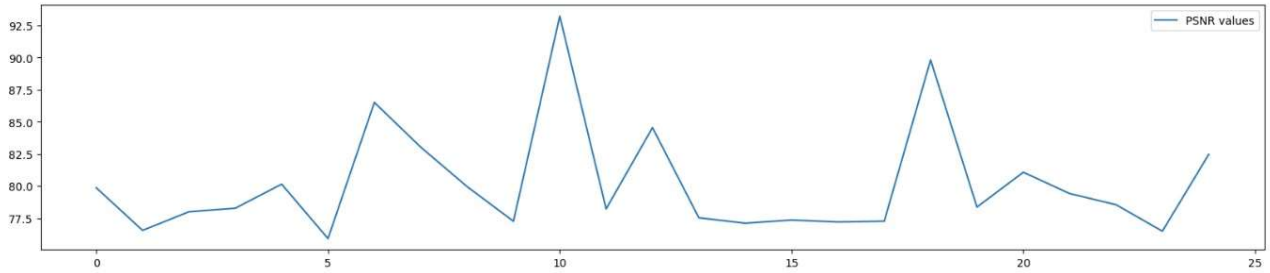

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
PSNR value 75.30735197984922
```

Fig. 25

Layer 10 Model:

```
3 plot_psnr(ml_10_psnr)
4 plot_output(test_high_image,test_low_image,ml_10_pred,ml_10_psnr)
```




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
Maximum psnr value: 93.2173304563211
Minimum PSNR value: 75.92321330141901
Average PSNR value: 80.17222361825993
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

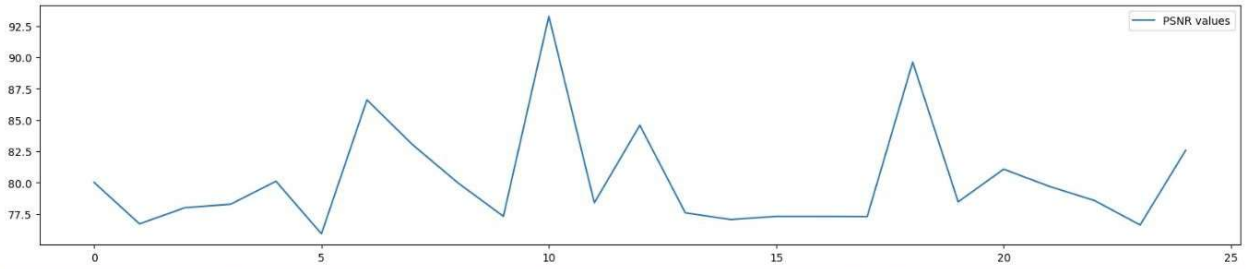
```
PSNR value 79.87398332281539
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
PSNR value 76.5602786237957
```

Fig. 26

Layer 15 Model:




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
Maximum psnr value: 93.25661942277631
Minimum PSNR value: 75.92811228773063
Average PSNR value: 80.22239996358779
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
PSNR value 80.02784081695664
```

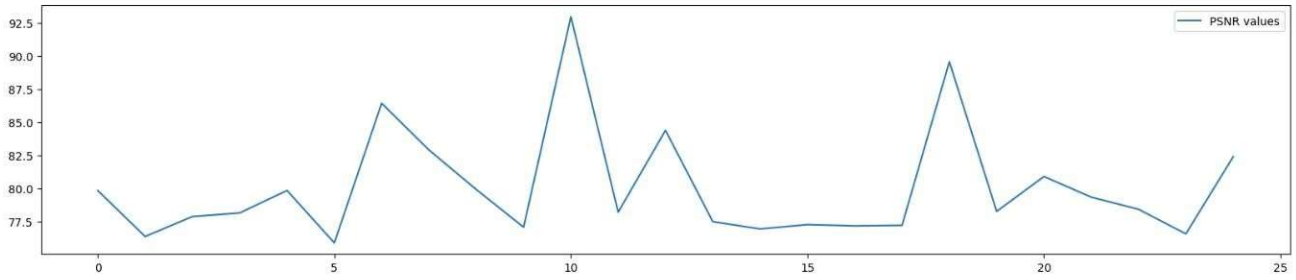

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
PSNR value 76.72142622208558
```

Fig. 27

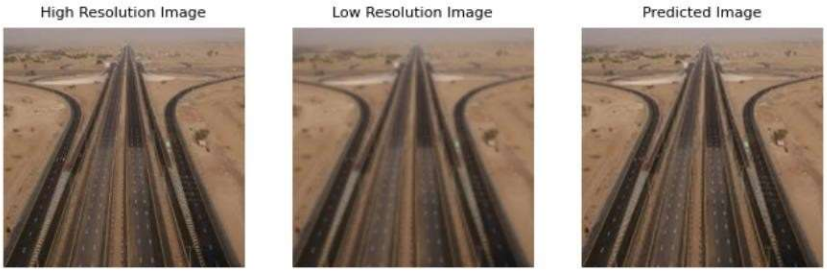
Layer 20 Model

```
4 plot_output(test_high_image,test_low_image,ml_20_pred,ml_20_psnr)
```




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
Maximum psnr value: 92.94281927548559
Minimum PSNR value: 75.89938128282526
Average PSNR value: 80.0588363106026
```




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
PSNR value 79.84382240689713
```

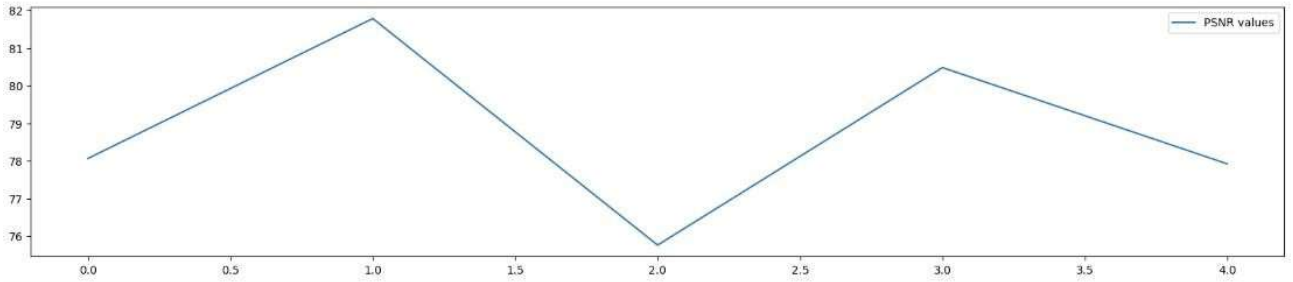

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

Fig. 28

And here the results we obtained when made predictions on the Set5 dataset using the above pretrained models:

Layer 5 Model:

```
plot_psnr(psnr_l5)
plot_output_final(set5_out,set5_inp,pred_l5,psnr_l5)
```




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).

Maximum psnr value: 81.77888095616224
Minimum PSNR value: 75.76313835219287
Average PSNR value: 78.8006443672707
```

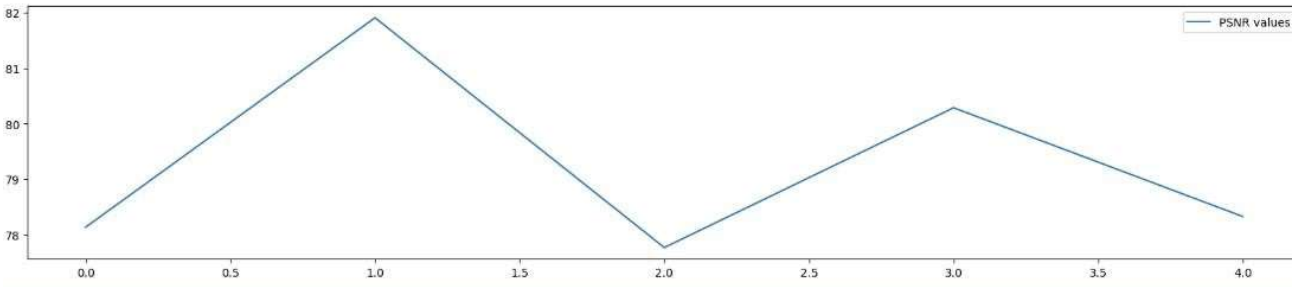

```
PSNR value 78.0640618743689

Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```


```
PSNR value 81.77888095616224
```

Fig. 29

Layer 10 Model:

```
plot_output_final(set5_out,set5_inp,pred_l10,psnr_l10)
```




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Maximum psnr value: 81.9099261310472
Minimum PSNR value: 77.77052907292087
Average PSNR value: 79.28723417905445
```

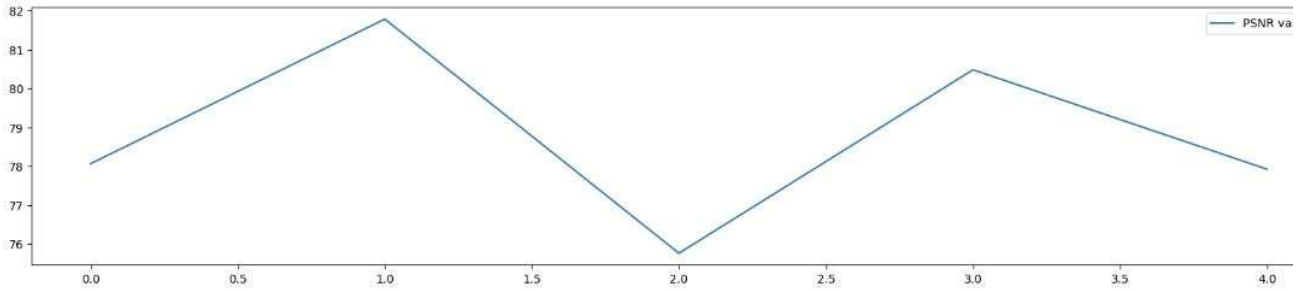

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
PSNR value 78.13814463510342
```


```
PSNR value 81.9099261310472
```

Fig. 30

Layer 15 Model:

```
4 plot_output_final(set5_out,set5_inp,pred_l15,psnr_l15)
```




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
Maximum psnr value: 81.77888095616224
Minimum PSNR value: 75.76313835219287
Average PSNR value: 78.8006443672707
```


```
PSNR value 78.0640618743689
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

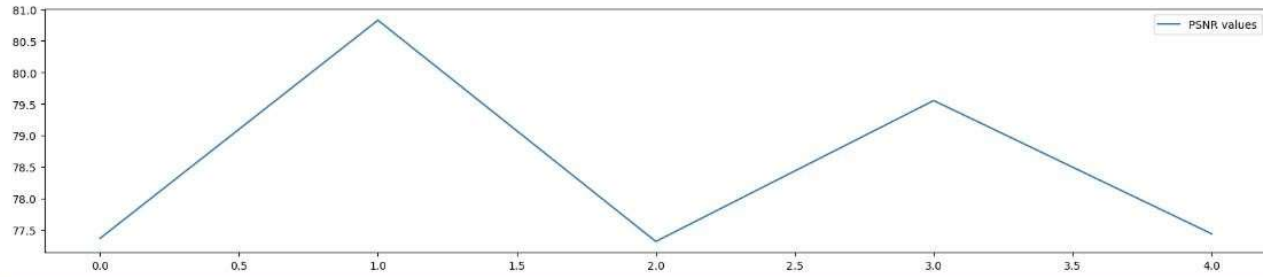

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
PSNR value 81.77888095616224
```

Fig. 31

Layer 20 Model:

```
4 plot_output_final(set5_out,set5_inp,pred_l20,psnr_l20)
```




```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
Maximum psnr value: 80.82844640657788
Minimum PSNR value: 77.31559282103818
Average PSNR value: 78.4999435413671
```


```
PSNR value 77.36513280655858
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
PSNR value 80.82844640657788
```

Fig. 32

## Conclusions we formed after observing these results:

1. Smaller the size of image, less computation time the network needs
2. The smaller the size of high-resolution image, lower PSNR scores will be achieved. This could be the fact that small sized images already have less data when compared to one's images which have larger dimension. The model could extract more features from images with larger dimension rather than the ones with lower sizes.
3. After a certain depth, the predicted image will reach its maximum attainable PSNR score.
4. An image of size 32x32x3 or 100x100x3 would not be needing all the 20 layers to create a higher resolution image. By our observing the model's performance we can say, a network of 5 or 10 layers deep is sufficient for small size images.
5. Images which are larger in size would benefit from having deeper networks.
6. Training deep networks of depth = 20 or more will be easier to train for smaller images but would require a lot of computation resources for a larger sized image.
7. After looking at results of our models for Set5, when trained on 256x256 sized images, we can notice the following:

Results of 20 Layer network


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```

```
PSNR value 80.82844640657788
```

Results from 5,10,15 Layers network respectively:


```
PSNR value 81.77888095616224
```


```
PSNR value 81.9099261310472
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
PSNR value 81.77888095616224
```

Fig. 34 a, 34b, 34c, 34d

We can see the predicted image from 20-layer network, it has sharper edges when compared to the other ones. As we look at bottom edge (near the stomach) region of the bird, we can clearly notice the difference in sharpness produced by the different models. From this we can conclude that lower depth networks provide with a decent PSNR score for small to moderately sized images but to predict finer details for large images, deep networks is a must.

As we know the computation of deep networks is slow, to counter this we could create a network with moderate depth and run the predictions twice through the network. This might work better than shallow networks and compute faster than deeper networks. This is just a speculation.

# 5. References:

## Lecture notes

Fig 1 : https://www.google.com/url?sa=i&url=https%3A%2F%2Ftowardsdatascience.com%2Freview-vdsr-super-resolution-f8050d49362f&psig=AOvVaw3WZyImRVR5dTwK9mKX8Rs9&ust=1668637291890000&source=images&cd=vfe&ved=0CBAQjhxqFwoTCNjvobqcsfsCFQAAAAAdAAAAABAM

Fig 2: https://www.google.com/url?sa=i&url=https%3A%2F%2Fwww.androidguys.com%2Ftips-tools%2Fwhat-is-ppi-and-why-is-it-important%2F&psig=AOvVaw1T6hy_875APJ5r59VZznrx&ust=1668651766612000&source=images&cd=vfe&ved=0CBAQjhxqFwoTCMDiv7fSsfsCFQAAAAAdAAAAABAN

Fig 3: https://www.cambridgeincolour.com/tutorials/image-interpolation.htm

Fig 4 – 34: Outputs from the code

Understanding VDSR: https://www.mathworks.com/help/deeplearning/ug/single-image-super-resolution-using-deep-learning.html

CV2 : https://chadrick-kwag.net/cv2-resize-interpolation-methods/

Reference code to understand the working: https://franciscofarinha.ca/post/vdsr_paper/ , https://www.kaggle.com/code/harshraone/super-resolution-using-multi-scale-learning

Dataset (Image Super Resolution) : https://www.kaggle.com/datasets/adityachandrasekhar/image-super-resolution

Dataset (Set5) : https://www.kaggle.com/datasets/ll01dm/set-5-14-super-resolution-dataset

Other references:

https://stackoverflow.com/questions/50760543/error-oom-when-allocating-tensor-with-shape

https://cloudinary.com/guides/bulk-image-resize/python-image-resize-with-pillow-and-opencv

https://www.tutorialkart.com/opencv/python/install-opencv-python-with-anaconda/

https://www.programcreek.com/python/example/122026/keras.backend.resize_images