08_16_23

August 17, 2023

1 Imports/Device Settings/Utils

1.1 RUN ONLY ONCE FROM HERE

[59]:

```
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
```

[60]:

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

[61]:

```
#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)
```

[62]:

```
#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:

[91]:

```
# 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
```

[92]:

```
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:

[93]:

```
# 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
```

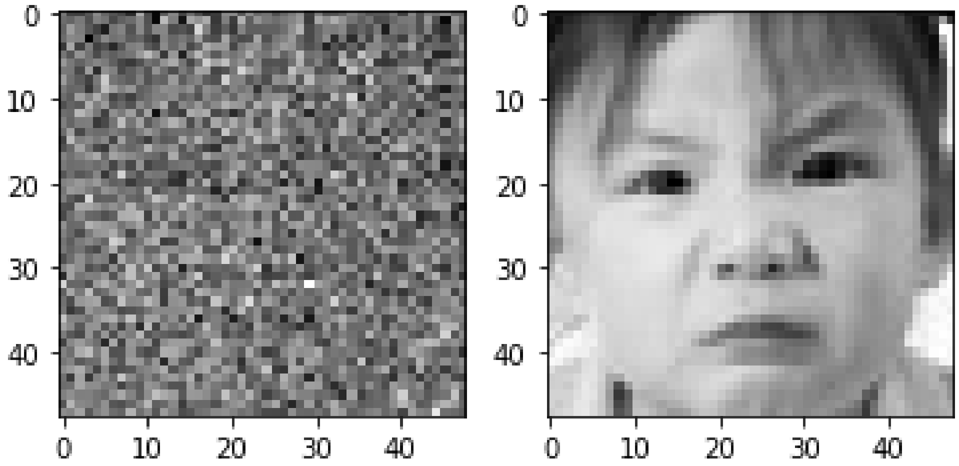
```
/burg/opt/anaconda3-2022.05/lib/python3.9/site-
packages/skimage/metrics/simple_metrics.py:163: RuntimeWarning: invalid value
encountered in double_scalars
  return 10 * np.log10((data_range ** 2) / err)
```

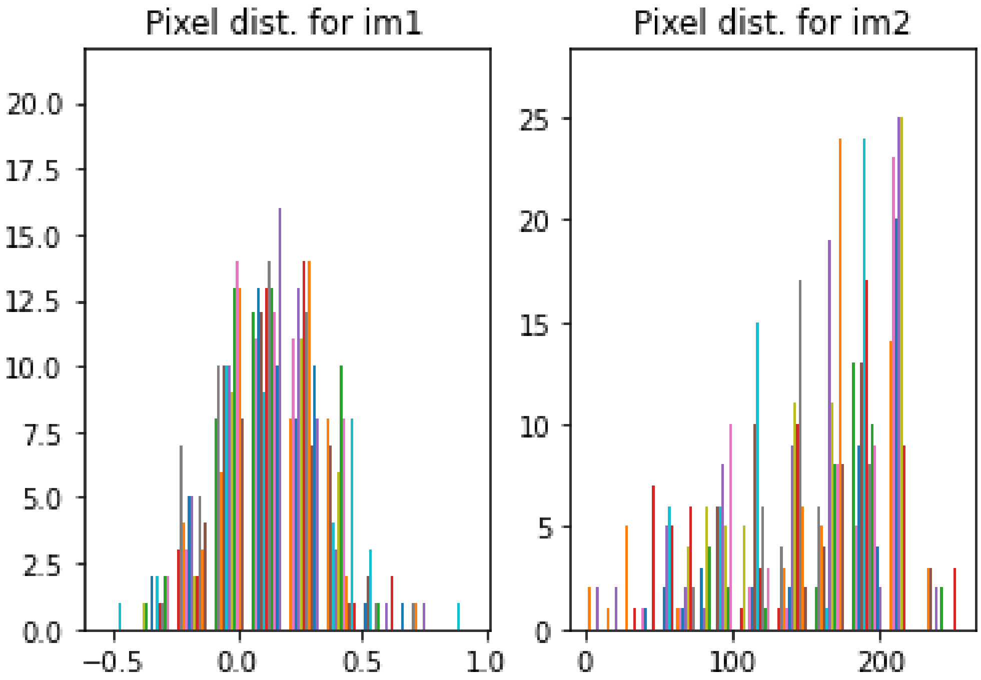
```
Avg. PSNR: 4.35
```

[94]:

```
# 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

[95]:

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

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

[96]:

```
# 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

[97]:

```
# 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 Twonet(nn.Module):
    def __init__(self):
        super(Twonet, 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.convTranspose1 = nn.ConvTranspose2d(in_channels=256,␣
↪out_channels=128, kernel_size=2, stride=2)

        self.conv4_1 = ConvBlock(in_channels=256, out_channels=128)
        self.conv4_2 = ConvBlock(in_channels=128, out_channels=128)

        self.convTranspose2 = nn.ConvTranspose2d(in_channels=128,␣
↪out_channels=64, kernel_size=2, stride=2)

        self.conv5_1 = ConvBlock(in_channels=128, out_channels=64)
        self.conv5_2 = 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))

        x = self.convTranspose1(x)

        x = torch.cat([skip_connect2, x], self.cat_dimension)
        x = self.conv4_2(self.conv4_1(x))
        x = self.convTranspose2(x)

        x = torch.cat([skip_connect1, x], self.cat_dimension)
        x = self.conv5_2(self.conv5_1(x))

        return x

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

[97]:

```
Twonet(
  (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()
  )
  (convTranspose1): ConvTranspose2d(256, 128, kernel_size=(2, 2), stride=(2, 2))
  (conv4_1): ConvBlock(
    (conv1): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (conv4_2): ConvBlock(
    (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (convTranspose2): ConvTranspose2d(128, 64, kernel_size=(2, 2), stride=(2, 2))
  (conv5_1): ConvBlock(
    (conv1): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (conv5_2): ConvBlock(
    (conv1): Conv2d(64, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
)
```

4 Optional Tests

[98]:

```
# 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

[99]:

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

[100]:

```
# 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: 7s
training loss: 1861.3309326171875, test loss: 735.1430053710938
epoch 1 complete. elapsed time: 14s
training loss: 730.4447631835938, test loss: 671.6201782226562
epoch 2 complete. elapsed time: 21s
training loss: 659.782470703125, test loss: 612.9658203125
epoch 3 complete. elapsed time: 28s
training loss: 613.9469604492188, test loss: 565.3809814453125
epoch 4 complete. elapsed time: 35s
training loss: 578.6698608398438, test loss: 536.9165649414062
epoch 5 complete. elapsed time: 42s
training loss: 549.6648559570312, test loss: 511.2002868652344
epoch 6 complete. elapsed time: 49s
training loss: 526.0184326171875, test loss: 483.69854736328125
epoch 7 complete. elapsed time: 56s
training loss: 504.69744873046875, test loss: 465.8990173339844
epoch 8 complete. elapsed time: 63s
training loss: 483.731689453125, test loss: 446.2145080566406
epoch 9 complete. elapsed time: 70s
training loss: 463.05633544921875, test loss: 431.4891662597656
epoch 10 complete. elapsed time: 77s
training loss: 444.573974609375, test loss: 417.3224182128906
epoch 11 complete. elapsed time: 84s
training loss: 428.67437744140625, test loss: 403.796630859375
epoch 12 complete. elapsed time: 91s
training loss: 414.38751220703125, test loss: 394.0511474609375
epoch 13 complete. elapsed time: 98s
training loss: 401.9185791015625, test loss: 383.3927307128906
epoch 14 complete. elapsed time: 105s
training loss: 390.56622314453125, test loss: 374.6682434082031
epoch 15 complete. elapsed time: 112s
training loss: 380.46270751953125, test loss: 366.3709716796875
epoch 16 complete. elapsed time: 119s
training loss: 371.0546875, test loss: 360.0040283203125
epoch 17 complete. elapsed time: 126s
training loss: 362.37994384765625, test loss: 352.8594665527344
epoch 18 complete. elapsed time: 133s
training loss: 354.3267517089844, test loss: 346.6192321777344
epoch 19 complete. elapsed time: 140s
training loss: 346.608154296875, test loss: 340.80279541015625
epoch 20 complete. elapsed time: 147s
training loss: 339.265380859375, test loss: 334.51715087890625
epoch 21 complete. elapsed time: 154s
```

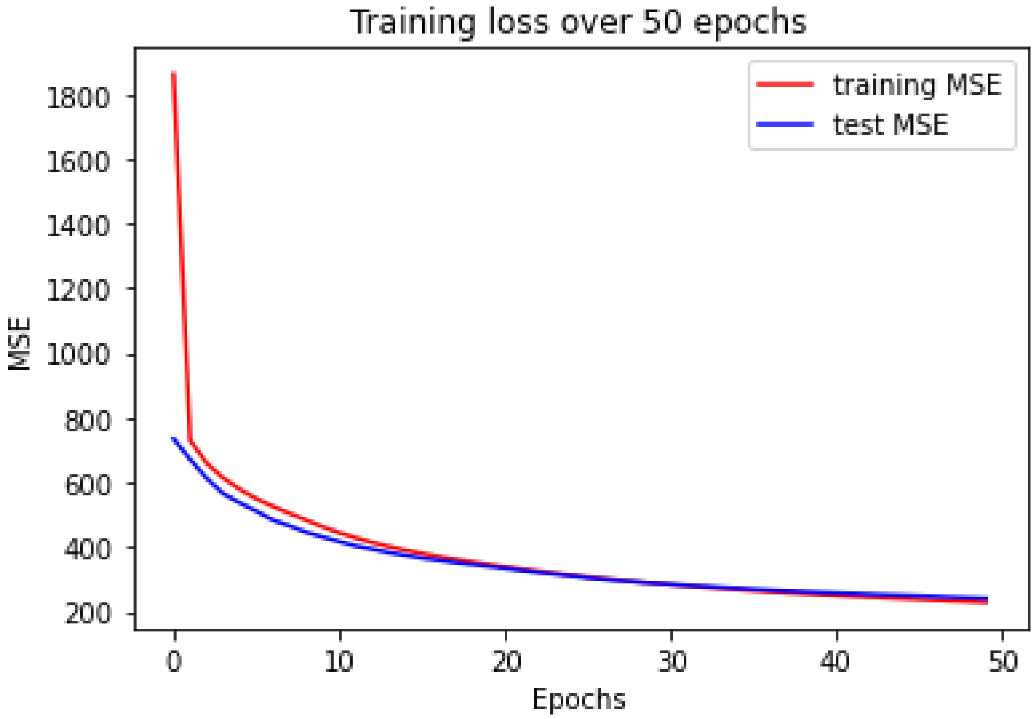
```
training loss: 332.2876281738281, test loss: 327.9779052734375
epoch 22 complete. elapsed time: 162s
training loss: 325.6748352050781, test loss: 322.1311340332031
epoch 23 complete. elapsed time: 169s
training loss: 319.4094543457031, test loss: 316.9410705566406
epoch 24 complete. elapsed time: 176s
training loss: 313.5443420410156, test loss: 311.3306884765625
epoch 25 complete. elapsed time: 183s
training loss: 308.10992431640625, test loss: 305.7515563964844
epoch 26 complete. elapsed time: 190s
training loss: 302.8777160644531, test loss: 300.5751037597656
epoch 27 complete. elapsed time: 197s
training loss: 297.8593444824219, test loss: 296.4984130859375
epoch 28 complete. elapsed time: 204s
training loss: 292.9192199707031, test loss: 292.3488464355469
epoch 29 complete. elapsed time: 211s
training loss: 288.485595703125, test loss: 288.4309997558594
epoch 30 complete. elapsed time: 219s
training loss: 284.235107421875, test loss: 285.108642578125
epoch 31 complete. elapsed time: 226s
training loss: 280.0912780761719, test loss: 281.8576965332031
epoch 32 complete. elapsed time: 234s
training loss: 276.2082824707031, test loss: 278.7071533203125
epoch 33 complete. elapsed time: 241s
training loss: 272.52239990234375, test loss: 275.643798828125
epoch 34 complete. elapsed time: 248s
training loss: 269.068603515625, test loss: 272.75372314453125
epoch 35 complete. elapsed time: 254s
training loss: 265.6302185058594, test loss: 269.83203125
epoch 36 complete. elapsed time: 261s
training loss: 262.4419860839844, test loss: 267.4181823730469
epoch 37 complete. elapsed time: 268s
training loss: 259.359130859375, test loss: 264.8430480957031
epoch 38 complete. elapsed time: 275s
training loss: 256.4273376464844, test loss: 262.6467590332031
epoch 39 complete. elapsed time: 282s
training loss: 253.5440673828125, test loss: 260.4901428222656
epoch 40 complete. elapsed time: 289s
training loss: 250.83164978027344, test loss: 258.5862731933594
epoch 41 complete. elapsed time: 296s
training loss: 248.2327880859375, test loss: 256.67889404296875
epoch 42 complete. elapsed time: 303s
training loss: 245.7191619873047, test loss: 254.65151977539062
epoch 43 complete. elapsed time: 310s
training loss: 243.3513641357422, test loss: 252.72213745117188
epoch 44 complete. elapsed time: 317s
training loss: 240.98834228515625, test loss: 251.26190185546875
epoch 45 complete. elapsed time: 324s
```

```
training loss: 238.8385467529297, test loss: 249.60926818847656
epoch 46 complete. elapsed time: 330s
training loss: 236.67416381835938, test loss: 247.65902709960938
epoch 47 complete. elapsed time: 337s
training loss: 234.61024475097656, test loss: 245.51751708984375
epoch 48 complete. elapsed time: 344s
training loss: 232.65182495117188, test loss: 243.7272186279297
epoch 49 complete. elapsed time: 351s
training loss: 230.67630004882812, test loss: 242.05047607421875
Done training.
```

6 Loss graph

[101]:

```python
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

[102]:

```
network.eval()
```

[102]:

```
Twonet(
  (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()
  )
  (convTranspose1): ConvTranspose2d(256, 128, kernel_size=(2, 2), stride=(2, 2))
  (conv4_1): ConvBlock(
    (conv1): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (conv4_2): ConvBlock(
    (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (convTranspose2): ConvTranspose2d(128, 64, kernel_size=(2, 2), stride=(2, 2))
  (conv5_1): ConvBlock(
    (conv1): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (conv5_2): ConvBlock(
    (conv1): Conv2d(64, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
)
```

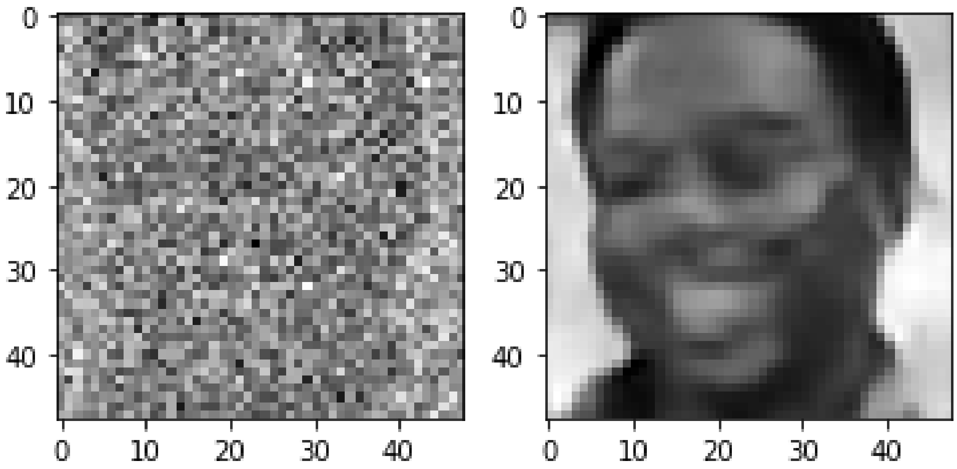
[103]:

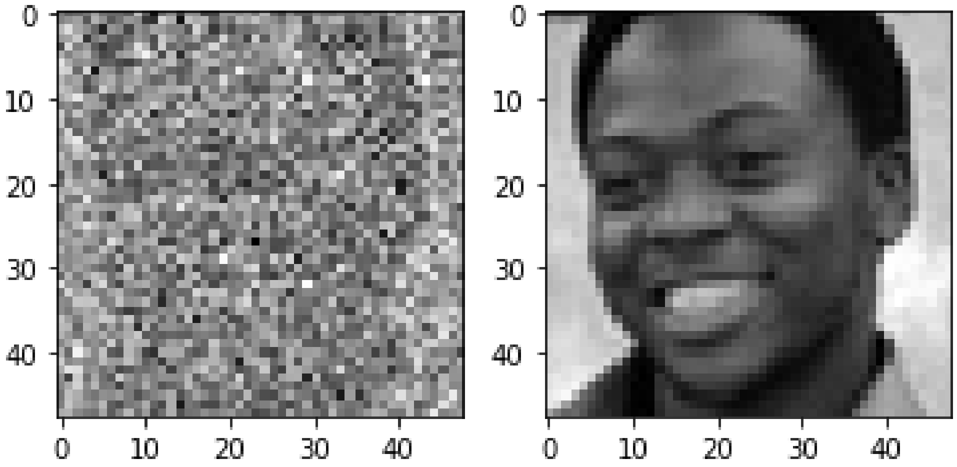
```
# Choose between [0, 19]
sample_number = 1
test_size = 100

# 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: 24.08
```

[104]:

```
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

[ ]: