VERY DEEP CONVOLUTION NETWORK FOR SUPER RESOLUTION

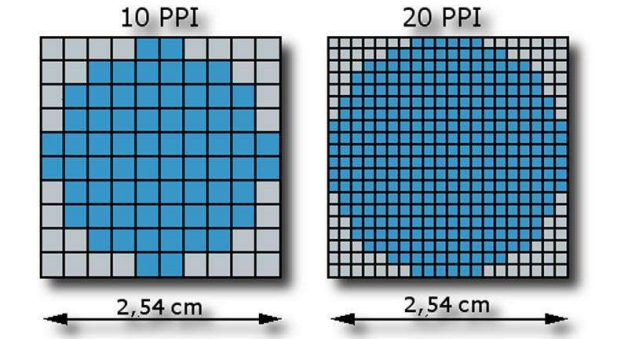
Deep-Learning Project CS 577

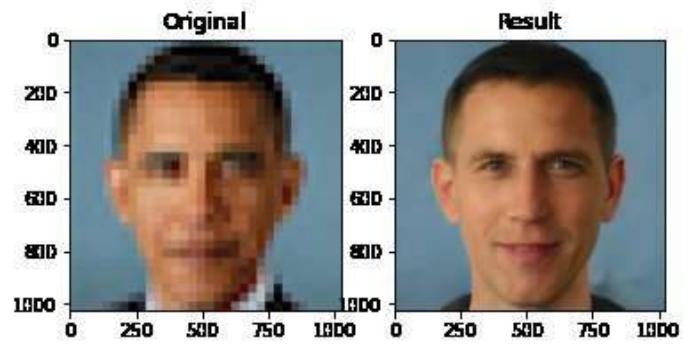
WHAT IS SUPER RESOLUTION

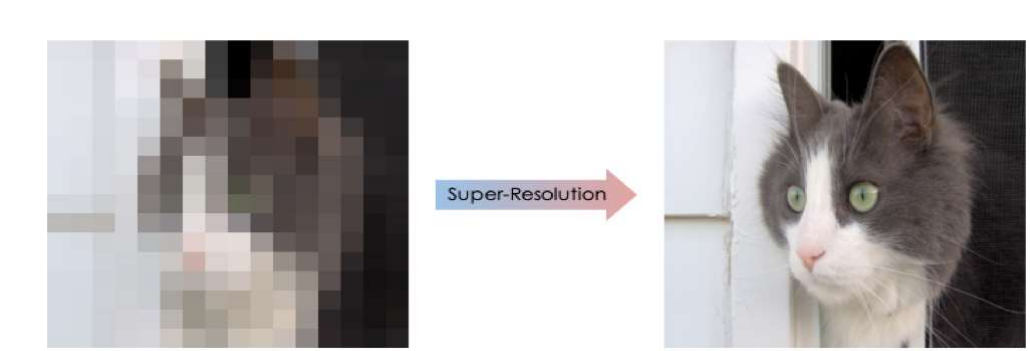
- Before getting into the nitty gritty of super resolutions, let us understand some concepts regarding images.

- What is an image?

- What is resolution?


10 PPI
20 PPI
2,54 cm
2,54 cm


Original
Result
0
200
400
600
800
1000
0
250
500
750
1000


Super-Resolution

INTRODUCTION

Image Super-Resolution: Low Resolution image to High Resolution image

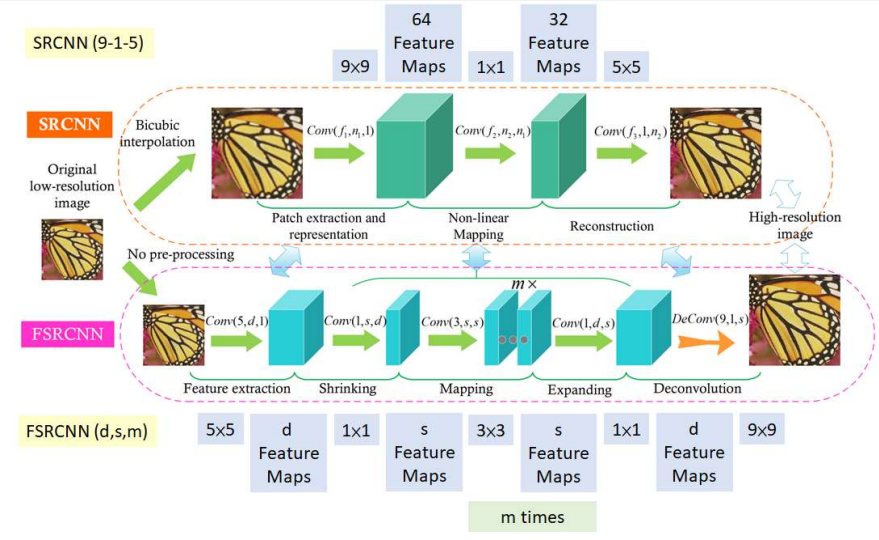
Usefulness: medical field, security, Biological feild.

Existing deep learning technique and its problem: SRCNN -small image regions, slow, single scale

LR image

HR image

PREVIOUS MODELS



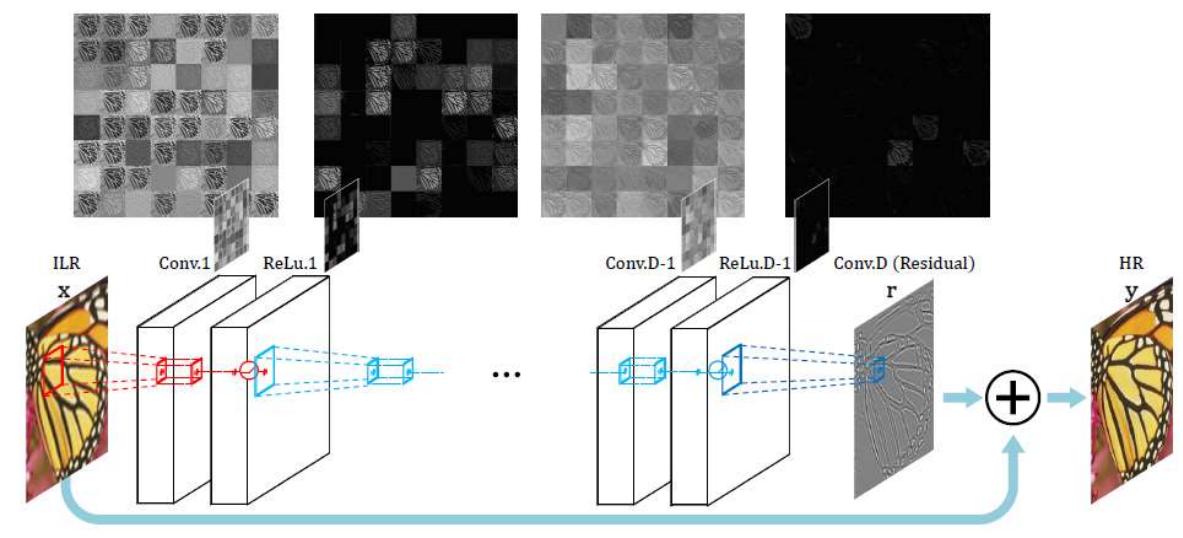
PROBLEM STATEMENT

How to implement a better Image Super-Resolution network that works better than SRCNN?

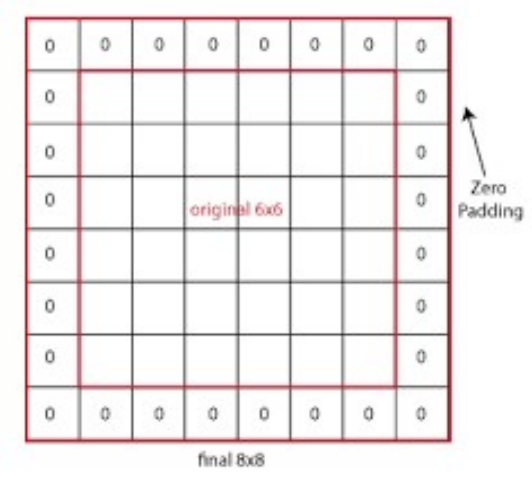
SRCNN	Our network
Small Images	Big Images
Slow	Fast
Uni-Scale	Multi-Scale

VDSR NETWORK

- Use a very deep convolutional network inspired by K.Simonyan and A.Zisserman
- Deep Convolution-layers network



- Issue with using a very deep network to predict dense outputs:
  - Size of the feature map gets reduced every time convolution operations are applied
- Proposed solution:
  - Use Zero Padding before convolutions



IMPLEMENTATION

```
train_label = train_x #Storing the original resoltuion img as train label
test_label = test_x #Storing the original resoltuion img as test label

print("Training set ground truth has shape: " + str(train_label.shape))
print("Test set ground truth has shape: " + str(test_label.shape))

#setting output width x height
w,h = train_label.shape[1:-1]
input_dim = train_label.shape[1:]
scale_factor = scale_f
#size_output_img = (w,h)
print('Width and Height of training and test data should be:',input_dim)
dw= int(w/scale_factor)
dh=int(h/scale_factor)
#print('Scale factor of for width and height:',dw,dh)
ds =(dw,dh)

#Creating Train_X and Test_X from the ground truth
#this is done so by reducing the size of copy of ground truth(train_label, test_label)
#Here the scale is 2, the training shape = 16x16x3 i.e half of 32x32x3
train_data = []
for img in train_label:
    img_re = cv2.resize(img, ds, interpolation=cv2.INTER_CUBIC)
    train_data.append(img_re)
train_data = np.asarray(train_data)

test_data = []
for img in test_label:
    img_re = cv2.resize(img,ds, interpolation=cv2.INTER_CUBIC)
    test_data.append(img_re)
 #train_data = np.array(train_data,dtype=float)
 b1 = train_data[:10000]
 b2 = train_data[10000:20000]
 b3 = train_data[20000:30000]
 b4 = train_data[30000:40000]
 b5 = train_data[40000:50000]
 b1 = tf.convert_to_tensor(b1,dtype=float)
 b1 = resize_images(b1,scale_factor,scale_factor,'channels_last',interpolation='bilinear')
 b2 = tf.convert_to_tensor(b2,dtype=float)
 b2 = resize_images(b2,scale_factor,scale_factor,'channels_last',interpolation='bilinear')
 b3 = tf.convert_to_tensor(b3,dtype=float)
 b3 = resize_images(b3,scale_factor,scale_factor,'channels_last',interpolation='bilinear')
 b4 = tf.convert_to_tensor(b4,dtype=float)
 b4 = resize_images(b4,scale_factor,scale_factor,'channels_last',interpolation='bilinear')
 b5 = tf.convert_to_tensor(b5,dtype=float)
 b5 = resize_images(b5,scale_factor,scale_factor,'channels_last',interpolation='bilinear')
 x_scaled_train= tf.concat([b1,b2,b3,b4,b5],0)
 test_data = tf.convert_to_tensor(test_data,dtype=float)
```

X- input image[Low Resolution]: For the model

Y - Label output[High Resolution :]

```
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)
```

```python
def vdsr(input_dim, l):
    LR = Input(shape=input_dim, name='input')
    X = ZeroPadding2D()(LR)
    X = Conv2D(64,(3,3), name='CONV1')(X)
    X = ReLU()(X)

    for i in range(l-2):
        X = ZeroPadding2D()(X)
        X = Conv2D(64, (3,3), name='CONV%i' % (i+2))(X)
        X = ReLU()(X)

    X = ZeroPadding2D()(X)
    residual = Conv2D(1, (3,3), name='CONV%i' % l)(X)

    out = Add()([LR, residual])

    return Model(LR, out)

```

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: 113,153
Trainable params: 113,153
Non-trainable params: 0

10, 15 and 20 Layer Deep Models

```
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

```
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

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

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]

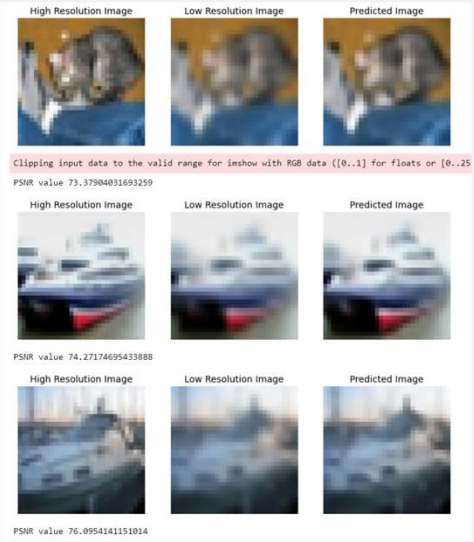
Total params: 667,073
Trainable params: 667,073
Non-trainable params: 0

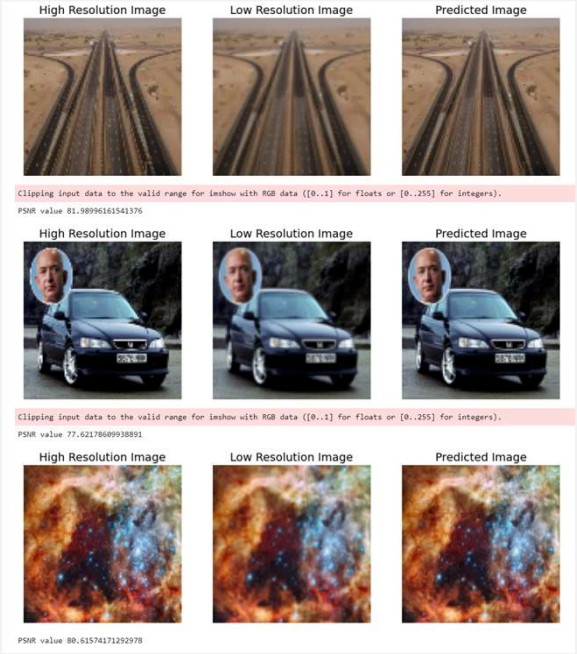
LOSS

- PSNR : Peak Signal to Noise Ratio

$$
\begin{aligned}
PSNR &= 10 \cdot \log_{10} \left(\frac{MAX_I^2}{MSE} \right) \\
&= 20 \cdot \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right) \\
&= 20 \cdot \log_{10}(MAX_I) - 10 \cdot \log_{10}(MSE).
\end{aligned}
$$

RESULTS AND DISCUSSION





GUESS THE MODEL?

```
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..
PSNR value 82.08008428346795
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..
PSNR value 78.02908746185197
```


```
PSNR value 80.73586079821227
```

```
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 [
PSNR value 79.84382240689713
```


```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [
PSNR value 76.3794976067739
```


```
PSNR value 77.88437684189171
```

```
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..
```

```
PSNR value 82.08008428346795
```


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

```
PSNR value 78.02908746185197
```


```
PSNR value 80.73586079821227
```

20 layer Model trained on 100x100x3 images

```
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 [
```

```
PSNR value 79.84382240689713
```


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

```
PSNR value 76.3794976067739
```


```
PSNR value 77.88437684189171
```

20 layer Model trained on 256x256x3 images

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
```


```
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
```

CONCLUSION

1. After a certain depth, the predicted image will reach its maximum attainable PSNR score.
2. 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.
3. Images which are larger in size would benefit from having deeper networks.
4. 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.
5. 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.

REFERENCES

- https://arxiv.org/pdf/1511.04587.pdf?fbclid=IwAR0gm942jqx5Beq9j3maZ90xeEw4qnWNzToiBUKHFCS7OYgTG__-hI_rx24
- http://mmlab.ie.cuhk.edu.hk/projects/SRCNN.html
- https://en.wikipedia.org/wiki/Super-resolution_imaging
- https://www.google.com/url?sa=i&url=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FPixel_aspect_ratio&psig=AOvVaw0ohHSfNwsq7YBYPaQEwhaW&ust=1668734123757000&source=images&cd=vfe&ved=0CA8QjRxqFwoTCOiHwomGtPsCFQAAAAAdAAAAABAJ
- https://cdn.mos.cms.futurecdn.net/PzPq6Pbn5RqgrWunhEx6rg.jpg
- https://miro.medium.com/max/1090/1*auit3UsBTjbnzZoLDEJUUg.png
- https://www.scientiamobile.com/wp-content/uploads/2018/12/Pixel-Density.jpg
- https://miro.medium.com/max/1060/1*rmQjKd4uETJtwYMRTDzmrg.png
- https://www.google.com/url?sa=i&url=https%3A%2F%2Fzentralwerkstatt.org%2Fblog%2Fpulse&psig=AOvVaw09cargbTBiNkZgNnrNz0DO&ust=1668735205473000&source=images&cd=vfe&ved=0CA8QjRxqFwoTCJD445qJtPsCFQAAAAAdAAAAABAK
- https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcRePQoj9vCAU-MxGdkQUWxAIVASBjLWZsFuFewuKnbSnwwEaKpuRV5MM7TtwlpG1PIRnWo&usqp=CAU

THANK YOU