 

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.re size',  target\_size = (224, 224),  batch\_size = 32,  class\_mode = 'categorical') test\_set =  test\_datagen.flow\_from\_directory('/content/test.resi ze',  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() |