6-Sept-24

PORTFOLIO

WEEK 5 - DOCUMENTATION

Tran Duc Anh Dang | 103995439
STUDIO CLASS: STUDIO 1-3

Abstract

This portfolio submission demonstrates the development and evaluation of deep learning models for image classification and object detection tasks using CNN, ResNet50, and Mask R-CNN. The submission includes labeled datasets, model outcomes, and source code structured as follows:

- Labeled Log Dataset: A set of 10 annotated images using the LabelMe tool and the corresponding JSON file.
- CNN Model: Test outcomes of a basic CNN model trained to classify images into "rust" and "no rust," with images and results saved in the folder “cnn_test.”
- ResNet50 Model: Test outcomes of a ResNet50 model for the same classification task, with results saved in “resnet50_test.”
- Mask R-CNN Model: Developed for detecting log objects in images. The model outputs detection results, including bounding boxes, confidence scores, and segmentation masks. The results are saved in the “rcnn_test” folder.
- Source Code: The source code for all models is located in the folder “code.”

You can find the requirements, documentation and source code file at:
Requirements: https://github.com/kinqsradio/COS40007-Artificial-Intelligence-for-Engineering/tree/main/week-05-portfolio/requirements
Documentation: https://github.com/kinqsradio/COS40007-Artificial-Intelligence-for-Engineering/tree/main/week-05-portfolio/docs
Code: https://github.com/kinqsradio/COS40007-Artificial-Intelligence-for-Engineering/tree/main/week-05-portfolio/code
dataset.json: https://github.com/kinqsradio/COS40007-Artificial-Intelligence-for-Engineering/tree/main/week-05-portfolio/my_coco_annotations
cnn_test: https://github.com/kinqsradio/COS40007-Artificial-Intelligence-for-Engineering/tree/main/week-05-portfolio/cnn_test
resnet50_test: https://github.com/kinqsradio/COS40007-Artificial-Intelligence-for-Engineering/tree/main/week-05-portfolio/resnet50_test
rcnn_test: https://github.com/kinqsradio/COS40007-Artificial-Intelligence-for-Engineering/tree/main/week-05-portfolio/rnn_test

Table of Contents

ABSTRACT.....1

TASK 1: DEVELOP CNN AND RESNET50 3

TASK 2: DEVELOP MASK RCNN FOR DETECTING LOG.....6

TASK 3: EXTENDING LOG LABELLING TO ANOTHER CLASS 8

Task 1: Develop CNN and Resnet50

1. First randomly take out 10 rust and 10 no rust images for testing (We call this as Test Set). So, your training set will not contain these 20 images. yes

```python
# Function to randomly select images for the test set
def select_images(src_folder, dest_folder, num_images=10):
    images = os.listdir(src_folder)
    selected_images = random.sample(images, num_images)
    for image in selected_images:
        shutil.move(os.path.join(src_folder, image), os.path.join(dest_folder, image))

# Move 10 images from each category to the test set
select_images(NO_RUST_DATASET_FOLDER, f'{TEST_DIR}/no_rust', num_images=10)
select_images(RUST_DATASET_FOLDER, f'{TEST_DIR}/rust', num_images=10)

print("Test set created with 10 rust and 10 no_rust images.")
```

[60] MagicPython

```
Test set created with 10 rust and 10 no_rust images.
```

```python
# Load and check dataset labels
dataset = ImageFolder(DATA_DIR)
print("Classes:", dataset.classes)
print("Class to index mapping:", dataset.class_to_idx)

# Check if the labels are correct
labels = [label for _, label in dataset]
print("Unique labels in the dataset:", set(labels))
```

[61] MagicPython

```
Classes: ['no_rust', 'rust']
Class to index mapping: {'no_rust': 0, 'rust': 1}
Unique labels in the dataset: {0, 1}
```

```python
# Data transformation
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])

# Load training dataset (only rust and no_rust)
train_dataset = datasets.ImageFolder(DATA_DIR, transform=transform)

# Load test dataset from the separate test directory
test_dataset = datasets.ImageFolder(TEST_DIR, transform=transform)

# Data loaders
train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=4, shuffle=False)

# Verify the classes and label indices
print(f"Training set size: {len(train_loader.dataset)}")
print(f"Test set size: {len(test_loader.dataset)}")
print(f"Classes in training set: {train_dataset.classes}")
print(f"Classes in test set: {test_dataset.classes}")
```

[63] MagicPython

```
Training set size: 570
Test set size: 40
Classes in training set: ['no_rust', 'rust']
Classes in test set: ['no_rust', 'rust']
```

2. Develop a simple CNN model similar to minst 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 correct classification of 20 images in the test set)

SimpleCNN

```python
# Simple CNN model
class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(3, 16, kernel_size=3, stride=1, padding=1)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, stride=1, padding=1)
        self.fc1 = nn.Linear(32 * 56 * 56, 128)
        self.fc2 = nn.Linear(128, 2)  # Output layer for 2 classes: rust and no_rust

    def forward(self, x):
        x = torch.relu(self.conv1(x))
        x = torch.max_pool2d(x, 2)
        x = torch.relu(self.conv2(x))
        x = torch.max_pool2d(x, 2)
        x = x.view(-1, 32 * 56 * 56)
        x = torch.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# Initialize the model, loss function, and optimizer
simplecnn = SimpleCNN()
simplecnn.to(device)
criterion = nn.CrossEntropyLoss()  # Use CrossEntropyLoss for multi-class classification
optimizer = optim.Adam(simplecnn.parameters(), lr=0.001)

print("SimpleCNN model initialized.")
```

[58] MagicPython

```
SimpleCNN model initialized.
```

Result Tables:

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

Accuracy: 100%

The result for **test outcome** containing **images** can be found at **cnn_test** folder!

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

Resnet50

```
# Training the ResNet50 model
# Load the pre-trained ResNet50 model
resnet50 = models.resnet50(pretrained=True)
num_ftrs = resnet50.fc.in_features
resnet50.fc = nn.Linear(num_ftrs, 2)  # Adjust the final layer for 2 classes: rust and no_rust

# Use the same loss function and optimizer
optimizer = optim.Adam(resnet50.parameters(), lr=0.001)

resnet50.to(device)
# Training the ResNet50 model
for epoch in range(num_epochs):
    resnet50.train()
    running_loss = 0.0
    for images, labels in train_loader:
        optimizer.zero_grad()
        outputs = resnet50(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    print(f'Epoch {epoch+1}/{num_epochs}, Loss: {running_loss/len(train_loader):.4f}')

# Save the ResNet50 model
torch.save(resnet50.state_dict(), 'week-05-portfolio/models/resnet50.pth')

print("ResNet50 model training complete. Model saved as 'week-05-portfolio/models/resnet50.pth'.")
```

[42] MagicPython

```
Epoch 1/10, Loss: 0.6413
Epoch 2/10, Loss: 0.4348
Epoch 3/10, Loss: 0.3388
Epoch 4/10, Loss: 0.3723
Epoch 5/10, Loss: 0.3126
Epoch 6/10, Loss: 0.3812
Epoch 7/10, Loss: 0.3050
Epoch 8/10, Loss: 0.2634
Epoch 9/10, Loss: 0.1976
Epoch 10/10, Loss: 0.2493
ResNet50 model training complete. Model saved as 'resnet50.pth'.
```

Result Tables:

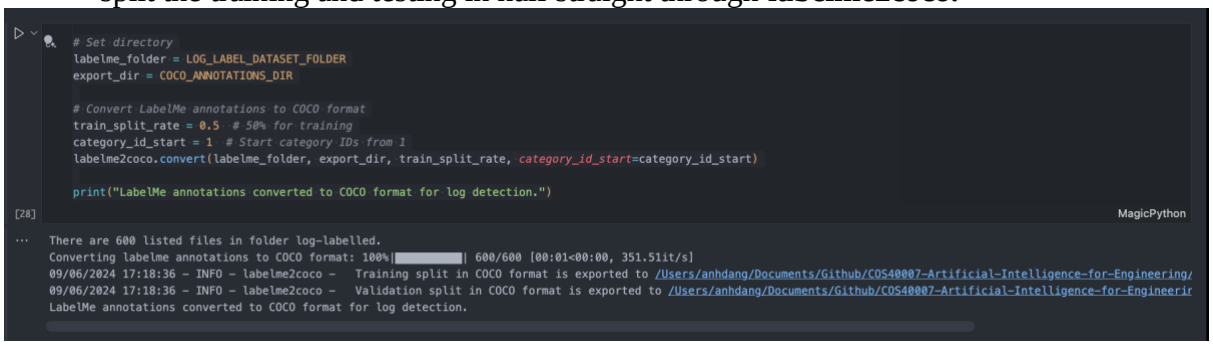
True Class	Predicted Class
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
0	0
1	1
1	1
1	1
1	1
1	1
1	1
1	0
1	1
1	1
1	1

Accuracy: 95%

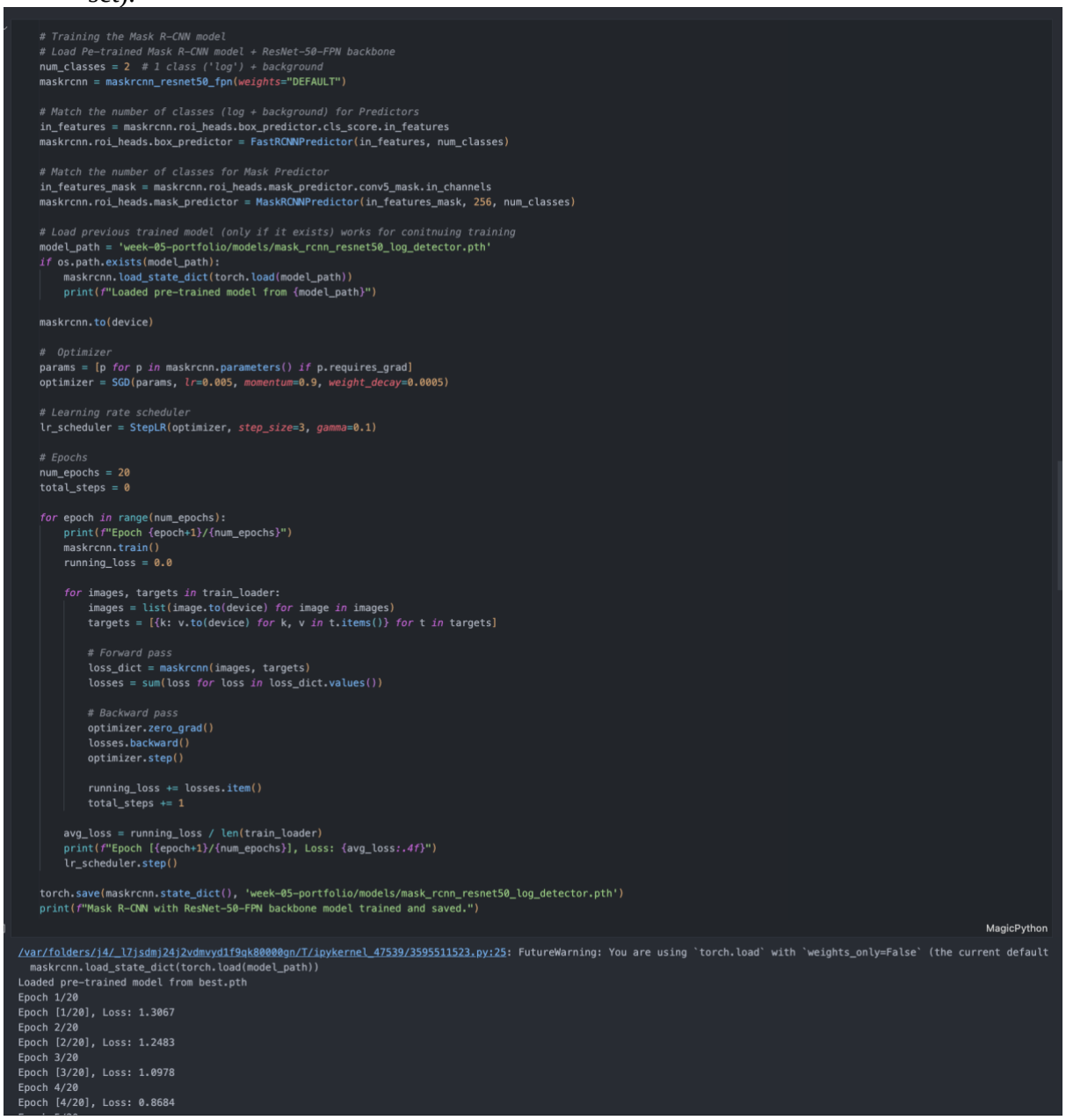
The result for **test outcome** containing **images** can be found at **resnet50_test** folder!

Task 2: Develop Mask RCNN for Detecting Log

1. First randomly take out 10 images for testing (We call this as Test Set)
   - For this task as I'm facing some hardware limitation on my Macbook, I have actually split the training and testing in half straight through **labelme2coco.**



2. Develop a Mask RCNN model using the labelled log as training data (excludes test set).



```python
# Training the Mask R-CNN model
# Load Pe-trained Mask R-CNN model + ResNet-50-FPN backbone
num_classes = 2  # 1 class ('log') + background
maskrcnn = maskrcnn_resnet50_fpn(weights="DEFAULT")

# Match the number of classes (log + background) for Predictors
in_features = maskrcnn.roi_heads.box_predictor.cls_score.in_features
maskrcnn.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

# Match the number of classes for Mask Predictor
in_features_mask = maskrcnn.roi_heads.mask_predictor.conv5_mask.in_channels
maskrcnn.roi_heads.mask_predictor = MaskRCNNPredictor(in_features_mask, 256, num_classes)

# Load previous trained model (only if it exists) works for conitnuing training
model_path = 'week-05-portfolio/models/mask_rcnn_resnet50_log_detector.pth'
if os.path.exists(model_path):
    maskrcnn.load_state_dict(torch.load(model_path))
    print(f"Loaded pre-trained model from {model_path}")

maskrcnn.to(device)

#  Optimizer
params = [p for p in maskrcnn.parameters() if p.requires_grad]
optimizer = SGD(params, lr=0.005, momentum=0.9, weight_decay=0.0005)

# Learning rate scheduler
lr_scheduler = StepLR(optimizer, step_size=3, gamma=0.1)

# Epochs
num_epochs = 20
total_steps = 0

for epoch in range(num_epochs):
    print(f"Epoch {epoch+1}/{num_epochs}")
    maskrcnn.train()
    running_loss = 0.0

    for images, targets in train_loader:
        images = list(image.to(device) for image in images)
        targets = [{k: v.to(device) for k, v in t.items()} for t in targets]

        # Forward pass
        loss_dict = maskrcnn(images, targets)
        losses = sum(loss for loss in loss_dict.values())

        # Backward pass
        optimizer.zero_grad()
        losses.backward()
        optimizer.step()

        running_loss += losses.item()
        total_steps += 1

    avg_loss = running_loss / len(train_loader)
    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {avg_loss:.4f}")
    lr_scheduler.step()

torch.save(maskrcnn.state_dict(), 'week-05-portfolio/models/mask_rcnn_resnet50_log_detector.pth')
print(f"Mask R-CNN with ResNet-50-FPN backbone model trained and saved.")
```

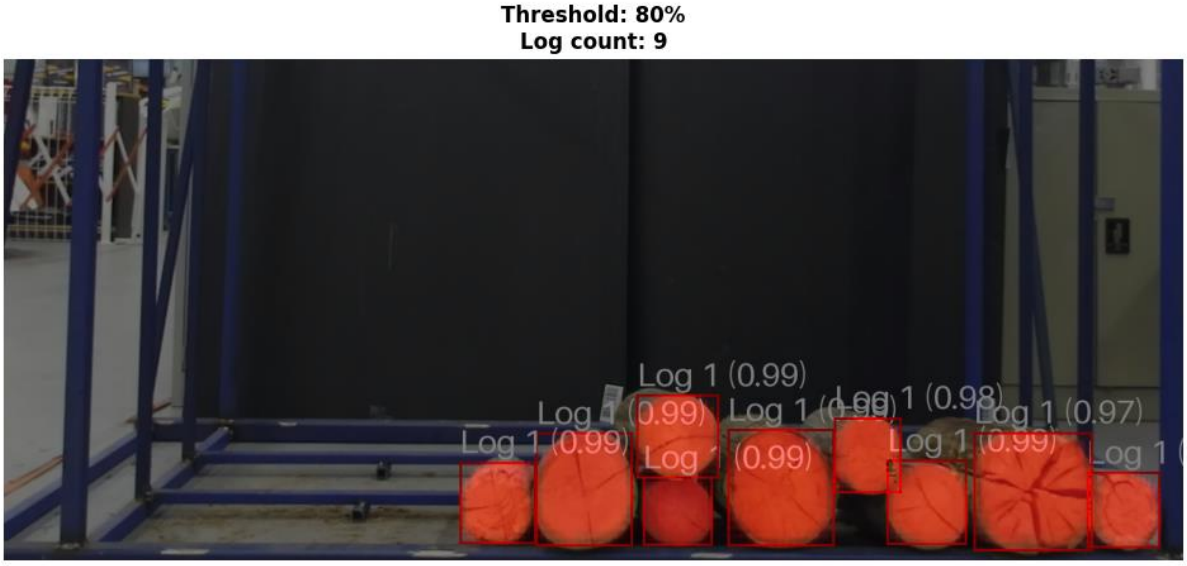
MagicPython

```
/var/folders/j4/_l7jsdmj24j2vdmvyd1f9qk80000gn/T/ipykernel_47539/3595511523.py:25: FutureWarning: You are using `torch.load` with `weights_only=False` (the current default
  maskrcnn.load_state_dict(torch.load(model_path))
Loaded pre-trained model from best.pth
Epoch 1/20
Epoch [1/20], Loss: 1.3067
Epoch 2/20
Epoch [2/20], Loss: 1.2483
Epoch 3/20
Epoch [3/20], Loss: 1.0978
Epoch 4/20
Epoch [4/20], Loss: 0.8684
```

Initialially I have trained the model around 17 epochs while having the train loss down to aproximately around 2.6 and then I further training the model for another 20 epochs and able to get the train loss down to around 0.6 which is a significantly improving from 2.6. This process took around 6-7 hours.

3. Test the model with Test set and generate images of detected log objects along with confidence score. For example, the test outcome of one image will look like similar to the following image. You will need to use OpenCV to produce such image
4. Write a python program that count number of detected logs in each output image (Log counting)

I have combined the answer for requirement 3 and 4 together and producing log counting and detected objects along with confidence score as well as segmentation which has been shown in portfolio requirements PDF as example. Additionally I have added threshold to filtered out bad or inaccurate results detected by the models. Below is an example of the output of the model.



The result for **test outcome** containing **images** can be found at **rcnn_test** folder!

Task 3: Extending Log Labelling to Another Class

The result this tasks you can find at **my_coco_annotations** and the json file required to submit is **dataset.json**