Name: Trung Kien Nguyen

Student ID: 104053642

Studio: 1- 3

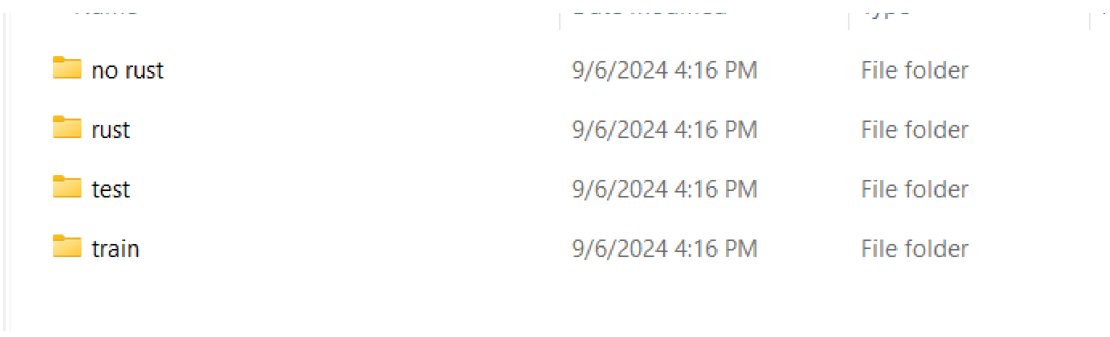
PORTFOLIO – WEEK 5

Step 1: Develop CNN and Resnet50

For step 1, the implementation file can be retrieved at: https://github.com/trungkiennguyen22082004/COS40007_Artificial_Intelligence_for_Engineering/blob/main/Studios/Studio%205/code/studio5_task1.py

Prepare data:

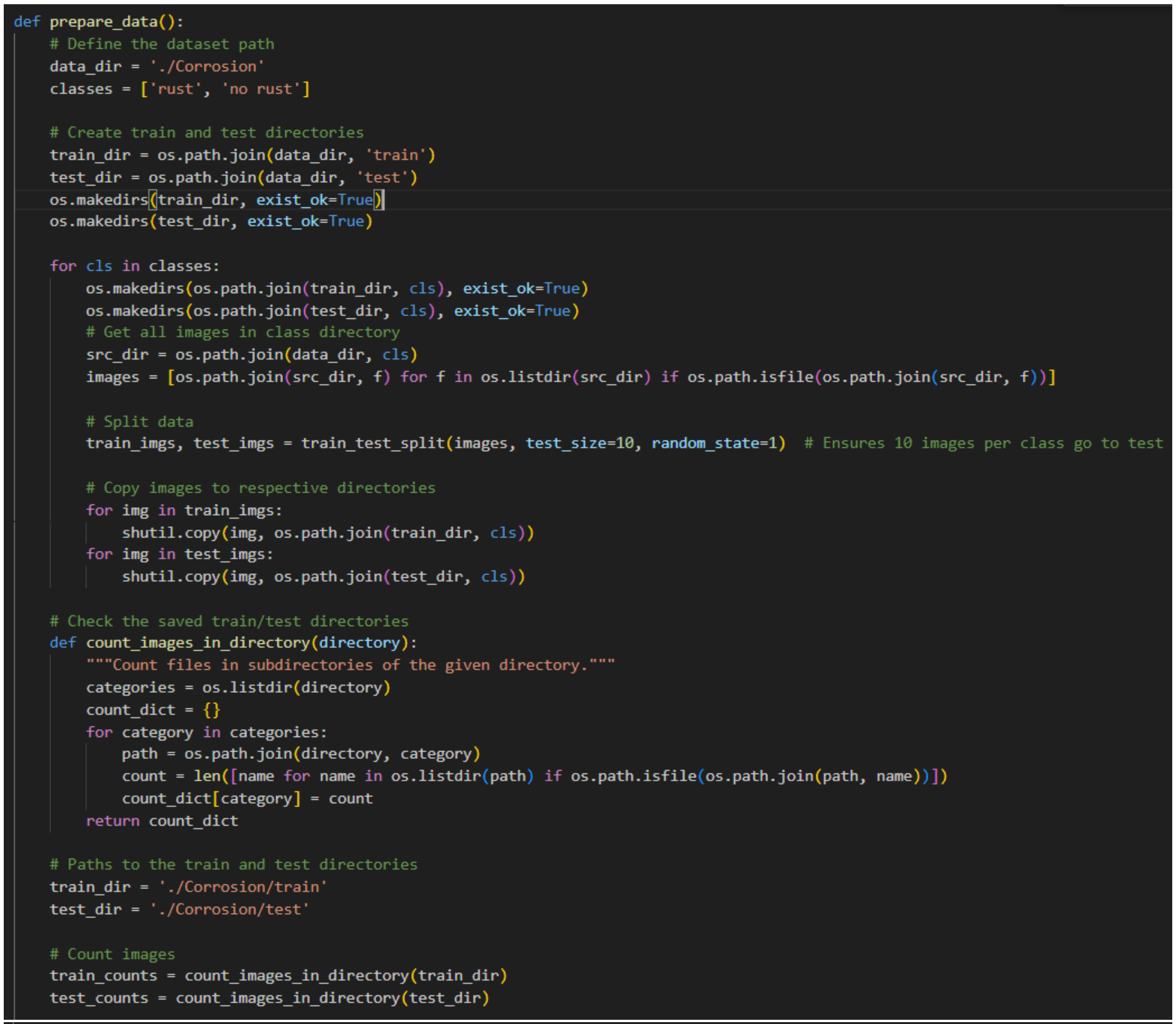
I have taken out 10 rust and 10 no rust images for testing from the directory Corrosion (https://liveswinburneeduau-my.sharepoint.com/personal/fforkan_swin_edu_au/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Ffforkan%5Fswin%5Fedu%5Fau%2FDocuments%2FCOS40007%2FCorrosion%2D20240820T124029Z%2D001%2FCorrosion&ga=1). The outputs are saved in two directories named "test" and "train" inside the Corrosion directory:



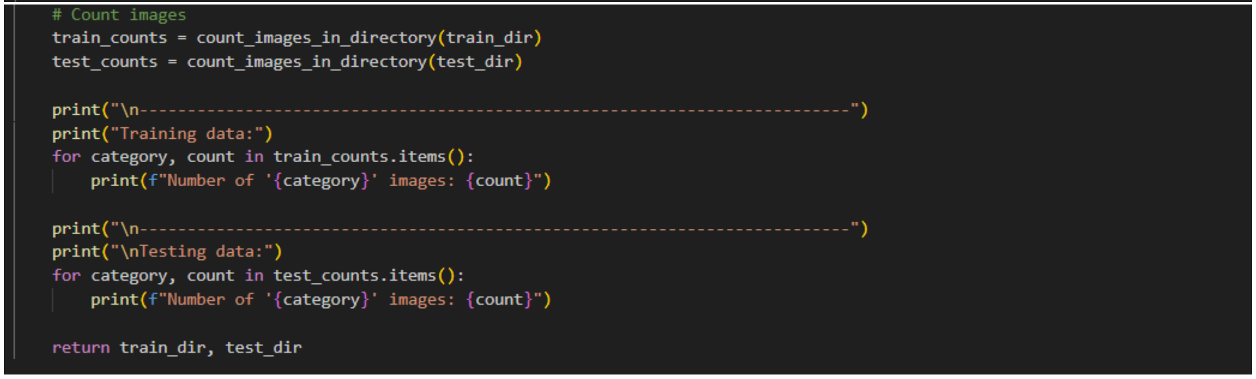
no rust	9/6/2024 4:16 PM	File folder
rust	9/6/2024 4:16 PM	File folder
test	9/6/2024 4:16 PM	File folder
train	9/6/2024 4:16 PM	File folder

- "test" folder: This include 10 images from the original "Corrosion/no rust" directory, saved in the "Corrosion/test/no rust" directory, and 10 images from the original "Corrosion/rust" directory, saved in the "Corrosion/test/rust" directory
- "train" folder: This include the other 5 images from the original "Corrosion/no rust" directory, saved in the "Corrosion/train/no rust" directory, and other 4 images from the original "Corrosion/rust" directory, saved in the "Corrosion/train/rust" directory

- Here is the code to implement that, using os (for managing the directories and files), shutil (for copying image files to appropriate destinations), sklearn.model_selection.train_test_split (for splitting dataset to prepare to save):

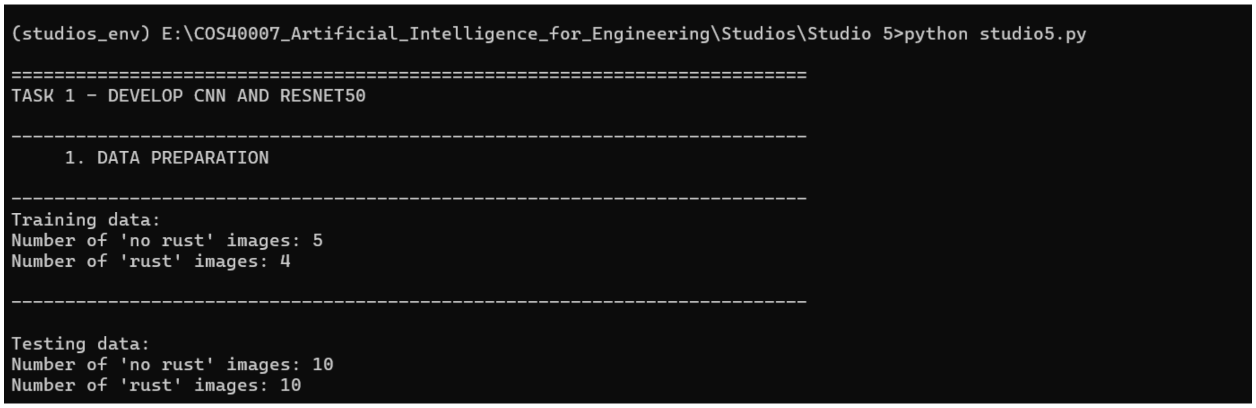


```python
def prepare_data():
    # Define the dataset path
    data_dir = './Corrosion'
    classes = ['rust', 'no rust']

    # Create train and test directories
    train_dir = os.path.join(data_dir, 'train')
    test_dir = os.path.join(data_dir, 'test')
    os.makedirs(train_dir, exist_ok=True)
    os.makedirs(test_dir, exist_ok=True)

    for cls in classes:
        os.makedirs(os.path.join(train_dir, cls), exist_ok=True)
        os.makedirs(os.path.join(test_dir, cls), exist_ok=True)
        # Get all images in class directory
        src_dir = os.path.join(data_dir, cls)
        images = [os.path.join(src_dir, f) for f in os.listdir(src_dir) if os.path.isfile(os.path.join(src_dir, f))]

        # Split data
        train_imgs, test_imgs = train_test_split(images, test_size=10, random_state=1)  # Ensures 10 images per class go to test

        # Copy images to respective directories
        for img in train_imgs:
            shutil.copy(img, os.path.join(train_dir, cls))
        for img in test_imgs:
            shutil.copy(img, os.path.join(test_dir, cls))

    # Check the saved train/test directories
    def count_images_in_directory(directory):
        """Count files in subdirectories of the given directory."""
        categories = os.listdir(directory)
        count_dict = {}
        for category in categories:
            path = os.path.join(directory, category)
            count = len([name for name in os.listdir(path) if os.path.isfile(os.path.join(path, name))])
            count_dict[category] = count
        return count_dict

    # Paths to the train and test directories
    train_dir = './Corrosion/train'
    test_dir = './Corrosion/test'

    # Count images
    train_counts = count_images_in_directory(train_dir)
    test_counts = count_images_in_directory(test_dir)
```



```python
    # Count images
    train_counts = count_images_in_directory(train_dir)
    test_counts = count_images_in_directory(test_dir)

    print("\n-------------------------------------------------------------------------")
    print("Training data:")
    for category, count in train_counts.items():
        print(f"Number of '{category}' images: {count}")

    print("\n-------------------------------------------------------------------------")
    print("\nTesting data:")
    for category, count in test_counts.items():
        print(f"Number of '{category}' images: {count}")

    return train_dir, test_dir
```

Here are the console outputs for this:

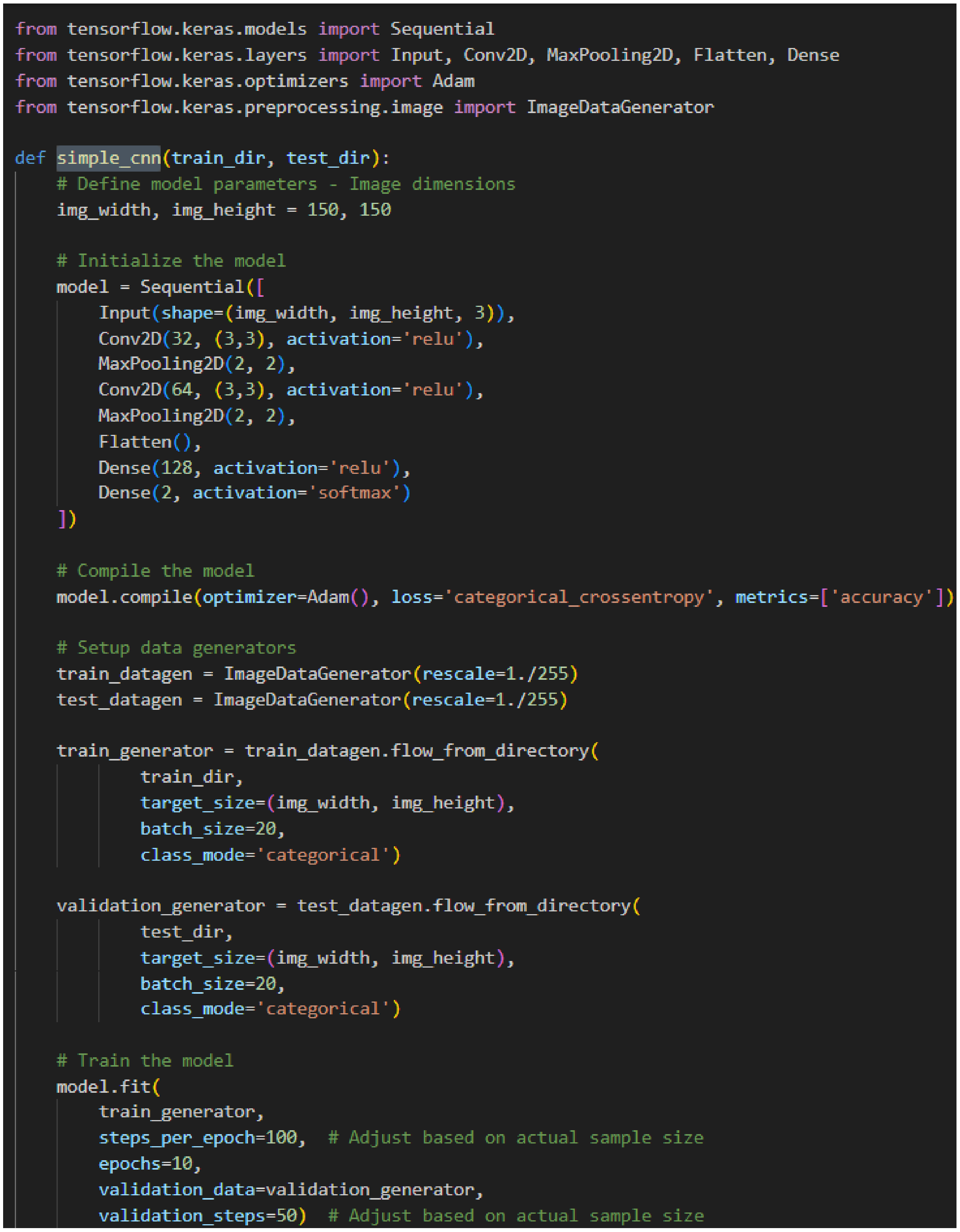


```
(studios_env) E:\COS40007_Artificial_Intelligence_for_Engineering\Studios\Studio 5>python studio5.py

==========================================================================
TASK 1 - DEVELOP CNN AND RESNET50

--------------------------------------------------------------------------
     1. DATA PREPARATION

--------------------------------------------------------------------------
Training data:
Number of 'no rust' images: 5
Number of 'rust' images: 4

--------------------------------------------------------------------------

Testing data:
Number of 'no rust' images: 10
Number of 'rust' images: 10
```

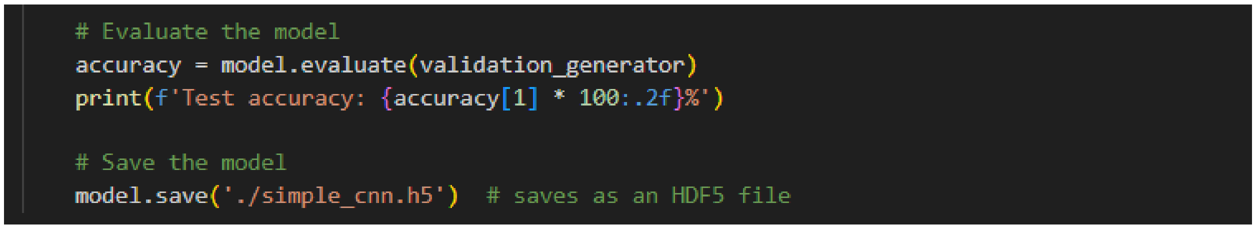
I have also committed and pushed the Studio 5 directory, including the output of images splitting, which can be seen in this link:
https://github.com/trungkiennguyen22082004/COS40007_Artificial_Intelligence_for_Engineering/tree/main/Studios/Studio%205/Corrosion

Simple CNN model

With the train/test data splitted and prepared from Corrosion folder, I have implemented a CNN model in the method simple_cnn, with the help from the package tensorflow.keras:



```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator

def simple_cnn(train_dir, test_dir):
    # Define model parameters - Image dimensions
    img_width, img_height = 150, 150

    # Initialize the model
    model = Sequential([
        Input(shape=(img_width, img_height, 3)),
        Conv2D(32, (3,3), activation='relu'),
        MaxPooling2D(2, 2),
        Conv2D(64, (3,3), activation='relu'),
        MaxPooling2D(2, 2),
        Flatten(),
        Dense(128, activation='relu'),
        Dense(2, activation='softmax')
    ])

    # Compile the model
    model.compile(optimizer=Adam(), loss='categorical_crossentropy', metrics=['accuracy'])

    # Setup data generators
    train_datagen = ImageDataGenerator(rescale=1./255)
    test_datagen = ImageDataGenerator(rescale=1./255)

    train_generator = train_datagen.flow_from_directory(
            train_dir,
            target_size=(img_width, img_height),
            batch_size=20,
            class_mode='categorical')

    validation_generator = test_datagen.flow_from_directory(
            test_dir,
            target_size=(img_width, img_height),
            batch_size=20,
            class_mode='categorical')

    # Train the model
    model.fit(
        train_generator,
        steps_per_epoch=100,  # Adjust based on actual sample size
        epochs=10,
        validation_data=validation_generator,
        validation_steps=50)  # Adjust based on actual sample size
```



```
# Evaluate the model
accuracy = model.evaluate(validation_generator)
print(f'Test accuracy: {accuracy[1] * 100:.2f}%')

# Save the model
model.save('./simple_cnn.h5')  # saves as an HDF5 file
```

The model has been defined as a Sequential model, meaning that layers are added in sequence:

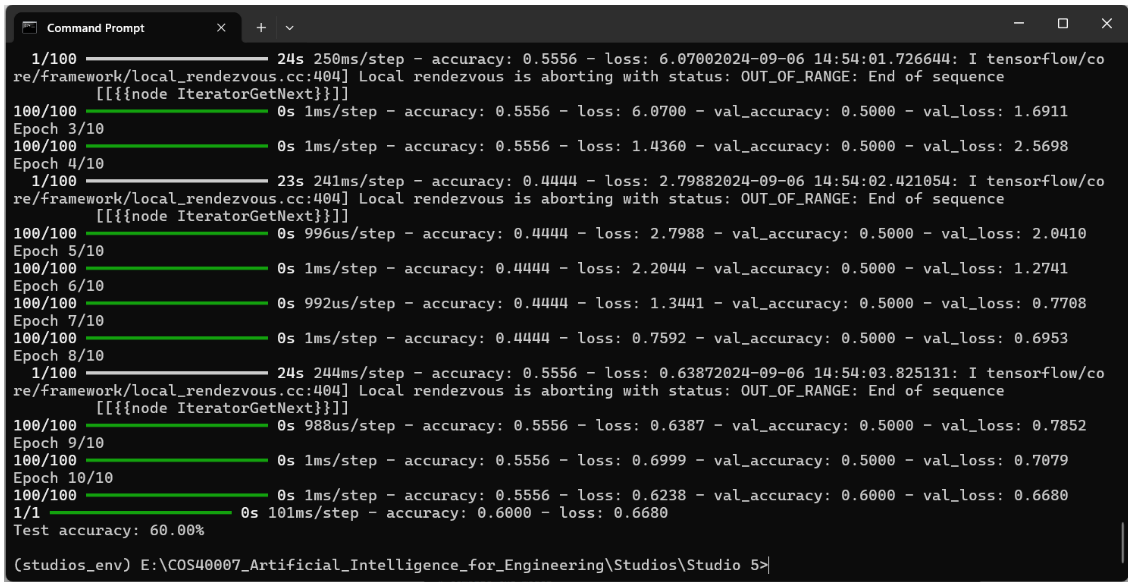
- Input: specify the shape of the input data, which includes image width, height, and 3 color channels (for RGB).
- Conv2D layers are convolutional layers that will help the model to extract features from the input images. There are two of conv2D layers, the first convolutional layer has 32 filters of size 3x3, and the second one has 64 filters of the same size, both using the ReLU activation function.
- MaxPooling2D is used to reduce spartial dimensions after each Conv2D layer
- Flatten convert any 2D features into 1D feature, to fit the next layer – Dense
- Dense layers are fully connected layers. There are two Dense layers used, one is with 128 units (ReLU), and the second is 2-unit (Softmax). So the outout will be the probability distribution between 2 class

The Adam optimizer is used, with a learning objective (loss) set to 'categorical_crossentropy' suitable for multi-class classification.

Next, ImnageDataGenerator from tensorflow.keras.preprocessing.image is used to scale image pixel values to the range from 0 to 1 for model input. These generators will handle loading and modifying images from the suitable directories to batches.

- train_generator will load images from train_dir, which is set to be './Corrosion/train' in the previous step. It resizes the image to a pre-determined size (150x150)
- validation_generator loads validation data in a similar manner from test_dir, which is ‘./Corrosion/test’

After setting up CNN model, I train it with 10 epoch (train_generator), then evaluate it with validation_generator. Here is the output for these tasks:



```
  1/100 ━━━━━━━━━━━━━━━━━━━━ 24s 250ms/step - accuracy: 0.5556 - loss: 6.07002024-09-06 14:54:01.726644: I tensorflow/co
re/framework/local_rendezvous.cc:404] Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
         [[{{node IteratorGetNext}}]]
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5556 - loss: 6.0700 - val_accuracy: 0.5000 - val_loss: 1.6911
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5556 - loss: 1.4360 - val_accuracy: 0.5000 - val_loss: 2.5698
Epoch 4/10
  1/100 ━━━━━━━━━━━━━━━━━━━━ 23s 241ms/step - accuracy: 0.4444 - loss: 2.79882024-09-06 14:54:02.421054: I tensorflow/co
re/framework/local_rendezvous.cc:404] Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
         [[{{node IteratorGetNext}}]]
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 996us/step - accuracy: 0.4444 - loss: 2.7988 - val_accuracy: 0.5000 - val_loss: 2.0410
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4444 - loss: 2.2044 - val_accuracy: 0.5000 - val_loss: 1.2741
Epoch 6/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 992us/step - accuracy: 0.4444 - loss: 1.3441 - val_accuracy: 0.5000 - val_loss: 0.7708
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.4444 - loss: 0.7592 - val_accuracy: 0.5000 - val_loss: 0.6953
Epoch 8/10
  1/100 ━━━━━━━━━━━━━━━━━━━━ 24s 244ms/step - accuracy: 0.5556 - loss: 0.63872024-09-06 14:54:03.825131: I tensorflow/co
re/framework/local_rendezvous.cc:404] Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
         [[{{node IteratorGetNext}}]]
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 988us/step - accuracy: 0.5556 - loss: 0.6387 - val_accuracy: 0.5000 - val_loss: 0.7852
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5556 - loss: 0.6999 - val_accuracy: 0.5000 - val_loss: 0.7079
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 1ms/step - accuracy: 0.5556 - loss: 0.6238 - val_accuracy: 0.6000 - val_loss: 0.6680
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 101ms/step - accuracy: 0.6000 - loss: 0.6680
Test accuracy: 60.00%

(studios_env) E:\COS40007_Artificial_Intelligence_for_Engineering\Studios\Studio 5>
```

Note that the test set is quite small, so the index 60% may not remain the same for all cases. The accuracy may vary between 50% to 60%

The simple CNN model after training is saved in the file simple_cnn.h5 (link: https://drive.google.com/file/d/1IqoA6381heVwb7voQ5cq1C4Uq_8yeiLG/view?usp=sharing)

ResNet50

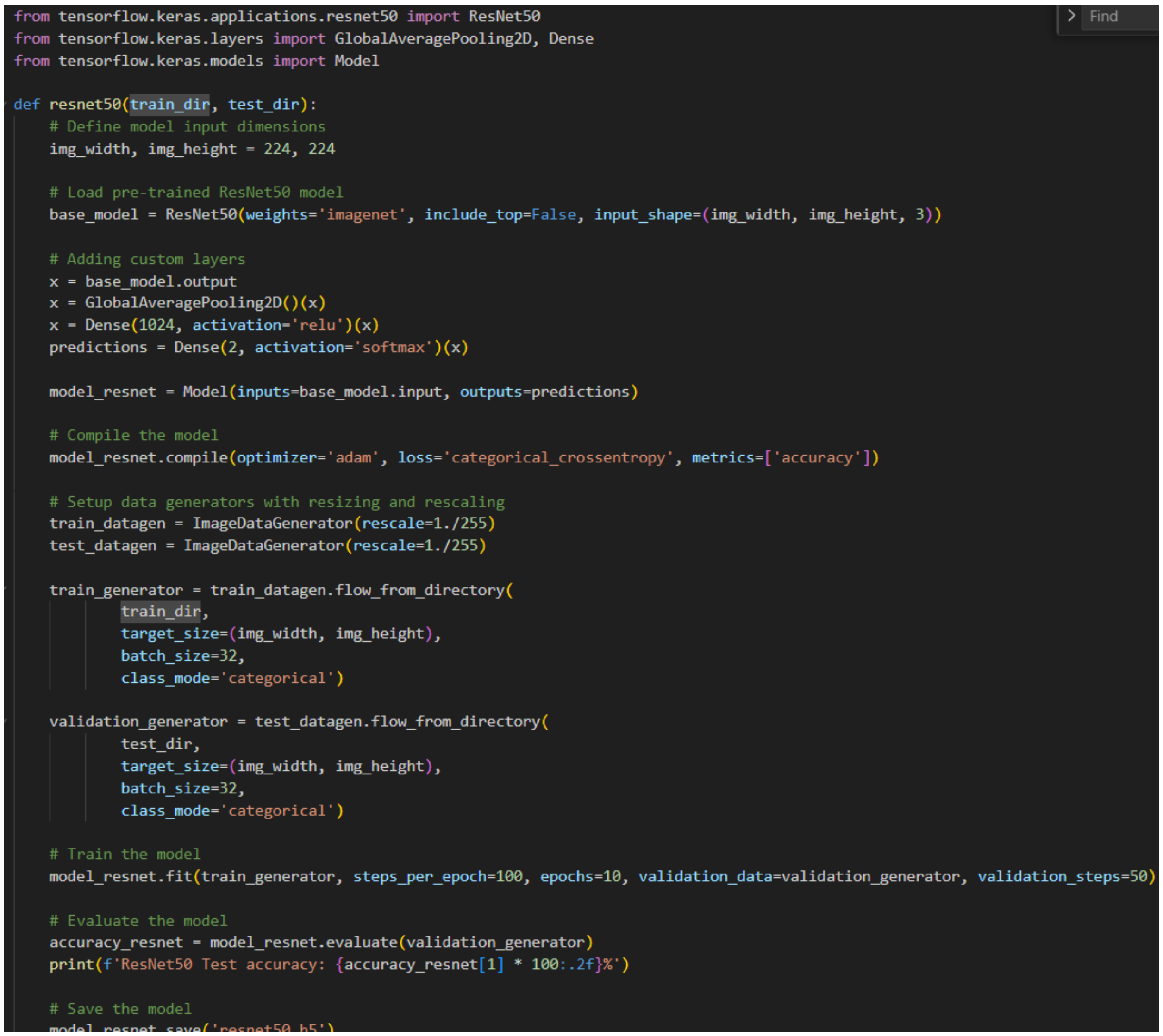
Using the same prepared dataset from Corrosion, I have also set up, trained and tested the Resnet50 model. Actually, I have defined a base model - a pre-trained one, with parameters as follows:

- weights: “imagenet”, which is a large visual database containinng over 14 million annotated images categorized into over 20,000 categories.
- input_shapes is (224, 224, 3), with 224x224 is the average dimensions of images data observed by me, while 3 represents the three basic color: Red, Green and Blue.

I then customized this base model with the following layers:

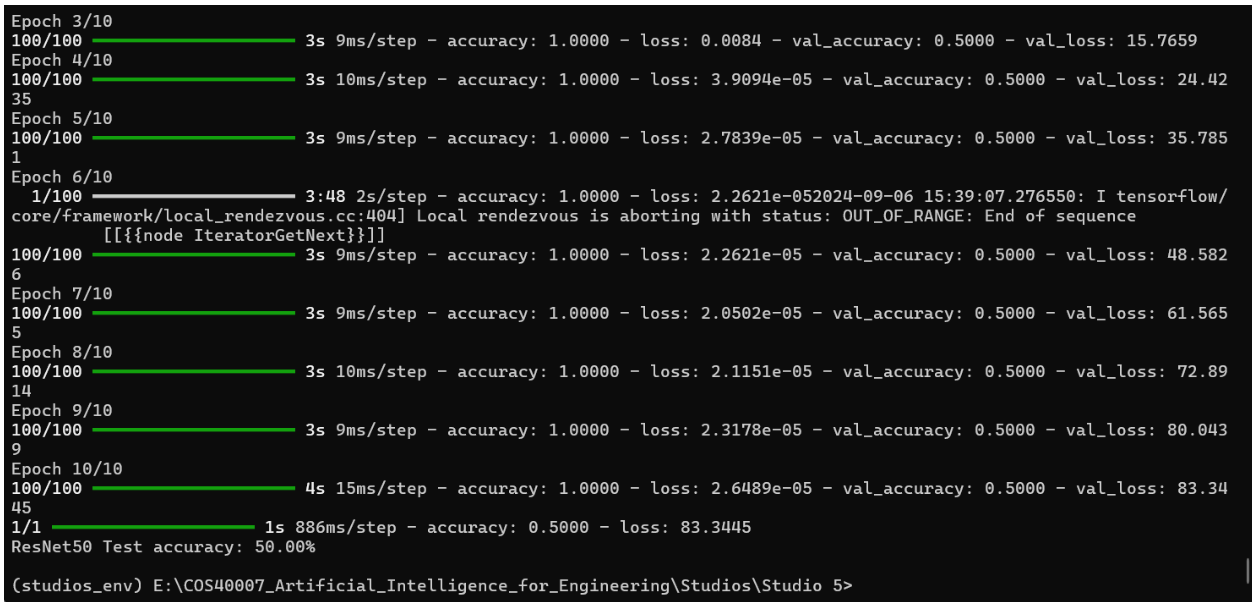
- a dense layer with 1024 units (ReLU activation function).
- A dense layse as the final layer (Softmax)m outputting probabilities for two classe similar to the above CNN

Then, I set up my final Resnet50 model (model_resnet), use the input from the base model. The train and test data is the same train_generator and validation_generator like the above CNN, with the epoch of 10 in training process. Here is the implementation in the resnet50 method:



```
from tensorflow.keras.applications.resnet50 import ResNet50
from tensorflow.keras.layers import GlobalAveragePooling2D, Dense
from tensorflow.keras.models import Model

def resnet50(train_dir, test_dir):
    # Define model input dimensions
    img_width, img_height = 224, 224

    # Load pre-trained ResNet50 model
    base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(img_width, img_height, 3))

    # Adding custom layers
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(1024, activation='relu')(x)
    predictions = Dense(2, activation='softmax')(x)

    model_resnet = Model(inputs=base_model.input, outputs=predictions)

    # Compile the model
    model_resnet.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])

    # Setup data generators with resizing and rescaling
    train_datagen = ImageDataGenerator(rescale=1./255)
    test_datagen = ImageDataGenerator(rescale=1./255)

    train_generator = train_datagen.flow_from_directory(
            train_dir,
            target_size=(img_width, img_height),
            batch_size=32,
            class_mode='categorical')

    validation_generator = test_datagen.flow_from_directory(
            test_dir,
            target_size=(img_width, img_height),
            batch_size=32,
            class_mode='categorical')

    # Train the model
    model_resnet.fit(train_generator, steps_per_epoch=100, epochs=10, validation_data=validation_generator, validation_steps=50)

    # Evaluate the model
    accuracy_resnet = model_resnet.evaluate(validation_generator)
    print(f'ResNet50 Test accuracy: {accuracy_resnet[1] * 100:.2f}%')

    # Save the model
    model_resnet.save('resnet50.h5')
```

Here is the output of training/testing ResNet50:



```
Epoch 3/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 1.0000 - loss: 0.0084 - val_accuracy: 0.5000 - val_loss: 15.7659
Epoch 4/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - accuracy: 1.0000 - loss: 3.9094e-05 - val_accuracy: 0.5000 - val_loss: 24.42
35
Epoch 5/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 1.0000 - loss: 2.7839e-05 - val_accuracy: 0.5000 - val_loss: 35.785
1
Epoch 6/10
  1/100 ━━━━━━━━━━━━━━━━━━━━ 3:48 2s/step - accuracy: 1.0000 - loss: 2.2621e-052024-09-06 15:39:07.276550: I tensorflow/
core/framework/local_rendezvous.cc:404] Local rendezvous is aborting with status: OUT_OF_RANGE: End of sequence
         [[{{node IteratorGetNext}}]]
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 1.0000 - loss: 2.2621e-05 - val_accuracy: 0.5000 - val_loss: 48.582
6
Epoch 7/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 1.0000 - loss: 2.0502e-05 - val_accuracy: 0.5000 - val_loss: 61.565
5
Epoch 8/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 10ms/step - accuracy: 1.0000 - loss: 2.1151e-05 - val_accuracy: 0.5000 - val_loss: 72.89
14
Epoch 9/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 3s 9ms/step - accuracy: 1.0000 - loss: 2.3178e-05 - val_accuracy: 0.5000 - val_loss: 80.043
9
Epoch 10/10
100/100 ━━━━━━━━━━━━━━━━━━━━ 4s 15ms/step - accuracy: 1.0000 - loss: 2.6489e-05 - val_accuracy: 0.5000 - val_loss: 83.34
45
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 886ms/step - accuracy: 0.5000 - loss: 83.3445
ResNet50 Test accuracy: 50.00%

(studios_env) E:\COS40007_Artificial_Intelligence_for_Engineering\Studios\Studio 5>
```

Similar to the above CNN model, the test dataset is quite small (10 images), so the accuracy can be from 50% to 60% while running multiple times

The trained Resnet 50 model is saved in "resnet50.h5" (link: https://drive.google.com/file/d/1piKyUEdOvssooywZ6U6MDbRUxxQN7qKN/view?usp=sharing)

Step 2: Develop Mask RCNN for detecting log

Here is the link for the implementation of step 2: https://github.com/trungkiennguyen22082004/COS40007_Artificial_Intelligence_for_Engineering/blob/main/Studios/Studio%205/code/studio5_task2.py

Prepare data

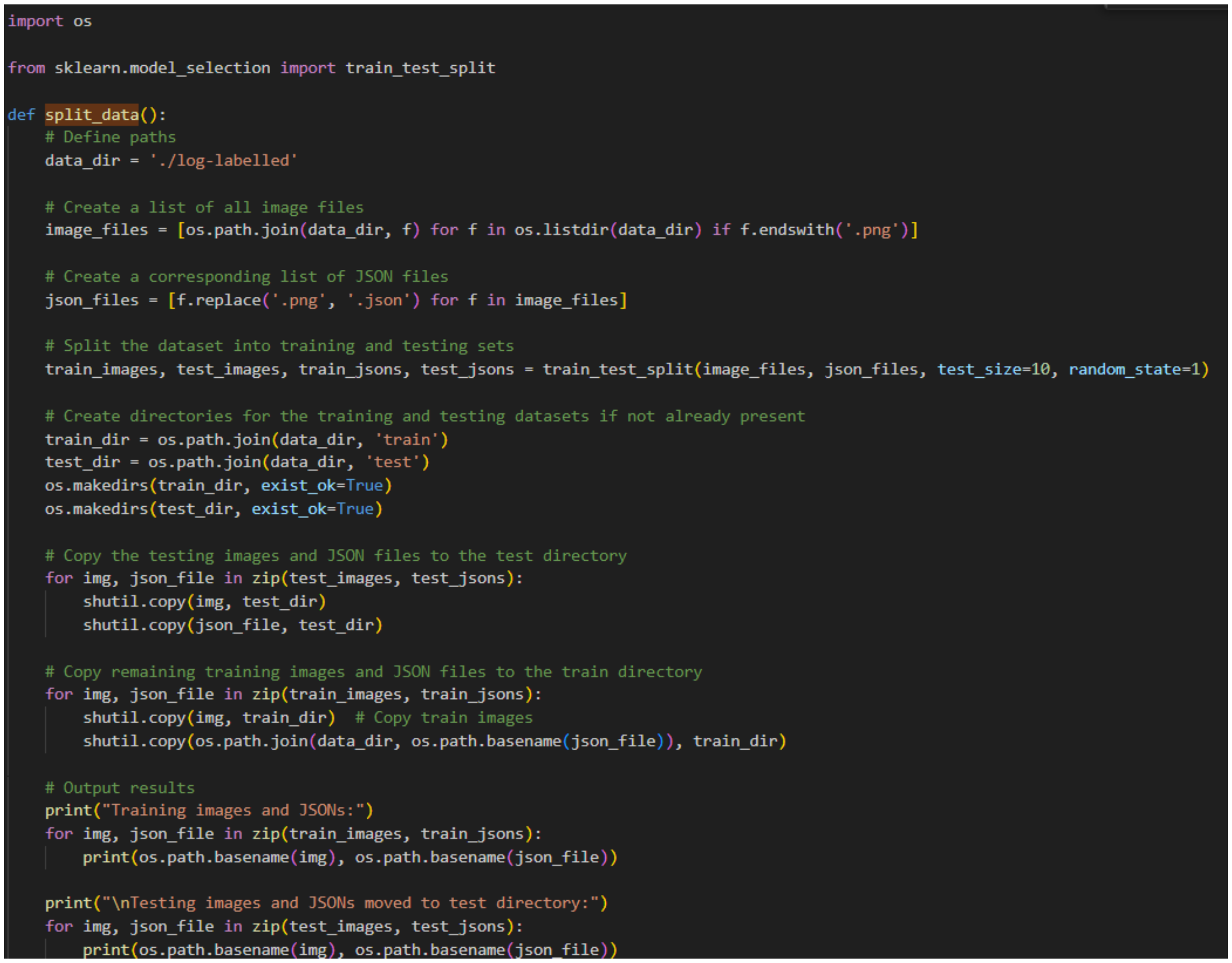
Split data

The data for this task is from the directory log-labelled (https://liveswinburneeduau-my.sharepoint.com/personal/fforkan_swin_edu_au/_layouts/15/onedrive.aspx?id=%2Fpersonal%2Ffforkan%5Fswin%5Fedu%5Fau%2FDocuments%2FCOS40007%2Flog%2Dlabelled&ga=1). This dataset included 600 image files, and 600 corresponding labelled JSON files for those images

Firstly, I have splitted data into test set and train set. The test set includes 10 images and 10 corresponding JSON labelled files, while the train set includes the other 590 pairs. You can view the log-labelled/train and log-labelled/test in the links below:

- log-labelled/train: https://github.com/trungkiennguyen22082004/COS40007_Artificial_Intelligence_for_Engineering/tree/main/Studios/Studio%205/log-labelled/train
- log-labelled/test: https://github.com/trungkiennguyen22082004/COS40007_Artificial_Intelligence_for_Engineering/tree/main/Studios/Studio%205/log-labelled/test

Here is the implementation for the splitting task, in the method split_data:



```python
import os

from sklearn.model_selection import train_test_split

def split_data():
    # Define paths
    data_dir = './log-labelled'

    # Create a list of all image files
    image_files = [os.path.join(data_dir, f) for f in os.listdir(data_dir) if f.endswith('.png')]

    # Create a corresponding list of JSON files
    json_files = [f.replace('.png', '.json') for f in image_files]

    # Split the dataset into training and testing sets
    train_images, test_images, train_jsons, test_jsons = train_test_split(image_files, json_files, test_size=10, random_state=1)

    # Create directories for the training and testing datasets if not already present
    train_dir = os.path.join(data_dir, 'train')
    test_dir = os.path.join(data_dir, 'test')
    os.makedirs(train_dir, exist_ok=True)
    os.makedirs(test_dir, exist_ok=True)

    # Copy the testing images and JSON files to the test directory
    for img, json_file in zip(test_images, test_jsons):
        shutil.copy(img, test_dir)
        shutil.copy(json_file, test_dir)

    # Copy remaining training images and JSON files to the train directory
    for img, json_file in zip(train_images, train_jsons):
        shutil.copy(img, train_dir)  # Copy train images
        shutil.copy(os.path.join(data_dir, os.path.basename(json_file)), train_dir)

    # Output results
    print("Training images and JSONs:")
    for img, json_file in zip(train_images, train_jsons):
        print(os.path.basename(img), os.path.basename(json_file))

    print("\nTesting images and JSONs moved to test directory:")
    for img, json_file in zip(test_images, test_jsons):
        print(os.path.basename(img), os.path.basename(json_file))
```

Convert to appropriate format for Touchvision's Mask RCNN

The library for Mask RCNN model is torchvision.models.detection. To set up this model, I first have to convert the train/test dataset to appropiate format. The method generate_coco_annotations use the pairs of image and corresponding labelled JSON files, to generate one final annotations JSON file. (That means, one annotations file for test dataset and one annotations file for train set).

A COCO annotations file provides the following key information:

- Images: meta data of each image in the dataset (file name, id, size, ...)
- Categories: This is different types of objects that could be recognized.
- Annotations:
    - Image ID
    - Categori ID show the type of annotated object
    - Segmentation: For segmentation tasks, it offers the object's pixel-level shape. This could take the shape of a binary mask or polygon.
    - Bounding box: specify the coordinate of the rectangle enclosing each object
    - Area
    - IsCrowd: if the objects distribution is crowd or not

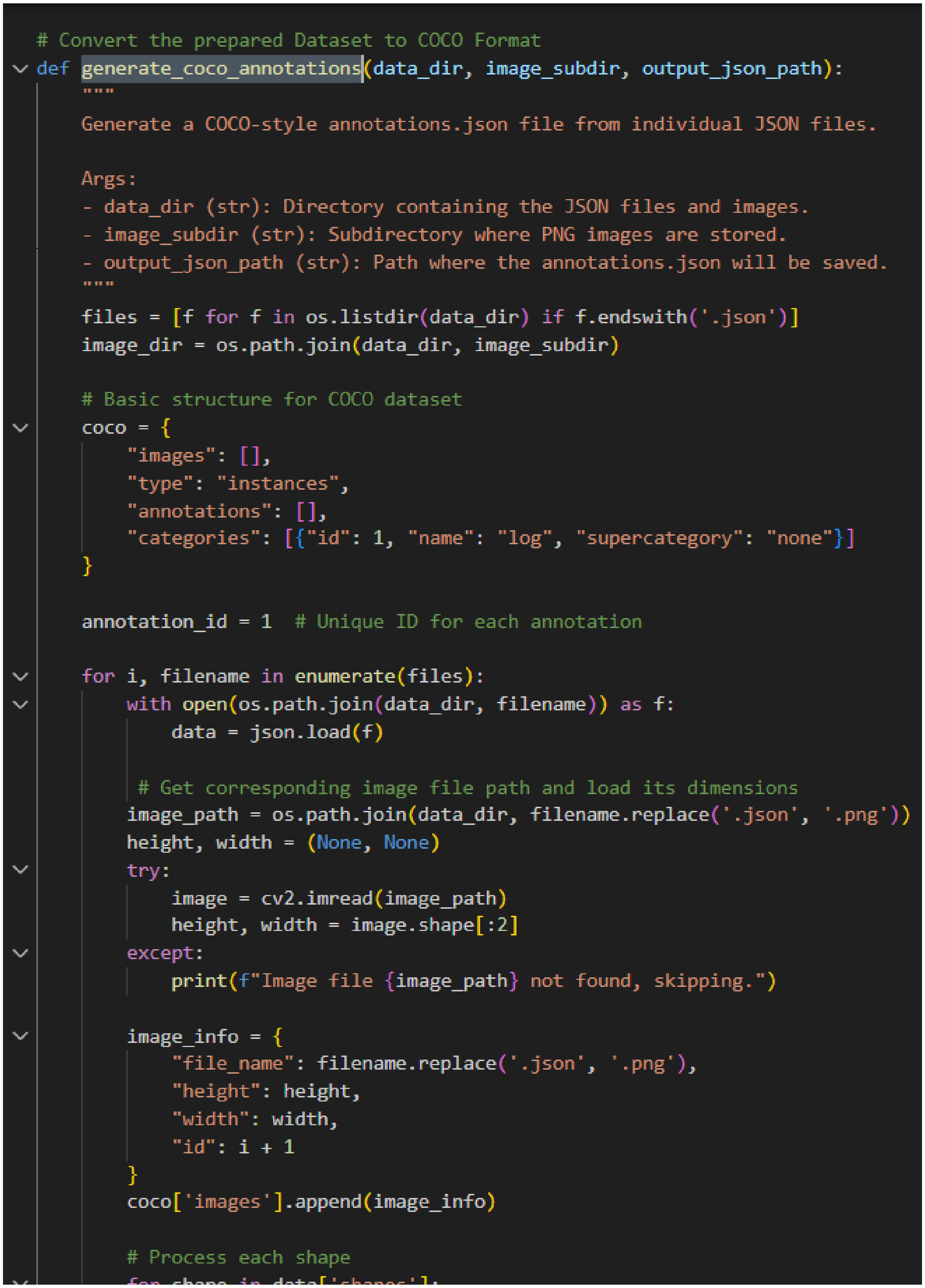
In terms of inplementation, after loading images and corresponding labelled jjson files with os, I have intialized the COCO structure, which is a dictionary (coco) including keys like images, type, annotations and categories

There is only one category in the dataset, which is “log”. I have also specified the corresponding id for this is 1.

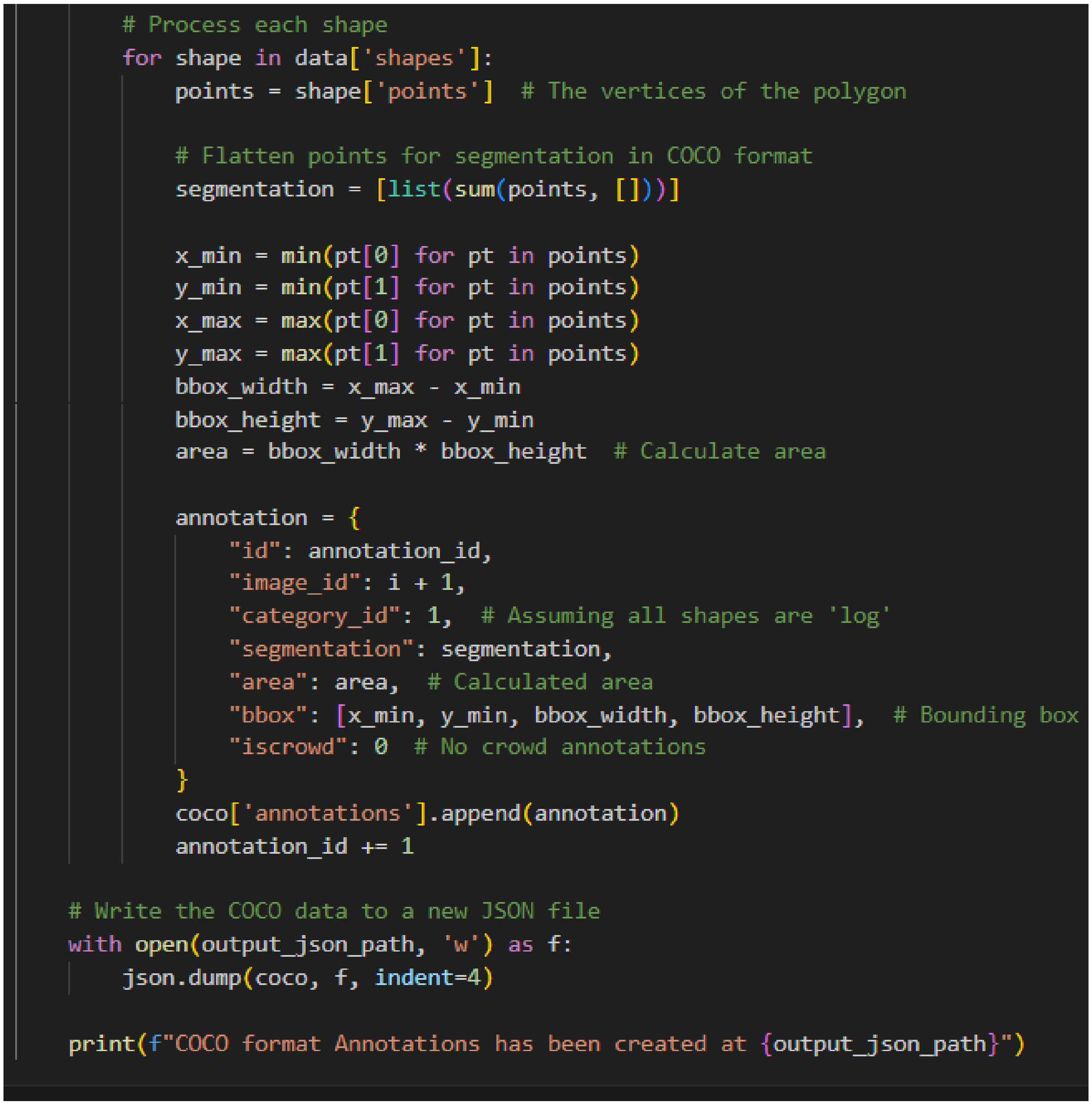
Then I processed the annotations and converted to the format that COCO required:

- Information about segmentation is compressed and kept.
- determines the minimum and maximum x and y coordinates from the points to compute the bounding box (bbox), after which the width and height are determined.
- determines the bounding box's area.
- Adds the following annotation data to coco['annotations']: id, calculated area, bbox, and iscrowd set to 0 (signifying that these are individual object instances rather than crowd annotations).

Here is the implementation of method generate_coco_annotations, which is used for both train set and test set:



```python
# Convert the prepared Dataset to COCO Format
def generate_coco_annotations(data_dir, image_subdir, output_json_path):
    """
    Generate a COCO-style annotations.json file from individual JSON files.

    Args:
    - data_dir (str): Directory containing the JSON files and images.
    - image_subdir (str): Subdirectory where PNG images are stored.
    - output_json_path (str): Path where the annotations.json will be saved.
    """
    files = [f for f in os.listdir(data_dir) if f.endswith('.json')]
    image_dir = os.path.join(data_dir, image_subdir)

    # Basic structure for COCO dataset
    coco = {
        "images": [],
        "type": "instances",
        "annotations": [],
        "categories": [{"id": 1, "name": "log", "supercategory": "none"}]
    }

    annotation_id = 1  # Unique ID for each annotation

    for i, filename in enumerate(files):
        with open(os.path.join(data_dir, filename)) as f:
            data = json.load(f)

         # Get corresponding image file path and load its dimensions
        image_path = os.path.join(data_dir, filename.replace('.json', '.png'))
        height, width = (None, None)
        try:
            image = cv2.imread(image_path)
            height, width = image.shape[:2]
        except:
            print(f"Image file {image_path} not found, skipping.")

        image_info = {
            "file_name": filename.replace('.json', '.png'),
            "height": height,
            "width": width,
            "id": i + 1
        }
        coco['images'].append(image_info)

        # Process each shape
```



```
        # Process each shape
        for shape in data['shapes']:
            points = shape['points']  # The vertices of the polygon

            # Flatten points for segmentation in COCO format
            segmentation = [list(sum(points, []))]

            x_min = min(pt[0] for pt in points)
            y_min = min(pt[1] for pt in points)
            x_max = max(pt[0] for pt in points)
            y_max = max(pt[1] for pt in points)
            bbox_width = x_max - x_min
            bbox_height = y_max - y_min
            area = bbox_width * bbox_height  # Calculate area

            annotation = {
                "id": annotation_id,
                "image_id": i + 1,
                "category_id": 1,  # Assuming all shapes are 'log'
                "segmentation": segmentation,
                "area": area,  # Calculated area
                "bbox": [x_min, y_min, bbox_width, bbox_height],  # Bounding box
                "iscrowd": 0  # No crowd annotations
            }
            coco['annotations'].append(annotation)
            annotation_id += 1

    # Write the COCO data to a new JSON file
    with open(output_json_path, 'w') as f:
        json.dump(coco, f, indent=4)

    print(f"COCO format Annotations has been created at {output_json_path}")
```

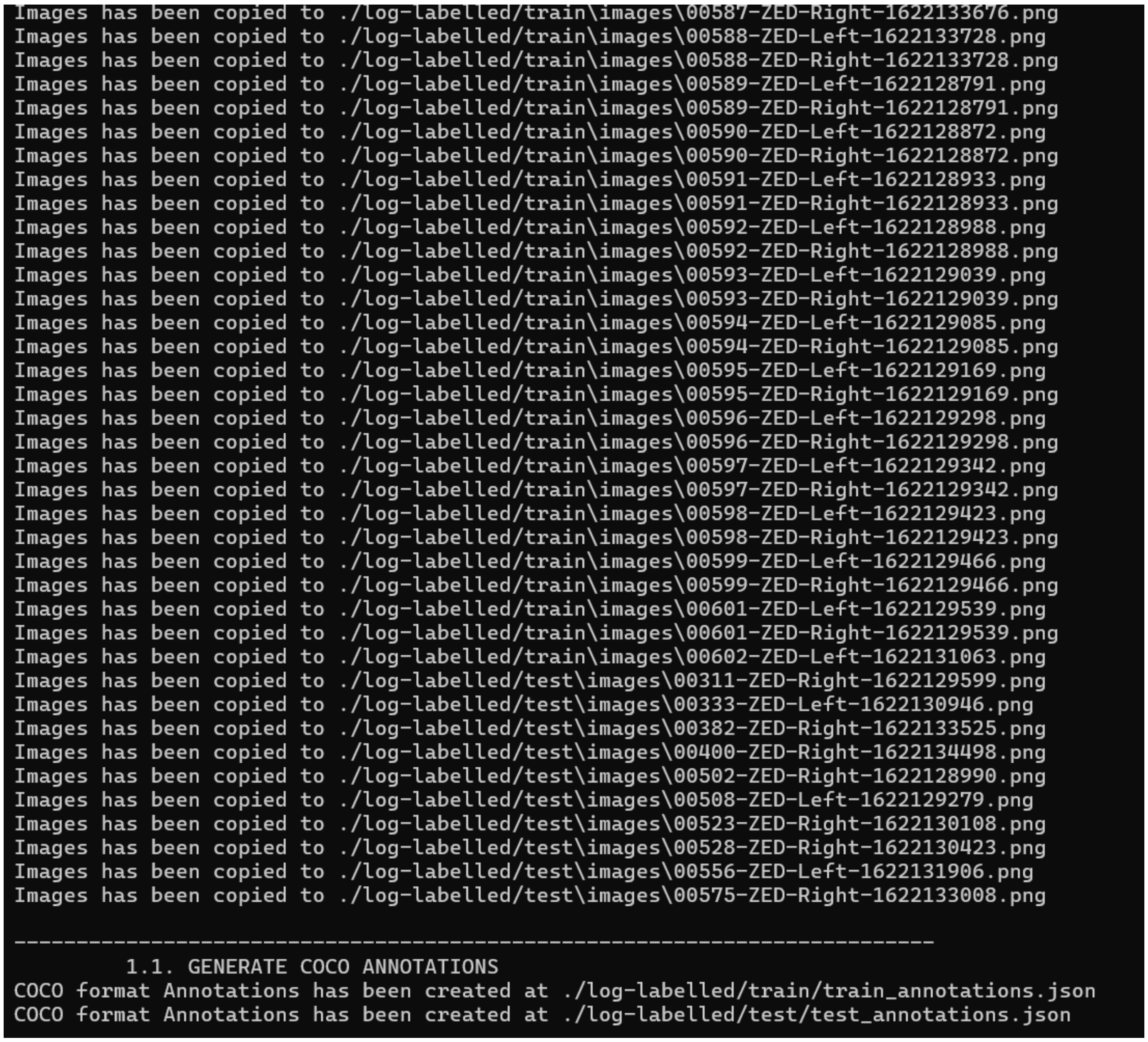
The two created annotations with coco format is saved in "log-labelled/train/" and "log-labelled/test/" directories respectively. Here are the links for those:

- train_annotations.json: https://github.com/trungkiennguyen22082004/COS40007_Artificial_Intelligence_for_Engineering/blob/main/Studios/Studio%205/log-labelled/train/train_annotations.json
- test_annotation.json: https://github.com/trungkiennguyen22082004/COS40007_Artificial_Intelligence_for_Engineering/blob/main/Studios/Studio%205/log-labelled/test/test_annotations.json

Also, I have create a method call setup_image_directories, using shutil and os to move all the images from each dataset (train/test) to separate folders of "images", just for easier model implementing later. Here are the links for:

- "log-labelled/test/images": https://github.com/trungkiennguyen22082004/COS40007_Artificial_Intelligence_for_Engineering/tree/main/Studios/Studio%205/log-labelled/test/images
- "log-labelled/train/images": https://github.com/trungkiennguyen22082004/COS40007_Artificial_Intelligence_for_Engineering/tree/main/Studios/Studio%205/log-labelled/train/images

Here is the console output for COCO-suite format converting:



```
Images has been copied to ./log-labelled/train\images\00587-ZED-Right-1622133676.png
Images has been copied to ./log-labelled/train\images\00588-ZED-Left-1622133728.png
Images has been copied to ./log-labelled/train\images\00588-ZED-Right-1622133728.png
Images has been copied to ./log-labelled/train\images\00589-ZED-Left-1622128791.png
Images has been copied to ./log-labelled/train\images\00589-ZED-Right-1622128791.png
Images has been copied to ./log-labelled/train\images\00590-ZED-Left-1622128872.png
Images has been copied to ./log-labelled/train\images\00590-ZED-Right-1622128872.png
Images has been copied to ./log-labelled/train\images\00591-ZED-Left-1622128933.png
Images has been copied to ./log-labelled/train\images\00591-ZED-Right-1622128933.png
Images has been copied to ./log-labelled/train\images\00592-ZED-Left-1622128988.png
Images has been copied to ./log-labelled/train\images\00592-ZED-Right-1622128988.png
Images has been copied to ./log-labelled/train\images\00593-ZED-Left-1622129039.png
Images has been copied to ./log-labelled/train\images\00593-ZED-Right-1622129039.png
Images has been copied to ./log-labelled/train\images\00594-ZED-Left-1622129085.png
Images has been copied to ./log-labelled/train\images\00594-ZED-Right-1622129085.png
Images has been copied to ./log-labelled/train\images\00595-ZED-Left-1622129169.png
Images has been copied to ./log-labelled/train\images\00595-ZED-Right-1622129169.png
Images has been copied to ./log-labelled/train\images\00596-ZED-Left-1622129298.png
Images has been copied to ./log-labelled/train\images\00596-ZED-Right-1622129298.png
Images has been copied to ./log-labelled/train\images\00597-ZED-Left-1622129342.png
Images has been copied to ./log-labelled/train\images\00597-ZED-Right-1622129342.png
Images has been copied to ./log-labelled/train\images\00598-ZED-Left-1622129423.png
Images has been copied to ./log-labelled/train\images\00598-ZED-Right-1622129423.png
Images has been copied to ./log-labelled/train\images\00599-ZED-Left-1622129466.png
Images has been copied to ./log-labelled/train\images\00599-ZED-Right-1622129466.png
Images has been copied to ./log-labelled/train\images\00601-ZED-Left-1622129539.png
Images has been copied to ./log-labelled/train\images\00601-ZED-Right-1622129539.png
Images has been copied to ./log-labelled/train\images\00602-ZED-Left-1622131063.png
Images has been copied to ./log-labelled/test\images\00311-ZED-Right-1622129599.png
Images has been copied to ./log-labelled/test\images\00333-ZED-Left-1622130946.png
Images has been copied to ./log-labelled/test\images\00382-ZED-Right-1622133525.png
Images has been copied to ./log-labelled/test\images\00400-ZED-Right-1622134498.png
Images has been copied to ./log-labelled/test\images\00502-ZED-Right-1622128990.png
Images has been copied to ./log-labelled/test\images\00508-ZED-Left-1622129279.png
Images has been copied to ./log-labelled/test\images\00523-ZED-Right-1622130108.png
Images has been copied to ./log-labelled/test\images\00528-ZED-Right-1622130423.png
Images has been copied to ./log-labelled/test\images\00556-ZED-Left-1622131906.png
Images has been copied to ./log-labelled/test\images\00575-ZED-Right-1622133008.png

--------------------------------------------------------------------------
         1.1. GENERATE COCO ANNOTATIONS
COCO format Annotations has been created at ./log-labelled/train/train_annotations.json
COCO format Annotations has been created at ./log-labelled/test/test_annotations.json
```

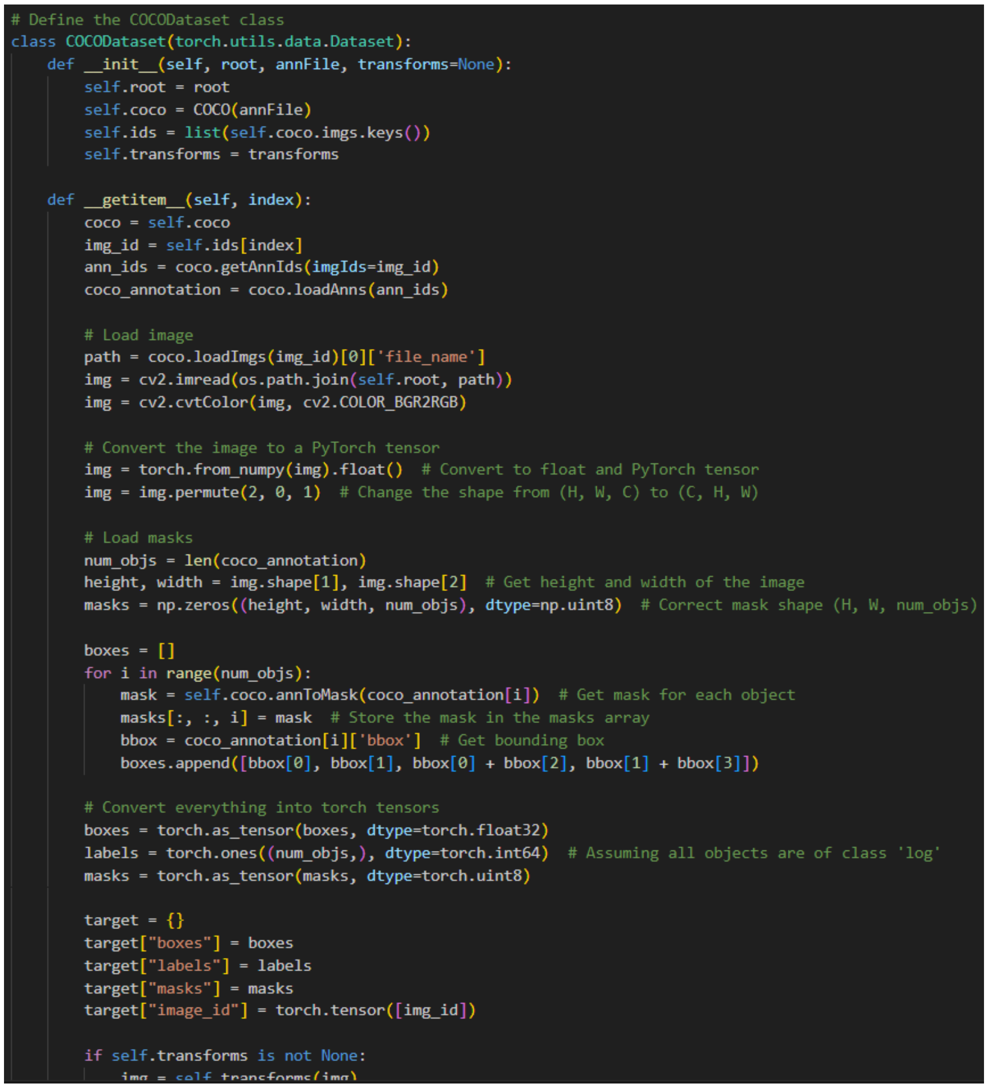
Mask RCNN model setting and traininig

The train_model method is responsible for this task.

Initially, I have implement the class of COCODataset (inheritted from torch.utils.data.Dataset), to summarise the images saved in the images file and the coco-suite annotations JSON file, from each dataset (train/test). This class has three methods that have been overidden by me:

- Constructer (__init__): The annotation file, dataset root directory, and any image transformations to be applied are loaded
- __getitem__: This fetchs a single data sample (e.g. when we index the dataset)
- __len__: simply return the length of the dataset

Here is the implementation for this class:



```
# Define the COCODataset class
class COCODataset(torch.utils.data.Dataset):
    def __init__(self, root, annFile, transforms=None):
        self.root = root
        self.coco = COCO(annFile)
        self.ids = list(self.coco.imgs.keys())
        self.transforms = transforms

    def __getitem__(self, index):
        coco = self.coco
        img_id = self.ids[index]
        ann_ids = coco.getAnnIds(imgIds=img_id)
        coco_annotation = coco.loadAnns(ann_ids)

        # Load image
        path = coco.loadImgs(img_id)[0]['file_name']
        img = cv2.imread(os.path.join(self.root, path))
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        # Convert the image to a PyTorch tensor
        img = torch.from_numpy(img).float()  # Convert to float and PyTorch tensor
        img = img.permute(2, 0, 1)  # Change the shape from (H, W, C) to (C, H, W)

        # Load masks
        num_objs = len(coco_annotation)
        height, width = img.shape[1], img.shape[2]  # Get height and width of the image
        masks = np.zeros((height, width, num_objs), dtype=np.uint8)  # Correct mask shape (H, W, num_objs)

        boxes = []
        for i in range(num_objs):
            mask = self.coco.annToMask(coco_annotation[i])  # Get mask for each object
            masks[:, :, i] = mask  # Store the mask in the masks array
            bbox = coco_annotation[i]['bbox']  # Get bounding box
            boxes.append([bbox[0], bbox[1], bbox[0] + bbox[2], bbox[1] + bbox[3]])

        # Convert everything into torch tensors
        boxes = torch.as_tensor(boxes, dtype=torch.float32)
        labels = torch.ones((num_objs,), dtype=torch.int64)  # Assuming all objects are of class 'log'
        masks = torch.as_tensor(masks, dtype=torch.uint8)

        target = {}
        target["boxes"] = boxes
        target["labels"] = labels
        target["masks"] = masks
        target["image_id"] = torch.tensor([img_id])

        if self.transforms is not None:
            img = self.transforms(img)
```

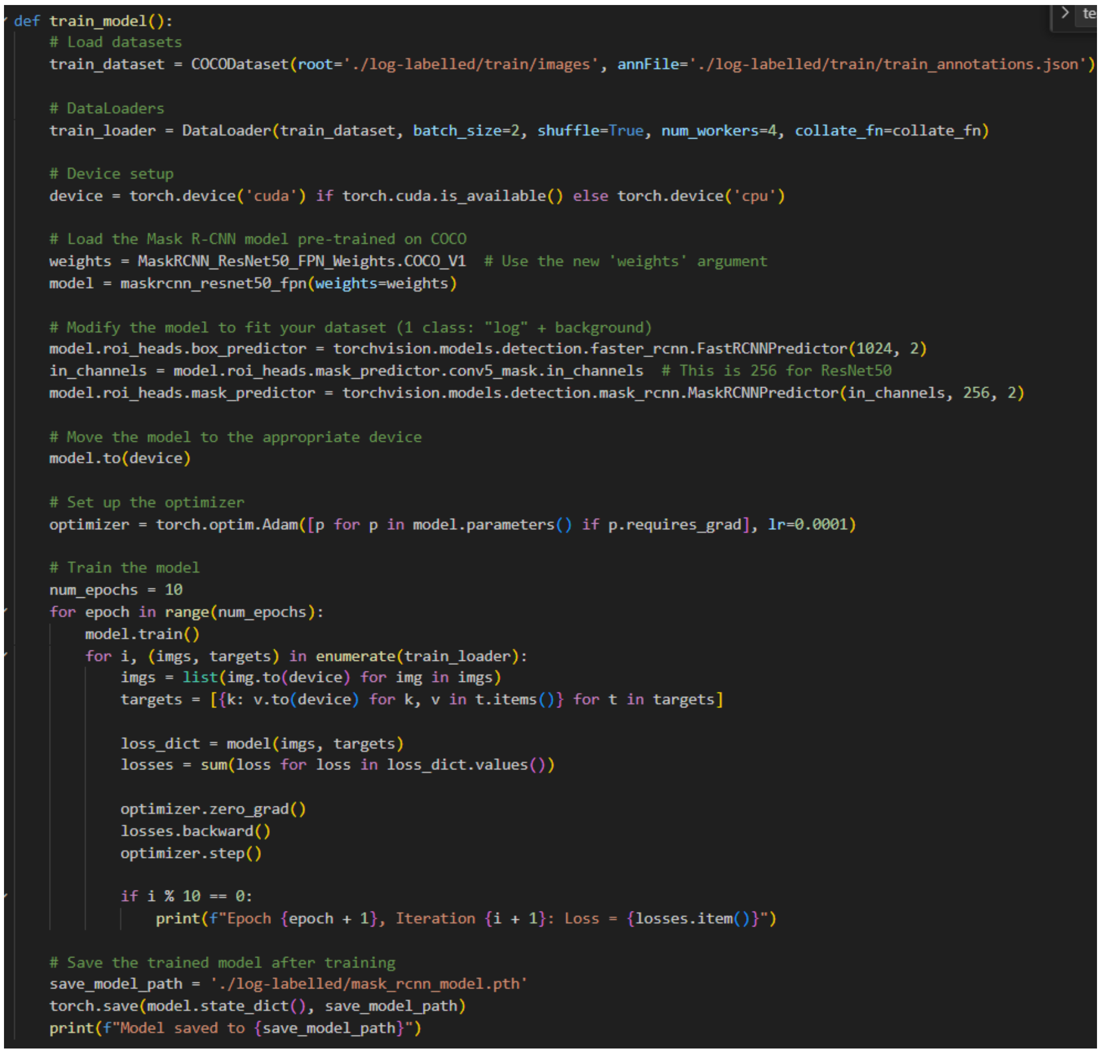
So, with the train dataset in the log-labelled folder, I have create the object of COCODataset for this.

The COCODataset instance cannot be used to input to the used mask rcnn model (maskrcnn_resnet50_fpn in torchvision.models.detection), so I have wraped them in the corresponding torch.utils.data.DataLoader instance. This class also allows to set up batch sizes splitting, which I have input the value of 2

The optimizer for the training process is also Adam (torch.optim.Adam)

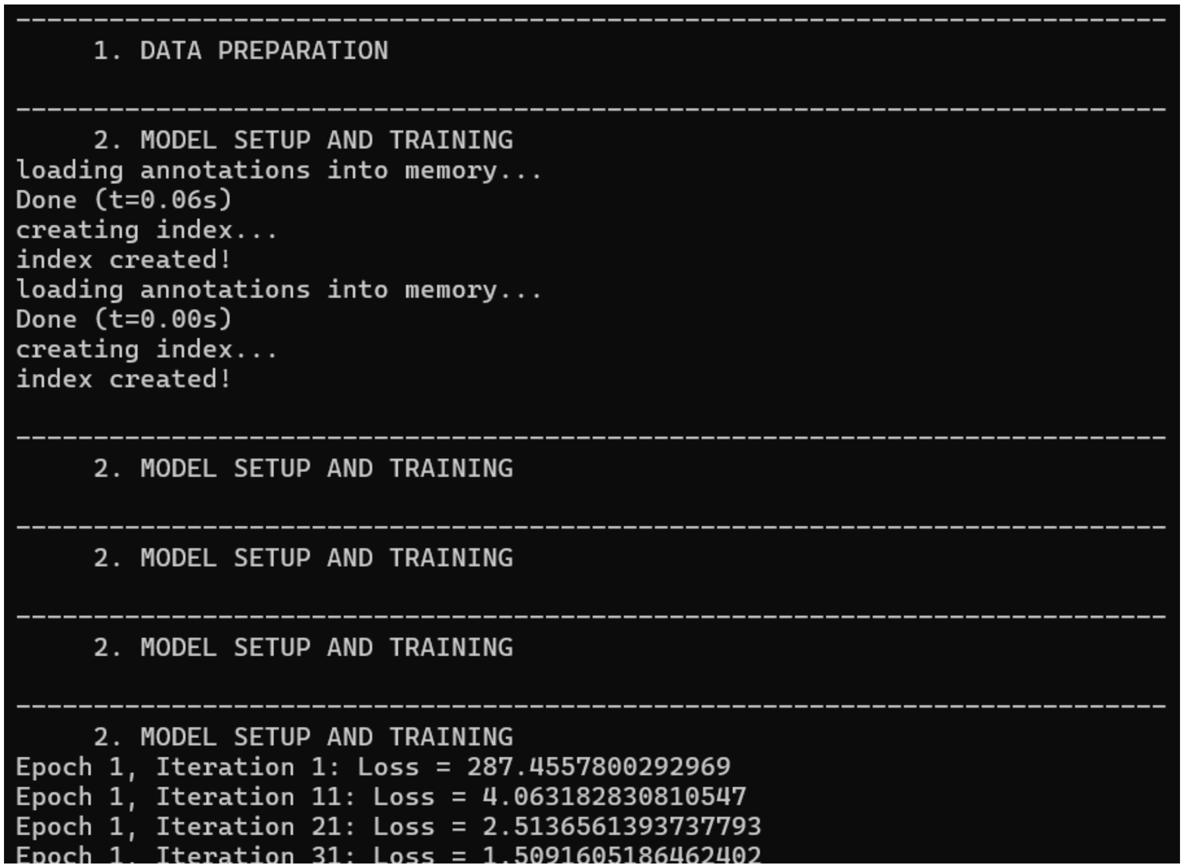
The number of epoch is 10.

The model is then saved (using torch.save) to the log-labelled folder, so that I can test it in the future without the need for re-training.

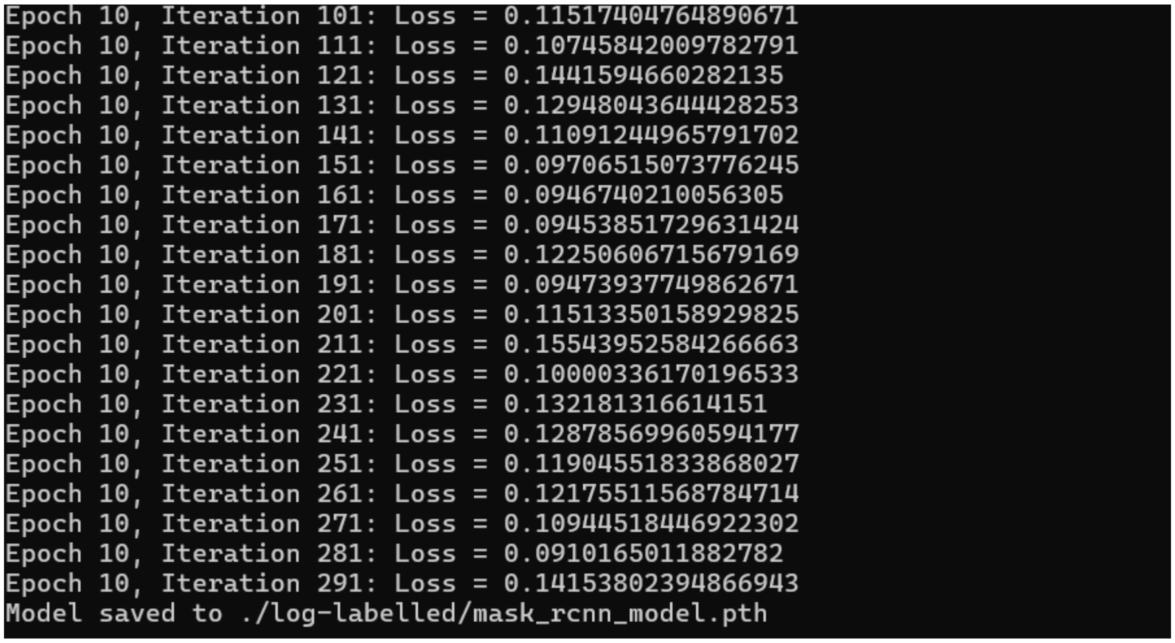


```
def train_model():
    # Load datasets
    train_dataset = COCODataset(root='./log-labelled/train/images', annFile='./log-labelled/train/train_annotations.json')

    # DataLoaders
    train_loader = DataLoader(train_dataset, batch_size=2, shuffle=True, num_workers=4, collate_fn=collate_fn)

    # Device setup
    device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')

    # Load the Mask R-CNN model pre-trained on COCO
    weights = MaskRCNN_ResNet50_FPN_Weights.COCO_V1  # Use the new 'weights' argument
    model = maskrcnn_resnet50_fpn(weights=weights)

    # Modify the model to fit your dataset (1 class: "log" + background)
    model.roi_heads.box_predictor = torchvision.models.detection.faster_rcnn.FastRCNNPredictor(1024, 2)
    in_channels = model.roi_heads.mask_predictor.conv5_mask.in_channels  # This is 256 for ResNet50
    model.roi_heads.mask_predictor = torchvision.models.detection.mask_rcnn.MaskRCNNPredictor(in_channels, 256, 2)

    # Move the model to the appropriate device
    model.to(device)

    # Set up the optimizer
    optimizer = torch.optim.Adam([p for p in model.parameters() if p.requires_grad], lr=0.0001)

    # Train the model
    num_epochs = 10
    for epoch in range(num_epochs):
        model.train()
        for i, (imgs, targets) in enumerate(train_loader):
            imgs = list(img.to(device) for img in imgs)
            targets = [{k: v.to(device) for k, v in t.items()} for t in targets]

            loss_dict = model(imgs, targets)
            losses = sum(loss for loss in loss_dict.values())

            optimizer.zero_grad()
            losses.backward()
            optimizer.step()

            if i % 10 == 0:
                print(f"Epoch {epoch + 1}, Iteration {i + 1}: Loss = {losses.item()}")

    # Save the trained model after training
    save_model_path = './log-labelled/mask_rcnn_model.pth'
    torch.save(model.state_dict(), save_model_path)
    print(f"Model saved to {save_model_path}")
```

And, these screenshots are the console outputs for the training process:



```
--------------------------------------------------------------------------
     1. DATA PREPARATION

--------------------------------------------------------------------------
     2. MODEL SETUP AND TRAINING
loading annotations into memory...
Done (t=0.06s)
creating index...
index created!
loading annotations into memory...
Done (t=0.00s)
creating index...
index created!

--------------------------------------------------------------------------
     2. MODEL SETUP AND TRAINING

--------------------------------------------------------------------------
     2. MODEL SETUP AND TRAINING

--------------------------------------------------------------------------
     2. MODEL SETUP AND TRAINING

--------------------------------------------------------------------------
     2. MODEL SETUP AND TRAINING
Epoch 1, Iteration 1: Loss = 287.4557800292969
Epoch 1, Iteration 11: Loss = 4.063182830810547
Epoch 1, Iteration 21: Loss = 2.5136561393737793
Epoch 1, Iteration 31: Loss = 1.5091605186462402
```



```
Epoch 10, Iteration 101: Loss = 0.11517404764890671
Epoch 10, Iteration 111: Loss = 0.10745842009782791
Epoch 10, Iteration 121: Loss = 0.1441594660282135
Epoch 10, Iteration 131: Loss = 0.12948043644428253
Epoch 10, Iteration 141: Loss = 0.11091244965791702
Epoch 10, Iteration 151: Loss = 0.09706515073776245
Epoch 10, Iteration 161: Loss = 0.0946740210056305
Epoch 10, Iteration 171: Loss = 0.09453851729631424
Epoch 10, Iteration 181: Loss = 0.12250606715679169
Epoch 10, Iteration 191: Loss = 0.09473937749862671
Epoch 10, Iteration 201: Loss = 0.11513350158929825
Epoch 10, Iteration 211: Loss = 0.15543952584266663
Epoch 10, Iteration 221: Loss = 0.10000336170196533
Epoch 10, Iteration 231: Loss = 0.132181316614151
Epoch 10, Iteration 241: Loss = 0.12878569960594177
Epoch 10, Iteration 251: Loss = 0.11904551833868027
Epoch 10, Iteration 261: Loss = 0.12175511568784714
Epoch 10, Iteration 271: Loss = 0.10944518446922302
Epoch 10, Iteration 281: Loss = 0.0910165011882782
Epoch 10, Iteration 291: Loss = 0.14153802394866943
Model saved to ./log-labelled/mask_rcnn_model.pth
```

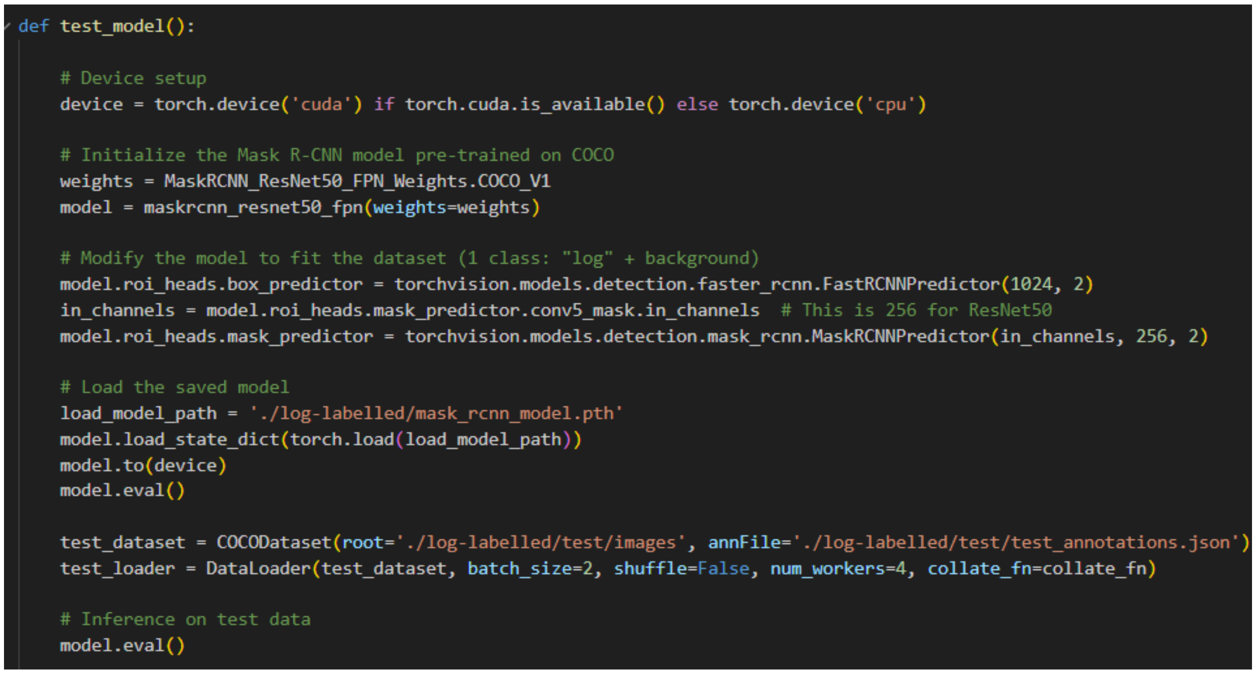
Here is the link for the trained Mask RCNN model:
https://drive.google.com/file/d/10qvGe3l9WtkqP4Hlk1D4TOop5dh1oP3I/view?usp=drive_link

Mask RCNN Testing and Evaluating

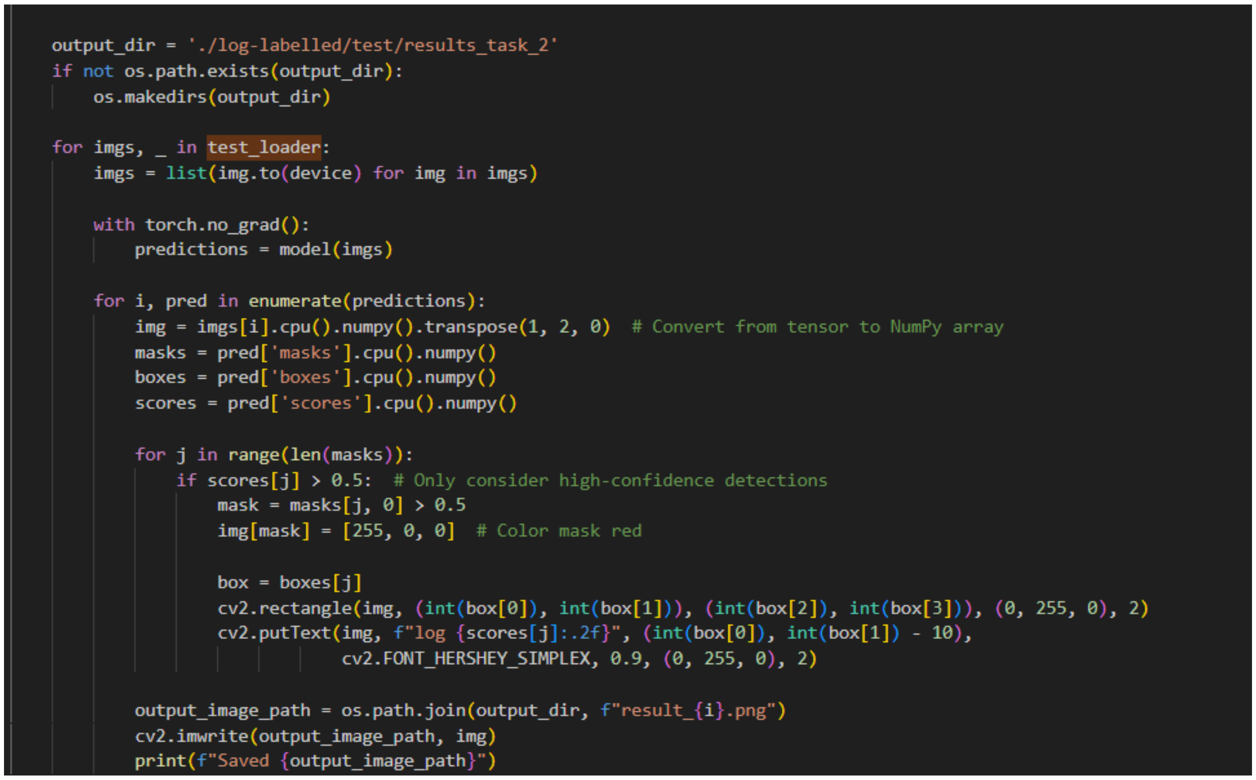
For testing the above model, I have implemented a method named “test_model” in the file studio5_task2.py.

- The function starts by loading the saved weights of the Mask R-CNN model, which named “mask_rcnn_model.pth” trained in the above stage
- Create COCODataset and DataLoader instance for test dataset
- Then I performed inference on test images (located in log-labelled/test directory). The method iterates over the test dataset, retrieving batches of images from test_loader
- The model predicts bounding boxes, masks, and confidence scores for each image.
- Visualization:
    - Check the confidence score. If detection is high-confidence (confidence score > 0.5), then we will process the visualization for that output
    - Color the detected log in red.
    - A green bounding box is drawn around the detected object
    - The output images are saved in the directory ‘log-labelled/test/results_task_2’

Here is the implementation for test_model



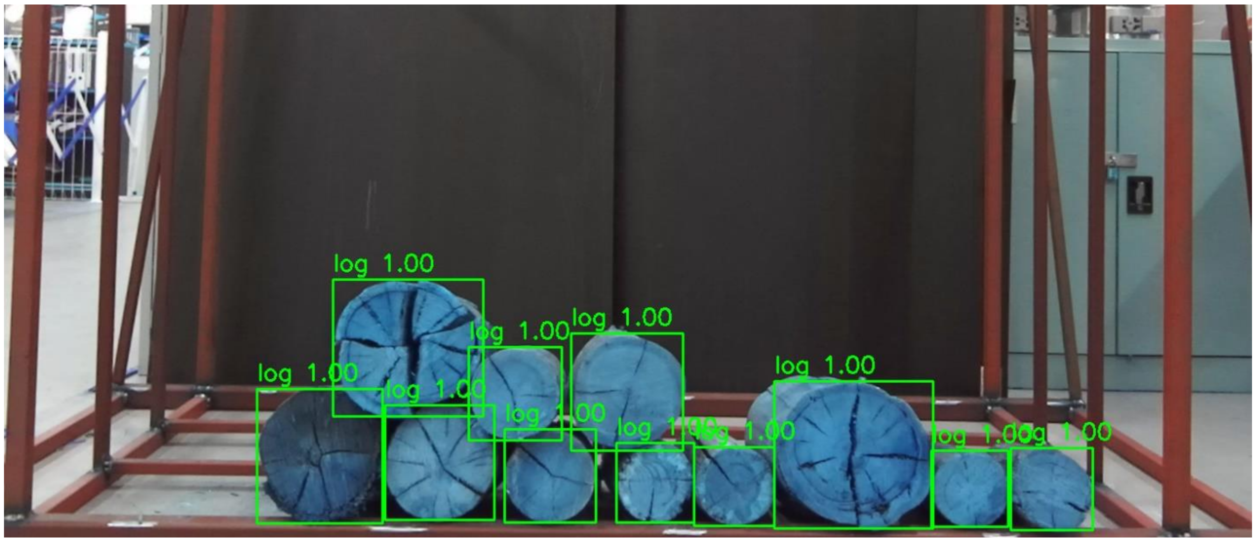
```
def test_model():

    # Device setup
    device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu')

    # Initialize the Mask R-CNN model pre-trained on COCO
    weights = MaskRCNN_ResNet50_FPN_Weights.COCO_V1
    model = maskrcnn_resnet50_fpn(weights=weights)

    # Modify the model to fit the dataset (1 class: "log" + background)
    model.roi_heads.box_predictor = torchvision.models.detection.faster_rcnn.FastRCNNPredictor(1024, 2)
    in_channels = model.roi_heads.mask_predictor.conv5_mask.in_channels  # This is 256 for ResNet50
    model.roi_heads.mask_predictor = torchvision.models.detection.mask_rcnn.MaskRCNNPredictor(in_channels, 256, 2)

    # Load the saved model
    load_model_path = './log-labelled/mask_rcnn_model.pth'
    model.load_state_dict(torch.load(load_model_path))
    model.to(device)
    model.eval()

    test_dataset = COCODataset(root='./log-labelled/test/images', annFile='./log-labelled/test/test_annotations.json')
    test_loader = DataLoader(test_dataset, batch_size=2, shuffle=False, num_workers=4, collate_fn=collate_fn)

    # Inference on test data
    model.eval()
```



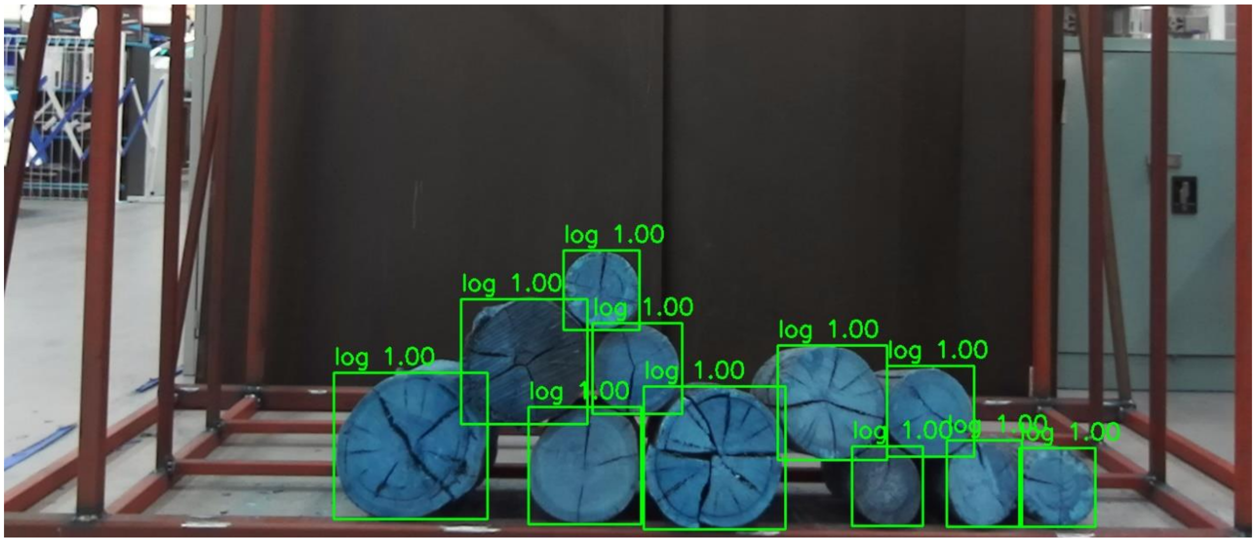
```python
output_dir = './log-labelled/test/results_task_2'
if not os.path.exists(output_dir):
    os.makedirs(output_dir)

for imgs, _ in test_loader:
    imgs = list(img.to(device) for img in imgs)

    with torch.no_grad():
        predictions = model(imgs)

    for i, pred in enumerate(predictions):
        img = imgs[i].cpu().numpy().transpose(1, 2, 0)  # Convert from tensor to NumPy array
        masks = pred['masks'].cpu().numpy()
        boxes = pred['boxes'].cpu().numpy()
        scores = pred['scores'].cpu().numpy()

        for j in range(len(masks)):
            if scores[j] > 0.5:  # Only consider high-confidence detections
                mask = masks[j, 0] > 0.5
                img[mask] = [255, 0, 0]  # Color mask red

                box = boxes[j]
                cv2.rectangle(img, (int(box[0]), int(box[1])), (int(box[2]), int(box[3])), (0, 255, 0), 2)
                cv2.putText(img, f"log {scores[j]:.2f}", (int(box[0]), int(box[1]) - 10),
                            cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)

        output_image_path = os.path.join(output_dir, f"result_{i}.png")
        cv2.imwrite(output_image_path, img)
        print(f"Saved {output_image_path}")
```

Here is the part of the console output for testing stage:

And, finally, some output generate images include:



result_0.png



result_1.png

You can find all of those via this link of ‘log-labelled/test/results_task_2’ directory, or “rcnn_test” directory as the requirements document has stated:
https://github.com/trungkiennguyen22082004/COS40007_Artificial_Intelligence_for_Engineering/tree/main/Studios/Studio%205/log-labelled/test/results_task_2

or

Step 3: Extending log labelling to another class

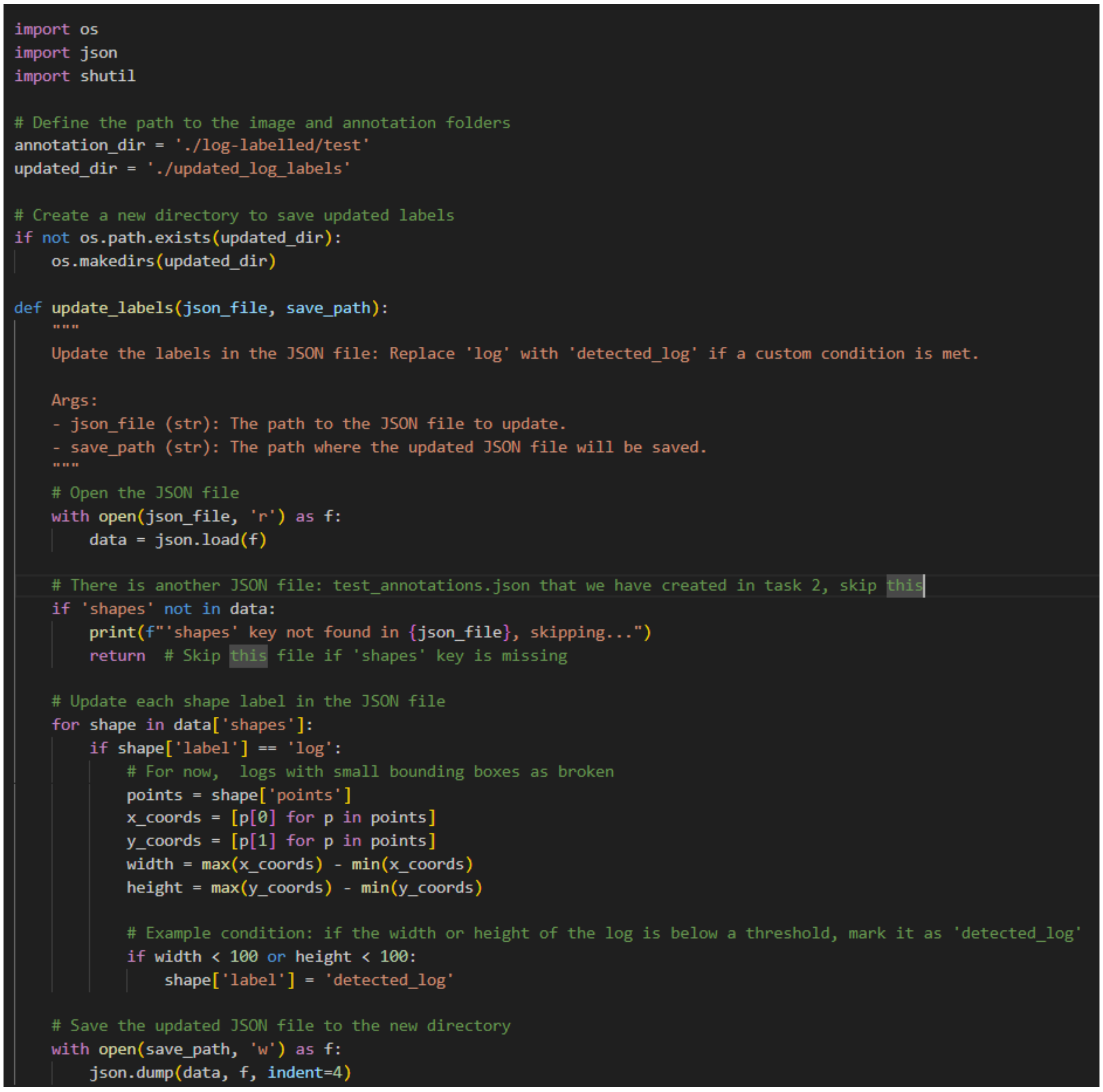
The purposes of this step is to update the labels for the 10 images. The implementation file for this can be retrieved from the following link:
https://github.com/trungkiennguyen22082004/COS40007_Artificial_Intelligence_for_Engineering/blob/main/Studios/Studio%205/code/studio5_task3.py

For this stage, I automate the process of checking the existing JSON labels and updating broken logs to detected_log:

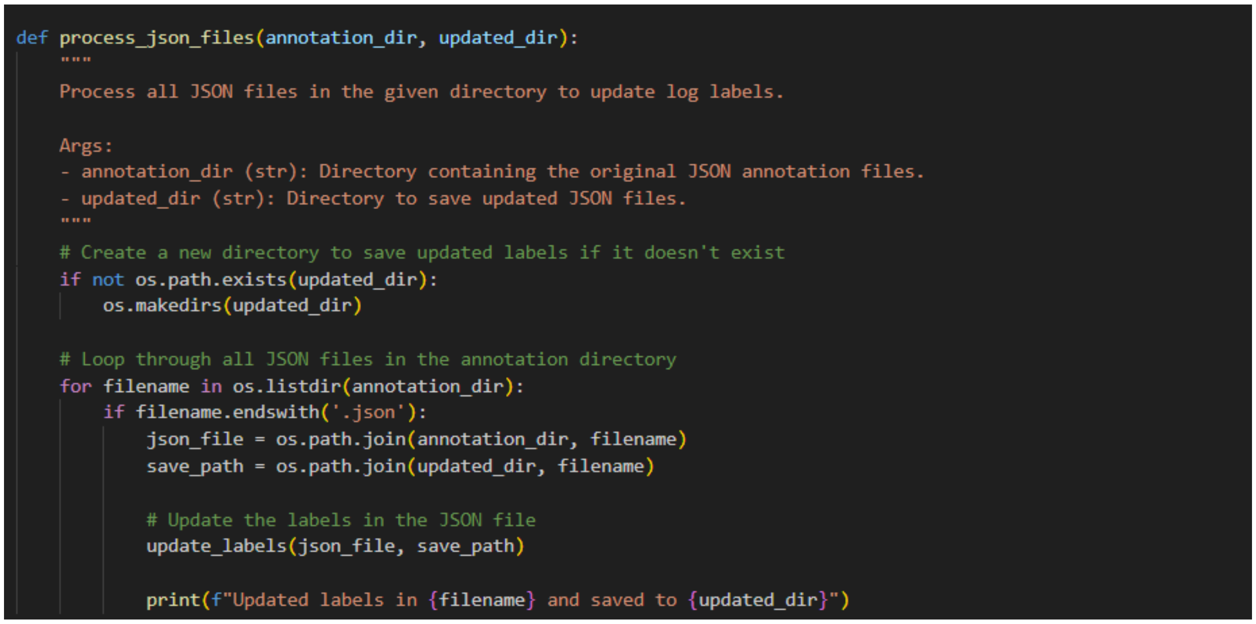
- Read all 20 files from the log-labelled/test directory (excluding the directories generated in Step 2), with 10 images (png) and 10 corresponding json files.
- The process_json_files iterate through each json files to proceed updating
- The update_labels function iterate through each shape in the JSON file and checks its label.
    - If the label is “log”, it calculates the bounding box size (width, hegiht) of the log, using the polygon points
    - If the size calculated is acceptable (not to be broken), determined if width or heigth is below 100 pixels, then the updated JSON file is saved.

- Created a new directory named “updated_log_labelled” to store the modified annotation files

Here is the implementation for this task:

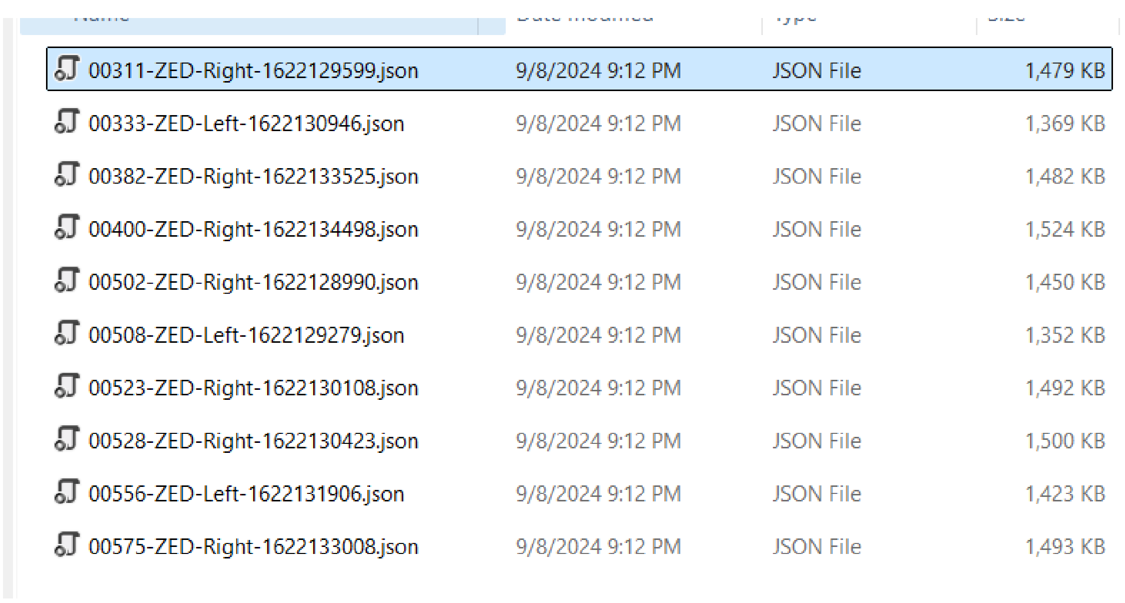


```python
import os
import json
import shutil

# Define the path to the image and annotation folders
annotation_dir = './log-labelled/test'
updated_dir = './updated_log_labels'

# Create a new directory to save updated labels
if not os.path.exists(updated_dir):
    os.makedirs(updated_dir)

def update_labels(json_file, save_path):
    """
    Update the labels in the JSON file: Replace 'log' with 'detected_log' if a custom condition is met.

    Args:
    - json_file (str): The path to the JSON file to update.
    - save_path (str): The path where the updated JSON file will be saved.
    """
    # Open the JSON file
    with open(json_file, 'r') as f:
        data = json.load(f)

    # There is another JSON file: test_annotations.json that we have created in task 2, skip this
    if 'shapes' not in data:
        print(f"'shapes' key not found in {json_file}, skipping...")
        return  # Skip this file if 'shapes' key is missing

    # Update each shape label in the JSON file
    for shape in data['shapes']:
        if shape['label'] == 'log':
            # For now,  logs with small bounding boxes as broken
            points = shape['points']
            x_coords = [p[0] for p in points]
            y_coords = [p[1] for p in points]
            width = max(x_coords) - min(x_coords)
            height = max(y_coords) - min(y_coords)

            # Example condition: if the width or height of the log is below a threshold, mark it as 'detected_log'
            if width < 100 or height < 100:
                shape['label'] = 'detected_log'

    # Save the updated JSON file to the new directory
    with open(save_path, 'w') as f:
        json.dump(data, f, indent=4)
```



```python
def process_json_files(annotation_dir, updated_dir):
    """
    Process all JSON files in the given directory to update log labels.

    Args:
    - annotation_dir (str): Directory containing the original JSON annotation files.
    - updated_dir (str): Directory to save updated JSON files.
    """
    # Create a new directory to save updated labels if it doesn't exist
    if not os.path.exists(updated_dir):
        os.makedirs(updated_dir)

    # Loop through all JSON files in the annotation directory
    for filename in os.listdir(annotation_dir):
        if filename.endswith('.json'):
            json_file = os.path.join(annotation_dir, filename)
            save_path = os.path.join(updated_dir, filename)

            # Update the labels in the JSON file
            update_labels(json_file, save_path)

            print(f"Updated labels in {filename} and saved to {updated_dir}")
```

Here is the output of updated annotations for 10 files:



00311-ZED-Right-1622129599.json	9/8/2024 9:12 PM	JSON File	1,479 KB
00333-ZED-Left-1622130946.json	9/8/2024 9:12 PM	JSON File	1,369 KB
00382-ZED-Right-1622133525.json	9/8/2024 9:12 PM	JSON File	1,482 KB
00400-ZED-Right-1622134498.json	9/8/2024 9:12 PM	JSON File	1,524 KB
00502-ZED-Right-1622128990.json	9/8/2024 9:12 PM	JSON File	1,450 KB
00508-ZED-Left-1622129279.json	9/8/2024 9:12 PM	JSON File	1,352 KB
00523-ZED-Right-1622130108.json	9/8/2024 9:12 PM	JSON File	1,492 KB
00528-ZED-Right-1622130423.json	9/8/2024 9:12 PM	JSON File	1,500 KB
00556-ZED-Left-1622131906.json	9/8/2024 9:12 PM	JSON File	1,423 KB
00575-ZED-Right-1622133008.json	9/8/2024 9:12 PM	JSON File	1,493 KB

You can check the updated json files in the updated_log_labels folder via this link: https://github.com/trungkiennguyen22082004/COS40007_Artificial_Intelligence_for_Engineering/tree/main/Studios/Studio%205/updated_log_labels