Data Collection and Preprocessing Phase

Date	June 26, 2025
Team ID	SWTID1750058607
Project Title	Early-Stage Disease Diagnosis System Using Human Nail Image
Maximum Marks	6 Marks

Data Preprocessing

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	The dataset contains images of human nails, categorized by various diseases and conditions. It is divided into a set with 655 images and a testing set with 183 images. Each disease category is organized into its own folder.The dataset contains images of human nails, categorized by various diseases and conditions. It is divided into a set with 655 images and a testing set with 183 images. Each disease category is organized into its own folder.
Resizing	Images are resized to a uniform dimension of 224x224 pixels to ensure they are suitable for input into the VGG16 model.
Normalization	Pixel values of the images are normalized to a specific range (typically 0 to 1) to help the model converge faster and perform more accurately. This is implicitly handled when loading the data and preparing it for the model.
Data Augmentation	This step, which could involve techniques like flipping, rotation, or zooming to artificially expand the dataset, was performed in this project.
Denoising	This step, which involves applying filters to reduce noise, was not performed in this project.

Edge Detection	This step, which applies algorithms to highlight edges, The layers of the VGG16 is responsible to detect edges
Color Space Conversion	Images are converted to the RGB color space to ensure consistency across the dataset, as some source images may be in different formats (e.g., RGBA or Grayscale).
Image Cropping	While resizing achieves a standard size, specific cropping to focus on regions of interest was not performed as a separate step.
Batch Normalization	Batch normalization is not applied as a data preprocessing step but is integrated as a layer within the deep learning model architecture to stabilize and accelerate training.

Data Preprocessing Code Screenshots

Loading Data

```
import os

from zipfile import ZipFile

# Unzipping train data

with ZipFile(’train.zip’, ’r’) as zip_ref:
zip_ref.extractall(’./’) # Extracts to current
directory

print(″Train Dataset extracted″)

# Unzipping test data

with ZipFile(’test.zip’,’r’) as zip_ref:
zip_ref.extractall(’./’) # Extracts to current
directory

print(″Test Dataset extracted″)

# Counting number of images on train data
file_count_train = 0

for path, dirnames, filenames in
os.walk(’./content/train’):

file_count_train += len(filenames)

print(path) # This prints the directory path
print(″Number of images in train:″,
file_count_train)
```

	`# Counting number of images on test data`
	`file_count_test = 0`
	
	`for path, dirnames, filenames in`
	`os.walk('./content/test'):`
	
	`file_count_test += len(filenames)`
	
	`print(path) # This prints the directory path`
	`print("Number of images in test:", file_count_test)`

Resizing	`import os`
	
	`from PIL import Image`
	
	`# Create resized directories if they don't exist`
	`os.makedirs('./train.resize', exist_ok=True)`
	`os.makedirs('./test.resize', exist_ok=True)`
	`print("Train resized folder made")`
	
	`print("Test resized folder made") # Corrected`
	`message`
	
	`# Function to resize images in a folder`
	
	`def resize_images_in_folder(original_folder,`
	`resized_folder, target_size=(224, 224)):`
	
	`for path, dirnames, filenames in`
	`os.walk(original_folder):`
	
	`# Create corresponding subfolder in resized_folder`
	`relative_path = os.path.relpath(path,`
	`original_folder)`
	`os.makedirs(os.path.join(resized_folder,`
	`relative_path), exist_ok=True)`
	
	`for filename in filenames:`
	
	`if filename.lower().endswith(('.png', '.jpg',`
	`'.jpeg', '.gif', '.bmp')): image_path =`
	`os.path.join(path, filename) save_path =`
	`os.path.join(resized_folder, relative_path,`
	`filename)`
	
	`try:`

	img = Image.open(image_path) img = img.resize(target_size) img = img.convert('RGB')
	
	# Ensure RGB format img.save(save_path) except Exception as e:
	
	print(f"Skipped (error): {image_path} - {e}")
	
	else:
	
	print(f"Skipped (not an image): {os.path.join(path, filename)}")
	
	# Resize training data
	
	print("Training Data Resized started...") resize_images_in_folder('./content/train', './train.resize')
	
	print("Training Data Resized completed.")
	
	# Resize test data
	
	print("Test Data Resized started...") resize_images_in_folder('./content/test', './test.resize')
	
	print("Test Data Resized completed.")
	
	# Counting resized images
	
	file_count_train_resized = 0
	
	for path, dirnames, filenames in os.walk('./train.resize'):
	
	file_count_train_resized += len(filenames) print("Number of images in train resized:", file_count_train_resized)
	
	file_count_test_resized = 0
	
	for path, dirnames, filenames in os.walk('./test.resize'):
	
	file_count_test_resized += len(filenames) print("Number of images in test resized:", file_count_test_resized) # Corrected message

Normalization	train_datagen = ImageDataGenerator(rescale = 1./255, shear_range = 0.2, zoom_range = 0.2, horizontal_flip = True) test_datagen = ImageDataGenerator(rescale = 1./255)
Data Augmentation	train_datagen = ImageDataGenerator(rescale = 1./255, shear_range = 0.2, zoom_range = 0.2, horizontal_flip = True) test_datagen = ImageDataGenerator(rescale = 1./255) train_set = train_datagen.flow_from_directory('/content/train.resize', target_size = (224, 224), batch_size = 32, class_mode = 'categorical') test_set = test_datagen.flow_from_directory('/content/test.resize', target_size = (224, 224), batch_size = 32, class_mode = 'categorical')

Edge Detection	The pre-trained model VGG16 uses its layers and detects the edges automatically from tensorflow.keras.applications.vgg16 import VGG16,preprocess_input vgg = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

Color Space Conversion	The VGG16 automatically does color space conversion. Initially manually color spacing was performed
	
	for folder in Train_subfolders:
	
	folderPath = os.path.join(ResizedFolderPath, folder)
	
	for image_name in os.listdir(folderPath):
	
	image_path = os.path.join(folderPath, image_name)
	
	if image_name.lower().endswith(('.png', '.jpg', '.jpeg')):
	
	try:
	
	img = Image.open(image_path).resize((224, 224)).convert('RGB')
	
	img_array = np.array(img)
	
	img_array = preprocess_input(img_array)
	
	X_train.append(img_array)
	
	y_train.append(Labels_map[folder])
	
	except Exception as e:
	
	print(f"Skipped {image_path}: {e}")
	
	X_train = np.array(X_train)
	
	y_train = np.array(y_train)
	
	Same for test was performed

Image Cropping	Image cropping was not performed ,an example code is shown showing to perform cropping
	import os
	
	from PIL import Image
	
	def apply_cropping_to_folder(input_folder, output_folder, x, y, width, height):
	os.makedirs(output_folder, exist_ok=True)
	
	for path, dirnames, filenames in os.walk(input_folder):
	
	relative_path = os.path.relpath(path, input_folder)
	os.makedirs(os.path.join(output_folder, relative_path), exist_ok=True)
	
	for filename in filenames:
	
	if filename.lower().endswith(('.png', '.jpg', '.jpeg', '.gif', '.bmp')):
	
	image_path = os.path.join(path, filename)
	
	save_path = os.path.join(output_folder, relative_path, filename)
	
	try:
	
	img = Image.open(image_path).convert('RGB')
	
	# PIL crop uses (left, upper, right, lower)
	cropped_img = img.crop((x, y, x + width, y + height))
	
	cropped_img.save(save_path)
	
	except Exception as e:
	
	print(f"Skipped cropping (error): {image_path} - {e}")
	
	# Example usage (cropping a central square of 200x200 from a 224x224 image):
	
	# x_center = (224 - 200) // 2
	
	# y_center = (224 - 200) // 2
	
	# apply_cropping_to_folder('./train.resize', './train.cropped', x_center, y_center, 200, 200)
	
	# apply_cropping_to_folder('./test.resize', './test.cropped', x_center, y_center, 200, 200)

Batch Normalization	from tensorflow.keras.layers import BatchNormalization
	from tensorflow.keras.models import Model
	from tensorflow.keras.layers import Input, Dense, Flatten, Dropout
	from tensorflow.keras.applications import VGG16
	# This section typically defines the model architecture.
	# Batch Normalization is a layer within the neural network, not a separate
	# pre-processing script applied directly to images before model input.
	# Example VGG16 model setup with Batch Normalization
	input_tensor = Input(shape=(224, 224, 3))
	base_model = VGG16(weights='imagenet', include_top=False, input_tensor=input_tensor)
	# Freeze the layers of the base model
	for layer in base_model.layers:
	layer.trainable = False
	# Add custom layers on top of VGG16
	x = Flatten()(base_model.output)
	x = Dense(256, activation='relu')(x)
	x = BatchNormalization()(x) # Applying BatchNormalization
	x = Dropout(0.5)(x)
	predictions = Dense(17, activation='softmax')(x) # Assuming 17 classes based on original document
	model = Model(inputs=base_model.input, outputs=predictions)
	# Compile the model

	# model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy']) # model.summary()