Digital project functions

Upload an image. (done)

def upload\_image(self):

        file\_path = tk.filedialog.askopenfilename(filetypes=[("Image files", "\*.jpg;\*.jpeg;\*.png;\*.svg")])

        if file\_path:

            self.image = Image.open(file\_path)

            self.image = self.image.resize((400, 400))  # for display

            self.display\_image(self.image)

    def display\_image(self, img):

        img = ImageTk.PhotoImage(img)

        self.image\_label.configure(image=img)

        self.image\_label.image = img

calculate its histogram and display it. (done)

def calculate\_histogram(self, img):

        if img.mode == 'RGB':

            r, g, b = img.split()

            histogram\_r = np.array(r.histogram())

            histogram\_g = np.array(g.histogram())

            histogram\_b = np.array(b.histogram())

            histogram = np.array([histogram\_r, histogram\_g, histogram\_b])

        else:

            histogram = np.array(img.histogram())

            histogram = histogram.reshape(1, -1)  # Convert to a 2D array

        return histogram

    def plot\_histogram(self, histogram):

        plt.figure(figsize=(6, 4))

        plt.title('Histogram')

        if histogram.shape[0] == 3:  # For RGB images

            colors = ('Red', 'Green', 'Blue')

            for i, color in enumerate(colors):

                plt.plot(histogram[i], color=color)

        else:  # For grayscale or other image modes

            plt.plot(histogram[0], color='black')

        plt.xlabel('Pixel Value')

        plt.ylabel('Frequency')

        plt.show()

    def display\_histogram(self):

        if self.image:

            histogram = self.calculate\_histogram(self.image)

            self.plot\_histogram(histogram)

apply histogram equalization and display both equalized image and its histogram. (the bar is done but the algorithm not)

    def show\_equalization\_slider(self):

        if self.image:

            self.equalization\_slider = tk.Scale(self.buttons\_frame, from\_=0, to=100, orient=tk.HORIZONTAL, label="Equalization %", command=self.perform\_histogram\_equalization)

            self.equalization\_slider.pack(side=tk.LEFT)

    def perform\_histogram\_equalization(self, value):

        if self.image:

            percentage = int(value)

            # Implement histogram equalization algorithm based on the percentage

            # Apply equalization to the image and display the result

            # here's a placeholder display of the image with the percentage

            equalized\_img = self.equalize\_histogram(self.image, percentage)

            self.display\_image(equalized\_img)

    def equalize\_histogram(self, img, percentage):

        # Implement histogram equalization algorithm based on the percentage

        # Apply equalization to the image and return the equalized image

        # This is a placeholder for the actual implementation

        return img

apply filtering (Sobel, Laplace) + user types parameters and display them (need to fix, the build in function runs good but the real function code does not run: need to add the lab code )

    # (build-in functions for Sobel and Laplace filters)

    def apply\_sobel(self):

        if self.image:

            sobel\_img = self.image.filter(ImageFilter.FIND\_EDGES)

            self.display\_image(sobel\_img)

    def apply\_laplace(self):

        if self.image:

            laplace\_img = self.image.filter(ImageFilter.CONTOUR)

            self.display\_image(laplace\_img)

    # (code need to fix, put the lab code instead of this)

    def apply\_sobel(self):

        if self.image:

            gray\_img = self.image.convert('L')  # Convert image to grayscale

            pixels = np.array(gray\_img)

            rows, cols = pixels.shape

            # Sobel operators for edge detection

            sobel\_x = np.array([[-1, 0, 1], [-2, 0, 2], [-1, 0, 1]])

            sobel\_y = np.array([[-1, -2, -1], [0, 0, 0], [1, 2, 1]])

            # Convolve image with the Sobel operators

            new\_pixels\_x = np.zeros((rows - 2, cols - 2))

            new\_pixels\_y = np.zeros((rows - 2, cols - 2))

            for i in range(rows - 2):

                for j in range(cols - 2):

                    window = pixels[i:i + 3, j:j + 3]

                    new\_pixels\_x[i, j] = np.sum(window \* sobel\_x)

                    new\_pixels\_y[i, j] = np.sum(window \* sobel\_y)

            # Combine x and y gradients to get the magnitude

            magnitude = np.sqrt(new\_pixels\_x\*\*2 + new\_pixels\_y\*\*2)

            magnitude \*= 255.0 / np.max(magnitude)  # Normalize the values

            sobel\_img = Image.fromarray(magnitude.astype(np.uint8))

            self.display\_image(sobel\_img)

    def apply\_laplace(self):

        if self.image:

            gray\_img = self.image.convert('L')  # Convert image to grayscale

            pixels = np.array(gray\_img)

            rows, cols = pixels.shape

            # Laplace operator for edge detection

            laplace\_operator = np.array([[0, 1, 0], [1, -4, 1], [0, 1, 0]])

            # Convolve image with the Laplace operator

            new\_pixels = np.zeros((rows - 2, cols - 2))

            for i in range(rows - 2):

                for j in range(cols - 2):

                    window = pixels[i:i + 3, j:j + 3]

                    new\_pixels[i, j] = np.sum(window \* laplace\_operator)

            laplace\_img = Image.fromarray(new\_pixels.astype(np.uint8))

            self.display\_image(laplace\_img)

apply Fourier transform of the image and display it (done)

def display\_fourier\_transform(self):

        if self.image:

            gray\_img = self.image.convert('L')  # Convert image to grayscale

            pixels = np.array(gray\_img)

            f\_transform = np.fft.fft2(pixels)  # Compute 2D Fourier Transform

            f\_transform\_shifted = np.fft.fftshift(f\_transform)  # Shift zero frequency component to center

            magnitude\_spectrum = 20 \* np.log(np.abs(f\_transform\_shifted))  # Calculate magnitude spectrum

            # Normalize values for displaying the image

            magnitude\_spectrum \*= 255.0 / np.max(magnitude\_spectrum)

            fourier\_img = Image.fromarray(magnitude\_spectrum.astype(np.uint8))

            self.display\_image(fourier\_img)

add noise (salt and pepper, periodic) + user types parameters and display noisy image. (done)

    def add\_salt\_pepper\_noise(self):

        if self.image:

            white\_pixels = int(self.white\_pixels\_entry.get())

            black\_pixels = int(self.black\_pixels\_entry.get())

            pixels = np.array(self.image)

            noisy\_pixels = self.add\_noise(pixels, white\_pixels, black\_pixels)

            noisy\_img = Image.fromarray(noisy\_pixels)

            self.display\_image(noisy\_img)

    def add\_noise(self, img, num\_white\_pixels, num\_black\_pixels):

        row, col = img.shape[:2]  # Getting the dimensions of the image

        for \_ in range(num\_white\_pixels):

            y\_coord = np.random.randint(0, row)  # Pick a random y coordinate

            x\_coord = np.random.randint(0, col)  # Pick a random x coordinate

            img[y\_coord][x\_coord] = 255  # Color that pixel to white

        for \_ in range(num\_black\_pixels):

            y\_coord = np.random.randint(0, row)  # Pick a random y coordinate

            x\_coord = np.random.randint(0, col)  # Pick a random x coordinate

            img[y\_coord][x\_coord] = 0  # Color that pixel to black

        return img

(you should add this code in the buttons code to make the user write the x, y numbers to apply the noise filter on the image after clicking on the Add Salt and Pepper Noise button)

        self.add\_noise\_button = tk.Button(self.buttons\_frame, text="Add Salt and Pepper Noise", command=self.add\_salt\_pepper\_noise)

        self.add\_noise\_button.pack(side=tk.LEFT)

        self.white\_pixels\_entry = tk.Entry(self.buttons\_frame)

        self.white\_pixels\_entry.pack(side=tk.LEFT)

        self.black\_pixels\_entry = tk.Entry(self.buttons\_frame)

        self.black\_pixels\_entry.pack(side=tk.LEFT)

periodic noise (done)

    def add\_periodic\_noise(self):

        if self.image:

            frequency = int(self.periodic\_frequency\_entry.get())

            img = np.array(self.image)

            noisy\_img = self.add\_periodic(img, frequency)

            noisy\_img = Image.fromarray(noisy\_img)

            self.display\_image(noisy\_img)

    def add\_periodic(self, img, frequency):

        row, col = img.shape[:2]  # Getting the dimensions of the image

        # Create a blank image to draw the periodic noise

        blank\_img = Image.new('RGB', (col, row), color='white')

        draw = ImageDraw.Draw(blank\_img)

        # Draw lines as an example of periodic noise

        for i in range(0, col, frequency):

            draw.line((i, 0, i, row), fill='black', width=2)

        noisy\_img = Image.alpha\_composite(Image.fromarray(img), blank\_img)

        return np.array(noisy\_img)

the buttons that run the function

        self.add\_periodic\_button = tk.Button(self.buttons\_frame, text="Add Periodic Noise", command=self.add\_periodic\_noise)

        self.add\_periodic\_button.pack(side=tk.LEFT)

        self.periodic\_frequency\_entry = tk.Entry(self.buttons\_frame)

        self.periodic\_frequency\_entry.pack(side=tk.LEFT)

remove S&P using median + user types parameters and display clean image. (done)

    def add\_periodic\_noise(self, frequency=20):

        if self.image:

            # Create a blank image to draw the periodic noise

            blank\_img = Image.new('RGB', (self.image.width, self.image.height), color='white')

            draw = ImageDraw.Draw(blank\_img)

            # Draw lines as an example of periodic noise

            for i in range(0, self.image.width, frequency):

                draw.line((i, 0, i, self.image.height), fill='black', width=2)

            noisy\_img = Image.alpha\_composite(self.image.convert('RGBA'), blank\_img.convert('RGBA'))

            self.display\_image(noisy\_img)

    def remove\_salt\_pepper\_noise(self):

        if self.image:

            pixels = np.array(self.image)

            filter\_size = int(self.median\_filter\_entry.get())

            denoised\_img = median\_filter(pixels, size=filter\_size)  # Apply median filter to remove salt and pepper noise

            denoised\_img = Image.fromarray(denoised\_img)

            self.display\_image(denoised\_img)

(you should add this code in the buttons code to make the user write the number to apply the median filter on the image after clicking on button)

        self.remove\_noise\_button = tk.Button(self.buttons\_frame, text="Remove Salt and Pepper Noise Using Median Filter", command=self.remove\_salt\_pepper\_noise)

        self.remove\_noise\_button.pack(side=tk.LEFT)

        self.median\_filter\_entry = tk.Entry(self.buttons\_frame)

        self.median\_filter\_entry.pack(side=tk.LEFT)