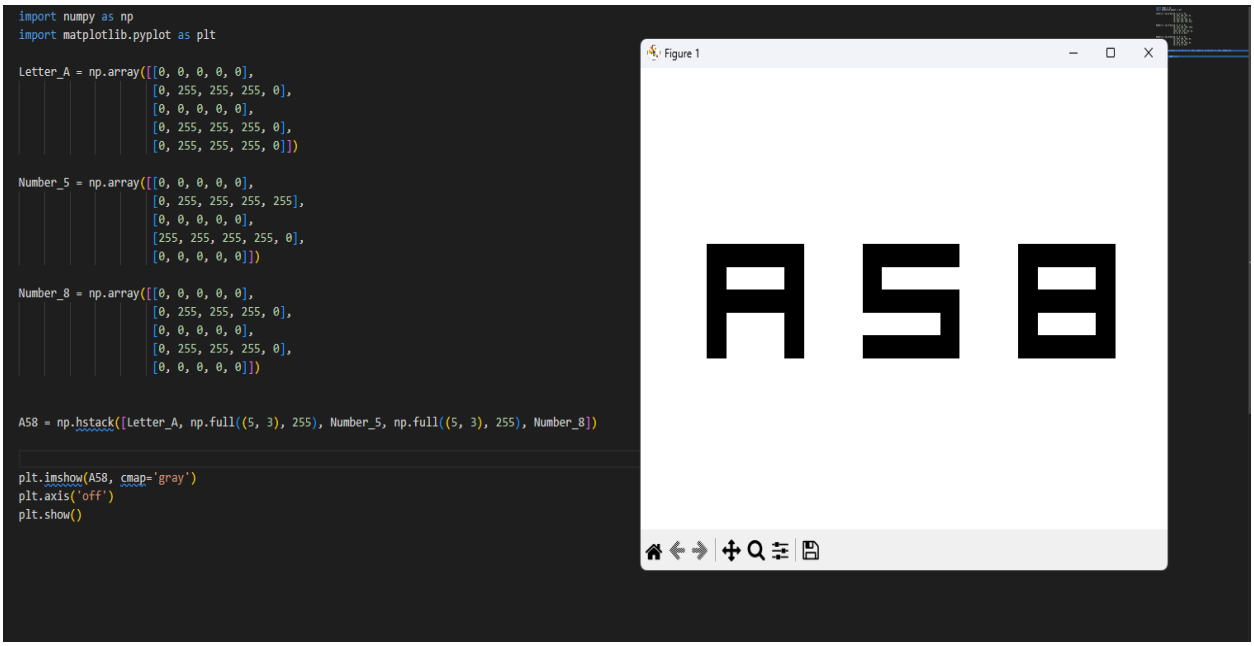
Task 01 code output:



Task 02 code:

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

source_folder = r"D:\CSE463 Homework 01\Dog Images"
destination_folder = r"D:\CSE463 Homework 01\Transformed Dog Images"

def process_image(image_path, save_path):
    img = cv2.imread(image_path)


    cropped_image = img[30:240, 10:277]
    cv2.imwrite(os.path.join(save_path, "cropped_" + os.path.basename(image_path)), cropped_image)

    flipped_image = cv2.flip(img, 0)
    cv2.imwrite(os.path.join(save_path, "flipped_" + os.path.basename(image_path)), flipped_image)

    rows, cols = img.shape[:2]
    M = cv2.getRotationMatrix2D((cols / 2, rows / 2), 90, 1)  # Rotate 90 degrees
    rotated_image = cv2.warpAffine(img, M, (cols, rows))
    cv2.imwrite(os.path.join(save_path, "rotated_" + os.path.basename(image_path)), rotated_image)

    width = int(img.shape[1] * 0.5)
    height = int(img.shape[0] * 0.5)
    dim = (width, height)
    resized_image = cv2.resize(img, dim, interpolation=cv2.INTER_AREA)  # Resize to half
    cv2.imwrite(os.path.join(save_path, "resized_" + os.path.basename(image_path)), resized_image)

    shift_x = 50
    shift_y = 30
    translation_matrix = np.float32([[1, 0, shift_x], [0, 1, shift_y]])
    translated_image = cv2.warpAffine(img, translation_matrix, (img.shape[1], img.shape[0]))
    cv2.imwrite(os.path.join(save_path, "translated_" + os.path.basename(image_path)), translated_image)

for filename in os.listdir(source_folder):
    if filename.endswith(".jpg"):
        file_path = os.path.join(source_folder, filename)
        process_image(file_path, destination_folder)
        print(f"Processed {filename}")

#For noise pictures and histogram of the noisy images
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt

source_folder = r"D:\CSE463 Homework 01\Dog Images"
noisy_folder = r"D:\CSE463 Homework 01\Noisy Dog Images"

def add_salt_and_pepper_noise(image, prob=0.05):
    """Add salt-and-pepper noise to an image."""
    noisy_image = np.copy(image)

    num_salt = np.ceil(prob * image.size * 0.5).astype(int)
    num_pepper = np.ceil(prob * image.size * 0.5).astype(int)

    # Apply salt noise (white pixels)
    coords = [np.random.randint(0, i - 1, num_salt) for i in image.shape]
    noisy_image[coords[0], coords[1]] = 255

    # Apply pepper noise (black pixels)
    coords = [np.random.randint(0, i - 1, num_pepper) for i in image.shape]
    noisy_image[coords[0], coords[1]] = 0

    return noisy_image

def process_image(image_path, noisy_path):
    img = cv2.imread(image_path)
    noisy_image = add_salt_and_pepper_noise(img)
    cv2.imwrite(os.path.join(noisy_path, "noisy_" + os.path.basename(image_path)), noisy_image)
    plt.figure(figsize=(10, 5))

    plt.subplot(1, 2, 1)
    plt.title("Original Image")
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.title("Noisy Image")
    plt.imshow(cv2.cvtColor(noisy_image, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()

    # Plot histogram of noisy image
    plt.figure(figsize=(10, 5))
    plt.hist(noisy_image.ravel(), bins=256, color='gray', alpha=0.7)
    plt.title("Histogram of Noisy Image")
    plt.xlabel("Pixel Intensity")
    plt.ylabel("Frequency")
    plt.xlim([0, 255])
    plt.grid()
    plt.show()

for filename in os.listdir(source_folder):
    if filename.endswith(".jpg"):
        file_path = os.path.join(source_folder, filename)
        process_image(file_path, noisy_folder)
        print(f"Processed {filename}")
```

Task 03:

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

image_paths = [
    r'D:\CSE463 Homework 01\Urban Images\urban 1.jpeg',
    r'D:\CSE463 Homework 01\Urban Images\urban 2.jpeg',
    r'D:\CSE463 Homework 01\Urban Images\urban 3.jpeg',
    r'D:\CSE463 Homework 01\Urban Images\urban 4.jpeg',
    r'D:\CSE463 Homework 01\Urban Images\urban 5.jpeg'
]

output_folder = r'D:\CSE463 Homework 01\Urban Images\Output'
for idx, path in enumerate(image_paths):
    img = cv2.imread(path)
    mean = 0
    std_dev = 25
    noise = np.random.normal(mean, std_dev, img.shape).astype(np.uint8)
    noisy_image = cv2.add(img, noise)

    cv2.imwrite(os.path.join(output_folder, f'original_image_{idx + 1}.jpeg'), img)
    cv2.imwrite(os.path.join(output_folder, f'noisy_image_{idx + 1}.jpeg'), noisy_image)


    plt.figure(figsize=(10, 5))
    plt.subplot(1, 2, 1)
    plt.title(f'Original Image {idx + 1}')
    plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
    plt.axis('off')

    plt.subplot(1, 2, 2)
    plt.title(f'Noisy Image {idx + 1}')
    plt.imshow(cv2.cvtColor(noisy_image, cv2.COLOR_BGR2RGB))
    plt.axis('off')

    plt.show()

    #histogram
    plt.figure(figsize=(5, 5))
    plt.hist(noisy_image.ravel(), bins=256, color='red', alpha=0.5, label='Noisy Image')
    plt.title(f'Histogram of Noisy Image {idx + 1}')
    plt.xlabel('Pixel Value')
    plt.ylabel('Frequency')
    plt.legend()
    plt.show()


#For blending the images

base_image = cv2.imread(image_paths[0])
height, width, channels = base_image.shape
accum_image = np.zeros((height, width, channels), dtype=np.float32)

for path in image_paths:
    img = cv2.imread(path)
    if img.shape != (height, width, channels):
        img = cv2.resize(img, (width, height))
    accum_image += img.astype(np.float32)


blended_image = (accum_image / len(image_paths)).astype(np.uint8)


cv2.imwrite("blended_image.jpg", blended_image)
plt.imshow(cv2.cvtColor(blended_image, cv2.COLOR_BGR2RGB))
```