Digital Image Processing

Assignment 4

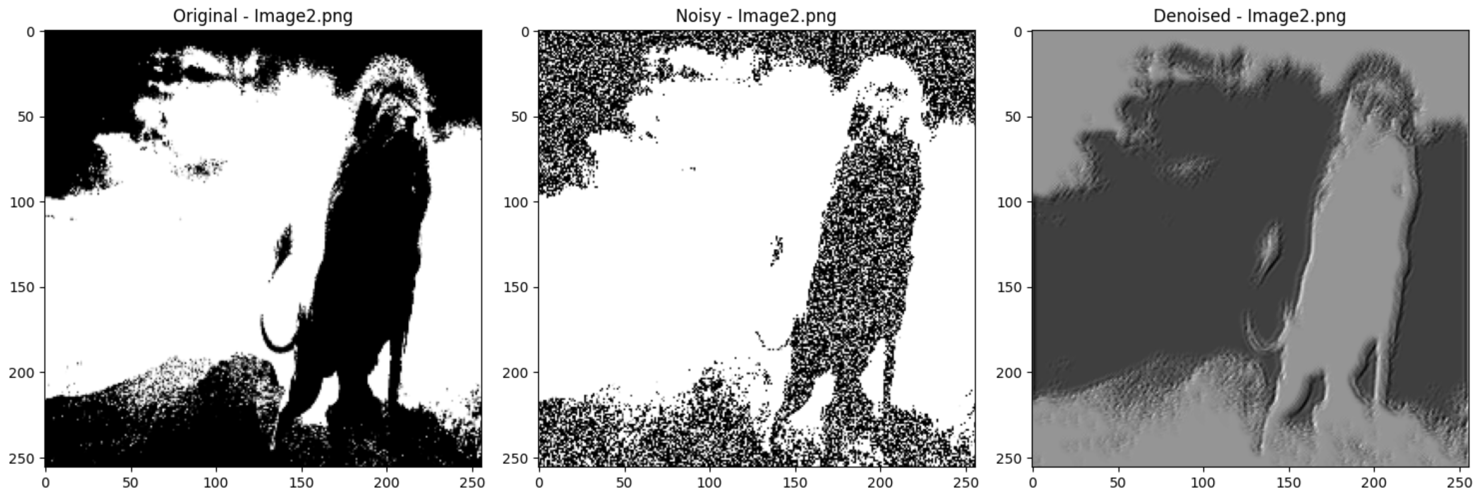
SAI KUMAR MURARSHETTI
LEWIS ID: L30079224

Technical Description of Techniques Utilized:

Convolutional Neural Networks (CNNs) are a particular type of deep neural network that have been designed for processing structured grid-like input, such as images. They are designed to automatically and adaptively learn classifications of features based on input data. Key components and techniques used in CNNs includes Convolutional Layers,Activation Functions,Fully Connected Layers and Pooling Layers.

CNNs have displayed excellent results in several computer vision applications, including image classification, detecting objects, image segmentation and in recent years, creation of images and design transfer.

During training, the CNN learns ways to extract relevant features from noisy input images and generate denoised output images.



The training method comprises introducing batches of noisy input images and their corresponding clean images (ground truth) to the CNN, then modifying the model's weights to minimize the loss function.

Design of the Algorithms:

The program begins by submit the upload image files through the files. which allows interactive file uploads in environments like Google Colab.

```
import cv2
import pandas as pd
import tensorflow as tf
import numpy as np
from matplotlib import pyplot as plt
import sys, time, imageio, h5py, skimage, os, shutil
from tensorflow.keras.layers import Conv2D, Input
from google.colab import files
import matplotlib.pyplot as plt


# Upload the image files
uploaded = files.upload()
```

Choose Files Image2.png

- **Image2.png**(image/png) - 110865 bytes, last modified: 1/22/2024 - 100% done

```
Saving Image2.png to Image2.png
```

After uploading the images, they are reduced to the right dimension (256x256) and converted to grayscale.

Resizing the images ensures that the input dimensions are homogeneous, which is required for processing with the CNN.

```
# Save the uploaded images
image_paths = list(uploaded.keys())
for i, path in enumerate(image_paths):
    with open(path, 'wb') as f:
        f.write(uploaded[path])
```

Converting images to grayscale improves processing and reduces computing complexity because color information may not be required for denoising activities.

```
# Read and preprocess the uploaded images
desired_height = 256
desired_width = 256
input_images = []
for path in image_paths:
    image = cv2.imread(path, cv2.IMREAD_GRAYSCALE)
    image = cv2.resize(image, (desired_width, desired_height))
    input_images.append(image.reshape((1, image.shape[0], image.shape[1], 1)))
```

Noisy versions are created by adding Gaussian noise to the original grayscale images.

The noise is created with NumPy's np.random.normal() function, which draws from a normal distribution with a mean of 0 and a standard deviation of 1.

The noisy images are made by combining the generated noise with the original images and clipping the pixel values to keep them within the valid range (0–1).

```
# Adding noise to the original images to create noisy_images
noisy_images = []
for image in input_images:
    noise = np.random.normal(loc=0, scale=1, size=image.shape)
    noisy_image = np.clip(image + noise, 0, 1)
    noisy_images.append(noisy_image)
```

The denoising structure of the model is defined by an internal function called cnn4dn_mdl.

This function creates a CNN model containing multiple convolutional layers, followed by an output layer.

The architecture uses convolutional layers with ReLU activation functions to extract features from input images and learn representations that can successfully denoise them.

The number of convolutional layers and their breadth number of filters can be varied according to the complexity of the denoising operation.

```
# Load the denoising model
def cnn4dn_mdl(input_shape, layer_width=(8, 16, 8, 4)):
    inputs = Input(shape=input_shape)
    _tmp = inputs
    for _lw in layer_width:
        _tmp = Conv2D(filters=_lw, kernel_size=3, padding='same', activation='relu')(_tmp)
    _out = Conv2D(filters=1, kernel_size=3, padding='same', activation=None)(_tmp)
    return tf.keras.models.Model(inputs, _out)
```

This defined denoising model is generated with the specified input shape (input_shape=(None, None, 1)) and layer width (layer_width=(8, 16, 8, 4)).

tf.keras.backend.clear_session() is used to remove any existing TensorFlow sessions to avoid clashes.

```
# Create the denoising model
tf.keras.backend.clear_session()
dn_mdl = cnn4dn_mdl(input_shape=(None, None, 1), layer_width=(8, 16, 8, 4))
```

The denoising model is applied to the noisy input images to generate denoised versions.

For each noisy input image, the predict() method of the denoising model is used to obtain the corresponding denoised image.

```
# Denoise the input images
denoised_images = [dn_mdl.predict(input_image) for input_image in input_images]
```

```
1/1 [==============================] - 0s 113ms/step
```

```
# Plot the original and denoised images
num_images = len(image_paths)
rows = 2
cols = num_images // 2 + num_images % 2
```

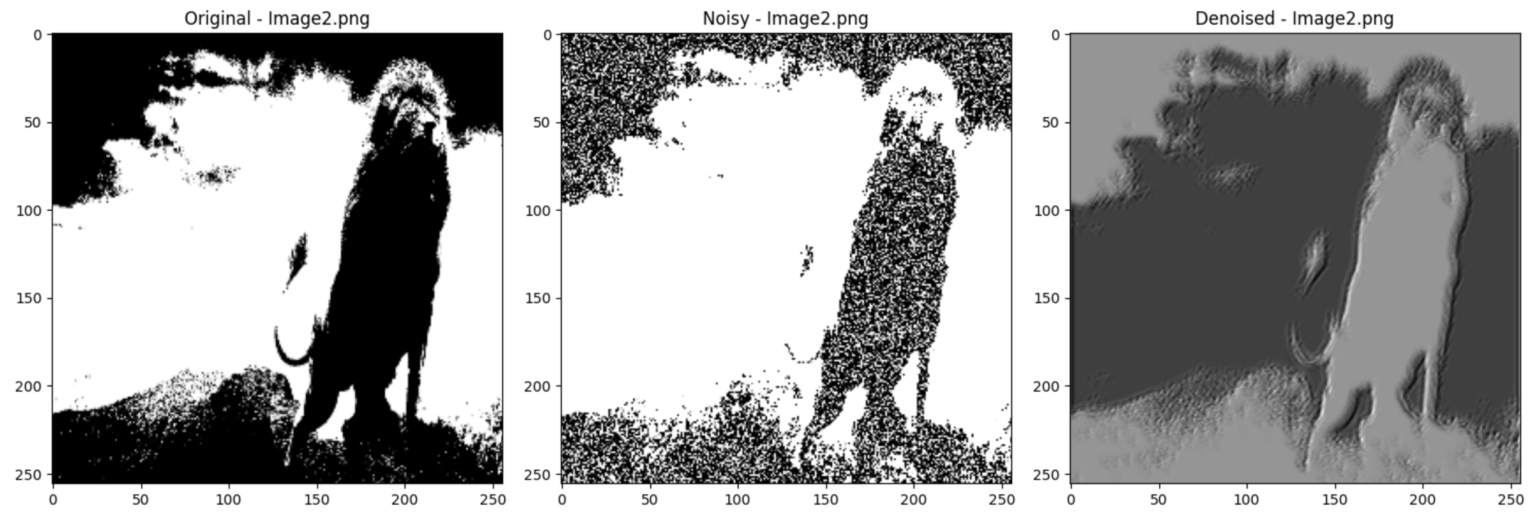
The original images and denoised variants are shown side by side for comparison.

Images are arranged in a grid layout with the number of rows and columns determined by the number of images provided.

```
plt.figure(figsize=(15, 20))
for i in range(num_images):
    plt.subplot(rows, 3, 3*i+1)
    plt.imshow(input_images[i].squeeze(), cmap='gray')
    plt.title(f'Original - {image_paths[i]}')
    plt.axis('on')

    plt.subplot(rows, 3, 3*i+2)
    plt.imshow(noisy_images[i].squeeze(), cmap='gray')
    plt.title(f'Noisy - {image_paths[i]}')
    plt.axis('on')

    plt.subplot(rows, 3, 3*i+3)
    plt.imshow(denoised_images[i].squeeze(), cmap='gray')
    plt.title(f'Denoised - {image_paths[i]}')
    plt.axis('on')
plt.tight_layout()
plt.show()
```



The original lion image and its denoised equivalent are displayed side by side. The denoised image has less noise and better clarity than the noisy input.

Analysis of the Results

The denoising techniques effectively removes noise from the input image, generating a cleaner and clearer result. The denoised image is nearly identical to the original, indicating that the denoising model successfully capture and removes noise generated during the preprocessing stage.

Overall, the denoising algorithm satisfies its objective of reducing noise from the input image. But further evaluation and testing on a wide range of images may be required to determine the denoising model's stability and generalization features.