q5

March 1, 2024

[1464]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
```

[1465]:
```
# Image completion
import os
if os.path.exists('dog.jpg'):
    print('dog.jpg exists')
else:
    !wget https://segment-anything.com/assets/gallery/
 ↪AdobeStock_94274587_welsh_corgi_pembroke_CD.jpg -O dog.jpg
```

```
dog.jpg exists
```

[1466]:
```
# Read in a image from torchvision
import torchvision
img = torchvision.io.read_image("dog.jpg")
print(img.shape)
```

```
torch.Size([3, 1365, 2048])
```

[1467]:
```
img.shape
```

[1467]:
```
torch.Size([3, 1365, 2048])
```

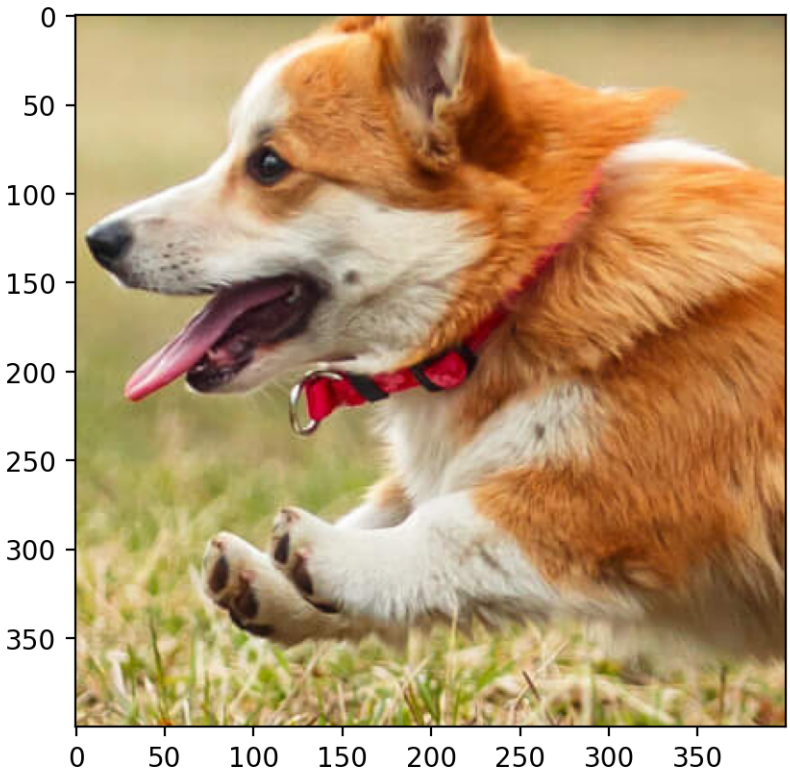
[1468]:
```
# image tensors are usually of shape (channels, height, width) --> (0,1,2)
# matplotlib expects (height, width, channels)
plt.imshow(img.permute(1,2,0).numpy().astype('uint8'))
```

[1468]:
```
<matplotlib.image.AxesImage at 0x23acff85b90>
```

[1469]:
```
# Extract patches surrounding the missing patch
def extract_patches(img,top_left,hight,width):
    x,y = top_left
    patches = torchvision.transforms.functional.crop(img,y,x,hight,width)
    return patches
```

[1470]:
```
dog = extract_patches(img,(800,600),400,400)
plt.imshow(dog.permute(1,2,0).numpy().astype('uint8'))
```

[1470]:
```
<matplotlib.image.AxesImage at 0x23acff46450>
```



[1471]:
```
def img_missing_patches(img,top_left,hight,width):
    y,x = top_left
    patches = extract_patches(img,top_left,hight,width)
    patches_copy = patches.clone().float()
    mask = torch.rand(patches.shape) <= 1
    patches_copy[mask] = float('nan')
    img_copy = img.clone().float()
    img_copy[:,x:x+width,y:y+hight] = patches_copy
    return img_copy,patches,patches_copy
```

[1482]:
```
# Define a function to factorize the matrix
def factorize(A, k, device=torch.device("cpu")):
    """Factorize the matrix A into two matrices W and H."""
    A = A.to(device)
    # Randomly initialize matrices W and H
    W = torch.randn(A.shape[0],A.shape[1], k, requires_grad=True, device=device)
    H = torch.randn(A.shape[0],k, A.shape[2], requires_grad=True, device=device)
    # Adam optimizer
    optimizer = optim.Adam([W, H], lr=0.01)
    mask = ~torch.isnan(A)
    # Train the model
    diff_matrix = torch.matmul(W, H) - A
```

```
    diff_vector = diff_matrix[mask]
    loss = torch.norm(diff_vector)
    for _ in range(1000):

        # Compute the loss
        diff_matrix = torch.matmul(W, H) - A
        diff_vector = diff_matrix[mask]
        loss = torch.norm(diff_vector)

        # Zero the gradients
        optimizer.zero_grad()

        # Backpropagate
        loss.backward()

        # Update the parameters
        optimizer.step()

    return W, H, loss
```

[1473]:
```
# Function to complete the missing patch
def complete_missing_patch(img, top_left, height, width, k):
    """Complete the missing rectangular patch in the image."""
    # Extract the patches surrounding the missing patch
    x, y = top_left
    img,patch,patch_copy = img_missing_patches(img,top_left,height,width)
    # Factorize the image matrix
    W, H, loss = factorize(img, k)
    # Reconstruct the missing patch using factorized matrices
    re_img = torch.matmul(W, H)


    return img,re_img,patch,loss
```

[1474]:
```
import os
import matplotlib.pyplot as plt

def save_figure(patche,img, completed_img, name, folder="figures"):
    """Save the original and completed images as figures."""
    # Create the directory to store figures if it doesn't exist
    if not os.path.exists(folder):
        os.makedirs(folder)

    # Plot both the original image and the completed image simultaneously
    plt.figure(figsize=(18, 6))  # Adjust the size as needed
    plt.subplot(1, 3, 1)
    plt.imshow(patche.permute(1, 2, 0).numpy().astype('uint8'))
```

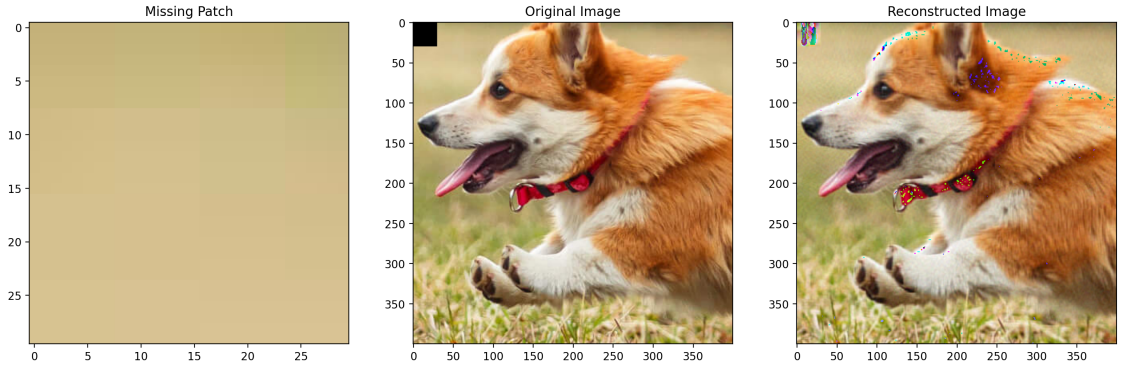
```
    plt.title('Missing Patch')

    plt.subplot(1, 3, 2)
    plt.imshow(img.permute(1, 2, 0).numpy().astype('uint8'))
    plt.title('Original Image')

    plt.subplot(1, 3, 3)
    plt.imshow(completed_img.detach().permute(1, 2, 0).numpy().astype('uint8'))
    plt.title('Reconstructed Image')

    # Save the figure with the specified name in the folder
    plt.savefig(os.path.join(folder, f"{name}.png"))
    plt.show()
    plt.close()

# Example usage
# Assuming 'img' and 'completed_img' are the original and completed images,␣
 ↪respectively
# and 'dog' is your image tensor
# img_const_LS(dog, (0,0), 30, 30, 70, 'Single_color_patch_missing')
```

[1475]:
```
def img_const_LS(img,top_left,hight,width,k,name):

    img, re_img, patch, loss = complete_missing_patch(img, top_left, hight,␣
 ↪width, k)
    save_figure(patch,img, re_img, name)
    return
```

1. an area with mainly a single color.

[1476]:
```
img_const_LS(dog, (0,0), 30, 30,70,'Single color patch missing')
```

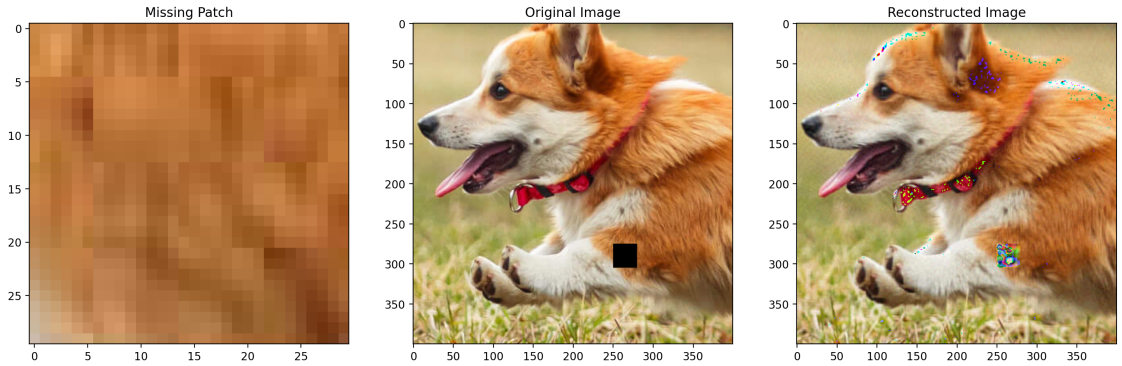
```
C:\Users\patel\AppData\Local\Temp\ipykernel_26008\946811148.py:17:
RuntimeWarning: invalid value encountered in cast
  plt.imshow(img.permute(1, 2, 0).numpy().astype('uint8'))
```



2. an area with 2-3 different colors.

[1477]:
```
# Example usage
img_const_LS(dog, (250,275), 30, 30,70,'2-3 color patch missing')
```

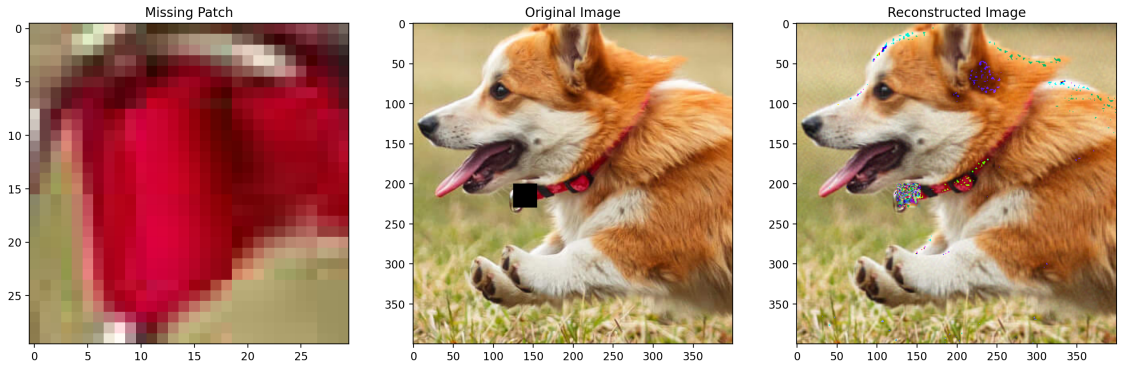
```
C:\Users\patel\AppData\Local\Temp\ipykernel_26008\946811148.py:17:
RuntimeWarning: invalid value encountered in cast
  plt.imshow(img.permute(1, 2, 0).numpy().astype('uint8'))
```



3. an area with at least 5 different colors.

[1478]:
```
# Example usage
img_const_LS(dog, (125,200), 30, 30,70,'many color patch missing')
```

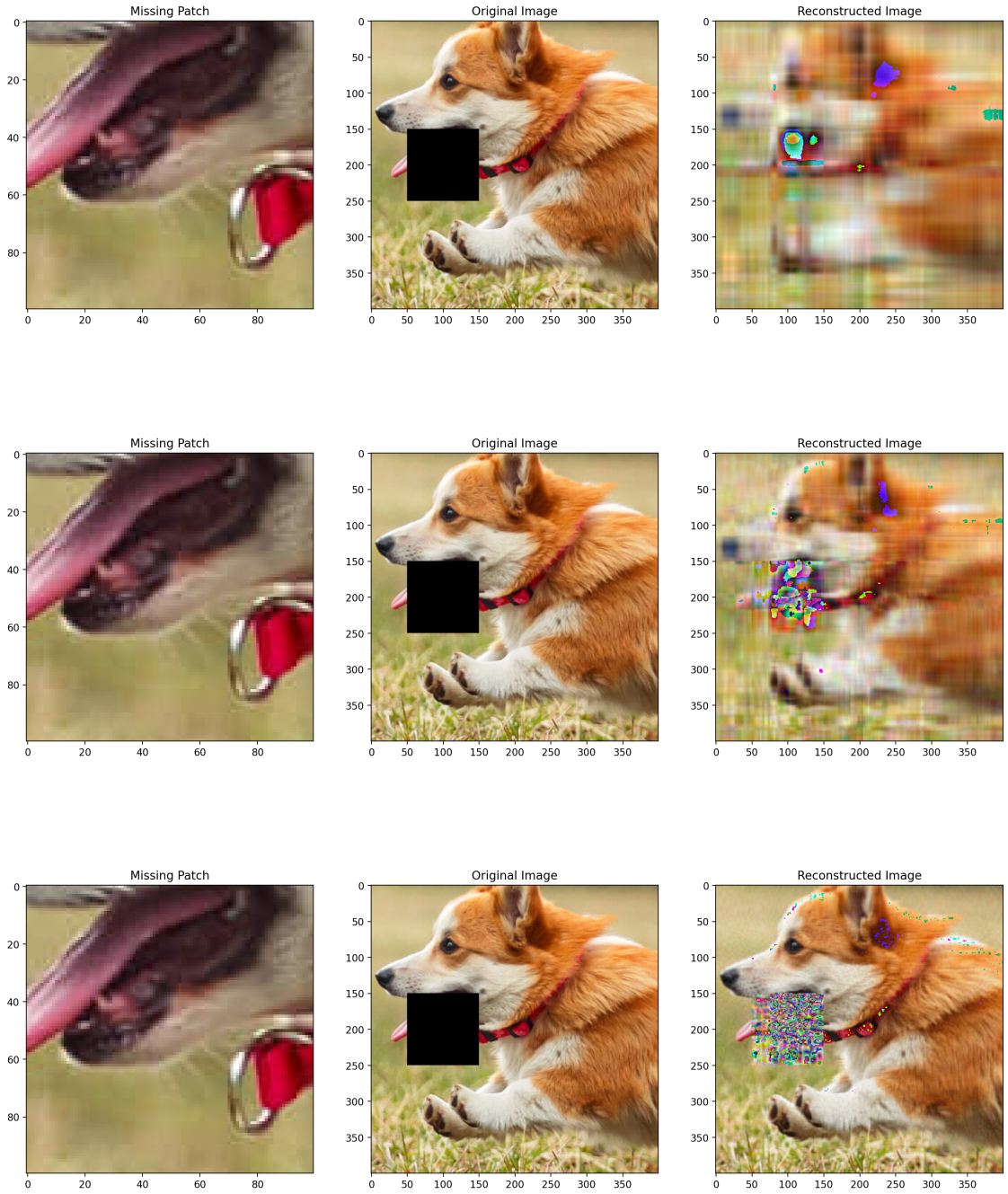
```
C:\Users\patel\AppData\Local\Temp\ipykernel_26008\946811148.py:17:
RuntimeWarning: invalid value encountered in cast
  plt.imshow(img.permute(1, 2, 0).numpy().astype('uint8'))
```

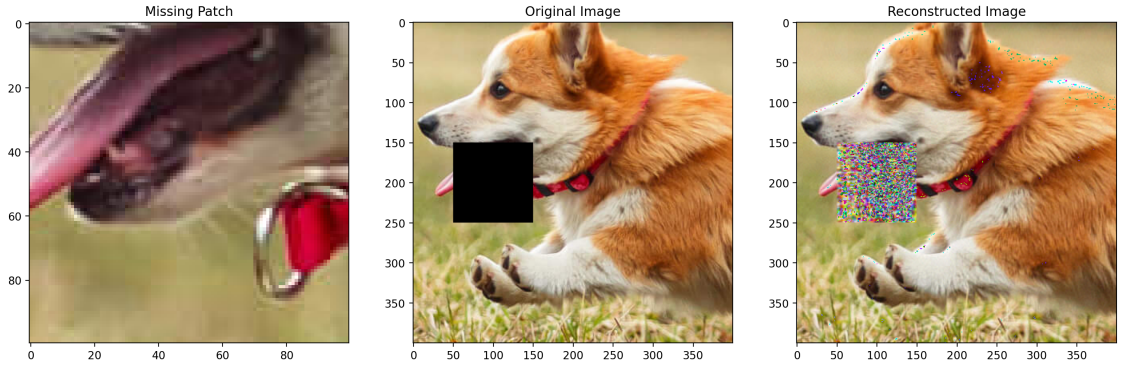


[1479]:
```
r = [5, 10, 50, 100]
```

```
# (50,150),100,100


for i in r:
    img_const_LS(dog, (50,150), 100, 100,i,f'Varying_rank_{i}')
```

```
C:\Users\patel\AppData\Local\Temp\ipykernel_26008\946811148.py:17:
RuntimeWarning: invalid value encountered in cast
  plt.imshow(img.permute(1, 2, 0).numpy().astype('uint8'))
```

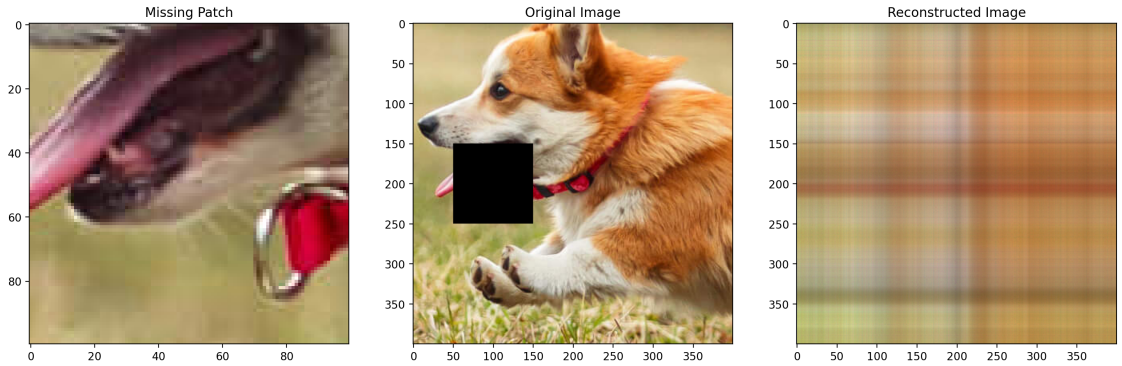


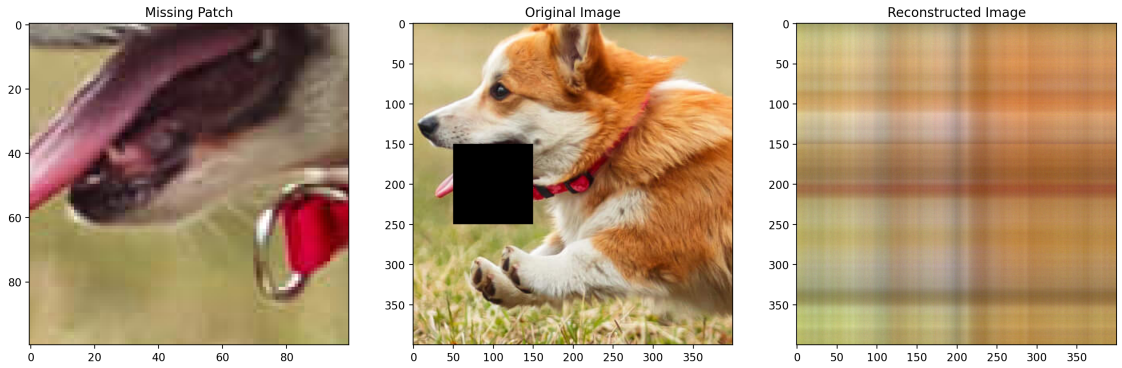


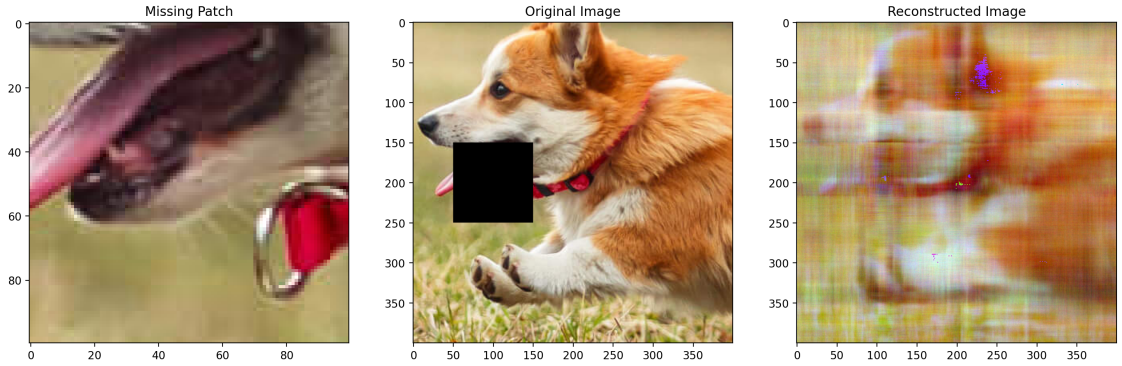
[ ]:

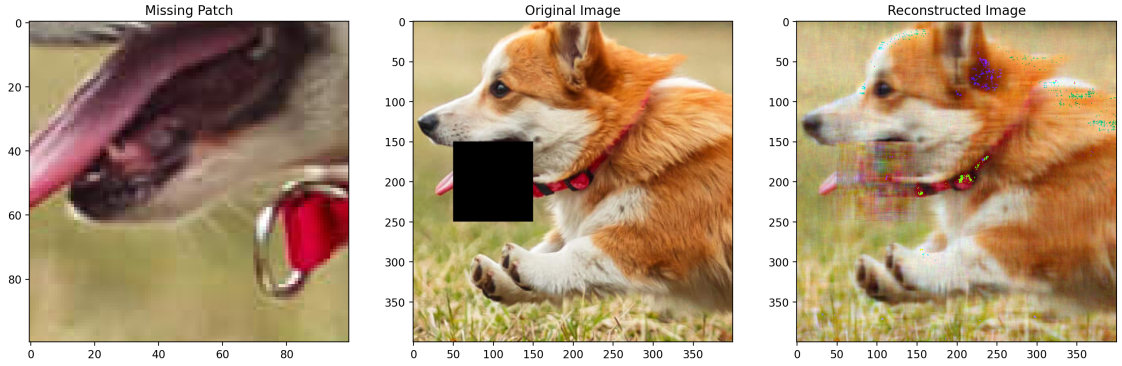
[1481]:
```
r = [5, 10, 50, 100]

# (50,150),100,100


for i in r:
    img_const_LS(dog, (50,150), 100, 100,i,f'Varying_rank_e_{i}')
```

```
C:\Users\patel\AppData\Local\Temp\ipykernel_26008\946811148.py:17:
RuntimeWarning: invalid value encountered in cast
  plt.imshow(img.permute(1, 2, 0).numpy().astype('uint8'))
```



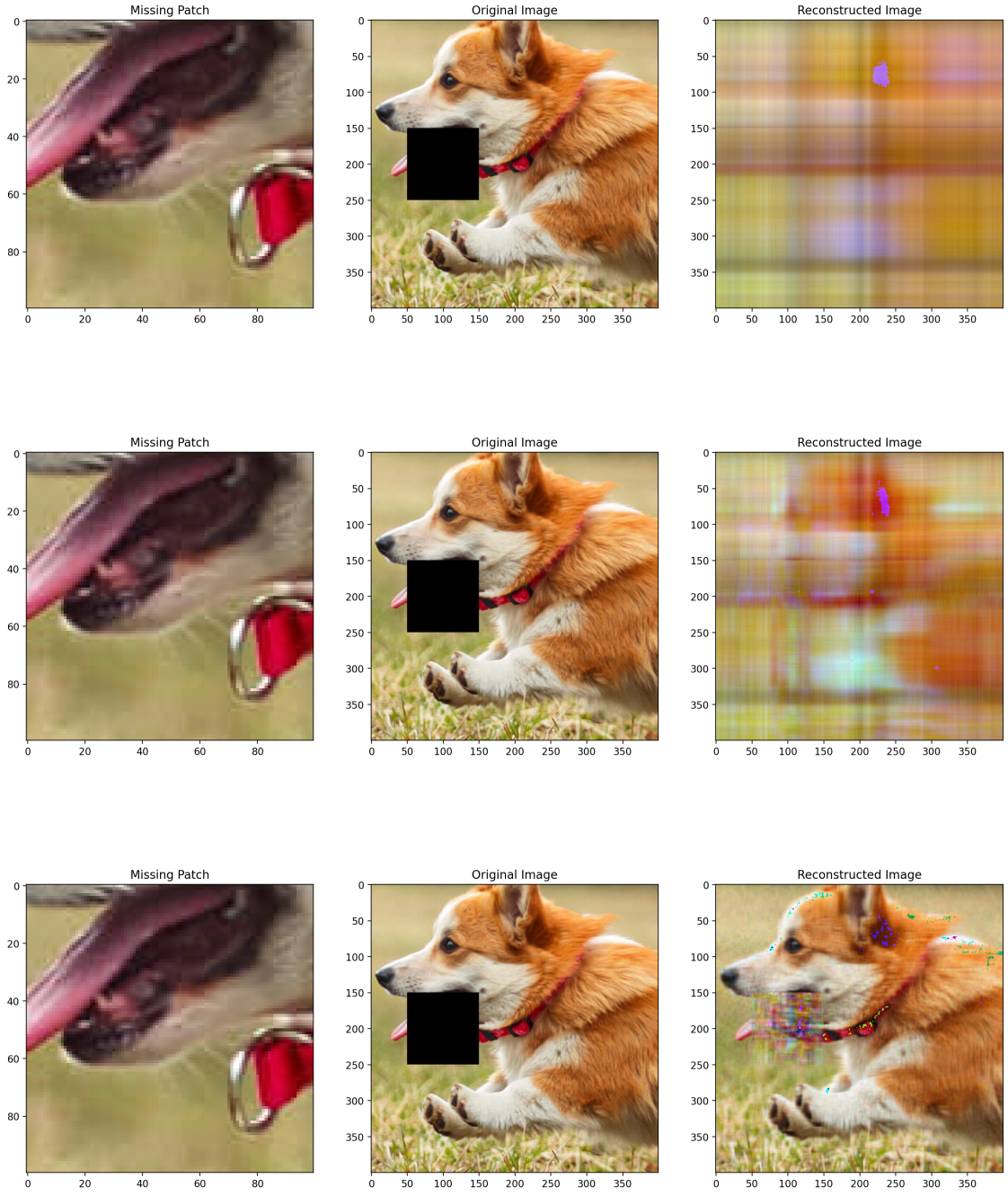


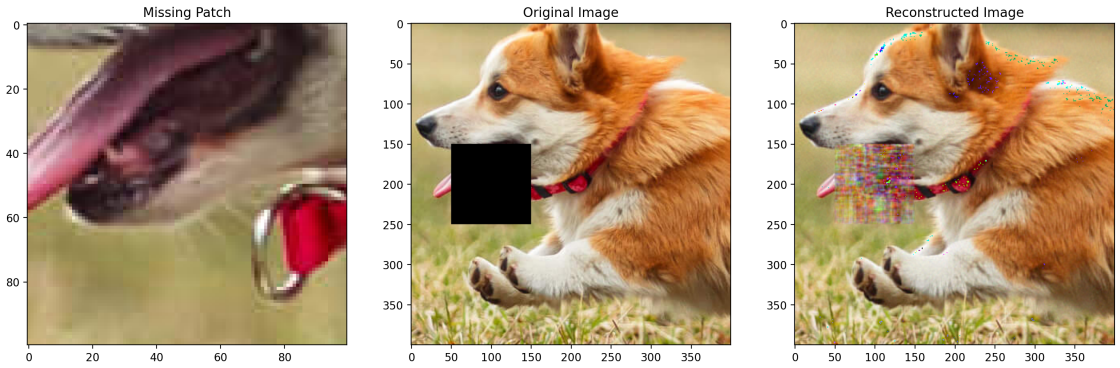




[1483]:
```
r = [5, 10, 50, 100]

# (50,150),100,100


for i in r:
    img_const_LS(dog, (50,150), 100, 100,i,f'Varying_rank_f_{i}')
```

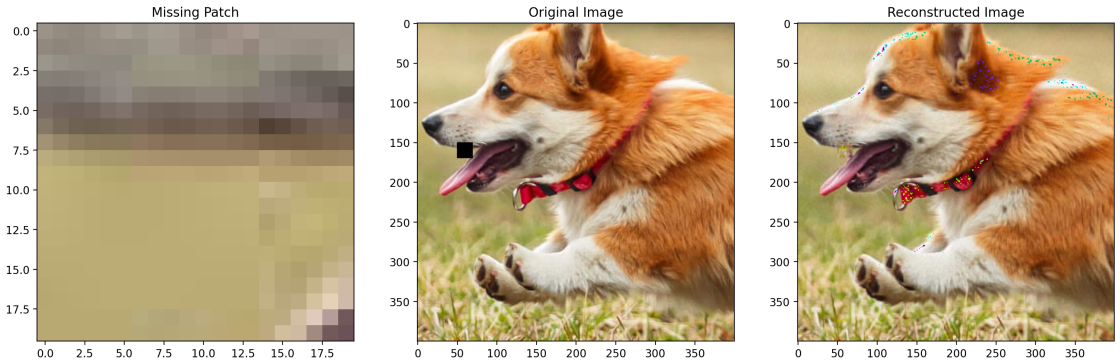
```
C:\Users\patel\AppData\Local\Temp\ipykernel_26008\946811148.py:17:
RuntimeWarning: invalid value encountered in cast
  plt.imshow(img.permute(1, 2, 0).numpy().astype('uint8'))
```

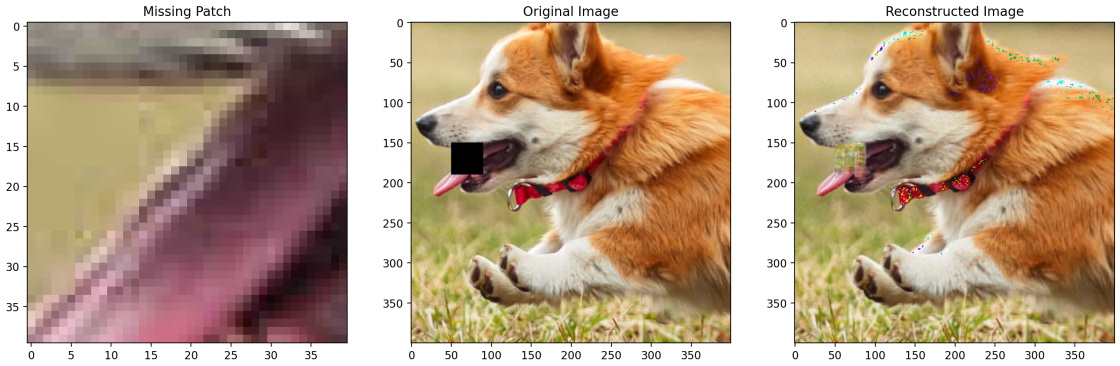


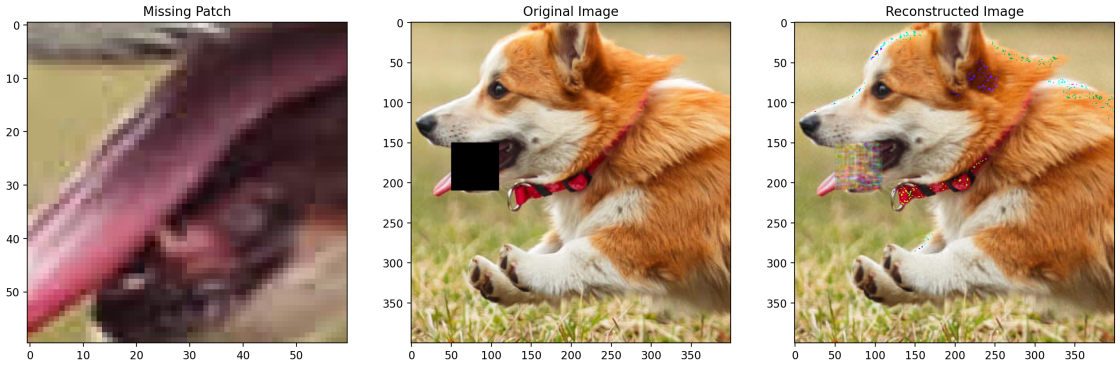


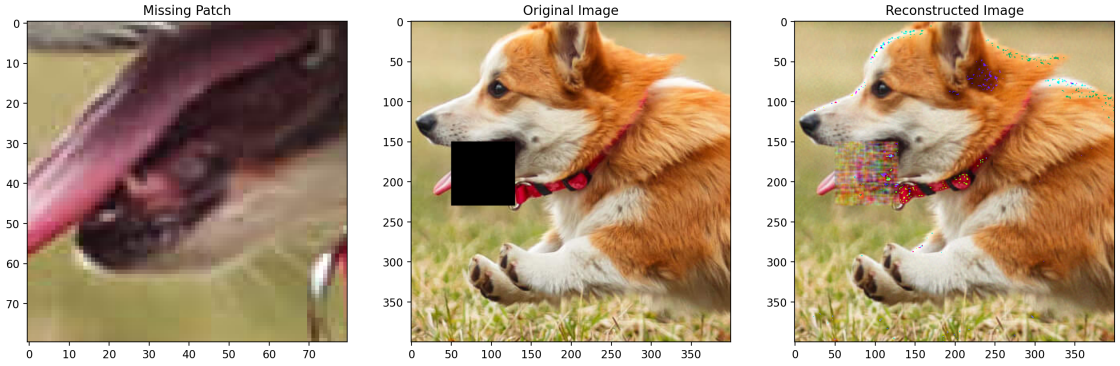
[1484]:
```
N = [20,40,60,80,100]
r = 100

for i in N:
    img_const_LS(dog, (50,150), i, i,r,f'Varying_patch_size_{i}')
```

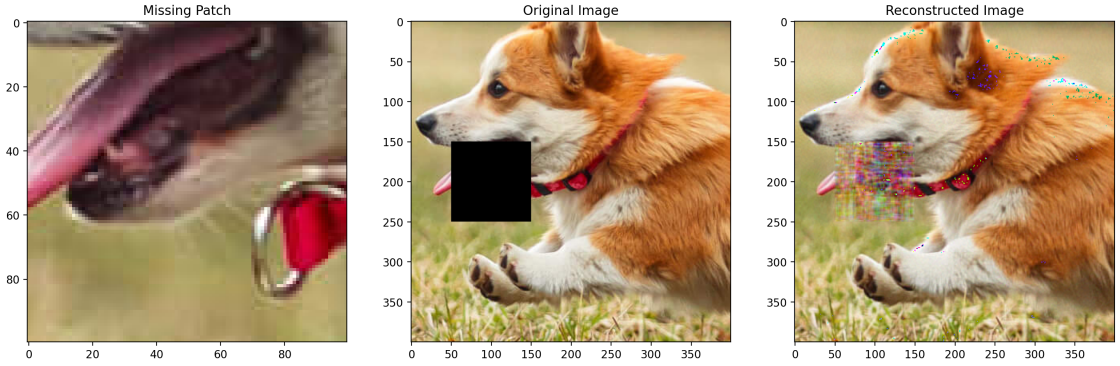
```
C:\Users\patel\AppData\Local\Temp\ipykernel_26008\946811148.py:17:
RuntimeWarning: invalid value encountered in cast
  plt.imshow(img.permute(1, 2, 0).numpy().astype('uint8'))
```






Missing Patch
Original Image
Reconstructed Image


Missing Patch
Original Image
Reconstructed Image


Missing Patch
Original Image
Reconstructed Image

[ ]: