main.py

```python
"""
Image Processing Project: Noise Analysis and Denoising

This project aims to analyze the effects of salt-and-pepper noise on an image and perform denoising operations
using an averaging technique. The project involves the following steps:

1. Read an original image.
2. Generate multiple noisy samples by adding varying amounts of salt-and-pepper noise.
3. Group the noisy samples into different sample sizes (1, 5, 10, 50, 100, 500).
4. For each sample size, calculate the average and variance of the noisy samples.
5. Denoise the samples by averaging them within each sample group.
6. Save the denoised images and the calculated statistics to a CSV file.
7. Generate a PDF report with the original image, denoised images, and sample statistics.

Usage:
   1. Ensure that the required dependencies are installed.
   2. Place the original image file named "original.jpg" in the project directory.
   3. Run the main.py script.
   4. The noisy samples will be generated in the "noisy_samples" folder.
   5. The denoised images will be saved in the "result" folder.
   6. The CSV file with statistics will be saved as "result/results.csv".
   7. The PDF report will be saved as "result/results.pdf".

Note: This project is part of an Image Processing class exercise and is intended for educational purposes.

Author: Amirhossein Gholizadeh
"""

import cv2
import numpy as np
import os
import csv
from noise_gen import add_salt_and_pepper_noise
from noise_var_calc import calculate_average_and_variance
from report_gen import generate_pdf_report

def main() -> None:
    # Reading the image
    image_path = "original.jpg"
    image = cv2.imread(image_path)

    # Create directories to save noisy samples and results
    os.makedirs("noisy_samples", exist_ok=True)
    os.makedirs("result", exist_ok=True)

    # Generate noisy samples
    samples = 500
    print(f"Generating {samples} noisy images...")
    for i in range(samples):
        noise_amount = np.random.uniform(0.01, 0.5)
        noisy_samples = add_salt_and_pepper_noise(image, amount=noise_amount)
        cv2.imwrite(f"noisy_samples/noisy_sample_{i}.jpg", noisy_samples)

    # Select sample groups and calculate statistics
    sample_sizes = [1, 5, 10, 50, 100, 500]
    results = []

    print("Averaging operation started...")
    # For each value of k (sample group size), perform the following steps.
    for k in sample_sizes:
        #Read the corresponding number of noisy samples (based on k). Each sample is loaded as a grayscale image.
        selected_samples = [cv2.imread(f"noisy_samples/noisy_sample_{i}.jpg", cv2.IMREAD_GRAYSCALE) for i in range(k)]
        # Calculate the average and variance for this group of noisy samples.
        avg, var = calculate_average_and_variance(np.mean(selected_samples, axis=0))
        results.append((k, avg, var))

        # Create resulting image for this sample group by averaging the selected samples
        resulting_image = np.mean(selected_samples, axis=0) # Based on the formula: g(x, y) = (1/k) * Σ(i=1 to k) gi(x, y)
        cv2.imwrite(f"result/result_for_{k}_samples.jpg", resulting_image)
        print(f"Saved the results for {k} smples in \"result\" folder")

    # Save results to a csv file
    csv_filename = "result/results.csv"
    # Write the sample group size, average, and average variance for each group.
    with open(csv_filename, "w", newline="") as csvfile:
        writer = csv.writer(csvfile)
        writer.writerow(["Sample Groups", "Average", "Average Variance"])
        for row in results:
            writer.writerow(row)

    generate_pdf_report(sample_sizes)
    print(f"Finished the operation successfully. To see the results please check the \"result\" folder")

if __name__ == "__main__":
    main()
```

noise_gen.py

```python
import numpy as np

def add_salt_and_pepper_noise(image, amount) -> list:
    """
    Adds salt-and-pepper noise to an image.
    :param image: Input image
    :param amount: Proportion of image pixels to replace with noise
    :return: Image with salt-and-pepper noise
    """

    # Converts the input image to a floating-point format (values between 0.0 and 1.0).
    # Divides each pixel value by 255 to normalize it to the [0, 1] range.
    image_float = np.float32(image) / 255.0

    # Creates random values for each pixel in the image.
    # These random values determine whether a pixel becomes “salt” (white) or “pepper” (black).
    random_values = np.random.rand(*image.shape[:2])

    # Set pixels to 1 (salt) or 0 (pepper) based on random values
    image_float[random_values < (amount / 2)] = 0 # If a random value is less than half of the specified amount, set the pixel value to 0 (black) for “pepper.”
    image_float[random_values > (1 - amount / 2)] = 1 # If a random value is greater than (1 - half of the amount), set the pixel value to 1 (white) for “salt.”

    # Converts the modified floating-point image back to an 8-bit format (values between 0 and 255).
    noisy_image = np.uint8(image_float * 255)
    return noisy_image
```

noise_var_calc.py

```python
import numpy as np

def calculate_average_and_variance(image) -> tuple[float, float]:
    """
    Calculates average and variance for an image based on these formulas:
    - E[g(x, y)] = f(x, y)
    - σ²g(x, y) = (1/k) * σ²η(x, y)
    :param image: Input image
    :return: Average and variance
    """
    # Computes the total number of pixels in the image.
    # image.shape[0] gives the height (number of rows), and image.shape[1] gives the width (number of columns).
    total_pixels = image.shape[0] * image.shape[1]

    # Computes the average pixel value (mean) for the entire image.
    average_value = np.sum(image) / total_pixels
    # Computes the sum of squared differences between each pixel value and the average.
    squared_diff_sum = np.sum((image - average_value) ** 2)
    # Computes the variance of pixel values in the image.
    variance_value = squared_diff_sum / total_pixels
    return average_value, variance_value
```

report_gen.py

```python
from reportlab.lib.pagesizes import landscape, A5
from reportlab.platypus import SimpleDocTemplate, Image, Paragraph, Spacer, Table, TableStyle
from reportlab.lib.styles import getSampleStyleSheet
from reportlab.lib.units import inch
import csv

def add_snapshot():
    from PyPDF2 import PdfReader, PdfWriter
    # Open the existing PDF
    existing_pdf = PdfReader("result/report.pdf")
    pdf_writer = PdfWriter()

    # Add all pages from the existing PDF to the writer
    for page in existing_pdf.pages:
        pdf_writer.add_page(page)

    # Open the PDF file to append
    pdf_to_append = PdfReader("snapshots/snapshots.pdf")

    # Append pages from the new PDF to the writer
    for page in pdf_to_append.pages:
        pdf_writer.add_page(page)

    # Write out the combined PDF
    with open("result/report.pdf", "wb") as out_file:
        pdf_writer.write(out_file)

def generate_pdf_report(samples) -> None:
    """
    Generate a PDF report with the original image, denoised images, and sample statistics.

    Args:
        samples (list): List of sample sizes used for denoising.

    This function creates a PDF report using the ReportLab library. The report includes the following elements:
        - Original image
        - Denoised images for each sample size, grouped in rows of 3
        - Titles for each denoised image indicating the sample size

    The PDF report is saved as "result/report.pdf".
    """

    print("Generating pdf report...")

    # Set up styles
    styles = getSampleStyleSheet()
    title_style = styles["Heading2"]
    subtitle_style = styles["Heading3"]

    # Set up the PDF document
    doc = SimpleDocTemplate("result/report.pdf", pagesize=landscape(A5), topMargin=0*inch, bottomMargin=0*inch)
    elements = []

    # Add the original image with title
    original_image = Image("original.jpg", width=3.5*inch, height=4*inch)
    elements.append(Paragraph("Author: Amirhossein Gholizadeh"))
    elements.append(Paragraph("Student Num: 4021119008"))
    elements.append(Paragraph("Course: Image Processing"))
    elements.append(Paragraph("Chapter: 2"))
    elements.append(Paragraph("Original Image", title_style))
    elements.append(original_image)

    # Add a spacer for bottom margin
    #elements.append(Spacer(0, 1*inch))

    # Add the denoised images with titles
    image_files = [f"result/result_for_{i}_samples.jpg" for i in samples]
    titles = ["k=1", "k=5", "k=10", "k=20", "k=50", "k=100"]
    table_data = []
    for i in range(0, len(image_files), 3):
        # Create a new row for the table
        row = []
        for j in range(3):
            # Check if there is an image file for this position
            if i + j < len(image_files):
                # Create an Image object for the denoised image file
                image = Image(image_files[i + j], width=1.5*inch, height=2*inch)
                # Create a Paragraph object for the title
                title = Paragraph(titles[i + j], subtitle_style)
                # Append the image and title as a list to the row
                row.append([image, title])
        # Append the row to the table data
        table_data.append(row)
    # Create a Table object with the table data and set column widths
    table = Table(table_data, colWidths=[2.5*inch, 2.5*inch, 2.5*inch])
    # Set the vertical alignment for all cells in the table
    table.setStyle(TableStyle([('VALIGN', (0, 0), (-1, -1), 'MIDDLE')]))
    elements.append(Spacer(0, 0.3*inch))
    elements.append(Paragraph("Denoised Images", title_style))
    elements.append(Spacer(0, 0.3*inch))
    elements.append(table)

    # Add the sample statistics table from CSV file
    with open('result/results.csv', 'r') as csv_file:
        csv_reader = csv.reader(csv_file)
        csv_data = list(csv_reader)

    # Create a table from the CSV data
    stats_table = Table(csv_data, colWidths=[inch, 1.5*inch, 1.5*inch])
    stats_table.setStyle(TableStyle([('ALIGN', (0, 0), (-1, -1), 'CENTER')]))
    elements.append(Spacer(0, 0.3*inch))
    elements.append(Paragraph("Sample Statistics", title_style))
    elements.append(Spacer(0, 0.3*inch))
    elements.append(stats_table)

    # Build the PDF document
    doc.build(elements)
    add_snapshot()
    print("PDF file 'report.pdf' generated successfully.")
```

Requirements:

chardet==5.2.0
numpy==1.26.4
opencv-python==4.9.0.80
pillow==10.3.0
PyPDF2==3.0.1
reportlab==4.2.0