08_17_23

August 17, 2023

1 Imports/Device Settings/Utils

[1]:
```
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
```

[2]:
```
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
```

[3]:
```
ground_truth_images = np.load("training.npy")
```

1.1 If you don't have the above dataset:

[ ]:
```
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)
```

1.2 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, (2304), '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]:
```python
print(np.shape(ground_truth_images))
print(np.shape(added_noise_images))

clean = torch.as_tensor(ground_truth_images, dtype=torch.float32)
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 : ]}
```

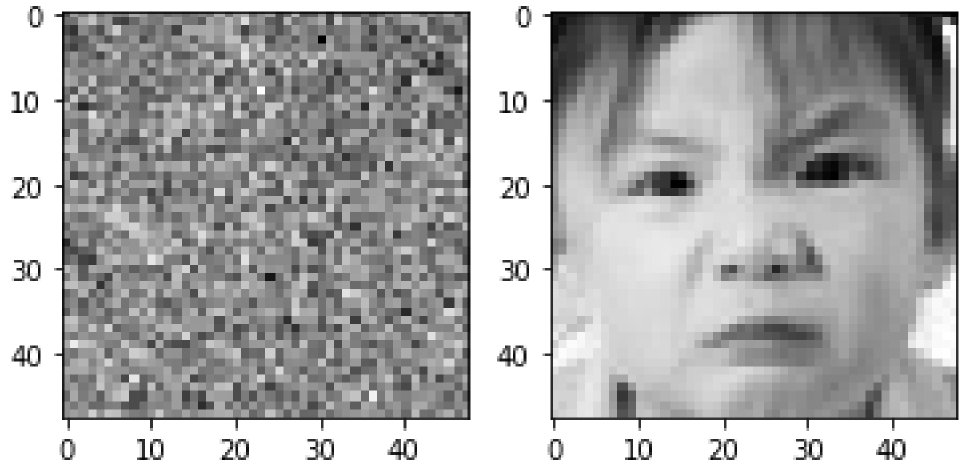
```
(28709, 1, 48, 48)
(28709, 1, 48, 48)
```

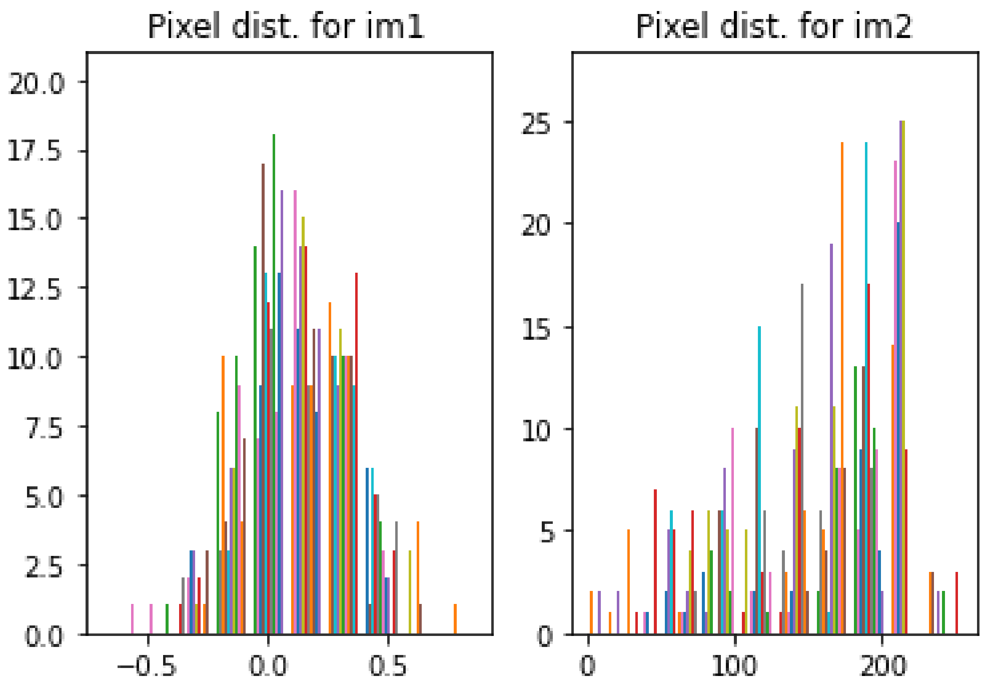
1.3 Then back to relevant calculations:

[7]:
```python
# A priori PSNR calculation
test_number = len(train["noisy"])
print(f"# of test images: {test_number}")

ante_psnr = utils.avg_psnr(300, train["noisy"], train["clean"])
print(f"Avg. PSNR: {round(ante_psnr, 2)}")
```

```
# of test images: 25000
Avg. PSNR: 4.58
```

[8]:
```python
# 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 Proof of inverse problem setup

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

columns = np.reshape(temp, (2304), 'F')
```

```
print(columns[0:10])

#temp = np.rot90(np.flip(temp, 1), 1)
#temp = np.reshape(temp, (48*48))
#print(temp[0:10])
```

[ ]:
```
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')

plt.imshow(Ay, cmap='gray')
```

[ ]:
```
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

[9]:
```
# 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

class UNet(nn.Module):
    def __init__(self):
        super(UNet, self).__init__()
        #1 if batched, 0 if unbatched
        self.cat_dim = 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.convTranspose1 = nn.ConvTranspose2d(in_channels=128,␣
 ↪out_channels=64, kernel_size=2, stride=2)

        #DONT FORGET TO ADD CONCATENATION CHANNELS
        self.conv3_1 = ConvBlock(in_channels=128, out_channels=64)
        self.conv3_2 = ConvBlock(in_channels=64, out_channels=1)

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

        skip_connect = x

        x = self.maxPool1(x)

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

        x = self.convTranspose1(x)

        x = torch.cat([skip_connect, x], self.cat_dim)

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

        return x

unet = UNet()
unet = unet.to(device)
unet.train()
```

```
[9]: UNet(
       (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()
  )
  (convTranspose1): ConvTranspose2d(128, 64, kernel_size=(2, 2), stride=(2, 2))
  (conv3_1): ConvBlock(
    (conv1): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (conv3_2): ConvBlock(
    (conv1): Conv2d(64, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
)
```

4 Optional Tests

[10]:
```
# OPTIONAL
# Testing input/output sizes
input = torch.randn(2, 1, IMAGE_DIM, IMAGE_DIM)
print("Input: ", np.shape(input))

output1 = unet(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 = 40
batch_size = 20
lr = 1e-5
loss_fn = nn.MSELoss()
optimizer = torch.optim.Adam(unet.parameters(), lr=lr)
```

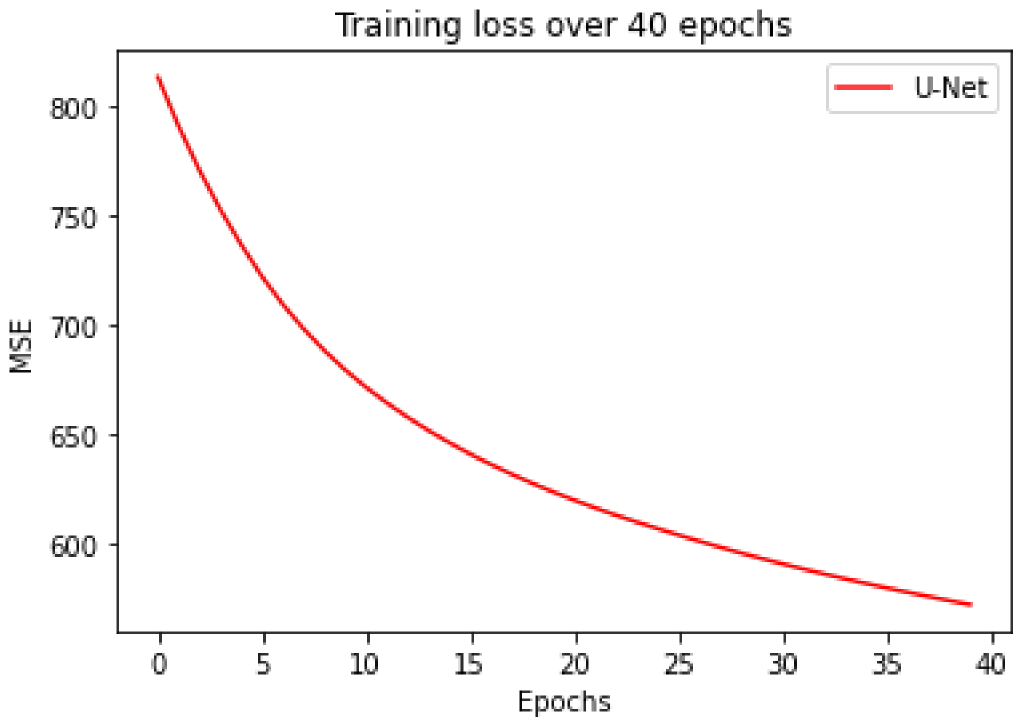
[14]:
```
# Training
loss_record = utils.train(
    x=train["noisy"],
    y=train["clean"],
    neural_network=unet,
    epochs=epochs,
    batch_size=batch_size,
    learning_rate=lr,
    loss_function=loss_fn,
    optimizer=optimizer
)
```

```
Starting training…
Epoch 0 complete. Elapsed time: 4s
Average loss: 813.6483154296875
Epoch 1 complete. Elapsed time: 9s
Average loss: 791.068359375
Epoch 2 complete. Elapsed time: 13s
Average loss: 770.9090576171875
Epoch 3 complete. Elapsed time: 17s
Average loss: 753.00048828125
Epoch 4 complete. Elapsed time: 22s
Average loss: 736.9688110351562
Epoch 5 complete. Elapsed time: 26s
Average loss: 722.4353637695312
Epoch 6 complete. Elapsed time: 30s
Average loss: 709.6864624023438
Epoch 7 complete. Elapsed time: 35s
Average loss: 698.4712524414062
Epoch 8 complete. Elapsed time: 39s
Average loss: 688.5208129882812
Epoch 9 complete. Elapsed time: 43s
Average loss: 679.658203125
Epoch 10 complete. Elapsed time: 48s
Average loss: 671.7188720703125
Epoch 11 complete. Elapsed time: 52s
Average loss: 664.544921875
Epoch 12 complete. Elapsed time: 56s
Average loss: 657.9549560546875
Epoch 13 complete. Elapsed time: 61s
Average loss: 651.8871459960938
Epoch 14 complete. Elapsed time: 65s
Average loss: 646.3385009765625
Epoch 15 complete. Elapsed time: 69s
Average loss: 641.1995849609375
Epoch 16 complete. Elapsed time: 74s
Average loss: 636.4400634765625
Epoch 17 complete. Elapsed time: 78s
```

```
Average loss: 631.9671630859375
Epoch 18 complete. Elapsed time: 83s
Average loss: 627.7855834960938
Epoch 19 complete. Elapsed time: 87s
Average loss: 623.8380126953125
Epoch 20 complete. Elapsed time: 91s
Average loss: 620.1175537109375
Epoch 21 complete. Elapsed time: 96s
Average loss: 616.6078491210938
Epoch 22 complete. Elapsed time: 100s
Average loss: 613.2723999023438
Epoch 23 complete. Elapsed time: 104s
Average loss: 610.1095581054688
Epoch 24 complete. Elapsed time: 109s
Average loss: 607.1044921875
Epoch 25 complete. Elapsed time: 113s
Average loss: 604.1787109375
Epoch 26 complete. Elapsed time: 117s
Average loss: 601.3329467773438
Epoch 27 complete. Elapsed time: 122s
Average loss: 598.5825805664062
Epoch 28 complete. Elapsed time: 126s
Average loss: 595.9321899414062
Epoch 29 complete. Elapsed time: 130s
Average loss: 593.4024047851562
Epoch 30 complete. Elapsed time: 135s
Average loss: 590.9552001953125
Epoch 31 complete. Elapsed time: 139s
Average loss: 588.5995483398438
Epoch 32 complete. Elapsed time: 144s
Average loss: 586.3338623046875
Epoch 33 complete. Elapsed time: 148s
Average loss: 584.1436157226562
Epoch 34 complete. Elapsed time: 152s
Average loss: 582.0146484375
Epoch 35 complete. Elapsed time: 157s
Average loss: 579.9557495117188
Epoch 36 complete. Elapsed time: 161s
Average loss: 577.9662475585938
Epoch 37 complete. Elapsed time: 165s
Average loss: 576.03076171875
Epoch 38 complete. Elapsed time: 170s
Average loss: 574.1450805664062
Epoch 39 complete. Elapsed time: 174s
Average loss: 572.3075561523438
Done training.
```

6 Loss graph

[15]:
```
epoch_record = np.arange(0, len(loss_record), 1)
plt.plot(epoch_record, loss_record, c="red", label="U-Net")
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

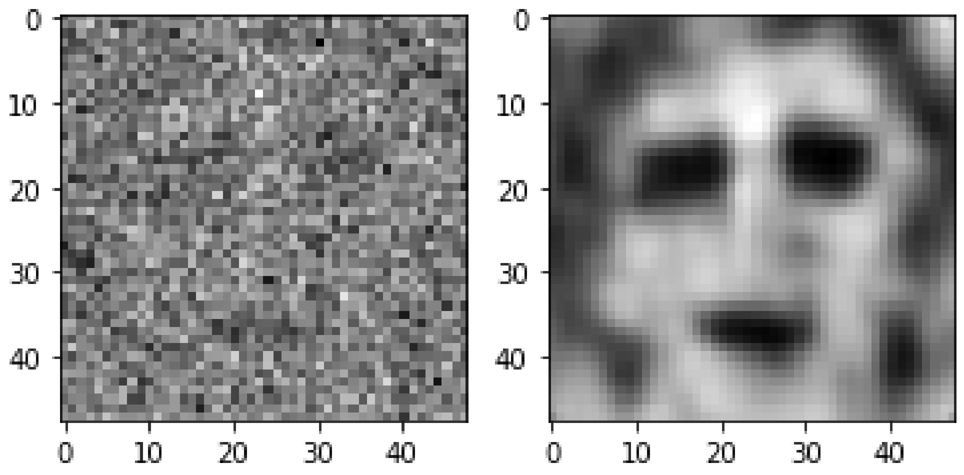
[16]:
```
unet.eval()
```

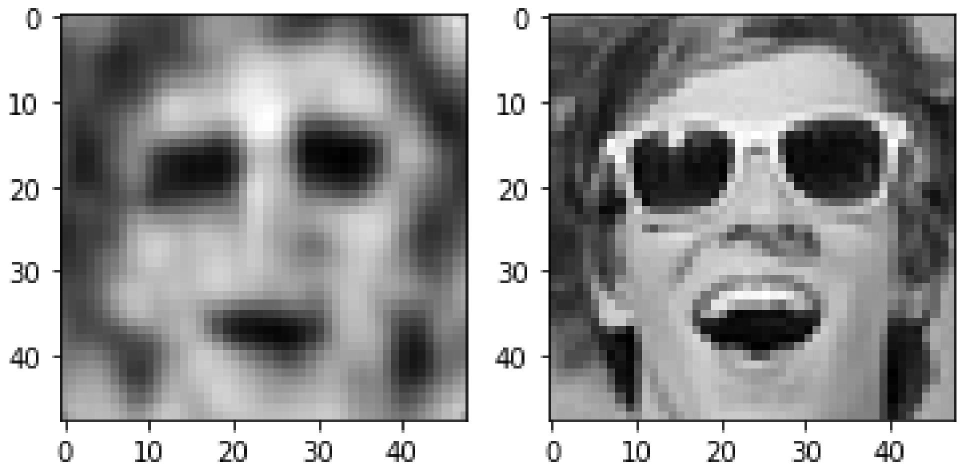
[16]:
```
UNet(
  (conv1_1): ConvBlock(
    (conv1): Conv2d(1, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (conv1_2): ConvBlock(
    (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (maxPool1): MaxPool2d(kernel_size=2, 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()
  )
  (convTranspose1): ConvTranspose2d(128, 64, kernel_size=(2, 2), stride=(2, 2))
  (conv3_1): ConvBlock(
    (conv1): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
  (conv3_2): ConvBlock(
    (conv1): Conv2d(64, 1, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (actv1): ReLU()
  )
)
```

[21]:
```
# Choose between [0, 19]
sample_number = 1
test_size = 100

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

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

```
Avg. PSNR: 20.37
```

[26]:
```
sample_no = 10
im0 = utils.detach(validation["noisy"][sample_no][0])
im1 = utils.detach(unet_pred[sample_no][0])
im2 = utils.detach(validation["clean"][sample_no][0])
utils.show_images(im0, im1)
utils.show_images(im1, im2)
```


0
10
20
30
40
0
10
20
30
40


0
10
20
30
40
0
10
20
30
40

```
[ ]:
```