08_16_23

August 17, 2023

1 Imports/Device Settings/Utils

1.1 RUN ONLY ONCE FROM HERE

[1]:

```python
import numpy as np
import torch
import torch.nn as nn
#import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import utils

from PIL import Image
#from skimage.metrics import peak_signal_noise_ratio as psnr

device = (
    "cuda"
    if torch.cuda.is_available()
    else "cpu"
)

print(f"Using {device} device")
torch.set_default_device(device)
torch.set_default_dtype(torch.float32)


IMAGE_DIM = 48
```

```
Using cuda device
```

[2]:

```python
ground_truth_images = np.load("training.npy")
```

[3]:

```python
#ground_truth_images = np.zeros((28709, 1, IMAGE_DIM, IMAGE_DIM), np.float32)
#utils.create_training_data("train/", ground_truth_images)

#normalizations step
#ground_truth_images = np.multiply(ground_truth_images, 1/255)
#np.save("training.npy", ground_truth_images)
```

[4]:

```
#revert normalization
ground_truth_images = np.multiply(ground_truth_images, 255)
clean = torch.as_tensor(ground_truth_images, dtype=torch.float32)
```

1.2 UNTIL HERE!!

1.3 Then back to relevant code:

[5]:

```
# Create A.T*y
# CHOOSE SAMPLING RATE HERE

p = IMAGE_DIM**2
n = int(0.5*p)
A = np.random.normal(loc=0, scale=1/n, size=(n, p))

added_noise_images = np.zeros((28709, 1, IMAGE_DIM, IMAGE_DIM), np.float32)

i = 0
for image in ground_truth_images:
    if (i%1000==0):
        print(f"Image {i}")
    x = ground_truth_images[i][0]
    x = np.reshape(x, (IMAGE_DIM**2), 'F')

    added_noise_images[i][0] = np.reshape(np.matmul(A.T, np.matmul(A, x)),␣
  ↪(IMAGE_DIM, IMAGE_DIM), 'F')
    i += 1

print("done")
```

```
Image 0
Image 1000
Image 2000
Image 3000
Image 4000
Image 5000
Image 6000
Image 7000
Image 8000
Image 9000
Image 10000
Image 11000
Image 12000
Image 13000
Image 14000
Image 15000
Image 16000
Image 17000
Image 18000
Image 19000
Image 20000
Image 21000
Image 22000
Image 23000
Image 24000
Image 25000
Image 26000
Image 27000
Image 28000
done
```

[6]:

```
noisy = torch.as_tensor(added_noise_images, dtype=torch.float32)

# Set training dictionary

train = {"noisy" : noisy[ : 25000],
        "clean" : clean[ : 25000]}

# Set validation dictionary
validation = {"noisy" : noisy[25000 : ],
        "clean" : clean[25000 : ]}
```

1.4 relevant calculations:

[7]:

```
# A priori PSNR calculation
test_number = len(train["noisy"])
print(f"# of test images: {test_number}")

ante_psnr = utils.avg_psnr(test_number, train["noisy"], train["clean"])
print(f"Avg. PSNR: {round(ante_psnr, 2)}")
```

```
# of test images: 25000
```

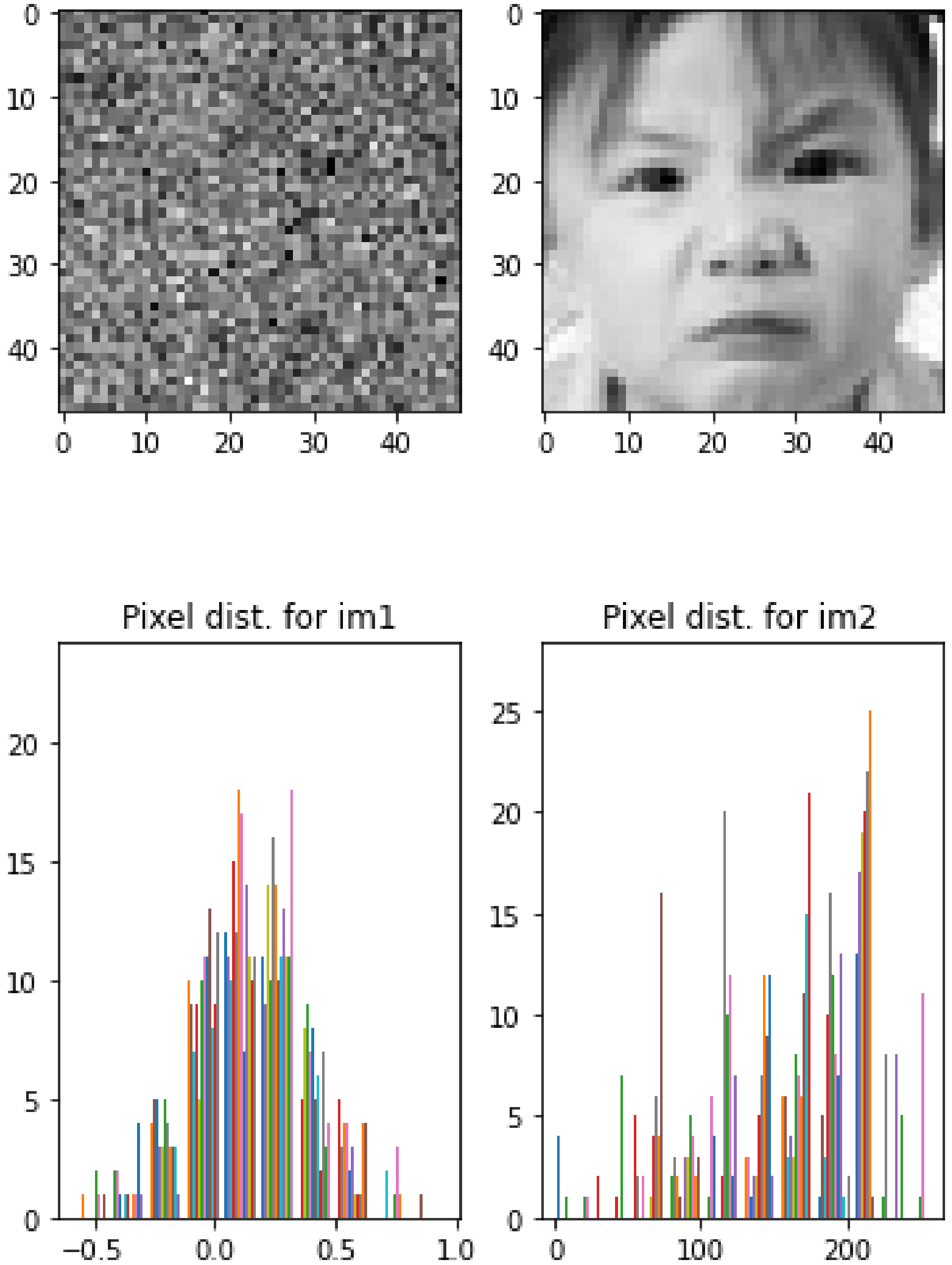
```
/burg/opt/anaconda3-2022.05/lib/python3.9/site-
packages/skimage/metrics/simple_metrics.py:163: RuntimeWarning: invalid value
encountered in double_scalars
  return 10 * np.log10((data_range ** 2) / err)
```

```
Avg. PSNR: 4.35
```

[8]:

```
# Optional - compare noisy vs. clean image
sample_no = 1
im1 = utils.detach(noisy[sample_no][0])
im2 = utils.detach(clean[sample_no][0])
utils.show_images(im1, im2)

# Optional - test pixel distributions
```

```
utils.show_images_hist(im1, im2)
```



2 Correctness of image flattening

[9]:

```
# temp = ground_truth_images[0][0]
# column1 = temp[0:10, 0]
# print(column1)

# columns = np.reshape(temp, (2304), 'F')
# print(columns[0:10])
```

[10]:

```
# p = len(temp)
# n = int(p/2)
# print(p, n)

# A = np.random.normal(loc=0, scale=1/n, size=(n, p))
# y = np.matmul(A, temp)
# Ay = np.matmul(A.T, y)
# Ay = np.reshape(Ay, (48, 48), 'F')

# true_image = ground_truth_images[0][0]
# test_image = Ay
# d_range = np.max([np.max(test_image) - np.min(test_image), np.max(true_image) - np.min(true_image)])
# print(psnr(true_image, test_image, data_range=d_range))
```

3 U-net architecture

[11]:

```
# Establish neural network model

class ConvBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(ConvBlock, self).__init__()
        self.conv1 = nn.Conv2d(in_channels, out_channels, kernel_size=3, stride=1, padding=1)
        self.actv1 = nn.ReLU()

    def forward(self, x):
        x = self.actv1(self.conv1(x))
        return x

# Establish neural network model

class Threenet(nn.Module):
    def __init__(self):
        super(Threenet, self).__init__()

        #1 if batched, 0 if unbatched
        self.cat_dimension = 1

        self.conv1_1 = ConvBlock(in_channels=1, out_channels=64)
        self.conv1_2 = ConvBlock(in_channels=64, out_channels=64)

        self.maxPool1 = nn.MaxPool2d(kernel_size=2)

        self.conv2_1 = ConvBlock(in_channels=64, out_channels=128)
        self.conv2_2 = ConvBlock(in_channels=128, out_channels=128)

        self.maxPool2 = nn.MaxPool2d(kernel_size=2)

        self.conv3_1 = ConvBlock(in_channels=128, out_channels=256)
        self.conv3_2 = ConvBlock(in_channels=256, out_channels=256)

        self.maxPool3 = nn.MaxPool2d(kernel_size=2)

        self.conv4_1 = ConvBlock(in_channels=256, out_channels=512)
        self.conv4_2 = ConvBlock(in_channels=512, out_channels=512)

        self.convTranspose1 = nn.ConvTranspose2d(in_channels=512, out_channels=256, kernel_size=2, stride=2)

        self.conv5_1 = ConvBlock(in_channels=512, out_channels=256)
        self.conv5_2 = ConvBlock(in_channels=256, out_channels=256)

        self.convTranspose2 = nn.ConvTranspose2d(in_channels=256, out_channels=128, kernel_size=2, stride=2)

        self.conv6_1 = ConvBlock(in_channels=256, out_channels=128)
        self.conv6_2 = ConvBlock(in_channels=128, out_channels=128)

        self.convTranspose3 = nn.ConvTranspose2d(in_channels=128, out_channels=64, kernel_size=2, stride=2)

        self.conv7_1 = ConvBlock(in_channels=128, out_channels=64)
        self.conv7_2 = ConvBlock(in_channels=64, out_channels=64)
        self.conv7_3 = ConvBlock(in_channels=64, out_channels=1)

        #DONT FORGET TO ADD CONCATENATION CHANNELS

    def forward(self, x):
        x = self.conv1_2(self.conv1_1(x))

        skip_connect1 = x

        x = self.maxPool1(x)
        x = self.conv2_2(self.conv2_1(x))

        skip_connect2 = x

        x = self.maxPool2(x)
        x = self.conv3_2(self.conv3_1(x))

        skip_connect3 = x

        x = self.maxPool3(x)
        x = self.conv4_2(self.conv4_1(x))

        x = self.convTranspose1(x)
        x = torch.cat([skip_connect3, x], self.cat_dimension)
        x = self.conv5_2(self.conv5_1(x))

        x = self.convTranspose2(x)
        x = torch.cat([skip_connect2, x], self.cat_dimension)
        x = self.conv6_2(self.conv6_1(x))

        x = self.convTranspose3(x)
        x = torch.cat([skip_connect1, x], self.cat_dimension)
        x = self.conv7_3(self.conv7_2(self.conv7_1(x)))

        return x

network = Threenet()
network = network.to(device)
network.train()
```

[11]:

```
Threenet(
  (conv1_1): ConvBlock(
    (conv1): Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (conv1_2): ConvBlock(
    (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (maxPool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (conv2_1): ConvBlock(
    (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (conv2_2): ConvBlock(
    (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (maxPool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (conv3_1): ConvBlock(
    (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (conv3_2): ConvBlock(
    (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (maxPool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (conv4_1): ConvBlock(
    (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (conv4_2): ConvBlock(
    (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (convTranspose1): ConvTranspose2d(512, 256, kernel_size=(2, 2), stride=(2, 2))
  (conv5_1): ConvBlock(
    (conv1): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (conv5_2): ConvBlock(
    (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (convTranspose2): ConvTranspose2d(256, 128, kernel_size=(2, 2), stride=(2, 2))
  (conv6_1): ConvBlock(
    (conv1): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (conv6_2): ConvBlock(
    (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (convTranspose3): ConvTranspose2d(128, 64, kernel_size=(2, 2), stride=(2, 2))
  (conv7_1): ConvBlock(
    (conv1): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (conv7_2): ConvBlock(
    (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (conv7_3): ConvBlock(
    (conv1): Conv2d(64, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
)
```

4 Optional Tests

[12]:

```
# OPTIONAL
# Testing input/output sizes
input = torch.randn(2, 1, IMAGE_DIM, IMAGE_DIM)
print("Input: ", np.shape(input))

output1 = network(input)
print("UNet Output: ", np.shape(output1))
```

```
Input:  torch.Size([2, 1, 48, 48])
UNet Output:  torch.Size([2, 1, 48, 48])
```

5 Training the Model

5.0.1 Set hyperparameters

[13]:

```
epochs = 50
batch_size = 20
lr = 1e-4
loss_fn = nn.MSELoss()
optimizer = torch.optim.Adam(network.parameters(), lr=lr)
```

[14]:

```
# Training
training_loss, test_loss = utils.train(
    x=train["noisy"],
    y=train["clean"],
    validation_x=validation["noisy"],
    validation_y=validation["clean"],
    neural_network=network,
    epochs=epochs,
    batch_size=batch_size,
    learning_rate=lr,
    loss_function=loss_fn,
    optimizer=optimizer
)
```

```
Starting training…
epoch 0 complete. elapsed time: 15s
```

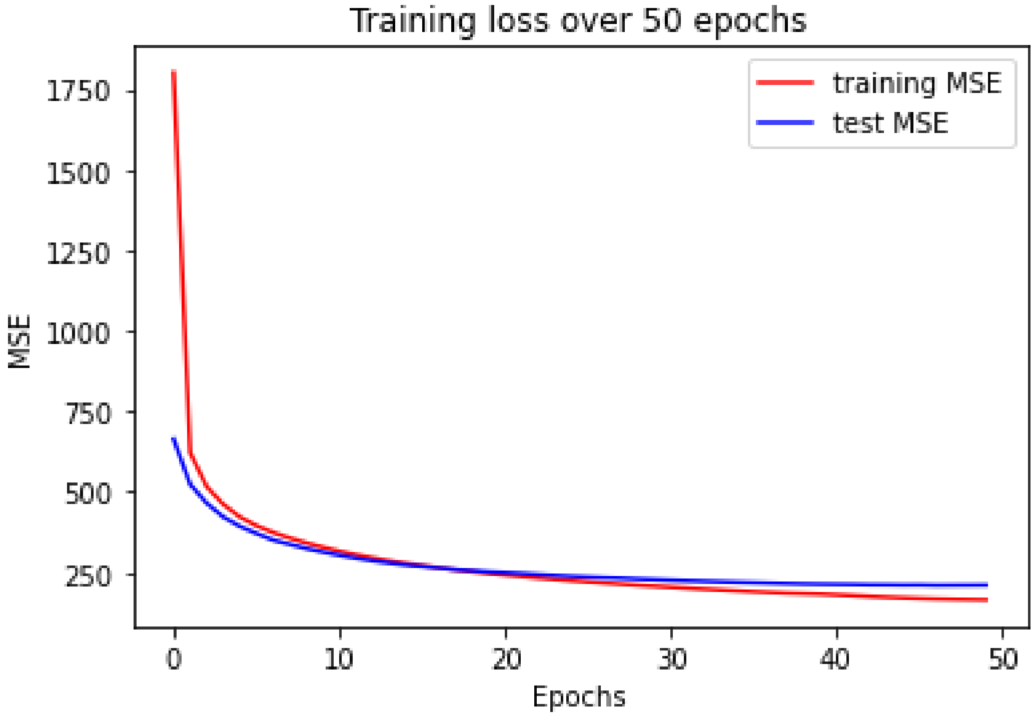
```
training loss: 1802.5513916015625, test loss: 665.0151977539062
epoch 1 complete. elapsed time: 25s
training loss: 622.411865234375, test loss: 523.985107421875
epoch 2 complete. elapsed time: 35s
training loss: 516.7144165039062, test loss: 465.3548583984375
epoch 3 complete. elapsed time: 45s
training loss: 461.79217529296875, test loss: 422.3830871582031
epoch 4 complete. elapsed time: 55s
training loss: 422.38433837890625, test loss: 393.26788330078125
epoch 5 complete. elapsed time: 65s
training loss: 395.4277038574219, test loss: 371.1346130371094
epoch 6 complete. elapsed time: 75s
training loss: 374.3458557128906, test loss: 350.6646423339844
epoch 7 complete. elapsed time: 84s
training loss: 357.1914367675781, test loss: 337.5110778808594
epoch 8 complete. elapsed time: 94s
training loss: 341.9107666015625, test loss: 324.4728088378906
epoch 9 complete. elapsed time: 104s
training loss: 328.2871398925781, test loss: 313.4806823730469
epoch 10 complete. elapsed time: 114s
training loss: 316.0379333496094, test loss: 303.58551025390625
epoch 11 complete. elapsed time: 124s
training loss: 304.9957275390625, test loss: 294.5194396972656
epoch 12 complete. elapsed time: 134s
training loss: 295.2705078125, test loss: 286.2991943359375
epoch 13 complete. elapsed time: 143s
training loss: 286.2917785644531, test loss: 279.5648498535156
epoch 14 complete. elapsed time: 153s
training loss: 278.4056396484375, test loss: 273.3607177734375
epoch 15 complete. elapsed time: 163s
training loss: 271.0382385253906, test loss: 268.5890808105469
epoch 16 complete. elapsed time: 173s
training loss: 264.4971923828125, test loss: 263.87164306640625
epoch 17 complete. elapsed time: 183s
training loss: 258.1697692871094, test loss: 258.84442138671875
epoch 18 complete. elapsed time: 193s
training loss: 252.35044860839844, test loss: 255.31797790527344
epoch 19 complete. elapsed time: 203s
training loss: 246.97091674804688, test loss: 251.5403289794922
epoch 20 complete. elapsed time: 213s
training loss: 241.95431518554688, test loss: 249.09732055664062
epoch 21 complete. elapsed time: 222s
training loss: 237.2635955810547, test loss: 246.48399353027344
epoch 22 complete. elapsed time: 232s
training loss: 232.85414123535156, test loss: 243.36354064941406
epoch 23 complete. elapsed time: 242s
training loss: 228.5993194580078, test loss: 240.6340789794922
epoch 24 complete. elapsed time: 252s
```

```
training loss: 224.567138671875, test loss: 237.85043334960938
epoch 25 complete. elapsed time: 262s
training loss: 220.77330017089844, test loss: 235.79922485351562
epoch 26 complete. elapsed time: 272s
training loss: 217.12429809570312, test loss: 233.82919311523438
epoch 27 complete. elapsed time: 282s
training loss: 213.58514404296875, test loss: 232.19166564941406
epoch 28 complete. elapsed time: 292s
training loss: 210.30291748046875, test loss: 229.68768310546875
epoch 29 complete. elapsed time: 301s
training loss: 207.23300170898438, test loss: 227.72613525390625
epoch 30 complete. elapsed time: 311s
training loss: 204.1021728515625, test loss: 225.81854248046875
epoch 31 complete. elapsed time: 321s
training loss: 201.107421875, test loss: 224.1342010498047
epoch 32 complete. elapsed time: 331s
training loss: 198.2167510986328, test loss: 222.49327087402344
epoch 33 complete. elapsed time: 341s
training loss: 195.44253540039062, test loss: 220.91770935058594
epoch 34 complete. elapsed time: 350s
training loss: 192.7393798828125, test loss: 219.41827392578125
epoch 35 complete. elapsed time: 360s
training loss: 190.14138793945312, test loss: 218.14453125
epoch 36 complete. elapsed time: 370s
training loss: 187.79193115234375, test loss: 216.89280700683594
epoch 37 complete. elapsed time: 379s
training loss: 185.7062225341797, test loss: 215.64926147460938
epoch 38 complete. elapsed time: 389s
training loss: 184.44827270507812, test loss: 214.45855712890625
epoch 39 complete. elapsed time: 398s
training loss: 183.015625, test loss: 213.6410675048828
epoch 40 complete. elapsed time: 408s
training loss: 180.52981567382812, test loss: 213.08062744140625
epoch 41 complete. elapsed time: 417s
training loss: 177.99095153808594, test loss: 212.54307556152344
epoch 42 complete. elapsed time: 427s
training loss: 175.25914001464844, test loss: 211.60459899902344
epoch 43 complete. elapsed time: 437s
training loss: 172.98565673828125, test loss: 211.47093200683594
epoch 44 complete. elapsed time: 446s
training loss: 170.97291564941406, test loss: 210.9196014404297
epoch 45 complete. elapsed time: 456s
training loss: 169.19891357421875, test loss: 211.03256225585938
epoch 46 complete. elapsed time: 465s
training loss: 167.78114318847656, test loss: 210.12930297851562
epoch 47 complete. elapsed time: 475s
training loss: 166.7620391845703, test loss: 210.3343963623047
epoch 48 complete. elapsed time: 484s
```

```
training loss: 165.90383911132812, test loss: 210.46292114257812
epoch 49 complete. elapsed time: 494s
training loss: 165.0670166015625, test loss: 210.6776123046875
Done training.
```

6 Loss graph

[15]:
```
epoch_record = np.arange(0, len(training_loss), 1)
plt.plot(epoch_record, training_loss, c="red", label="training MSE")
plt.plot(epoch_record, test_loss, c="blue", label="test MSE")
plt.legend(loc="upper right")
plt.title(f"Training loss over {epochs} epochs")
plt.xlabel("Epochs")
plt.ylabel("MSE")
plt.show()
```



7 Tests/PSNR Calculations

[16]:
```
network.eval()
```

[16]:
```
Threenet(
  (conv1_1): ConvBlock(
    (conv1): Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (conv1_2): ConvBlock(
    (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (maxPool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (conv2_1): ConvBlock(
    (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (conv2_2): ConvBlock(
    (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (maxPool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (conv3_1): ConvBlock(
    (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (conv3_2): ConvBlock(
    (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (maxPool3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  (conv4_1): ConvBlock(
    (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (conv4_2): ConvBlock(
    (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (convTranspose1): ConvTranspose2d(512, 256, kernel_size=(2, 2), stride=(2, 2))
  (conv5_1): ConvBlock(
    (conv1): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (conv5_2): ConvBlock(
    (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (convTranspose2): ConvTranspose2d(256, 128, kernel_size=(2, 2), stride=(2, 2))
  (conv6_1): ConvBlock(
    (conv1): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (conv6_2): ConvBlock(
    (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (convTranspose3): ConvTranspose2d(128, 64, kernel_size=(2, 2), stride=(2, 2))
  (conv7_1): ConvBlock(
    (conv1): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (conv7_2): ConvBlock(
    (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (conv7_3): ConvBlock(
    (conv1): Conv2d(64, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
)
```

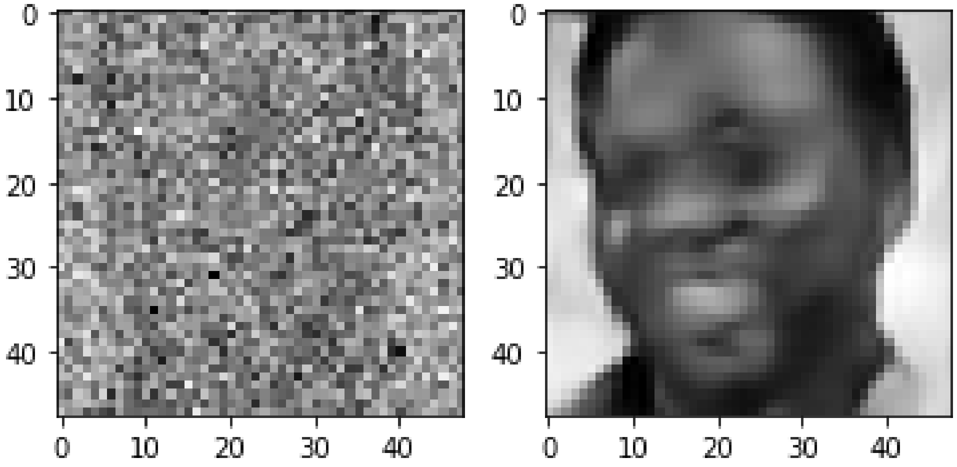
[17]:

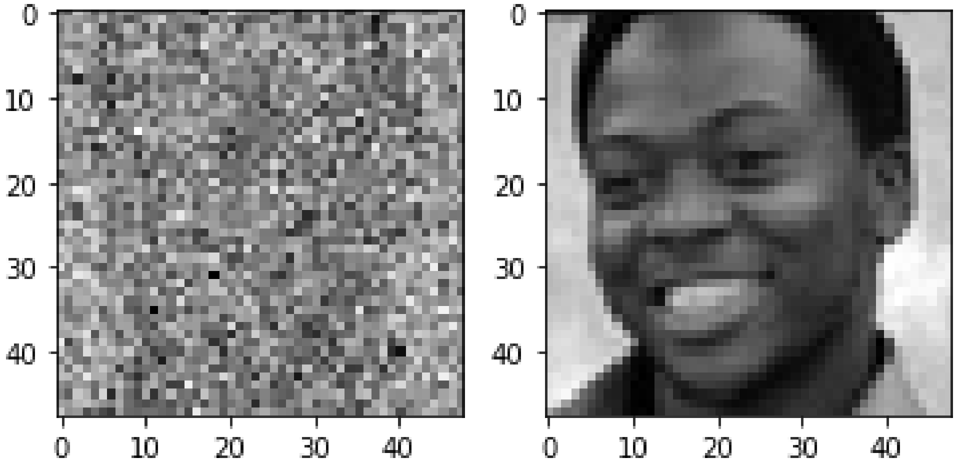
```python
# Choose between [0, 19]
sample_number = 1
test_size = 100

# Test visual recovery results
with torch.no_grad():
    predictions = network(validation["noisy"][0:test_size])

post_psnr = utils.avg_psnr(test_size, predictions, validation["clean"])
print(f"Avg. PSNR: {round(post_psnr, 2)}")
```

```
Avg. PSNR: 24.84
```

[18]:

```python
sample_no = 29
im0 = utils.detach(validation["noisy"][sample_no][0])
im1 = utils.detach(predictions[sample_no][0])
im2 = utils.detach(validation["clean"][sample_no][0])
utils.show_images(im0, im1)
utils.show_images(im0, im2)
```


0
10
20
30
40
0
10
20
30
40
0
10
20
30
40
0
10
20
30
40


0
10
20
30
40
0
10
20
30
40
0
10
20
30
40
0
10
20
30
40

```
[ ]:
```