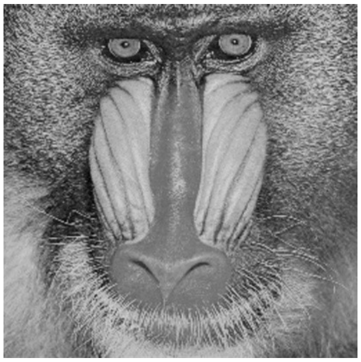
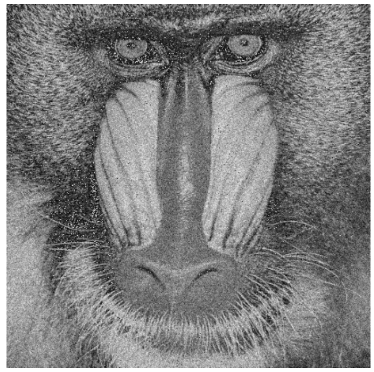
Caden Thompson | Homework 3 | 4/17/25

1)

Original



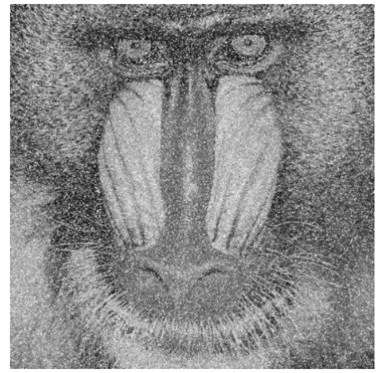
AWGN



PSNR value is 28.39dB

MSE value is 94.29dB

AWGN and S&P Noise



PSNR value is 28.33dB

MSE value is 95.50dB

Original

AWGN

PSNR value is 28.39dB

MSE value is 94.13dB

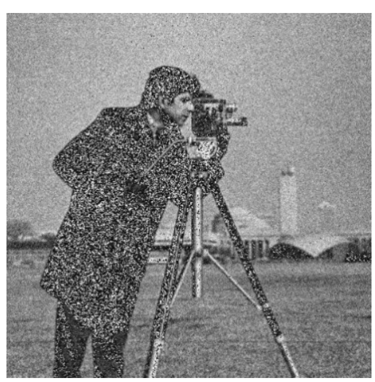
AWGN and S&P Noise

PSNR value is 28.33dB

MSE value is 95.47dB

Original

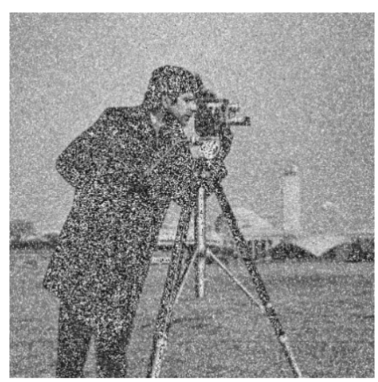
AWGN



PSNR value is 28.38dB

MSE value is 94.32dB

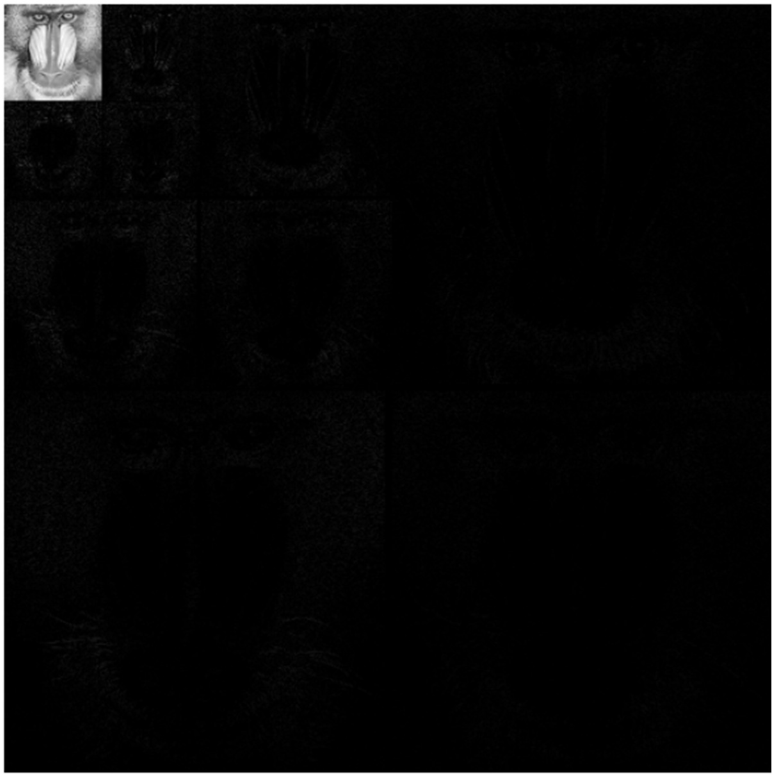
AWGN and S&P Noise



PSNR value is 28.30dB

MSE value is 96.16dB

Flattened Wavelet Coefficients (3-Level, db2)



Flattened Wavelet Coefficients (3-Level, db2)

Flattened Wavelet Coefficients (3-Level, db2)

Results and Discussion:

For each image as more noise was added the MSE increased. This can be seen when comparing the MSE for each image when just AWGN was added and when both AWGN and Salt and Pepper noise was added. The Peak Signal to Noise Ration for each image was around 28.3 dB with the only real constant being a lower PSNR when adding both noises as compared to just AWGN.

After applying 2D Discrete Wavelet Transform and 3 levels of decomposition the images output are rather hard to depict in this format. The individual images were much more visible when output in the terminal, but once saved and pasted into a word document the images are very hard to depict due to their dark color. The three images all had very similar results.

Caden Thompson Homework 3 4/17/25

Code for AWGN Noise

```python
#https://github.com/behnamasadi/PythonTutorial/blob/master/signal_system/white_noise_gaussian_noise.ipynb
def AddGaussianNoise():
    # original image
   # imagepath = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/mandril_color.tif'
    #imagepath = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/lena_gray_256.tif'
    imagepath = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/cameraman.tif'
    f = cv2.imread(imagepath, 0)
    f = f/255
    # create gaussian noise
    x, y = f.shape
    mean = 0
    var = .01
    sigma = np.sqrt(var)
    n = np.random.normal(loc=mean,
                         scale=sigma,
                         size=(x,y))
    # add a gaussian noise
    g = f + n
    g = (g*255).round().astype(np.uint8)
    plt.hist(g.flat)
    plt.xlim([0,255]); plt.ylim([0,60000])
    plt.xlabel('pixel value'); plt.ylabel('frequency')
    plt.show()
    os.chdir('C:/Users/caden/CompVision')
    filename = 'HW3TestCam2.jpg'
    #saves file
    cv2.imwrite(filename,g)
    return g
```

Code for PSNR and MSE Calculation

```python
#https://www.geeksforgeeks.org/python-peak-signal-to-noise-ratio-psnr/
def PSNR(original, compressed):
    mse = np.mean((original - compressed) ** 2)
    if(mse == 0):  # MSE is zero means no noise is present in the signal .
                   # Therefore PSNR have no importance.
        return 100
    max_pixel = 255.0
    psnr = 20 * log10(max_pixel / sqrt(mse))
    return psnr, mse
```

Code for S&P Noise

```python
def AddSaltandPepper(opt):

    imagepath = 'C:/Users/caden/CompVision/HW3Test2.jpg'
    img = cv2.imread(imagepath, 0)
    img = img/255
    # blank image
    x,y = img.shape
    g = np.zeros((x,y), dtype=np.float32)
    # salt and pepper amount
    if(opt == 0):
        pepper = .1
        salt = 1
    if(opt == 1):
        pepper = 0
        salt = .9
    if(opt == 2):
        pepper = 0.1
        salt = 0.9
    # create salt and peper noise image
    for i in range(x):
        for j in range(y):
            rdn = np.random.random()
            if rdn < pepper:
                g[i][j] = 0
            elif rdn > salt:
                g[i][j] = 1
            else:
                g[i][j] = img[i][j]
    #used for final image formating
    g = (g*255).round().astype(np.uint8)
    cv2.imshow('image with noise', g)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    plt.hist(g.flat)
    plt.xlim([0,255]); plt.ylim([0,60000])
    plt.xlabel('pixel value'); plt.ylabel('frequency')
    plt.show()
    return g
```

Code for DWT

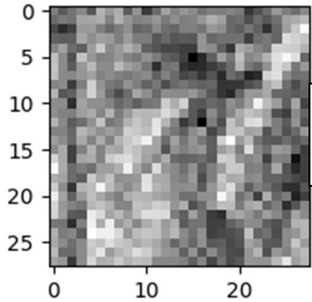
```python
def Wavelet(img):
    #pywt.wavedec2(img, 'db1', mode='symmetric', level=3)
    #https://pywavelets.readthedocs.io/en/latest/ref/dwt-coefficient-handling.html

    #cam = pywt.data.camera()
    coeffs = pywt.wavedecn(img, wavelet='db2', level=3)
    arr, coeff_slices = pywt.coeffs_to_array(coeffs)
    plt.figure(figsize=(8, 8))
    plt.imshow(np.abs(arr), cmap='gray')
    plt.title("Flattened Wavelet Coefficients (3-Level, db2)")
    plt.axis('off')
    plt.show()
    os.chdir('C:/Users/caden/CompVision')
   # filename = 'HW3TestWave.jpg'
    g = arr
    #cv2.imwrite(filename,g)
```

Imported Libraries

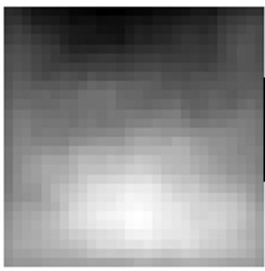
```python
# libraries
import cv2
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats.kde import gaussian_kde
import matplotlib.pyplot as plt
import os
from math import log10, sqrt
import pywt
```

2)

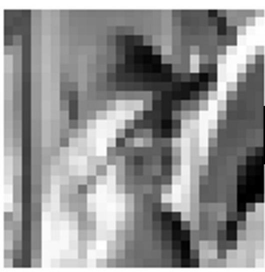
Hard Thresholding



Gaussian Noise Applied



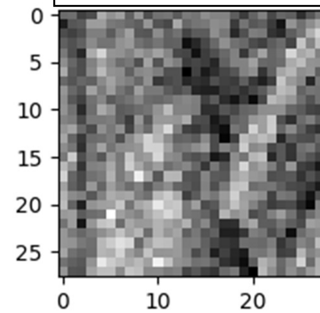
Denoised Image



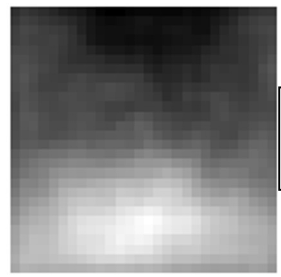
Original 28x28

PSNR value is 64.45 dB

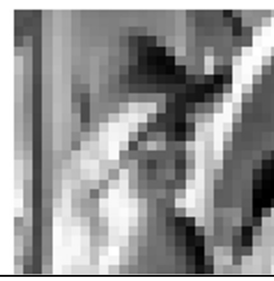
Soft Thresholding



Gaussian Noise Applied



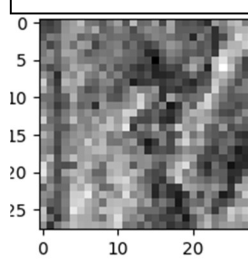
Denoised Image



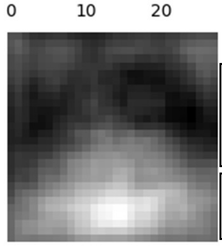
Original 28x28

PSNR value is 64.55 dB

Firm/Semi-Soft Thresholding



Gaussian Noise Applied



Denoised Image

PSNR value is 64.56 dB

Original 28x28

Results and Discussion:

For the neural network I used a Pytorch autoencoder framework that was originally designed to denoise images of numbers. I changed the dataset used for training to a dataset that was composed of random objects. The dataset was resized to 28x28 images, and I will explain the reasoning behind that later. Using this dataset the NN was trained on 128 images. After training I used the lena image to test the network. The lena test image was also resized to 28x28. The images were all resized due to computational complexity. Originally I had the network set up to use 256x256 sized images, but it repeatedly crashed my computer. I resized to smaller images to aid in computation and training time. As seen above the Soft thresholding technique produced a slightly higher PSNR value. Overall the results seem very similar visually. None of the denoised images resulted in a quality image that could be distinguished as a denoised version of the lena image. The network overall performed very poorly at denoising the image, and I believe this was due to a few reasons. The first reason was the training data. The training data used was a dataset that was resized from its original size and was a collection of random objects, not human faces. Additionally, the test image was resized and the gaussian noise that was added had a much larger impact at this size making the image very difficult to make out visually. Finally the network was only trained on 10 epochs. Using a better dataset, larger images, and longer training times would increase the effectiveness of this neural network. I also tested an additional thresholding technique called firm/semisoft thresholding and the results were rather similar, but the image was marginally better.

Caden Thompson Homework 3 4/17/25

Retrieving dataset and batch size for training and testing. This code also resizes the images to 28x28

```python
opt = 0
if(opt == 0):
    threshold = 'soft'
elif(opt == 1):
    threshold = 'hard'
device = 'cuda' if torch.cuda.is_available() else 'cpu'
#https://www.geeksforgeeks.org/denoising-autoencoders-in-machine-learning/
transform = transforms.Compose([
    transforms.Resize((28, 28)),
    transforms.Grayscale(num_output_channels=1),
    transforms.ToTensor(),
])
mnist_dataset_train = datasets.CIFAR10(
    root='./data',  train=True, download=True, transform=transform)
mnist_dataset_test = datasets.CIFAR10(
     root='./data',  train=False, download=True, transform=transform)
batch_size = 128
train_loader = torch.utils.data.DataLoader(
    mnist_dataset_train, batch_size=batch_size, shuffle=True)
imagepath = 'C:/Users/caden/Downloads/standard_test_images/standard_test_images/lena_gray_256.tif'
data1 = Image.open(imagepath)
resizetransfrom = transforms.Resize((28, 28))
tensor_data = transform(resizetransfrom(data1))
test_loader = torch.utils.data.DataLoader(
    mnist_dataset_test, batch_size=1, shuffle=False)
```

Runs the neural network and outputs the final result.

```python
epochs = 10
model = DAE().to(device)
optimizer = optim.Adam(model.parameters(), lr=1e-2)
criterion = nn.MSELoss()
for epoch in range(1, epochs + 1):
    train(epoch, model, train_loader, optimizer, True)
tensor_data = tensor_data.to(device)
optimizer.zero_grad()
data_noise = 0.1 * torch.randn(tensor_data.shape).to(device)
data_noise = tensor_data + data_noise
recon_batch = model(data_noise.to(device))
plt.figure(figsize=(20, 12))
for i in range(1):
    plt.subplot(3, 5, 1+i)
    plt.imshow(data_noise[i, :, :].view(
        28, 28).detach().numpy(), cmap='binary')
    plt.subplot(3, 5, 6+i)
    plt.imshow(recon_batch[i, :].view(28, 28).detach().numpy(), cmap='binary')
    plt.axis('off')
    plt.subplot(3, 5, 11+i)
    plt.imshow(tensor_data[i, :, :].view(28, 28).detach().numpy(), cmap='binary')
    plt.axis('off')
plt.show()
value, mse = PSNR(tensor_data[i, :, :].view(28, 28).detach().numpy(), recon_batch[i, :].view(28, 28).detach().numpy())
print(f"PSNR value is {value} dB")
```

Neural Network Moduel: This code outlines the autoencoder and implements the soft and hard thresholding for the activation function in the testing mode. This code also trains the model on the dataset.

```python
class DAE(nn.Module):
    def __init__(self):
        super().__init__()

        self.fc1 = nn.Linear(784, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 128)

        self.fc4 = nn.Linear(128, 256)
        self.fc5 = nn.Linear(256, 512)
        self.fc6 = nn.Linear(512, 784)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()
    def encode(self, x):
        h1 = self.relu(self.fc1(x))
        h2 = self.relu(self.fc2(h1))
        out = self.relu(self.fc3(h2))
        if not self.training:
            out_np = out.detach().cpu().numpy()
            for i in range(out_np.shape[0]):
                out_np[i] = pywt.threshold(out_np[i], 0.5, mode=threshold)
            out = torch.from_numpy(out_np).to(out.device).float()
        return out
    def decode(self, z):
        h4 = self.relu(self.fc4(z))
        h5 = self.relu(self.fc5(h4))
        out = self.fc6(h5)
        if not self.training:
            out_np = out.detach().cpu().numpy()
            for i in range(out_np.shape[0]):
                out_np[i] = pywt.threshold(out_np[i], 0.5, mode=threshold)
            out = torch.from_numpy(out_np).to(out.device).float()
        return self.sigmoid(out)
    def forward(self, x):
        q = self.encode(x.view(-1, 784))
        return self.decode(q)
def train(epoch, model, train_loader, optimizer,  cuda=True):
    model.train()
    train_loss = 0
    for batch_idx, (data, _) in enumerate(train_loader):
        data.to(device)
        optimizer.zero_grad()
        data_noise = torch.randn(data.shape).to(device)
        data_noise = data + data_noise
        recon_batch = model(data_noise.to(device))
        loss = criterion(recon_batch, data.view(data.size(0), -1).to(device))
        loss.backward()
        train_loss += loss.item() * len(data)
        optimizer.step()
        if batch_idx % 100 == 0:
            print('Train Epoch: {} [{}/{} ({:.0f}%)]\tLoss: {:.6f}'.format(epoch, batch_idx * len(data), len(train_loader.dataset),
                                                                           100. * batch_idx /
                                                                           len(train_loader),
                                                                           loss.item()))
    print('====&gt; Epoch: {} Average loss: {:.4f}'.format(
        epoch, train_loss / len(train_loader.dataset)))
```

Imported Libraries

```python
import torch.utils.data
from torchvision import datasets, transforms
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from torch import nn, optim
from torch.utils.data import DataLoader
from PIL import Image
import torch.nn.functional as F
import cv2

from scipy.stats.kde import gaussian_kde

import os
from math import log10, sqrt
import pywt
```

References:

https://www.geeksforgeeks.org/python-peak-signal-to-noise-ratio-psnr/

https://www.geeksforgeeks.org/denoising-autoencoders-in-machine-learning/

https://github.com/behnamasadi/PythonTutorial/blob/master/signal_system/white_noise_gaussian_noise.ipynb

https://pywavelets.readthedocs.io/en/latest/ref/dwt-coefficient-handling.html

https://pywavelets.readthedocs.io/en/latest/ref/thresholding-functions.html