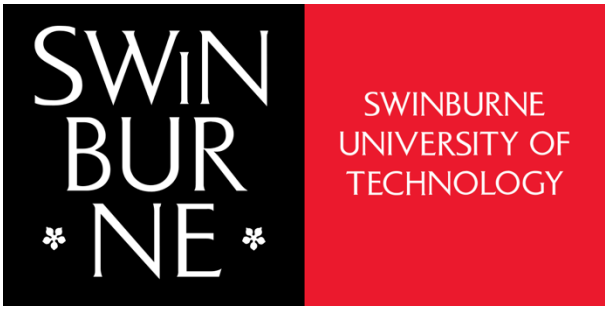

Artificial Intelligence for Engineering

Portfolio Assessment 3

Develop an AI model by your own decision

Xuan Tuan Minh Nguyen - 103819212
Studio 1-6

Task 1. Develop CNN and ResNet50

1. Randomly take out 10 rust and 10 no-rust images for testing

Detailed Task: First, randomly take out 10 rust and 10 no-rust images for testing (We call this a Test Set). So, your training set will not contain these 20 images. Yes

```
[ ]  def count_images_in_directory(directory):
       """
       Utility function to count images in a given directory
       :param directory: The directory to count images in
       :return: None
       """
       total_count = 0
       for root, dirs, files in os.walk(directory):
         file_count = len(files)
         print(f"{root}: {file_count} images")
         total_count += file_count
```

Figure 1: Utility Function to count images in directory

```
[ ]  BASE_PATH = os.getcwd() + '/Corrosion-dataset'

     def data_preparation():
       """
       Prepare the data for training and testing
       :return: train_dir, test_dir
       """
       # Get the corrosion dataset and define two main classses, which are rust and no rust
       data_path = os.path.join(BASE_PATH, 'Corrosion')
       target_classes = ['rust', 'no rust']

       # Make train and test directory
       train_dir = os.path.join(data_path, 'train')
       test_dir = os.path.join(data_path, 'test')

       if not os.path.exists(train_dir):
         os.makedirs(train_dir)
       if not os.path.exists(test_dir):
         os.makedirs(test_dir)

       # Walk through the rust and no rust directory
       for c in target_classes:
         # Make rust and no rust folder if not exsits
         if not os.path.exists(os.path.join(train_dir, c)):
           os.makedirs(os.path.join(train_dir, c))
         if not os.path.exists(os.path.join(test_dir, c)):
           os.makedirs(os.path.join(test_dir, c))
         labeled_dir = os.path.join(data_path, c)
         # Append images into an array for spliting
         images = []
         for file in os.listdir(labeled_dir):
           if os.path.isfile(os.path.join(labeled_dir, file)):
             images.append(os.path.join(labeled_dir, file))

         # Using train_test_split to split 10 images for both rust and no rust
         train_images, test_images = train_test_split(images, test_size=10, random_state=1)

         # Put imags to the rust and no rust folder
         for image in train_images:
           shutil.copy(image, os.path.join(train_dir, c))
         for image in test_images:
           shutil.copy(image, os.path.join(test_dir, c))

       print(len(images))

       return train_dir, test_dir
```

Figure 2: Main function to split dataset

```
print("Starting data preparation process...")
train_dir, test_dir = data_preparation()
count_images_in_directory(train_dir)
count_images_in_directory(test_dir)
print("Data preparation completed.")
```

Figure 3: Execute process

```
Starting data preparation process...
15
/content/drive/MyDrive/Colab Notebooks/COS40007/Assignment 4/Corrosion-dataset/Corrosion/train: 0 images
/content/drive/MyDrive/Colab Notebooks/COS40007/Assignment 4/Corrosion-dataset/Corrosion/train/rust: 4 images
/content/drive/MyDrive/Colab Notebooks/COS40007/Assignment 4/Corrosion-dataset/Corrosion/train/no rust: 5 images
/content/drive/MyDrive/Colab Notebooks/COS40007/Assignment 4/Corrosion-dataset/Corrosion/test: 0 images
/content/drive/MyDrive/Colab Notebooks/COS40007/Assignment 4/Corrosion-dataset/Corrosion/test/rust: 10 images
/content/drive/MyDrive/Colab Notebooks/COS40007/Assignment 4/Corrosion-dataset/Corrosion/test/no rust: 10 images
Data preparation completed.
```

Figure 4: Result after execute the code

2. Develop a simple CNN model

Detailed Task: Develop a simple CNN model similar to mnist classification but train with the provided corrosion data with class "rust" and "no rust" (that excludes Test Set). Once the model is trained and saved, test with your Test dataset and measure the accuracy (using the correct classification of 20 images in the test set).
Note: I have trained this model using 20 epochs, 100 steps per epoch

```
[50] def with_basic_cnn(train_dir, test_dir):
        """
        Function to build, train and test the basic CNN model
        :param train_dir: The directory of the training data
        :param test_dir: The directory of the testing data
        :return: model_builder
        """
        # Define target image's width and height
        image_width, image_height = 32, 32
        # Create a sequential builder to build the model
        model_builder = Sequential([
          Input(shape=(image_width, image_height, 3)),
          Conv2D(32, (3, 3), activation='relu'),
          MaxPooling2D((2, 2)),
          Conv2D(64, (3, 3), activation='relu'),
          MaxPooling2D((2, 2)),
          Conv2D(128, (3, 3), activation='relu'),
          MaxPooling2D((2, 2)),
          Flatten(),
          Dense(128, activation='relu'),
          Dense(2, activation='softmax')
        ])
        model_builder.summary()
        # Compile with adam and categorical crossentropy
        model_builder.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
        # Convert train and test dataset using ImageDataGenerator
        train_datagen = ImageDataGenerator(rescale=1./255)
        test_datagen = ImageDataGenerator(rescale=1./255)
        train_generator = train_datagen.flow_from_directory(train_dir, target_size=(image_width, image_height), batch_size=32, class_mode='categorical')
        test_generator = test_datagen.flow_from_directory(test_dir, target_size=(image_width, image_height), batch_size=32, class_mode='categorical')
        # Fith with 20 epochs (100 steps per epoch)
        history = model_builder.fit(train_generator, epochs=20, steps_per_epoch=100, validation_data=test_generator, validation_steps=50)
        # Plot the accuracy
        plot_accuracy(history, "Basic CNN")
        show_true_and_predicted_classes(model_builder, test_generator)
        # Eval and save predicted images to the folder
        print(f"Test accuracy of Basic CNN: {model_builder.evaluate(test_generator)[1] * 100:.2f}%")
        print(f"Saving Predicted CNN to {BASE_PATH}/cnn_test")
        save_predicted_images(model_builder, test_generator, BASE_PATH + "/cnn_test")
        return model_builder
```

Figure 5: Codebase for building, training and predicting CNN model

```
[51] cnn_model = with_basic_cnn(train_dir, test_dir)
     cnn_model.save(BASE_PATH + "/models/simple_cnn.keras")
```

Model: "sequential_6"

Layer (type)	Output Shape	Param #
conv2d_9 (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d_9 (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_10 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_10 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_11 (Conv2D)	(None, 4, 4, 128)	73,856
max_pooling2d_11 (MaxPooling2D)	(None, 2, 2, 128)	0
flatten_3 (Flatten)	(None, 512)	0
dense_12 (Dense)	(None, 128)	65,664
dense_13 (Dense)	(None, 2)	258

```
Total params: 159,170 (621.76 KB)
Trainable params: 159,170 (621.76 KB)
Non-trainable params: 0 (0.00 B)
Found 9 images belonging to 2 classes.
Found 20 images belonging to 2 classes.
Epoch 1/20
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDa
  self._warn_if_super_not_called()
  1/100 ━━━━━━━━━━━━━━━━━━━━ 8:53 5s/step - accuracy: 0.4444 - loss: 0.7023/usr/local/lib/python3.11/dist-packages/keras/sr
  self._interrupted_warning()
100/100 ━━━━━━━━━━━━━━━━━━━━ 8s 26ms/step - accuracy: 0.4444 - loss: 0.7023 - val_accuracy: 0.5000 - val_loss: 0.6827
Epoch 2/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5556 - loss: 0.6769 - val_accuracy: 0.5000 - val_loss: 0.6839
Epoch 3/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.5556 - loss: 0.6691 - val_accuracy: 0.5000 - val_loss: 0.6905
Epoch 4/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step - accuracy: 0.5556 - loss: 0.6644 - val_accuracy: 0.5000 - val_loss: 0.6871
Epoch 5/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5556 - loss: 0.6549 - val_accuracy: 0.5000 - val_loss: 0.6763
Epoch 6/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5556 - loss: 0.6452 - val_accuracy: 0.5000 - val_loss: 0.6688
Epoch 7/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5556 - loss: 0.6357 - val_accuracy: 0.5000 - val_loss: 0.6650
Epoch 8/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5556 - loss: 0.6213 - val_accuracy: 0.5000 - val_loss: 0.6680
```

Figure 6: Training and evaluating process

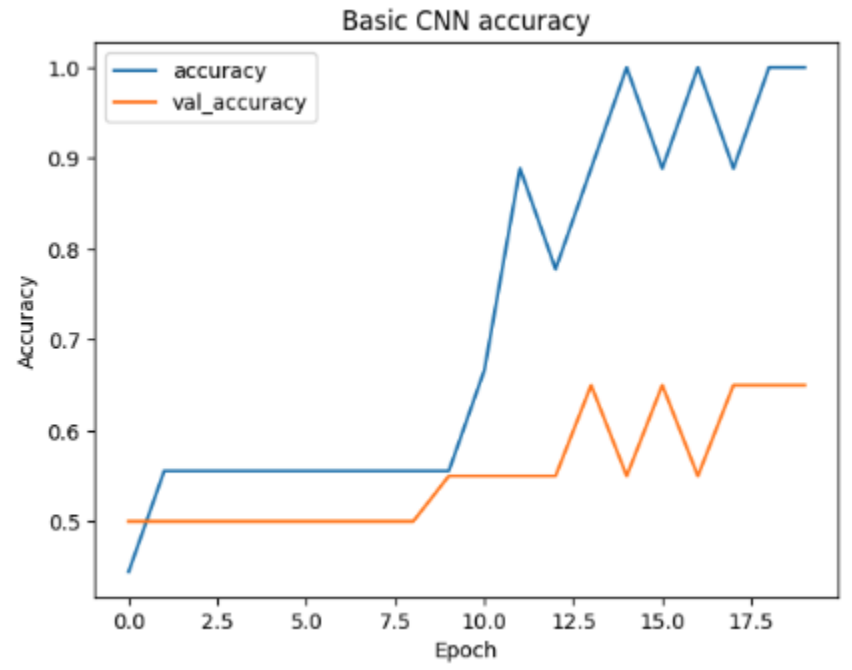

Figure 7: Basic CNN Accuracy

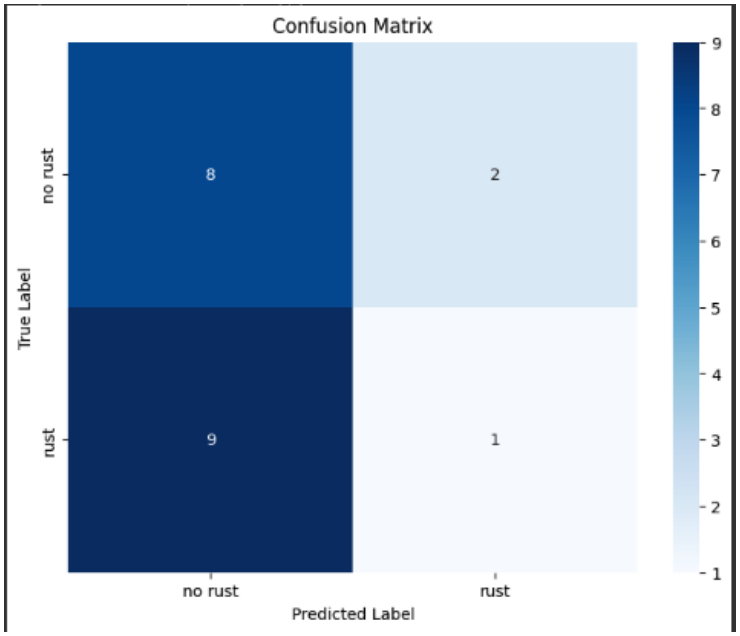

Figure 8: Confusion Matrix of Basic CNN

True Class	Predicted Class
0	0
0	0
0	0
0	0
0	1
0	0
0	0
0	1
0	0
0	0
1	1
1	0

1	0
1	0
1	0
1	0
1	0
1	0
1	0
1	0

Table 1: True class and Predicted class for Basic CNN

Overall Accuracy: 65.00%

3. Develop a more complex CNN, Restnet50

Detailed Task: Now develop a more complex CNN, Restnet50 and train with the same dataset as in step 2 test with the Test dataset and measure the accuracy (using 20 images in the test set)

Note: I have trained this model using 20 epochs, 100 steps per epoch

```python
def with_resnet50(train_dir, test_dir):
    """
    Function to build, train and test the ResNet50 model
    :param train_dir: The directory of the training data
    :param test_dir: The directory of the testing data
    :return: model_builder
    """
    # Define target image's width and height
    image_width, image_height = 224, 224
    # Get the initial ResNet50 and freeze it
    initial_resnet50 = ResNet50(weights='imagenet', include_top=False, input_shape=(image_width, image_height, 3))
    initial_resnet50.trainable = False
    # Create a sequential builder for initial ResNet50 and add some more layers
    model_builder = Sequential([
        initial_resnet50,
        GlobalAveragePooling2D(),
        Dense(128, activation='relu'),
        Dense(2, activation='softmax')
    ])
    model_builder.summary()
    # Compile with adam and categorical crossentropy
    model_builder.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
    # Convert train and test dataset using ImageDataGenerator
    train_datagen = ImageDataGenerator(rescale=1./255)
    test_datagen = ImageDataGenerator(rescale=1./255)

    train_generator = train_datagen.flow_from_directory(train_dir, target_size=(image_width, image_height), batch_size=32, class_mode='categorical')
    test_generator = test_datagen.flow_from_directory(test_dir, target_size=(image_width, image_height), batch_size=32, class_mode='categorical')
    print("Test generator: ", test_generator)
    # Fith with 20 epochs (100 steps per epoch)
    history = model_builder.fit(train_generator, epochs=20, steps_per_epoch=100, validation_data=test_generator, validation_steps=50)
    # Plot the accuracy
    plot_accuracy(history, "ResNet50")
    show_true_and_predicted_classes(model_builder, test_generator)
    # Eval and save predicted images to the folder
    print(f"Test accuracy of ResNet50: {model_builder.evaluate(test_generator)[1] * 100:.2f}%")
    print(f"Saving Predicted CNN to {BASE_PATH}/resnet50_test")
    save_predicted_images(model_builder, test_generator, BASE_PATH + "/resnet50_test")
    return model_builder
```

Figure 9: Codebase for building, training and predicting ResNet50 model

```
resnet_model = with_resnet50(train_dir, test_dir)
resnet_model.save(BASE_PATH + "/models/resnet50.keras")
```

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 7, 7, 2048)	23,587,712
global_average_pooling2d_3 (GlobalAveragePooling2D)	(None, 2048)	0
dense_14 (Dense)	(None, 128)	262,272
dense_15 (Dense)	(None, 2)	258

```
Total params: 23,850,242 (90.98 MB)
Trainable params: 262,530 (1.00 MB)
Non-trainable params: 23,587,712 (89.98 MB)
Found 9 images belonging to 2 classes.
Found 20 images belonging to 2 classes.
Test generator:  <keras.src.legacy.preprocessing.image.DirectoryIterator object at 0x7eab4cb97810>
Epoch 1/20
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` c
  self._warn_if_super_not_called()
  1/100 ━━━━━━━━━━━━━━━━━━━━ 31:13 19s/step - accuracy: 0.4444 - loss: 0.8576/usr/local/lib/python3.11/dist-packages/keras/src/trai
  self._interrupted_warning()
100/100 ━━━━━━━━━━━━━━━━━━━━ 26s 73ms/step - accuracy: 0.4444 - loss: 0.8576 - val_accuracy: 0.5000 - val_loss: 0.7756
Epoch 2/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5556 - loss: 0.7260 - val_accuracy: 0.5000 - val_loss: 0.8554
Epoch 3/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5556 - loss: 0.7874 - val_accuracy: 0.5000 - val_loss: 0.7679
Epoch 4/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5556 - loss: 0.7202 - val_accuracy: 0.5000 - val_loss: 0.6987
Epoch 5/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5556 - loss: 0.6840 - val_accuracy: 0.5000 - val_loss: 0.7125
Epoch 6/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4444 - loss: 0.7225 - val_accuracy: 0.5000 - val_loss: 0.7204
Epoch 7/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4444 - loss: 0.7341 - val_accuracy: 0.5000 - val_loss: 0.7010
Epoch 8/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4444 - loss: 0.7013 - val_accuracy: 0.5000 - val_loss: 0.7017
Epoch 9/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.5556 - loss: 0.6807 - val_accuracy: 0.5000 - val_loss: 0.7350
Epoch 10/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5556 - loss: 0.6948 - val_accuracy: 0.5000 - val_loss: 0.7597
Epoch 11/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5556 - loss: 0.7104 - val_accuracy: 0.5000 - val_loss: 0.7487
Epoch 12/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5556 - loss: 0.7023 - val_accuracy: 0.5000 - val_loss: 0.7182
Epoch 13/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5556 - loss: 0.6834 - val_accuracy: 0.5000 - val_loss: 0.6981
Epoch 14/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.5556 - loss: 0.6782 - val_accuracy: 0.5000 - val_loss: 0.6973
Epoch 15/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step - accuracy: 0.4444 - loss: 0.6888 - val_accuracy: 0.5000 - val_loss: 0.6999
Epoch 16/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4444 - loss: 0.6948 - val_accuracy: 0.5000 - val_loss: 0.6968
Epoch 17/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 1s 4ms/step - accuracy: 0.4444 - loss: 0.6864 - val_accuracy: 0.5000 - val_loss: 0.6973
Epoch 18/20
100/100 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.6667 - loss: 0.6757 - val_accuracy: 0.5000 - val_loss: 0.7095
```

Figure 10: Training and evaluating process for ResNet50

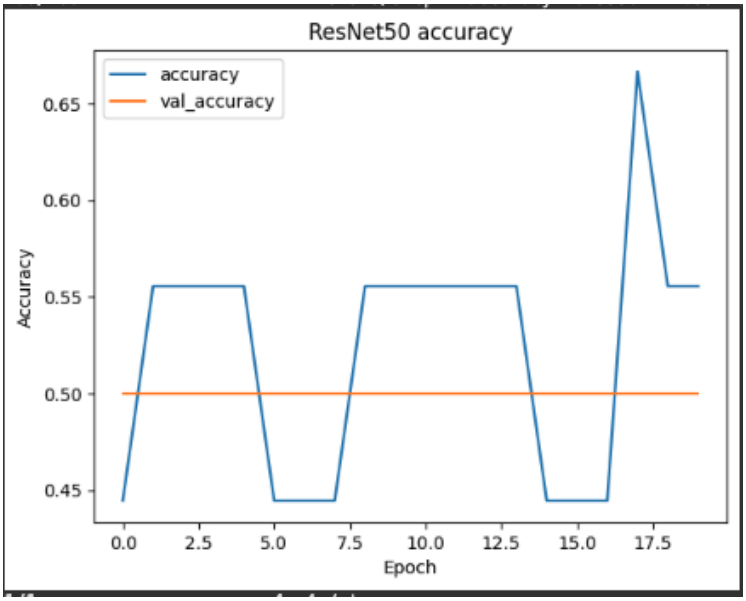

Figure 11: ResNet50 Accuracy

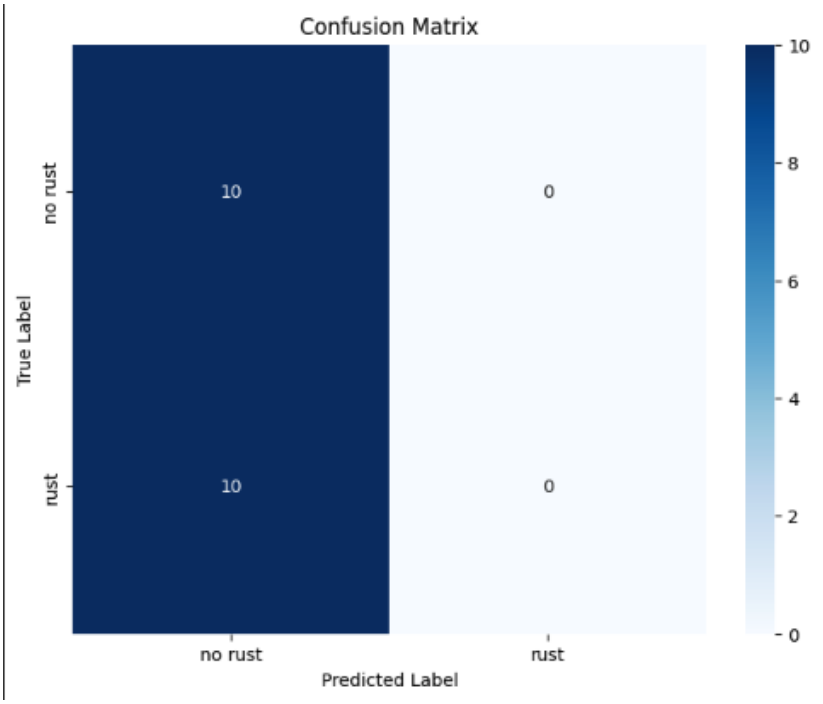

Figure 12: Confusion Matrix of ResNet50

True Class	Predicted Class
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
1	0

1	0
1	0
1	0
1	0
1	0
1	0
1	0
1	0
1	0

Table 2: True class and Predicted class for ResNet50

Overall Accuracy: 50%

Task 2: Develop Mask RCNN for detecting wooden log

1. Data Processing

Detailed Task: First, randomly take out 10 images for testing (We call this a Test Set)

```
def use_test_images():
  """
  Utility function to randomly extract 10 images for testing
  :return: None
  """
  _images = []
  for file in os.listdir(DS_LOG_PATH):
    if file.endswith('.png'):
      _images.append(file)

  _test_images = random.sample(_images, 10)
  return _test_images

def convert_from_labelme_to_coco():
  """
  Function to convert from labelme to coco
  :return: None
  """
  labelme2coco.convert(labelme_folder=DS_LOG_PATH, export_dir=DS_CONVERTED_PATH, train_split_rate=0.9, category_id_start=1)
```

Figure 13: Function to get 10 images for testing and another function to convert from labelme to coco

2. Models Development

Detailed Task: Develop a Mask RCNN model using the labelled wooden log as training data (excludes test set). Note that you may need to convert the labelme annotation to coco annotation to train mask RCNN. Here is a reference link:

```python
def with_model_config():
    """
    Utility function to define config for the Mask RCNN
    :return: detectron2.config.Config
    """
    _config = config.get_cfg()
    # Merge from config file using COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x
    _config.merge_from_file(model_zoo.get_config_file("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml"))
    # Force to run with cuda, if not you can set it with CPU
    _config.MODEL.DEVICE = "cuda"
    # Model output directory
    _config.MODEL_OUTPUT_DIR = ML_OUT_PATH
    # Config model's weights using COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x
    _config.MODEL.WEIGHTS = model_zoo.get_checkpoint_url("COCO-InstanceSegmentation/mask_rcnn_R_50_FPN_3x.yaml")
    # Number of classes for Region of Interests (RoI)
    _config.MODEL.ROI_HEADS.NUM_CLASSES = 1
    # Define batch size per image, recommend 128
    _config.MODEL.ROI_HEADS.BATCH_SIZE_PER_IMAGE = 128
    # Max iteration between epochs
    _config.SOLVER.MAX_ITER = 100
    # Base LR
    _config.SOLVER.BASE_LR = 0.00025
    # Image per batch
    _config.SOLVER.IMS_PER_BATCH = 2
    # Number of workers, recommended 2
    _config.DATALOADER.NUM_WORKERS = 2
    # If there are no log_train or log_val, assign it as Train and Test datasets
    if not _config.DATASETS.TRAIN == ("log_train",) and not _config.DATASETS.TEST == ("log_val",):
      _config.DATASETS.TRAIN = ("log_train",)
      _config.DATASETS.TEST = ("log_val",)

    return _config
```

Figure 14: Function to define config for Mask RCNN model

```python
def extract_log(img_dir, json_file):
    """
Chạy ô (⌘/Ctrl+Enter)
ô chưa được thực thi trong phiên này
    ...extract necessary data from coco dataset for registering datasets via detectron2.data.DatasetCatalog.register
    ...contains images
    :json_file: Directory to put the val and train data in json format
    :return: Extracted Logs
    """
    # Load coco data
    _json_dir = os.path.join(img_dir, json_file)
    _imported_coco_data = json.load(open(_json_dir))
    extracted_logs = []

    # Looking for images tag in coco data
    for img in _imported_coco_data['images']:
      temp = {}
      # Set file name to the right path
      fName = os.path.join(img_dir, img['file_name'])
      temp["file_name"] = fName
      # Set the image_id using id tag in json
      temp["image_id"] = img['id']
      # Set the height using height tag in json
      temp["height"] = img['height']
      # Set the width using width tag in json
      temp["width"] = img['width']

      # Extract important annotations metadata inside annotations tag if image_id is equal to id
      annots = [a for a in _imported_coco_data['annotations'] if a['image_id'] == img['id']]
      annotations = []

      for ann in annots:
        # Convert and append to the annotations array
        inner_anno = {
            "bbox": ann['bbox'],
            "bbox_mode": structures.BoxMode.XYWH_ABS,
            "segmentation": ann['segmentation'],
            "category_id": ann['category_id'] - 1
        }
        annotations.append(inner_anno)
      # Then set it as annotations
      temp["annotations"] = annotations
      # Append to the extracted logs array, end of pipeline
      extracted_logs.append(temp)
    # Return the extracted logs
    return extracted_logs
```

Figure 15: Function to extract necessary informations from coco dataset to register with detectron2 API

```python
[ ] def reg_datasets():
      """
      Function to register extracted informations to detectron2
      :return: None
      """
      keys = ["train", "val"]
      for k in keys:
        data.DatasetCatalog.register("log_" + k, lambda k=k: extract_log(DS_CONVERTED_PATH, f"{k}.json"))
        data.MetadataCatalog.get("log_" + k).set(thing_classes=["log"])
```

Figure 16: Function to register extracted informations with detectron2 API

```python
[ ] def visualiser(with_predictor, img_path, out_path):
      """
      Function to write the predicted images to a specific output path
      :with_predictor: Trained ML Model
      :img_path: Image location
      :out_path: Path to write predicted image
      :return: Output of predictor
      """

      # Read and predict image
      imr = cv2.imread(img_path)
      out = with_predictor(imr)

      # Get predicted boxes and scores
      instances = out["instances"].to("cpu")
      boxes = instances.pred_boxes.tensor.numpy()
      scores = instances.scores.numpy()

      # Iterate and draw output box with the score
      for box, score in zip(boxes, scores):
        x1, y1, x2, y2 = box.astype(int)
        box_color = (69, 173, 130)
        text_color = (0, 0, 0)
        cv2.rectangle(imr, (x1, y1), (x2, y2), box_color, 2)
        cv2.putText(imr, f"log: {score:.2f}", (x1, y1-10), cv2.FONT_HERSHEY_PLAIN, 0.5, text_color, 2)

      # Write the processed image to the output path
      cv2.imwrite(out_path, imr)
      return out
```

Figure 17: Function to process and save predicted images

```python
[ ] ensure_folder_exists()
    cfg = with_model_config()
    IS_TRAIN = True

    if IS_TRAIN:
      convert_from_labelme_to_coco()
      reg_datasets()

      trainer = engine.DefaultTrainer(cfg)
      trainer.resume_or_load(resume=False)

      trainer.train()

      checker = checkpoint.DetectionCheckpointer(trainer.model, save_dir=cfg.MODEL_OUTPUT_DIR)
      checker.save("final_model")

    cfg.MODEL.WEIGHTS = os.path.join(cfg.MODEL_OUTPUT_DIR, "final_model.pth")
    cfg.MODEL.ROI_HEADS.SCORE_THRESH_TEST = 0.66

    pred = engine.DefaultPredictor(cfg)
    test_imgs = use_test_images()

    for img in test_imgs:
      img_path = os.path.join(DS_LOG_PATH, img)
      out_path = os.path.join(IMG_OUT_PATH, f"result_{img}")

      outputs = visualiser(pred, img_path, out_path)

      print(f"Detected logs in {img} are: {counter(outputs)}")
```

Figure 18: Training and Testing Process

3. Models Evaluation

Detailed Task: Test the model with the Test set and generate images of detected log objects and confidence score. For example, the test outcome of one image will look similar to the following image. You will need to use OpenCV to produce such an image

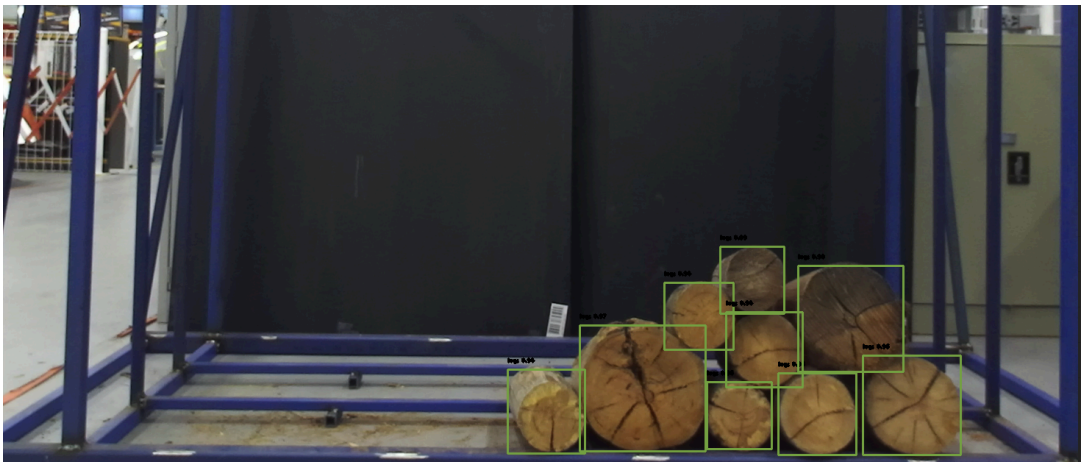

Figure 19: A screenshot showing the detected log objects with confidential score

4. Detected Log Counter

Detailed tasks: Write a Python program that counts the number of detected logs in each output image (Log counting)

```
def counter(outs):
  """
  Function to count number of logs
  """
  return len(outs["instances"])
```

Figure 20: Function to count number of logs

```
Detected logs in 00353-ZED-Left-1622131977.png are: 11
Detected logs in 00450-ZED-Left-1622131421.png are: 8
Detected logs in 00371-ZED-Left-1622132909.png are: 11
Detected logs in 00601-ZED-Left-1622129539.png are: 10
Detected logs in 00578-ZED-Right-1622133155.png are: 9
Detected logs in 00493-ZED-Left-1622133722.png are: 10
Detected logs in 00482-ZED-Left-1622133126.png are: 10
Detected logs in 00566-ZED-Left-1622132559.png are: 10
Detected logs in 00484-ZED-Left-1622133240.png are: 8
Detected logs in 00403-ZED-Left-1622128970.png are: 9
```

Figure 21: Counting results

Task 3: Extending log labelling to another class

Detailed Task: In Studio 5, Activity 1, you labelled the log with 10 images. Can you update the labels of those 10 images using labelme and replace the label of the logs that are broken as detected_log? Then save the updated label for all 10 images.

```
[ ]  def extend_labelling(json_file, path_to_save, log_threshold=100):
       orig_data = json.load(open(json_file, 'r'))
       if 'shapes' not in orig_data:
         print(f'Can not find shapes property in {orig_data}, ignoring...')
         return

       for s in orig_data['shapes']:
         if 'log' in s['label']:
           points = s['points']
           x_coordinations = []
           y_coordinations = []
           for p in points:
             x_coordinations.append(p[0])
             y_coordinations.append(p[1])
           width = max(x_coordinations) - min(x_coordinations)
           height = max(y_coordinations) - min(y_coordinations)

           if(width < log_threshold) or (height < log_threshold):
             s['label'] = 'detected_log'

       with open(path_to_save, 'w') as file:
         json.dump(orig_data, file, indent=2)
```

Figure 22: Function to find and extends the detected_log

```
[ ]  def process_extend_json():
       for f in os.listdir(ORIGINAL_PATH):
         if f.endswith('.png') or f.endswith('.jpg') or f.endswith('.jpeg'):
           shutil.copy(os.path.join(ORIGINAL_PATH, f), os.path.join(UPDATED_PATH, f))
         if f.endswith('.json'):
           print(f"Extending labels for {f}... ")
           json_file = os.path.join(ORIGINAL_PATH, f)
           save_path = os.path.join(UPDATED_PATH, f)

           extend_labelling(json_file, save_path)

           print(f"Finished extending labels for {f} and saved to {save_path}")

[ ]  process_extend_json()
```

Figure 23: Function to process and save both images and logs file to a new folder

```
Extending labels for 00310-ZED-Right-1622129553.json...
Finished extending labels for 00310-ZED-Right-1622129553.json and saved to /content/drive/MyDrive/Colab Notebooks/COS40007/Assignment4/extracted-converted-log-labelled/00310-ZED-Right-1622129553.json
Extending labels for 00310-ZED-Left-1622129553.json...
Finished extending labels for 00310-ZED-Left-1622129553.json and saved to /content/drive/MyDrive/Colab Notebooks/COS40007/Assignment4/extracted-converted-log-labelled/00310-ZED-Left-1622129553.json
Extending labels for 00314-ZED-Left-1622129763.json...
Finished extending labels for 00314-ZED-Left-1622129763.json and saved to /content/drive/MyDrive/Colab Notebooks/COS40007/Assignment4/extracted-converted-log-labelled/00314-ZED-Left-1622129763.json
Extending labels for 00311-ZED-Right-1622129599.json...
Finished extending labels for 00311-ZED-Right-1622129599.json and saved to /content/drive/MyDrive/Colab Notebooks/COS40007/Assignment4/extracted-converted-log-labelled/00311-ZED-Right-1622129599.json
Extending labels for 00312-ZED-Left-1622129636.json...
Finished extending labels for 00312-ZED-Left-1622129636.json and saved to /content/drive/MyDrive/Colab Notebooks/COS40007/Assignment4/extracted-converted-log-labelled/00312-ZED-Left-1622129636.json
Extending labels for 00311-ZED-Left-1622129599.json...
Finished extending labels for 00311-ZED-Left-1622129599.json and saved to /content/drive/MyDrive/Colab Notebooks/COS40007/Assignment4/extracted-converted-log-labelled/00311-ZED-Left-1622129599.json
Extending labels for 00313-ZED-Right-1622129715.json...
Finished extending labels for 00313-ZED-Right-1622129715.json and saved to /content/drive/MyDrive/Colab Notebooks/COS40007/Assignment4/extracted-converted-log-labelled/00313-ZED-Right-1622129715.json
Extending labels for 00312-ZED-Right-1622129636.json...
Finished extending labels for 00312-ZED-Right-1622129636.json and saved to /content/drive/MyDrive/Colab Notebooks/COS40007/Assignment4/extracted-converted-log-labelled/00312-ZED-Right-1622129636.json
Extending labels for 00313-ZED-Left-1622129715.json...
Finished extending labels for 00313-ZED-Left-1622129715.json and saved to /content/drive/MyDrive/Colab Notebooks/COS40007/Assignment4/extracted-converted-log-labelled/00313-ZED-Left-1622129715.json
Extending labels for 00309-ZED-Right-1622129510.json...
Finished extending labels for 00309-ZED-Right-1622129510.json and saved to /content/drive/MyDrive/Colab Notebooks/COS40007/Assignment4/extracted-converted-log-labelled/00309-ZED-Right-1622129510.json
```

Figure 24: Processing results

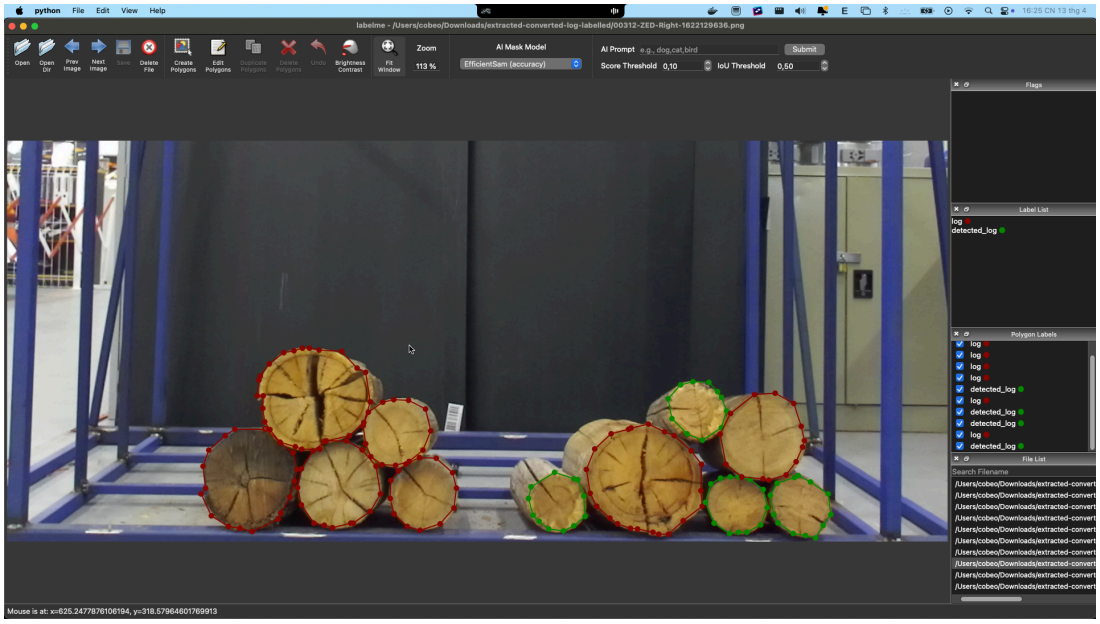

Figure 25: Screenshot of log and detected_log

Appendix

- Source code + datasets: https://github.com/cobeo2004/COS40007/tree/main/Assignment3
- Decistion Tree rules: https://github.com/cobeo2004/COS40007/blob/main/Assignment3/tree_rules.txt
- Saved models: https://github.com/cobeo2004/COS40007/tree/main/Assignment3/models