Portfolio Week 5 Report

Student Name: Dang Khoa Le
Student ID: 103844421
Tutorial: Tuesday 6.30-8.30 PM

Task 1: Developing CNN and ResNet50 for Rust Classification

Introduction

The goal of this task is to develop and evaluate two deep learning models (a simple CNN and a more complex ResNet50) for classifying rust and no-rust images from a given structural asset dataset. The dataset consists of labeled images categorized into two classes: "rust" and "no rust." The following steps were performed:

1. **Dataset Preparation**:
    - The dataset was split into training, validation, and test sets. The test set (Test Set Rust folder) consists of 20 randomly selected images (10 rust and 10 no rust) that were excluded from the training set.
2. **Model Development**:
    - **Simple CNN Model**: A convolutional neural network (CNN) similar to MNIST classification was developed and trained on the "Corrosion" dataset.
    - **ResNet50 Model**: A more complex CNN based on the ResNet50 architecture was developed and trained on the same dataset.
3. **Model Evaluation**:
    - Both models were evaluated using the test set to measure their accuracy.

Dataset

The dataset used for this task includes the following folders:

- **Corrosion/rust**: Contains images labeled as rust.
- **Corrosion/no rust**: Contains images labeled as no rust.
- **Test Set Rust**: Contains a mix of rust and no rust images used for model testing.

Model 1: Simple CNN

Architecture

The simple CNN model was designed as follows:

- **Input Layer**: 28x28x3 (RGB image).
- **Convolutional Layer**: 32 filters, 3x3 kernel size, ReLU activation.
- **MaxPooling Layer**: 2x2 pool size.
- **Flatten Layer**: Flattens the input to a 1D vector.
- **Dense Layer**: 64 units, ReLU activation.
- **Output Layer**: 1 unit, sigmoid activation (binary classification).

Training

The model was trained for 10 epochs with the following settings:

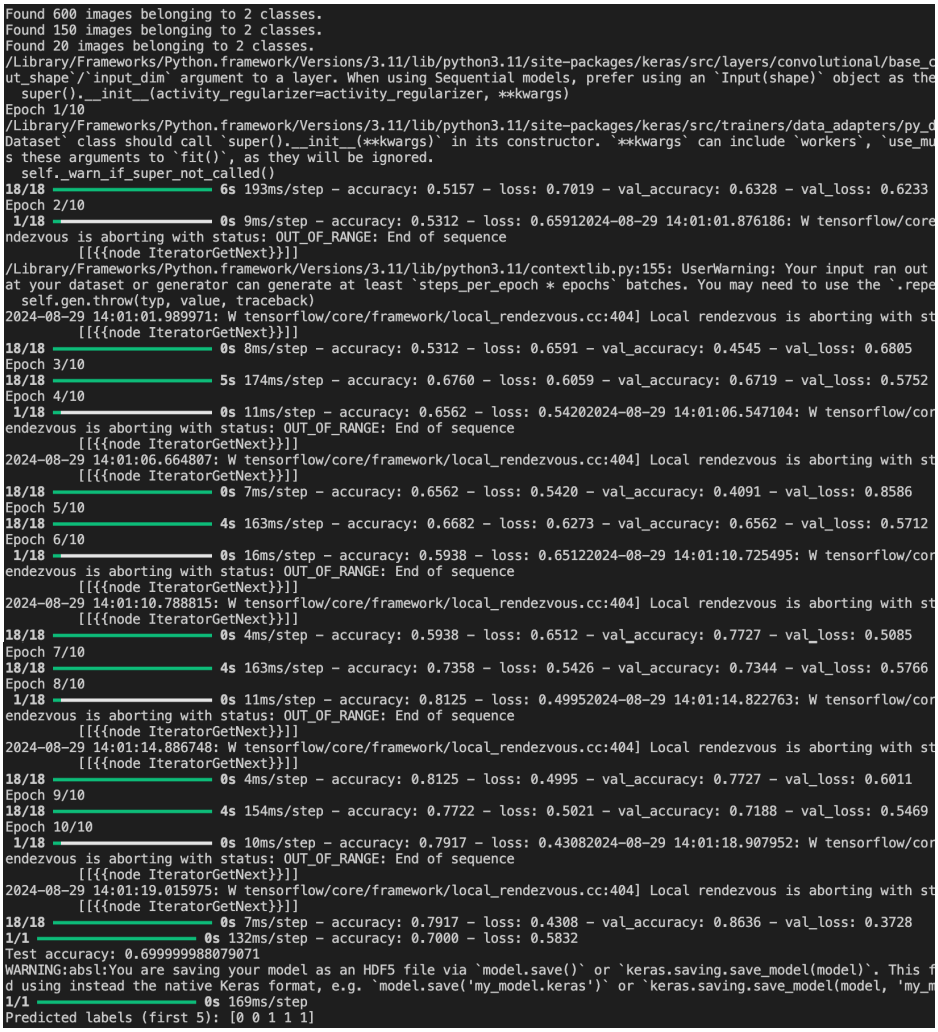
- **Optimizer**: Adam.
- **Loss Function**: Binary Crossentropy.
- **Metrics**: Accuracy.

Results

- **Training Accuracy**: The model achieved a training accuracy that varied over the epochs.
- **Validation Accuracy**: Validation accuracy was monitored to evaluate model performance.

- **Test Accuracy**: The final test accuracy achieved by the model was approximately 70%.

CNN model result (10 Epoch)



Prediction

Predicted labels (first 5): [0 0 1 1 1]

Simple CNN model saved as ‘rust_cnn_model.h5’.

Model 2: ResNet50

Architecture

The ResNet50 model was designed with the following components:

- **Base Model**: ResNet50 pre-trained on ImageNet, with the top layer removed.
- **Dense Layer**: 128 units, ReLU activation.
- **Output Layer**: 1 unit, sigmoid activation.

Training

The model was trained for 10 epochs with early stopping and model checkpointing:

- **Optimizer**: Adam.
- **Loss Function**: Binary Crossentropy.
- **Metrics**: Accuracy.
- **Early Stopping**: Monitoring validation loss with patience of 3 epochs.

Results

- **Training Accuracy**: The model's training accuracy fluctuated over the epochs.
- **Validation Accuracy**: The model's performance on the validation set was closely monitored.
- **Test Accuracy**: The final test accuracy achieved by the ResNet50 model was **50%**.

RestNet50 model saved as ‘rust_resnet50_model.h5’.

Best model during training was saved as ‘best_rust_resnet50_model.keras’.

ResNet50 model results (10 Epochs)

```
Epoch 1/10
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/keras/src/trainers/data_adapters/py_
Dataset` class should call `super().__init__(**kwargs)` in its constructor. `**kwargs` can include `workers`, `use_m
s these arguments to `fit()`, as they will be ignored.
  self._warn_if_super_not_called()
18/18 ━━━━━━━━━━━━━━━━━━━━ 51s 2s/step - accuracy: 0.4824 - loss: 0.7600 - val_accuracy: 0.5469 - val_loss: 0.6898
Epoch 2/10
 1/18 ━━━━━━━━━━━━━━━━━━━━ 29s 2s/step - accuracy: 0.5312 - loss: 0.69332024-08-29 13:53:43.763040: W tensorflow/cor
ndezvous is aborting with status: OUT_OF_RANGE: End of sequence
         [[{{node IteratorGetNext}}]]
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/contextlib.py:155: UserWarning: Your input ran out
at your dataset or generator can generate at least `steps_per_epoch * epochs` batches. You may need to use the `.rep
  self.gen.throw(typ, value, traceback)
2024-08-29 13:53:44.933736: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is aborting with s
         [[{{node IteratorGetNext}}]]
18/18 ━━━━━━━━━━━━━━━━━━━━ 4s 163ms/step - accuracy: 0.5312 - loss: 0.6933 - val_accuracy: 0.5455 - val_loss: 0.6878
Epoch 3/10
18/18 ━━━━━━━━━━━━━━━━━━━━ 42s 2s/step - accuracy: 0.4789 - loss: 0.7048 - val_accuracy: 0.5156 - val_loss: 0.6962
Epoch 4/10
 1/18 ━━━━━━━━━━━━━━━━━━━━ 42s 2s/step - accuracy: 0.4062 - loss: 0.68592024-08-29 13:54:31.356746: W tensorflow/cor
ndezvous is aborting with status: OUT_OF_RANGE: End of sequence
         [[{{node IteratorGetNext}}]]
2024-08-29 13:54:32.865750: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is aborting with s
         [[{{node IteratorGetNext}}]]
18/18 ━━━━━━━━━━━━━━━━━━━━ 6s 186ms/step - accuracy: 0.4062 - loss: 0.6859 - val_accuracy: 0.4545 - val_loss: 0.6876
Epoch 5/10
18/18 ━━━━━━━━━━━━━━━━━━━━ 42s 2s/step - accuracy: 0.5595 - loss: 0.6990 - val_accuracy: 0.5547 - val_loss: 0.7051
Epoch 6/10
 1/18 ━━━━━━━━━━━━━━━━━━━━ 26s 2s/step - accuracy: 0.4062 - loss: 0.80362024-08-29 13:55:18.076970: W tensorflow/cor
ndezvous is aborting with status: OUT_OF_RANGE: End of sequence
         [[{{node IteratorGetNext}}]]
2024-08-29 13:55:19.153029: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is aborting with s
         [[{{node IteratorGetNext}}]]
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 65ms/step - accuracy: 0.4062 - loss: 0.8036 - val_accuracy: 0.4545 - val_loss: 0.7346
Epoch 7/10
18/18 ━━━━━━━━━━━━━━━━━━━━ 49s 3s/step - accuracy: 0.5209 - loss: 0.6925 - val_accuracy: 0.5469 - val_loss: 0.6862
Epoch 8/10
 1/18 ━━━━━━━━━━━━━━━━━━━━ 32s 2s/step - accuracy: 0.5625 - loss: 0.68962024-08-29 13:56:10.005239: W tensorflow/cor
ndezvous is aborting with status: OUT_OF_RANGE: End of sequence
         [[{{node IteratorGetNext}}]]
2024-08-29 13:56:11.522381: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is aborting with s
         [[{{node IteratorGetNext}}]]
18/18 ━━━━━━━━━━━━━━━━━━━━ 10s 450ms/step - accuracy: 0.5625 - loss: 0.6896 - val_accuracy: 0.3636 - val_loss: 0.731
Epoch 9/10
18/18 ━━━━━━━━━━━━━━━━━━━━ 44s 2s/step - accuracy: 0.5692 - loss: 0.6833 - val_accuracy: 0.6172 - val_loss: 0.6945
Epoch 10/10
 1/18 ━━━━━━━━━━━━━━━━━━━━ 21s 1s/step - accuracy: 0.6250 - loss: 0.67162024-08-29 13:57:03.322163: W tensorflow/cor
ndezvous is aborting with status: OUT_OF_RANGE: End of sequence
         [[{{node IteratorGetNext}}]]
2024-08-29 13:57:04.798760: W tensorflow/core/framework/local_rendezvous.cc:404] Local rendezvous is aborting with s
         [[{{node IteratorGetNext}}]]
18/18 ━━━━━━━━━━━━━━━━━━━━ 3s 89ms/step - accuracy: 0.6250 - loss: 0.6716 - val_accuracy: 0.3636 - val_loss: 0.7425
1/1 ━━━━━━━━━━━━━━━━━━━━ 1s 1s/step - accuracy: 0.5000 - loss: 0.7560
Test accuracy: 0.5
```

Model Summary

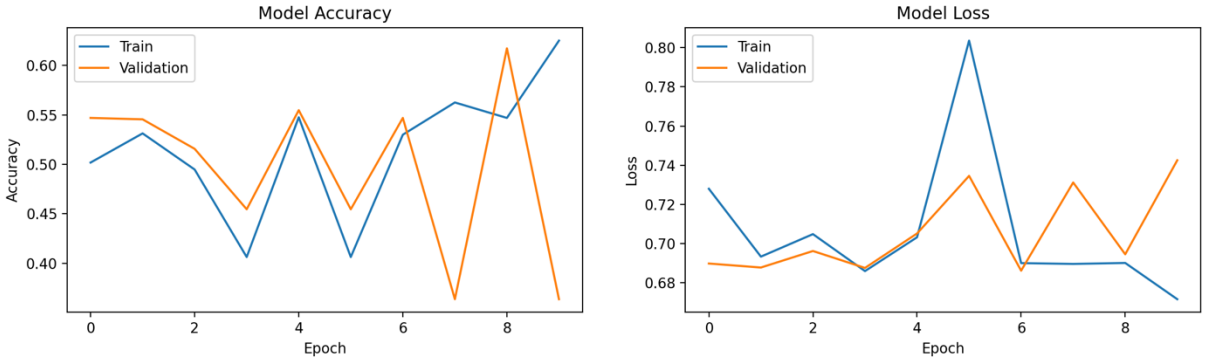
Model: "sequential_1"

Layer (type)	Output Shape	Param #
resnet50 (Functional)	(None, 2048)	23,587,712
dense_2 (Dense)	(None, 128)	262,272
dense_3 (Dense)	(None, 1)	129

Total params: 24,374,917 (92.98 MB)
Trainable params: 262,401 (1.00 MB)
Non-trainable params: 23,587,712 (89.98 MB)
Optimizer params: 524,804 (2.00 MB)

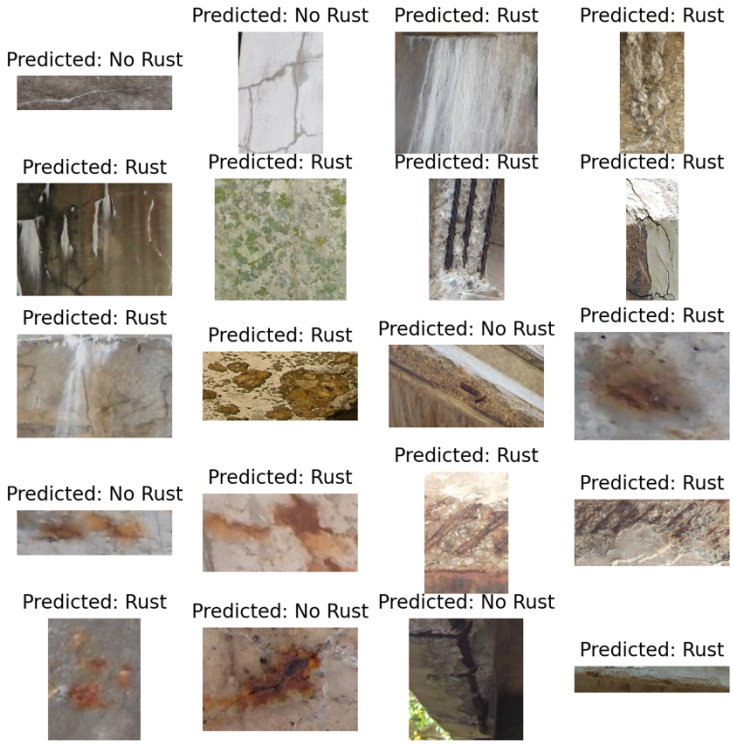
Accuracy and Loss

The graphs below show the ResNet50 model's accuracy and loss over the epochs.



Rust and No Rust Predictions

The following figure shows the predictions made by the ResNet50 model on a subset of test images (20 test images), yields relatively low accuracy of prediction in return.



Task 2: Develop Mask RCNN for detecting log

Introduction

The goal of this task is to develop and evaluate Mask R-CNN for detecting logs. The following steps were performed:

Technical Usage: Due to Mask RCNN in Keras compatibility issue, this assessment has been done using Pytorch version of Mask RCNN.

Dependencies and Helpers installations:

These helper scripts was run to install necessary dependencies used to support Mask RCNN – Pytorch version:

curl -O https://raw.githubusercontent.com/pytorch/vision/main/references/detection/engine.py
curl -O https://raw.githubusercontent.com/pytorch/vision/main/references/detection/utils.py
curl -O https://raw.githubusercontent.com/pytorch/vision/main/references/detection/coco_eval.py

curl -O https://raw.githubusercontent.com/pytorch/vision/main/references/detection/coco_utils.py
curl -O https://raw.githubusercontent.com/pytorch/vision/main/references/detection/transforms.py

As well as these dependencies' installations:
Install Torch and Torchvision for Mask R-CNN:
pip install torch torchvision
Install pycocotools for COCO evaluation:
pip install pycocotools
Install OpenCV for image processing:
pip install opencv-python
Install labelme2coco for label conversion:
pip install labelme2coco

Dataset Preparation

- **Dataset Composition**: The dataset comprises two folders:
  - **log-labelled**: Contains images and corresponding LabelMe configuration files (.json) for training. This folder is used to train the Mask R-CNN model.
  - **Test Set Log**: Contains 10 images and their respective LabelMe configuration files, which are reserved for testing the trained Mask R-CNN model.
- **Label Conversion**: The LabelMe annotations from the log-labelled folder were converted to COCO format using the labelme2coco library. This conversion was necessary to train the Mask R-CNN model, which relies on COCO-formatted annotations.

Model Development

- **Dataset Class Definition**: A custom LogDataset class was implemented to load the dataset. This class handles loading images, processing annotations (including masks and bounding boxes), and preparing the data for training.
- **Mask R-CNN Model Configuration**:
  - The Mask R-CNN model was configured using a ResNet-50 backbone with Feature Pyramid Networks (FPN). The model was modified to output predictions for one class (log) plus the background.
  - The model was trained using the Pytorch framework, and an SGD optimizer with a learning rate of 0.005, momentum of 0.9, and weight decay of 0.0005 was used.

Training

- **Training Setup**: The model was trained on a subset of 20 images from the log-labelled folder for one epoch to ensure a manageable runtime.
- **Training Output**: The training process completed in approximately 6 minutes per epoch, with the loss reducing from an initial value of 6.5321 to a final value of 4.7646. The primary sources of loss were the mask loss and the classifier loss, which are expected as these components are crucial for accurate object detection and segmentation.

Testing and Log Counting

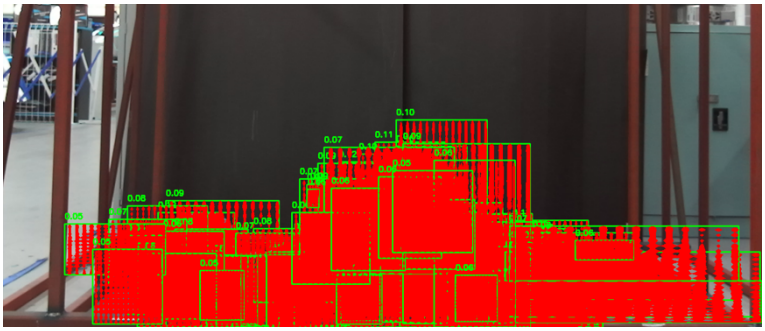
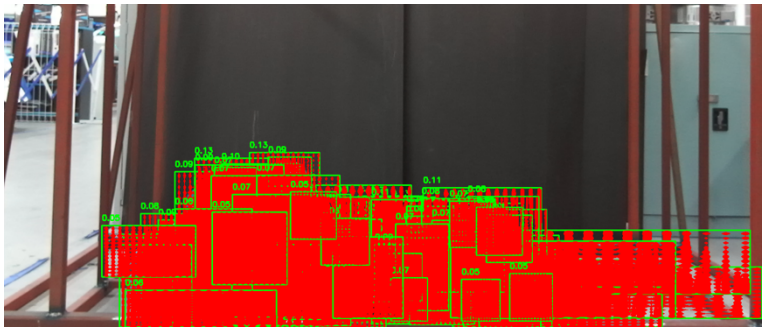
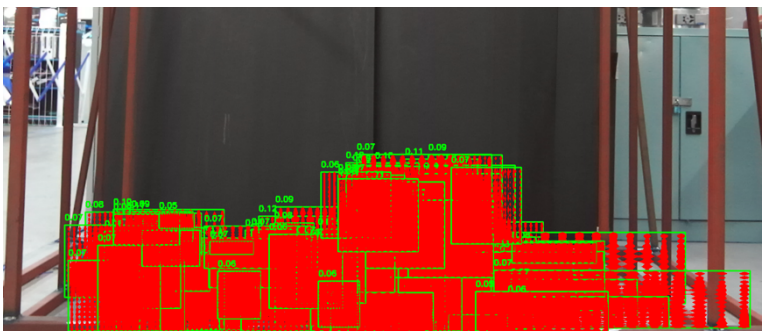
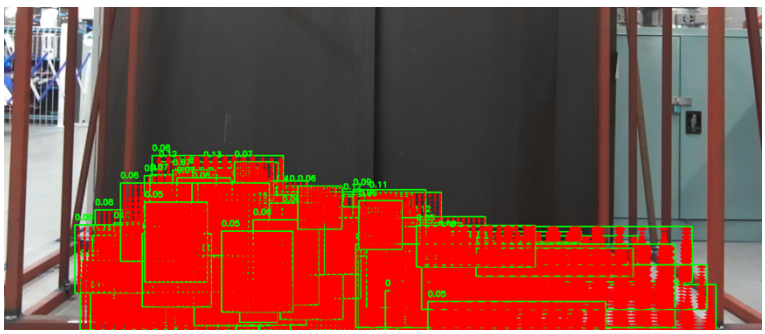
- **Testing**: The trained model was tested on the 10 images from the Test Set Log folder. The model successfully detected logs in each image, with the number of detected logs ranging from 50 to 75 per image.
- **Visualization**: For each test image, bounding boxes and masks were drawn around detected logs, and confidence scores were displayed on the images. The processed

images were saved as output_image_i.png, where i ranges from 0 to 9. OpenCV test output image was saved and allocated in Log Detection folder.

Test Result.

```
khoale@khoas-MacBook-Pro-2 A4 % python3 a4task2.py
Step 1 completed ------------------------------------------------------------------
Step 2 completed ------------------------------------------------------------------
Step 3 completed ------------------------------------------------------------------
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/torchvision/transforms/v2/_deprecated.py:41: UserWarning: The tr
ansform `ToTensor()` is deprecated and will be removed in a future release. Instead, please use `v2.Compose([v2.ToImage(), v2.ToDtype(torch.floa
t32, scale=True)])`.
  warnings.warn(
Epoch: [0]  [0/5]  eta: 0:05:52  lr: 0.001254  loss: 6.5321 (6.5321)  loss_classifier: 1.2234 (1.2234)  loss_box_reg: 0.7564 (0.7564)  loss_mask
: 4.1294 (4.1294)  loss_objectness: 0.4007 (0.4007)  loss_rpn_box_reg: 0.0221 (0.0221)  time: 70.5540  data: 0.2196
^[[B^[[AEpoch: [0]  [4/5]  eta: 0:01:15  lr: 0.005000  loss: 4.2196 (4.7646)  loss_classifier: 0.8075 (0.8619)  loss_box_reg: 0.7345 (0.7339)  l
oss_mask: 2.1163 (2.8070)  loss_objectness: 0.3593 (0.3302)  loss_rpn_box_reg: 0.0237 (0.0315)  time: 75.7255  data: 0.4418
Epoch: [0] Total time: 0:06:18 (75.7300 s / it)
Training complete!
Step 4 completed ------------------------------------------------------------------
Detected 69 logs in the image.
Detected 71 logs in the image.
Detected 61 logs in the image.
Detected 75 logs in the image.
Detected 55 logs in the image.
Detected 62 logs in the image.
Detected 73 logs in the image.
Detected 71 logs in the image.
Detected 50 logs in the image.
Detected 67 logs in the image.
```

Log Detection Outputs

Above are the demonstrations of the first 4 log output images (output_image_i.png with I ranges from 0 to 3).

Model Evaluation

- **Run Time**: The training process was efficient, taking approximately 6 minutes per epoch. This was achieved by using a reduced subset of 20 images, a single epoch of training, and a batch size of 4. However, the time required for training can be considered significant due to the computational complexity of Mask R-CNN models.
- **Loss Analysis**: The final loss values indicated that the model was learning to detect and segment logs effectively, though further training could potentially improve accuracy.
- **Log Detection Results**: The model detected a varying number of logs across the test images, with detected counts ranging from 50 to 75 logs per image. These results suggest that the model has a low-accuracy detection accuracy, though the variance in

detection counts may indicate challenges in accurately segmenting overlapping logs or small logs.

Suggestions: Due to time, storage and resource insensitivity, we opted in to only take a subset size of 20 training images, and only 1 epoch loop iterated for this task (which already take approximately 30 minutes worth of training, a complete test would cost approximately 3 hours on my training device), hence, the outcome result come with relatively low accuracy with log count detection and overlappings. Hence, it's recommended to train data with the completed log-labelled dataset (590 images and configurations in json file) to allow better training outcome, as well as raising the num_epochs variable (number of epochs iterated in the loop) from STEP 4, given from the a4task2.py script.

Conclusion

The Mask R-CNN model was successfully developed and trained to detect logs in a set of test images. However, the training process was not efficient, further optimization could be beneficial for improving detection accuracy and reducing variance in log counts across test images.

Task 3: Extending log labelling to another class

Task Overview

In this task, the objective is to extend the existing log labelling by updating the label for broken logs in the dataset. Specifically, logs that are broken in the images need to be relabelled as "detected_log." The task involves both manual label adjustment using LabelMe and automated label updating through a Python script (a4task3.py).

Technical Analysis

1. **Manually Update Labels Using labelme:**
   - Before running the Python script, I will manually open directory 'log-images', which storing each image and its corresponding JSON configuration file in the previously configured (sketched the polygonal annotation specifying the log) using LabelMe tool.
   - Once the image is loaded, inspect the logs and identify any broken logs. Change their label from "log" to "broken".
   - Save the updated labels for each image. This will modify the corresponding JSON annotation file for each image.

The above image illustrate the modification of broken logs' labelling from "log" to "broken".

2. **Python Script (a4task3.py):**
   - After manually updating the labels to "broken" using labelme, the next step is to run the Python script (a4task3.py). This script automatically processes the updated labels and changes all occurrences of "broken" to "detected_log."
   - The script iterates over all JSON files in the log-images folder and makes the necessary label changes.

Script Explanation:

1. **Directory Setup:**
   - The directory log-images is specified, which contains the images (.png files) and their corresponding label annotations (.json files) that were updated using labelme.
2. **Loading and Processing JSON Files:**
   - The script iterates through all JSON files in the log-images folder. Each JSON file contains annotation data for its corresponding image.
   - The update_labels function opens each JSON file and reads its content using Python's json module.
3. **Identifying and Updating Labels:**
   - The script checks each shape (object annotation) in the JSON file for the label 'broken'.
   - If a shape is labelled as 'broken', it is updated to 'detected_log'.
4. **Saving the Updated JSON Files:**
   - After updating all relevant labels in a JSON file, the modified annotation data is saved back to the same JSON file, ensuring that the changes are persistent.
5. **Console Output:**
   - For each processed JSON file, a message is printed to the console, confirming that the labels in that file have been successfully updated.
6. **Completion:**
   - Once all JSON files have been processed, a final message is printed to the console indicating that all labels have been updated.

```
khoale@khoas-MacBook-Pro-2 A4 % python3 a4task3.py
Updated labels in log-images/img_0008.json
Updated labels in log-images/img_0024.json
Updated labels in log-images/img_0012.json
Updated labels in log-images/img_0004.json
Updated labels in log-images/img_0028.json
Updated labels in log-images/img_0022.json
Updated labels in log-images/img_0017.json
Updated labels in log-images/img_0001.json
Updated labels in log-images/img_0020.json
Updated labels in log-images/img_0026.json
All labels updated successfully.
```

Above is the script output indicating all broken logs' labels from each of these json configuration files has been updated successfully into 'detected_log'.

Appendix

1. **Source Codes:** Python script files are provided in 'code' folder, which also includes the codes given from downloading necessary dependencies and helpers. There are 3 scripts files for task 1 to 3, named as 'a4task1.py', 'a4 task2.py' and 'a4task3.py'.
2. **Training Datasets:** The training datasets used including the folders of 'Corrosion' for task 1, 'log-labelled' for task 2, and 'log-images' for task 3.
3. **Test Datasets:** Datasets used for testing includes "Test Set Rust" for task 1 and "Test Set Log" for task 2.
4. **Results:**
   a. cnn_test: Folder contains CNN model's result saved as 'rust_cnn_model.h5'.
   b. resent50_test: Folder contains ResNet50 model's result saved as 'rust_resnet50_model.h5' and its best model as 'best_rust_resnet50_model.keras'.

c. rcnn_test: Folder contain its sub-folder ‘Log Detection’ with 10 images of log count detection from task 2.

d. export/coco: ‘export’ folder with its sub-folder ‘coco’ contains dataset.json, which is the converted configuration of the log training dataset’s configuration from LabelMe to COCO format.

5. Cloud: All of these files are saved on Google Drive, accessible via this link: https://drive.google.com/drive/folders/1lAYMPHBU49OCyb5fsr3ieT1XKzK45lwu?usp=sharing

Notice: If you wish to run the script files, please make sure to return all scripts from ‘code’ folder back to the parent directory so it could import and use datasets.