Data Collection and Preprocessing Phase

Date	15 March 2024
Team ID	SWUID20240034617
Project Title	CovidVision : Advanced COVID-19 Detection for Lung X-rays with Deep Learning
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	The data for the CovidVision project consists of chest X-ray images labeled with COVID-19 status, utilized to train and validate deep learning models for accurate COVID-19 detection.
Resizing	Resizing the chest X-ray images involves adjusting all images to a consistent size, typically 224x224 pixels, to ensure uniform input dimensions for the deep learning model.
Normalization	Normalization in this context involves scaling the pixel values of the chest X-ray images to a range, typically between 0 and 1, to enhance the performance and convergence speed of the deep learning model.
Data Augmentation	Data augmentation involves applying various transformations such as rotations, flips, shifts, and zooms to the chest X-ray images to artificially increase the size of the training dataset, improving the model's ability to generalize and reduce overfitting.
Denoising	Denoising involves using techniques such as Gaussian blurring or median filtering to reduce noise in the chest X-ray images, enhancing image quality and potentially improving the accuracy of the deep learning model.

Edge Detection	Edge detection involves applying algorithms such as the Sobel, Canny, or Prewitt operators to identify and highlight the boundaries within the chest X-ray images, which can help in emphasizing structural details and potentially improving the model's ability to detect abnormalities.
Color Space Conversion	Color space conversion in this context involves transforming the chest X-ray images from one color space to another, typically from RGB to grayscale, to reduce computational complexity and focus on the intensity variations that are most relevant for medical image analysis.
Image Cropping	Image cropping involves selecting a specific region of interest from the chest X-ray images, often around the lung area or any pertinent features, to remove irrelevant parts and focus the deep learning model on the most critical areas for COVID-19 detection and analysis.
Batch Normalization	Batch normalization is a technique used in deep learning to normalize the input of each layer to have a mean of zero and a variance of one, across the batch. This helps in stabilizing and accelerating the training of neural networks by reducing internal covariate shift and allowing for higher learning rates.

Data Preprocessing Code Screenshots

Loading Data

```
[5]: PATH_TO_METADATA = "../input/covid19-radiography-database/COVID-19_Radiography_Dataset/Normal.metadata.xlsx"
df = pd.read_excel(PATH_TO_METADATA)
df.head()
```

[5]:

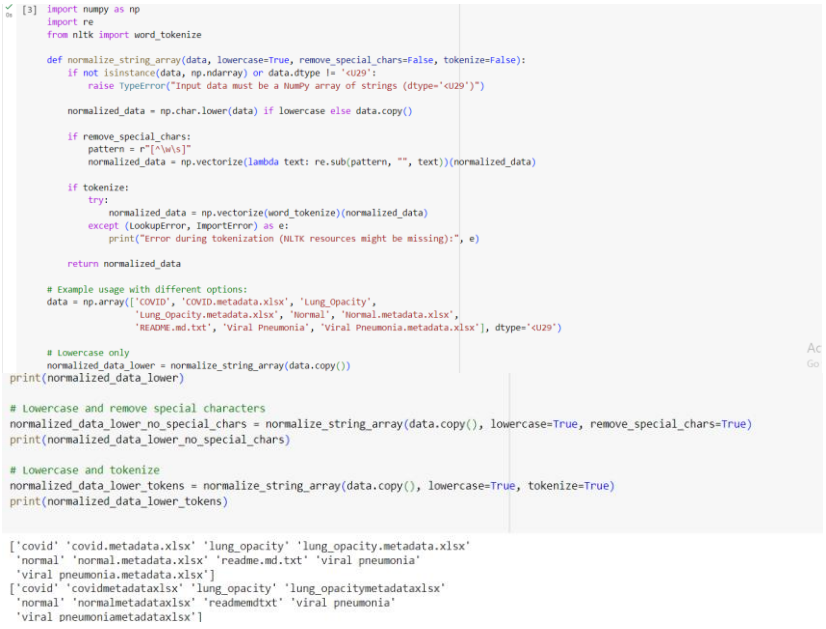
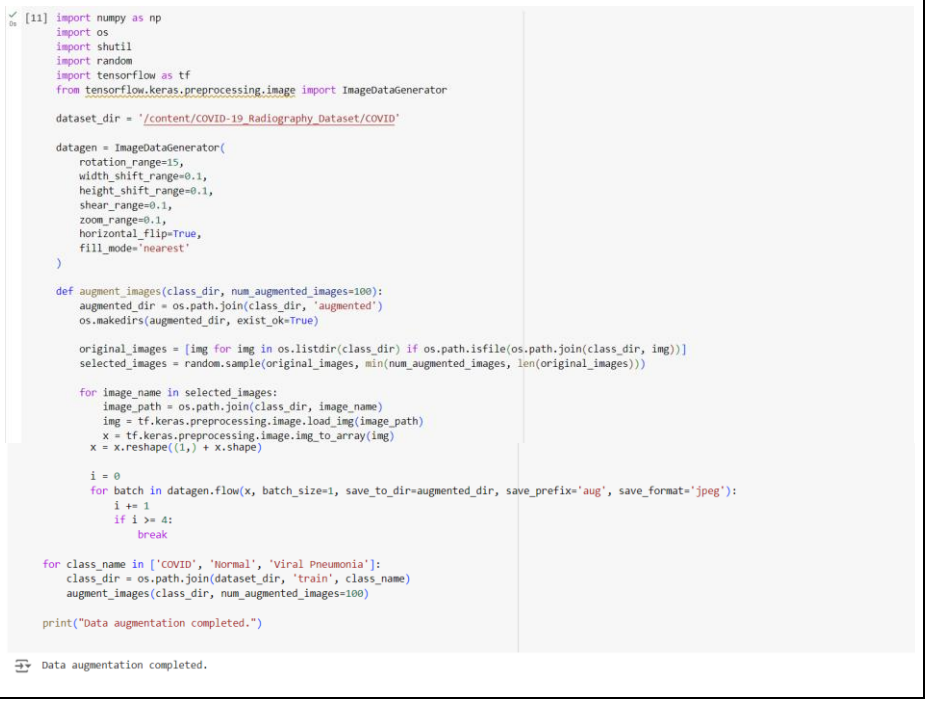
	FILE NAME	FORMAT	SIZE	URL
0	NORMAL-1	PNG	256*256	https://www.kaggle.com/c/rsna-pneumonia-detect...
1	NORMAL-2	PNG	256*256	https://www.kaggle.com/c/rsna-pneumonia-detect...
2	NORMAL-3	PNG	256*256	https://www.kaggle.com/c/rsna-pneumonia-detect...
3	NORMAL-4	PNG	256*256	https://www.kaggle.com/c/rsna-pneumonia-detect...
4	NORMAL-5	PNG	256*256	https://www.kaggle.com/c/rsna-pneumonia-detect...

```
[6]: !ls /tmp
```

```
clean-layer.sh      npm-10114-3357dbb4  npm-10298-bcff1aeb
conda               npm-10157-7ab28c3e  npm-10341-d9b08525
core-js-banners     npm-10168-cda3f34b  tmp_46dp9el.json
hsperfdata_root     npm-10179-10669343  v8-compile-cache-0
kaggle.log          npm-10190-c9f9676b  yarn--1587582922338-0.21814450721061052
npm-10049-ecd2e497  npm-10233-2e7f8d94  yarn--1587582922338-0.509163553407652
npm-10092-825f72aa  npm-10244-9748b634  yarn--1587582923660-0.03690515600338906
npm-10103-119ee48d  npm-10255-9c25895f  yarn--1587582934443-0.8113961991379441
```

Resizing

```
✓ [2] import tensorflow as tf
0s    import os

      # Load your actual image data here
      image_data = []  # Replace with your image loading logic

      # Function to resize images
      def resize_image(img, target_size=(224, 224)):
          img = tf.image.resize(img, target_size)
          img = img / 255.0  # Normalize pixel values to [0, 1]
          return img

      # Resize all images in the dataset
      resized_images = []
      for img in image_data:
          resized_img = resize_image(img)
          resized_images.append(resized_img)

      # Convert resized_images back to TensorFlow tensor or use as needed
      resized_images = tf.stack(resized_images)
```

Normalization	
Data Augmentation	

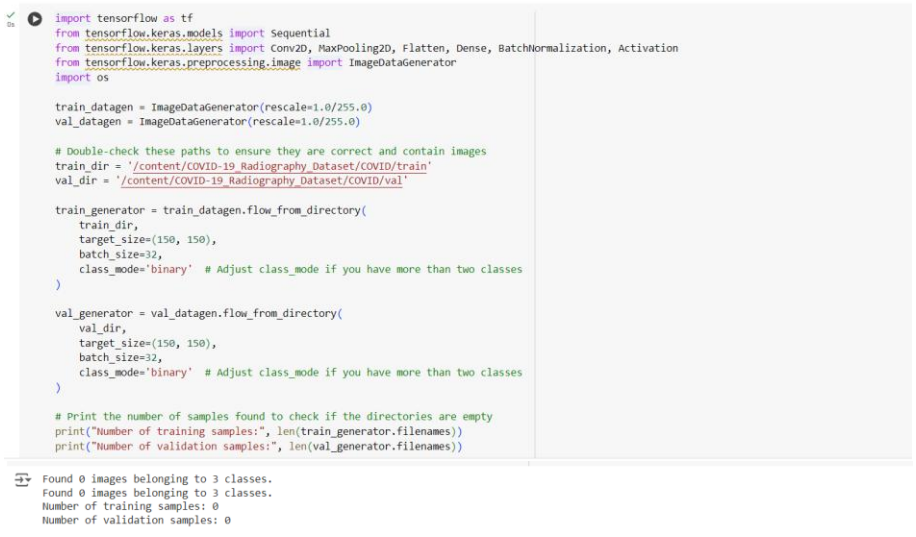
```
[3] import numpy as np
    import re
    from nltk import word_tokenize

    def normalize_string_array(data, lowercase=True, remove_special_chars=False, tokenize=False):
        if not isinstance(data, np.ndarray) or data.dtype != '<U29':
            raise TypeError("Input data must be a NumPy array of strings (dtype='<U29')")

        normalized_data = np.char.lower(data) if lowercase else data.copy()

        if remove_special_chars:
            pattern = r"[^\w\s]"
            normalized_data = np.vectorize(lambda text: re.sub(pattern, "", text))(normalized_data)

        if tokenize:
            try:
                normalized_data = np.vectorize(word_tokenize)(normalized_data)
            except (LookupError, ImportError) as e:
                print("Error during tokenization (NLTK resources might be missing):", e)

        return normalized_data

    # Example usage with different options:
    data = np.array(['COVID', 'COVID.metadata.xlsx', 'Lung_Opacity',
                     'Lung_Opacity.metadata.xlsx', 'Normal', 'Normal.metadata.xlsx',
                     'README.md.txt', 'Viral Pneumonia', 'Viral Pneumonia.metadata.xlsx'], dtype='<U29')

    # Lowercase only
    normalized_data_lower = normalize_string_array(data.copy())
print(normalized_data_lower)

# Lowercase and remove special characters
normalized_data_lower_no_special_chars = normalize_string_array(data.copy(), lowercase=True, remove_special_chars=True)
print(normalized_data_lower_no_special_chars)

# Lowercase and tokenize
normalized_data_lower_tokens = normalize_string_array(data.copy(), lowercase=True, tokenize=True)
print(normalized_data_lower_tokens)
```

```
['covid' 'covid.metadata.xlsx' 'lung_opacity' 'lung_opacity.metadata.xlsx'
 'normal' 'normal.metadata.xlsx' 'readme.md.txt' 'viral pneumonia'
 'viral pneumonia.metadata.xlsx']
['covid' 'covidmetadataxlsx' 'lung_opacity' 'lung_opacitymetadataxlsx'
 'normal' 'normalmetadataxlsx' 'readmemdtxt' 'viral pneumonia'
 'viral pneumoniametadataxlsx']
```

```
[11] import numpy as np
     import os
     import shutil
     import random
     import tensorflow as tf
     from tensorflow.keras.preprocessing.image import ImageDataGenerator

     dataset_dir = '/content/COVID-19_Radiography_Dataset/COVID'

     datagen = ImageDataGenerator(
         rotation_range=15,
         width_shift_range=0.1,
         height_shift_range=0.1,
         shear_range=0.1,
         zoom_range=0.1,
         horizontal_flip=True,
         fill_mode="nearest"
     )

     def augment_images(class_dir, num_augmented_images=100):
         augmented_dir = os.path.join(class_dir, 'augmented')
         os.makedirs(augmented_dir, exist_ok=True)

         original_images = [img for img in os.listdir(class_dir) if os.path.isfile(os.path.join(class_dir, img))]
         selected_images = random.sample(original_images, min(num_augmented_images, len(original_images)))

         for image_name in selected_images:
             image_path = os.path.join(class_dir, image_name)
             img = tf.keras.preprocessing.image.load_img(image_path)
             x = tf.keras.preprocessing.image.img_to_array(img)
           x = x.reshape((1,) + x.shape)

           i = 0
           for batch in datagen.flow(x, batch_size=1, save_to_dir=augmented_dir, save_prefix='aug', save_format='jpeg'):
               i += 1
               if i >= 4:
                   break

 for class_name in ['COVID', 'Normal', 'Viral Pneumonia']:
     class_dir = os.path.join(dataset_dir, 'train', class_name)
     augment_images(class_dir, num_augmented_images=100)

 print("Data augmentation completed.")
```

```
Data augmentation completed.
```

Denoising	
Edge Detection	
Color Space Conversion	

Image Cropping	
Batch Normalization	

```
import os
import cv2

dataset_dir = '/content/COVID-19_Radiography_Dataset/COVID'

def crop_images(class_dir, x, y, w, h):
    cropped_dir = os.path.join(class_dir, 'cropped')
    os.makedirs(cropped_dir, exist_ok=True)

    for image_name in os.listdir(class_dir):
        image_path = os.path.join(class_dir, image_name)
        if os.path.isfile(image_path):
            img = cv2.imread(image_path)
            cropped_img = img[y:y+h, x:x+w]
            cropped_image_path = os.path.join(cropped_dir, image_name)
            cv2.imwrite(cropped_image_path, cropped_img)

crop_x = 100
crop_y = 50
crop_width = 300
crop_height = 400

for class_name in ['COVID', 'Normal', 'Viral Pneumonia']:
    class_dir = os.path.join(dataset_dir, 'train', class_name)
    crop_images(class_dir, crop_x, crop_y, crop_width, crop_height)

print("Image cropping completed.")
```

```
Image cropping completed.
```

```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, BatchNormalization, Activation
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os

train_datagen = ImageDataGenerator(rescale=1.0/255.0)
val_datagen = ImageDataGenerator(rescale=1.0/255.0)

# Double-check these paths to ensure they are correct and contain images
train_dir = '/content/COVID-19_Radiography_Dataset/COVID/train'
val_dir = '/content/COVID-19_Radiography_Dataset/COVID/val'

train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary'  # Adjust class_mode if you have more than two classes
)

val_generator = val_datagen.flow_from_directory(
    val_dir,
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary'  # Adjust class_mode if you have more than two classes
)

# Print the number of samples found to check if the directories are empty
print("Number of training samples:", len(train_generator.filenames))
print("Number of validation samples:", len(val_generator.filenames))
```

```
Found 0 images belonging to 3 classes.
Found 0 images belonging to 3 classes.
Number of training samples: 0
Number of validation samples: 0
```