Data Collection and Preprocessing Phase

Date	29 august 2024
Team ID	166
Project Title	Deep learning techniques for breast cancer prediction
Maximum Marks	6 Marks

Preprocessing Template

The images will be preprocessed by resizing, normalizing, augmenting, denoising, adjusting contrast, detecting edges, converting color space, cropping, batch normalizing, and whitening data. These steps will enhance data quality, promote model generalization, and improve convergence during neural network training, ensuring robust and efficient performance across various computer vision tasks.

Section	**Description**
Data Overview	Clinical data Histopathological data Imaging data Genomic data Lifestyle data Laboratory results Survival and recurrance data
Resizing	48 ,48
Normalization	1049columns*2rows,263columns*2rows
Data Augmentation	`rotation_range=40`: Rotates the image randomly within a range of 40 degrees. `width_shift_range=0.2`: Shifts the image horizontally by up to 20% of the total width. `height_shift_range=0.2`: Shifts the image vertically by up to 20% of the total height. `shear_range=0.2`: Applies random shear transformation.

	`zoom_range=0.2`: Randomly zooms in on the image by up to 20%. `horizontal_flip=True`: Flips the image horizontally with a 50% chance. `fill_mode='nearest'`: Fills in missing pixels after a transformation using the nearest neighbor interpolation.
Denoising	**Gaussian Blur**:`cv2.GaussianBlur(img, (5, 5), 0)`: Applies a Gaussian filter with a 5x5 kernel size to smooth the image and reduce noise. **Median Filter**:`cv2.medianBlur(img, 5)`: Replaces each pixel value with the median value of the surrounding 5x5 neighborhood. Works well for removing "salt-and-pepper" noise. **Non-Local Means Denoising**;`cv2.fastNlMeansDenoisingColored(img, None, 10, 10, 7, 21)`: Removes noise while preserving the details of the image. This method is especially effective for color images.
Edge Detection	**Canny Edge Detection**: o `cv2.Canny(img, 100, 200)`: This method detects edges by using a multi-stage process. The two thresholds (100, 200) define the hysteresis for detecting strong and weak edges. o It's one of the most popular edge detection algorithms due to its accuracy and noise sensitivity. **Sobel Operator**: o `cv2.Sobel(img, cv2.CV_64F, 1, 0, ksize=5)` applies the Sobel filter in the x-direction (for vertical edges), and `cv2.Sobel(img, cv2.CV_64F, 0, 1, ksize=5)` applies it in the y-direction (for horizontal edges). o `cv2.addWeighted(sobel_x, 0.5, sobel_y, 0.5, 0)` combines the x and y direction results.

	Laplacian of Gaussian (LoG): o `cv2.Laplacian(img, cv2.CV_64F)`: This method calculates the second derivative of the image and is useful for detecting both horizontal and vertical edges. It's sensitive to noise, so it's often used with Gaussian smoothing.
Color Space Conversion	• **Grayscale**: • `cv2.COLOR_BGR2GRAY`: Converts the image from BGR (Blue-Green-Red, default OpenCV format) to Grayscale. The result is a single channel image that represents intensity (brightness) levels. • **HSV (Hue, Saturation, Value)**: • `cv2.COLOR_BGR2HSV`: This color space separates image intensity from color information, making it useful for tasks like object tracking or segmentation based on color. • **Hue** represents color, **Saturation** represents intensity of color, and **Value** represents brightness. • **LAB (CIELAB)**: • `cv2.COLOR_BGR2LAB`: This is a perceptually uniform color space where **L** stands for lightness, **A** represents green-to-red, and **B** represents blue-to-yellow. It is often used in tasks like color correction and balancing. • **YCrCb (Luminance-Chrominance)**: • `cv2.COLOR_BGR2YCrCb`: This color space is used in video compression. **Y** represents luminance (brightness), while **Cr** and **Cb** represent chrominance (color differences).
Image Cropping	• **Thresholding**: Converts the image to binary (black and white) for easier contour detection. You can adjust the thresholding values based on the intensity of the objects in the

	image. • **Contour Detection**: Finds the boundaries of objects in the image. • **Bounding Box**: We use `cv2.boundingRect()` to get the smallest rectangle that encloses the largest detected contour (object of interest). • **Cropping**: The bounding box coordinates (x, y, w, h) are used to crop the image to focus on the object.
Batch Normalization	• **BatchNormalization Layer**: The `BatchNormalization()` layer is added after the `Dense` layer but before the activation function in most cases. This ensures that the inputs are normalized before being passed to the activation function. • **Normalization**: During training, batch normalization standardizes the inputs for each mini-batch by subtracting the batch mean and dividing by the batch standard deviation. • **Scale and Shift**: After normalization, learnable parameters `gamma` (scale) and `beta` (shift) are introduced to allow the network to scale and shift the normalized output if needed.
Data Preprocessing Code Screenshots	

Loading Data	import pandas as pd
	
	# Load CSV file
	df = pd.read_csv('path_to_dataset.csv')
	
	# Display first few rows
	print(df.head())
	
	# Install the package if not already installed
	install.packages("readr")
	
	# Load library
	library(readr)
	
	# Load CSV file
	df <- read_csv('path_to_dataset.csv')
	
	# Display first few rows
	head(df)

Resizing	# Function to read a file and print its contents def read_file(file_path):

	try:
	with open(file_path, 'r') as file:
	contents = file.read()
	print(contents)
	except FileNotFoundError:
	print(f"Error: The file '{file_path}' was not found.")
	except Exception as e:
	print(f"An error occurred: {e}")
	
	# Example usage
	file_path = 'example.txt' # Change this to your file path
	read_file(file_path)

Normalization	import numpy as np
	import pandas as pd
	from sklearn.preprocessing import MinMaxScaler
	
	# Sample data
	data = {
	'A': [10, 20, 30, 40, 50],
	'B': [5, 10, 15, 20, 25],
	'C': [100, 200, 300, 400, 500]
	}
	
	# Create a DataFrame
	df = pd.DataFrame(data)
	
	# Display original data
	print("Original Data:")
	print(df)
	
	# Initialize the MinMaxScaler
	scaler = MinMaxScaler()
	
	# Normalize the data
	normalized_data = scaler.fit_transform(df)
	
	# Create a DataFrame with normalized data
	normalized_df = pd.DataFrame(normalized_data,
	columns=df.columns)
	
	# Display normalized data
	print("\nNormalized Data:")
	print(normalized_df)

Data Augmentation	import numpy as np
	from keras.preprocessing.image import ImageDataGenerator
	import matplotlib.pyplot as plt
	
	# Sample image data (for demonstration, you would load your actual image data)
	# Here, we create a dummy image (a 64x64 RGB image)
	sample_image = np.random.rand(64, 64, 3) # Randomly generated image
	
	# Reshape the image to add an extra dimension (batch size)
	sample_image = sample_image.reshape((1, 64, 64, 3))
	
	# Initialize the ImageDataGenerator with augmentation parameters
	datagen = ImageDataGenerator(
	  rotation_range=40, # Randomly rotate images in the range (degrees, 0 to 180)
	  width_shift_range=0.2, # Randomly translate images horizontally
	  height_shift_range=0.2, # Randomly translate images vertically
	  shear_range=0.2, # Shear angle in counter-clockwise direction in degrees
	  zoom_range=0.2, # Randomly zoom into images
	  horizontal_flip=True, # Randomly flip images
	  fill_mode='nearest' # Fill in newly created pixels after transformations
	)
	
	# Generate augmented images
	augmented_images = datagen.flow(sample_image, batch_size=1)
	
	# Display original and augmented images
	plt.figure(figsize=(12, 6))
	
	# Show original image
	plt.subplot(2, 4, 1)
	plt.title("Original Image")
	plt.imshow(sample_image[0])
	
	# Show augmented images
	for i in range(8):
	  augmented_image =

	next(augmented_images)[0].astype('float32') # Get the next augmented image
	plt.subplot(2, 4, i + 2)
	plt

Denoising	import numpy as np
	import matplotlib.pyplot as plt
	from skimage import io
	from skimage.filters import gaussian
	from skimage.util import random_noise
	
	# Load an example image (you can replace this with your image path)
	# For demonstration, we will use a sample image from skimage
	image = io.imread('https://image.shutterstock.com/image-photo/image-250nw-183919131.jpg')
	
	# Add random noise to the image
	noisy_image = random_noise(image, var=0.1) # Increase variance for more noise
	
	# Denoise the image using Gaussian filter
	denoised_image = gaussian(noisy_image, sigma=1, multichannel=True)
	
	# Display original, noisy, and denoised images
	plt.figure(figsize=(15, 5))
	
	# Original Image
	plt.subplot(1, 3, 1)
	plt.title("Original Image")
	plt.imshow(image)
	plt.axis('off')
	
	# Noisy Image
	plt.subplot(1, 3, 2)
	plt.title("Noisy Image")
	plt.imshow(noisy_image)
	plt.axis('off')
	
	# Denoised Image
	plt.subplot(1, 3, 3)
	plt.title("Denoised Image")
	plt.imshow(denoised_image)
	plt.axis('off')

	plt.tight_layout() plt.show()

Edge Detection	import numpy as np
	import matplotlib.pyplot as plt
	from skimage import io, filters, color
	
	# Load an example image (you can replace this with your image path)
	image = io.imread('https://image.shutterstock.com/image-photo/image-250nw-183919131.jpg')
	
	# Convert the image to grayscale
	gray_image = color.rgb2gray(image)
	
	# Apply Canny edge detection
	edges = filters.sobel(gray_image) # Alternatively, you can use skimage.feature.canny
	
	# Display original and edge-detected images
	plt.figure(figsize=(10, 5))
	
	# Original Image
	plt.subplot(1, 2, 1)
	plt.title("Original Image")
	plt.imshow(image)
	plt.axis('off')
	
	# Edge Detected Image
	plt.subplot(1, 2, 2)
	plt.title("Edge Detected Image")
	plt.imshow(edges, cmap='gray')
	plt.axis('off')
	
	plt.tight_layout()
	plt.show()

Color Space Conversion	import cv2
	import matplotlib.pyplot as plt
	
	# Load an example image (you can replace this with your image path)
	image = cv2.imread('https://image.shutterstock.com/image-photo/image-250nw-183919131.jpg')

	# Convert from BGR to RGB (OpenCV loads images in BGR format)
	image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
	
	# Convert RGB to Grayscale
	image_gray = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2GRAY)
	
	# Convert RGB to HSV
	image_hsv = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2HSV)
	
	# Convert RGB to LAB
	image_lab = cv2.cvtColor(image_rgb, cv2.COLOR_RGB2Lab)
	
	# Display the original and converted images
	plt.figure(figsize=(15, 10))
	
	# Original Image
	plt.subplot(2, 2, 1)
	plt.title("Original Image (RGB)")
	plt.imshow(image_rgb)
	plt.axis('off')
	
	# Grayscale Image
	plt.subplot(2, 2, 2)
	plt.title("Grayscale Image")
	plt.imshow(image_gray, cmap='gray')
	plt.axis('off')
	
	# HSV Image
	plt.subplot(2, 2, 3)
	plt.title("HSV Image")
	plt.imshow(image_hsv)
	plt.axis('off')
	
	# LAB Image
	plt.subplot(2, 2, 4)
	plt.title("LAB Image")
	plt.imshow(image_lab)
	plt.axis('off')
	
	plt.tight_layout()
	plt.show()

Image Cropping	import cv2
	import matplotlib.pyplot as plt
	
	# Load an example image (you can replace this with your image path)
	image = cv2.imread('https://image.shutterstock.com/image-photo/image-250nw-183919131.jpg')
	
	# Convert from BGR to RGB (OpenCV loads images in BGR format)
	image_rgb = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
	
	# Define the coordinates for cropping (x_start, y_start, width, height)
	x_start, y_start = 50, 50 # Top-left corner of the cropping rectangle
	width, height = 200, 150 # Dimensions of the cropping rectangle
	
	# Crop the image using NumPy slicing
	cropped_image = image_rgb[y_start:y_start + height, x_start:x_start + width]
	
	# Display the original and cropped images
	plt.figure(figsize=(10, 5))
	
	# Original Image
	plt.subplot(1, 2, 1)
	plt.title("Original Image")
	plt.imshow(image_rgb)
	plt.axis('off')
	
	# Cropped Image
	plt.subplot(1, 2, 2)
	plt.title("Cropped Image")
	plt.imshow(cropped_image)
	plt.axis('off')
	
	plt.tight_layout()
	plt.show()

Batch Normalization	import numpy as np import matplotlib.pyplot as plt from keras.datasets import mnist from keras.models import Sequential from keras.layers import Conv2D, BatchNormalization, Flatten,

	Dense, MaxPooling2D, Dropout
	from keras.utils import to_categorical
	
	# Load MNIST dataset
	(x_train, y_train), (x_test, y_test) = mnist.load_data()
	
	# Preprocess the data
	x_train = x_train.reshape((x_train.shape[0], 28, 28, 1)).astype('float32') / 255
	x_test = x_test.reshape((x_test.shape[0], 28, 28, 1)).astype('float32') / 255
	y_train = to_categorical(y_train, 10)
	y_test = to_categorical(y_test, 10)
	
	# Build a simple CNN model with Batch Normalization
	model = Sequential()
	model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
	model.add(BatchNormalization()) # Apply Batch Normalization
	model.add(MaxPooling2D(poo