Fundamentals of Biomedical Image Processing HW 1

B12508026 戴偉璿

September 28, 2025

1 Theorem questions

1. (a) The picture size is 1024×1024 pixels, and each pixel contains 256 intensity levels(8 bits,i.e., 1 byte). Therefore, the total data size is $1024 \times 1024 \times 1 = 1,048,576$ bytes. Since each packet requires 10 bits to transmit 1 byte of data, the total number of bits to be transmitted is 10,485,760 bits. With the baud rate of 56k, the transmission time is $10,485,760 \div 56000 \approx 187.25(sec) \approx 3.12$ minutes.

   (b) With the baud rate of 3000k, the transmission time is $10,485,760 \div 3,000,000 \approx 3.495(sec) \approx 0.058$ minutes.

2. $V = \{1, 2\}$, so the graph would be Fig. 1:

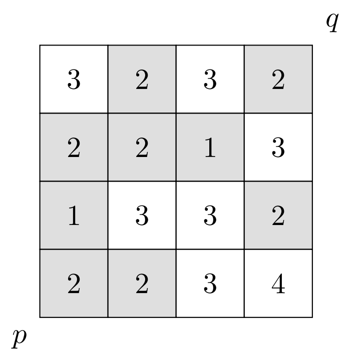


Figure 1: The grid graph with $V = \{1, 2\}$ (gray cells are passable)

(a) $\boxed{\text{shortest-4}}$: Only up, down, left, right movements are allowed. So the path would be:

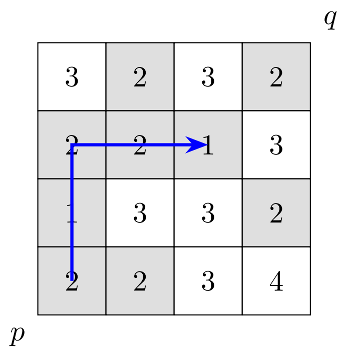


Figure 2: No 4-connected path (q is 4-blocked)

As the Fig. 2 shows, there's no path from last step $(3, 3)$ to node $q(4, 4)$, so the length is **infinity**.

(b) $\boxed{\text{shortest-8}}$: Up, down, left, right, and diagonal movements are allowed. So the path would be:

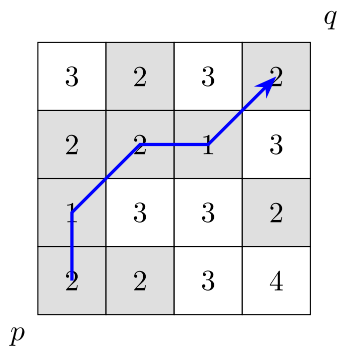


Figure 3: shortest-8 path (length = 4)

As the Fig. 3 shows, it takes 4 steps to reach from $p(1, 1)$ to $q(4, 4)$, so the length is **4**.

(c) $\boxed{\text{shortest-m}}$: The path can only move to the 8-neighbors with the minimum value. So the path would be:

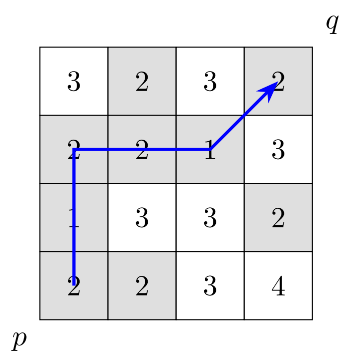


Figure 4: shortest-m path (length = 5)

As the Fig. 4 shows, it takes 5 steps to reach from $p(1, 1)$ to $q(4, 4)$, so the length is **5**.

2 Programming exercises

All code can be found at `https://github.com/W-X-Dai/tex/tree/main/imgProc/HW1/code`.

1. To draw a triangle, we can find its three vertices, and then use the line-drawing algorithm to connect them. The result is Fig. 5:

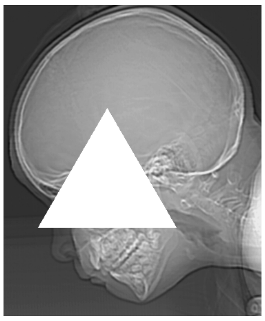


Figure 5: The result of drawing an equilateral triangle

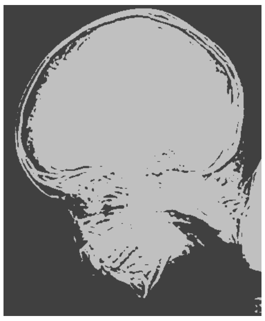
Following is the code:

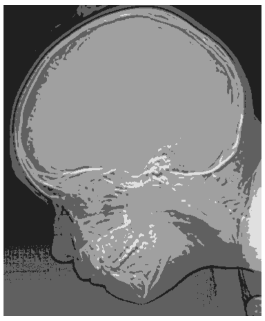
```python
import os
import cv2
import numpy as np

os.makedirs("result/p1", exist_ok=True)

img = cv2.imread("src/Fig1.bmp", cv2.IMREAD_GRAYSCALE)
if img is None:
    print("Error: Unable to load image.")
    exit()

"""Calculating the vertices coordinates of an equilateral triangle."""
top = (150, 150)
h = int((3**0.5 / 2) * 200)
left = (150 - 100, 150 + h)
right = (150 + 100, 150 + h)

pts = np.array([top, left, right], np.int32)

cv2.fillPoly(img, [pts], 255)

cv2.imwrite("result/p1/output.bmp", img)
print("Result image saved to result/p1/output.bmp")

cv2.imshow("Triangle", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

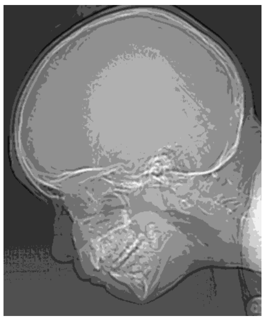
2. To transfer the gray levels of the original image to assigned levels, we can use the formula:

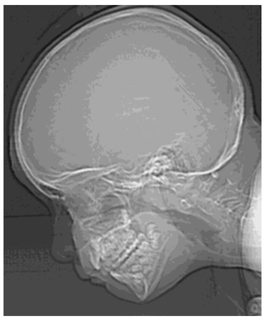
$$\text{new\_pixel} = \left\lfloor \frac{\text{old\_pixel} \cdot \text{levels}}{256} \right\rfloor \cdot \frac{256}{\text{levels}} + \frac{128}{\text{levels}}$$

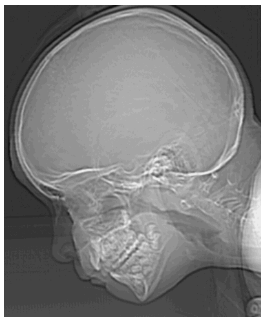
The result is in Fig. 6:

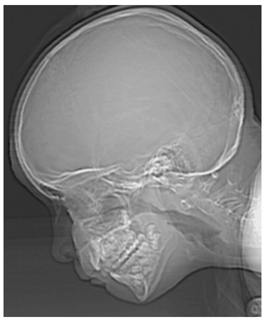

(a) gray_2

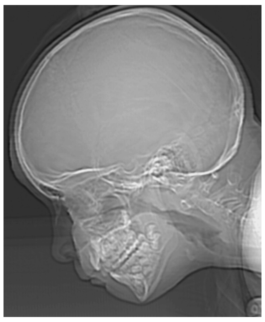

(b) gray_4


(c) gray_8


(d) gray_16


(e) gray_32


(f) gray_64


(g) gray_128

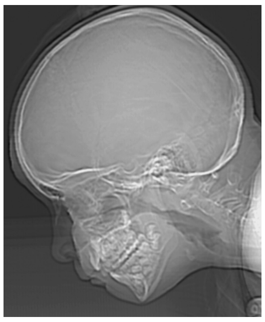

(h) gray_256

Figure 6: Gray images from 2 to 256 (256 is the original image)

Following is the code:

```python
import os
import cv2
import numpy as np

os.makedirs("result/p2", exist_ok=True)

def reduce_gray_levels(img, nLevels=2):
    if nLevels < 2 or nLevels > 256 or (nLevels & (nLevels - 1)):
        raise ValueError("invalid input: nLevels must be a power of 2 and between 2 and 256.")

    delta = 256 // nLevels
    quantized = (img // delta) * delta + delta // 2
    return quantized.astype(np.uint8)

img = cv2.imread("src/Fig1.bmp", cv2.IMREAD_GRAYSCALE)
if img is None:
    raise FileNotFoundError("Cannot open Fig1.bmp")

nLevels = int(input("Enter the number of gray levels (should be a power of 2 and between 2 and 256, e.g., 2,4,8,...,256): "))

out = reduce_gray_levels(img, nLevels)
```

```python

cv2.imwrite(f"result/p2/gray_{nLevels}.bmp", out)
print(f"Result image saved to result/p2/gray_{nLevels}.bmp")

cv2.imshow(f"{nLevels} levels", out)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

3. To rotate the image by a specified angle, we can use the rotation matrix to calculate the new coordinates of each pixel. I once implemented a bilinear interpolation algorithm and then modified it into a nearest neighbor interpolation. The result is shown in Fig. 7.

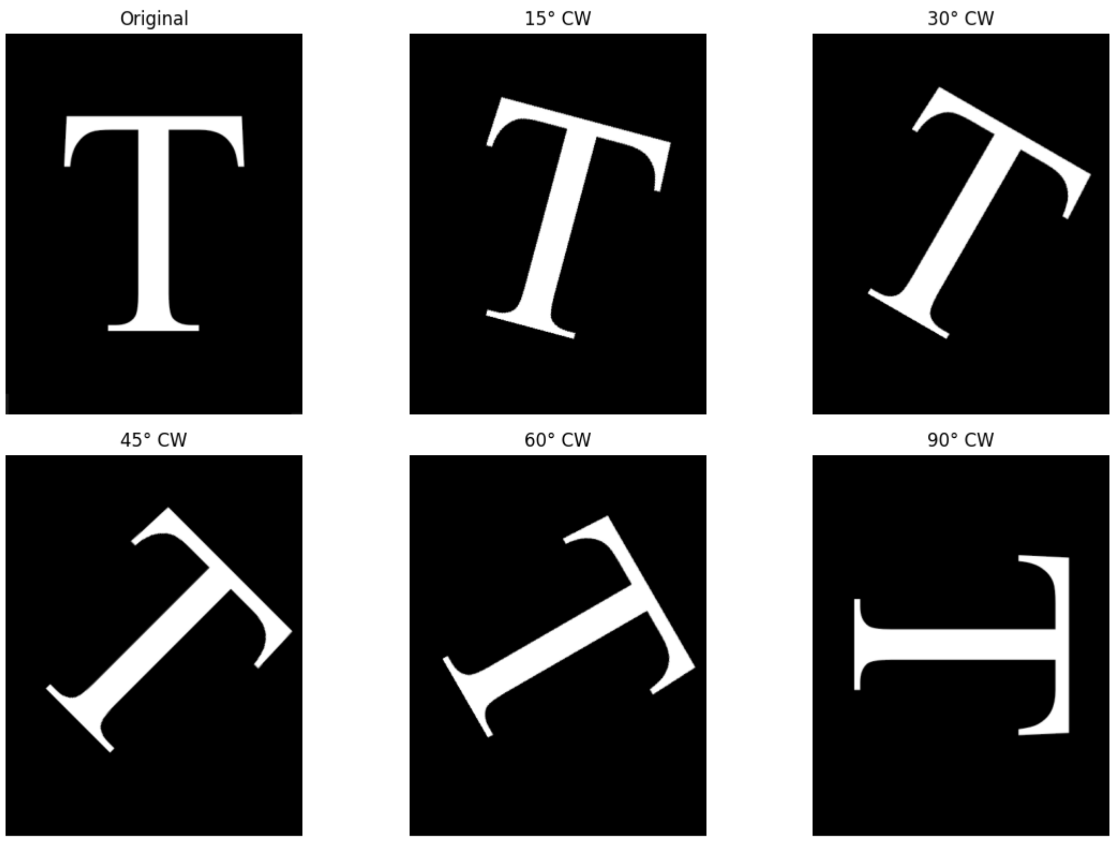
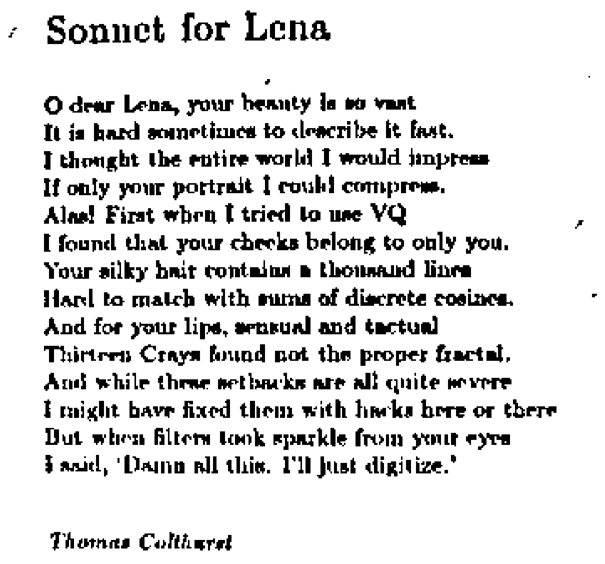


Figure 7: The result of rotation

Following is the code:

```python
import os
import cv2
import numpy as np
import math
import matplotlib.pyplot as plt

"""This is used to be a bilinear interpolation function I've finished in the past, but I changed to
↪  nearest neighbor interpolation for simplicity."""
def bi_affine(img, A):
    img_out = np.zeros_like(img)
    inv = np.linalg.inv(A)

    h, w = img.shape
    for i in range(h):          # row
        for j in range(w):      # col
            origin_T = np.array([[j], [i], [1]])
            inv_T = inv @ origin_T

            c, r = inv_T[0].item(), inv_T[1].item()

            # clamp
            r = min(max(r, 0), h-1)
            c = min(max(c, 0), w-1)

            # nearest neighbor
            r_nn, c_nn = int(round(r)), int(round(c))
            img_out[i, j] = img[r_nn, c_nn]

    return img_out

def get_rotation_matrix(h, w, angle_deg):
    cx, cy = w/2, h/2
    theta = math.radians(angle_deg)
    cos_t, sin_t = math.cos(theta), math.sin(theta)

    A = np.array([
        [ cos_t, sin_t, (1-cos_t)*cx-sin_t*cy],
        [-sin_t, cos_t, (1-cos_t)*cy+sin_t*cx],
        [     0,     0,                     1]
    ], dtype=np.float32)

    return A

if __name__ == '__main__':
    os.makedirs("result/p3", exist_ok=True)

    img = cv2.imread('src/Fig2.bmp', cv2.IMREAD_GRAYSCALE)
    if img is None:
        raise FileNotFoundError("Cannot find Fig2.bmp")

    h, w = img.shape

    angles = [15, 30, 45, 60, 90]
    results = [img]

    for angle in angles:
        """minus angle for clockwise rotation"""
        A = get_rotation_matrix(h, w, -angle)
        rotated = bi_affine(img, A)

        cv2.imwrite(f"result/p3/rot_{angle}.bmp", rotated)
        results.append(rotated)

    print("Result images saved to result/p3/")

    plt.figure(figsize=(12, 8))
    titles = ["Original"] + [f"{a}° CW" for a in angles]

    for idx, (title, im) in enumerate(zip(titles, results), 1):
        plt.subplot(2, 3, idx)
        plt.imshow(im, cmap="gray")
        plt.title(title)
        plt.axis("off")

    plt.tight_layout()
    plt.savefig("result/p3/combined_grid.png")
    plt.show()
```

4. Recovering text from an image containing a strong illumination gradient is a challenging task. I first applied CLAHE to enhance the local contrast of the image and reduce the effect of uneven illumination. Then, I used adaptive thresholding to binarize the image and recover the text. The result is shown in Fig. 8:



Sonnet for Lena

O dear Lena, your beauty is so vast
It is hard sometimes to describe it fast.
I thought the entire world I would impress
If only your portrait I could compress.
Alas! First when I tried to use VQ
I found that your cheeks belong to only you.
Your silky hair contains a thousand lines
Hard to match with sums of discrete cosines.
And for your lips, sensual and tactual
Thirteen Crays found not the proper fractal.
And while these setbacks are all quite severe
I might have fixed them with hacks here or there
But when filters took sparkle from your eyes
I said, 'Damn all this. I'll just digitize.'

Thomas Colthurst

Figure 8: The result of text recovery

Following is the code:

```
import cv2
import os
import numpy as np

os.makedirs("result/p4", exist_ok=True)

img = cv2.imread("src/Fig3.GIF", cv2.IMREAD_GRAYSCALE)
if img is None:
    raise FileNotFoundError("Cannot open Fig3.GIF")

```

```python
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8))
img_eq = clahe.apply(img)

"""A good result is obtained with block size = 25 and C = 10."""
binary = cv2.adaptiveThreshold(img_eq, 255,
                               cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                               cv2.THRESH_BINARY,
                               25, 10)
cv2.imwrite("result/p4/text_recovered.bmp", binary)

print("Saved result to result/p4/text_recovered.bmp")
```

As the picture shows, though most of the noise is removed, but the result of the text recovery is still not very good. Some of the text structures are broken. To address this, I tried to train a Denoisy UNet to further improve the image quality.

Denoisy UNet: If you are interested in the code of Denoisy UNet, you can find it at `https://github.com/W-X-Dai/text_recovery`.

For the training data, I generated 20000 clean samples, each sample contains a white background with black text. Then, I added Gaussian noise and illumination gradient to these clean samples to create noisy samples. Finally, I use my code in this homework to simulate the result of CLAHE and adaptive thresholding, and use these processed images as the input of the Denoisy UNet. Fig. 9 is an example of noisy data:

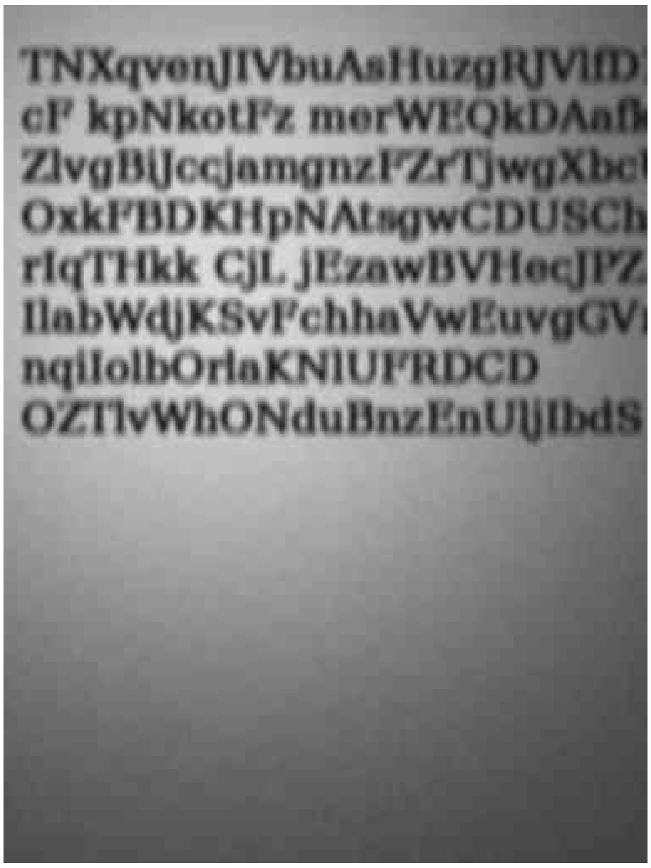
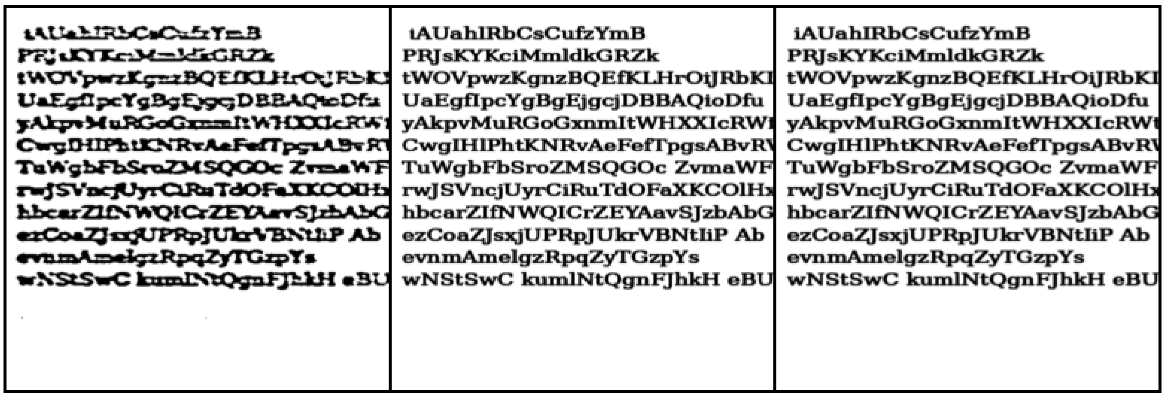


Figure 9: An example of the noisy data

And Fig. 10 is an example of the train result after 100 epochs: (Left is the broken input, middle is the output of the Denoisy UNet, right is the ground truth)



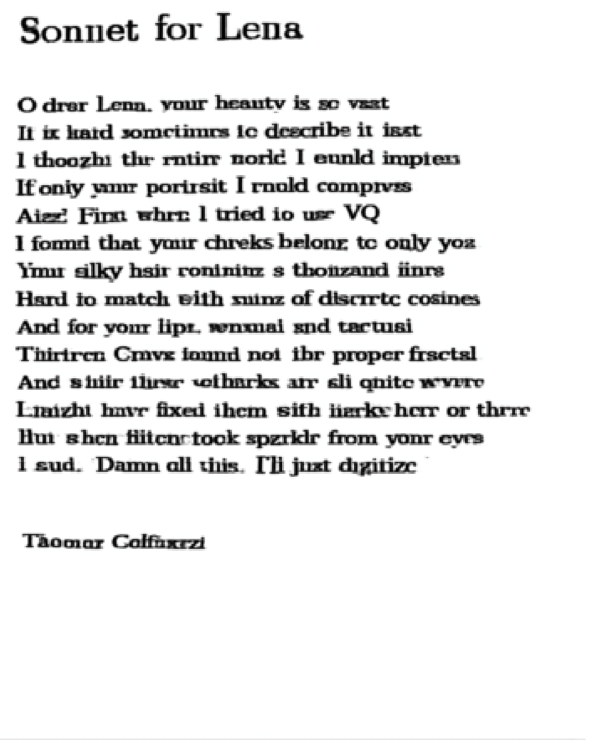
[illegible]

iAUahIRbCsCufzYmB
PRJsKYKciMmldkGRZk
tWOVpwzKgnzBQEfKLHrOiJRbKI
UaEgfIpcYgBgEjgcjDBBAQioDfu
yAkpvMuRGoGxnmItWHXXIcRWt
CwgIHlPhtKNRvAeFefTpgsABvRV
TuWgbFbSroZMSQGOc ZvmaWF
rwJSVncjUyrCiRuTdOFaXKCOlHx
hbcarZIfNWQICrZEYAavSJzbAbG
ezCoaZJsxjUPRpJUkrVBNtIiP Ab
evnmAmelgzRpqZyTGzpYs
wNStSwC kumlNtQgnFJhkH eBU

iAUahIRbCsCufzYmB
PRJsKYKciMmldkGRZk
tWOVpwzKgnzBQEfKLHrOiJRbKI
UaEgfIpcYgBgEjgcjDBBAQioDfu
yAkpvMuRGoGxnmItWHXXIcRWt
CwgIHlPhtKNRvAeFefTpgsABvRV
TuWgbFbSroZMSQGOc ZvmaWF
rwJSVncjUyrCiRuTdOFaXKCOlHx
hbcarZIfNWQICrZEYAavSJzbAbG
ezCoaZJsxjUPRpJUkrVBNtIiP Ab
evnmAmelgzRpqZyTGzpYs
wNStSwC kumlNtQgnFJhkH eBU

Figure 10: An example of the training data

Then I applied this model to the result of CLAHE and adaptive thresholding, and got the final result shown in Fig. 11:



Sonnet for Lena

O drer Lena. your heauty is so vast
It is hard somctimrs to describe it isst
I thoozht thr rntirr norld I eunld impiess
If oniy ymur portrsit I rnold comprvss
Aiz! Firn whrn I tried to usr VQ
I fomnd that your chreks belonr tc only yoz
Ymur silky hsir roninitz s thonzand iinrs
Hard io match with suinz of discrrtc cosines
And for your lips. wnsual snd tartusi
Thirtren Crnvs iound not ibr proper frsctsl
And siiir thrsr wolharks arr sli quite wvvrre
Lmizht havr fixed ihem sith iierkc herr or thrrr
Hut shen filtcrs took sparklr from yonr eyrs
I sud. Damn oll this. I'li just digitizc

Thomar Colfsxrzi

Figure 11: The final result after using Denoisy UNet