Portfolio Task - Week 5

COS40007 - Artificial Intelligence for Engineering

Student Information:

Name: Arnob Ghosh

Student ID: 103494114

Date: 7th September 2024

Tutorial Class: Monday 4:30-6:30

Ondrive Link: https://liveswinburneeduau-my.sharepoint.com/:f:/g/

personal/103494114_student_swin_edu_au/

ErHIePWVH0dKgAhmXJaN_KABcAOTu6JMKiJINSZgswFBHg?e=fnSE5n

Abstract

This portfolio submission uses CNN, ResNet50, and Mask R-CNN to demonstrate the development and assessment of deep learning models for image classification and object detection tasks. The tagged datasets, model outputs, and well-structured source code are included as follows:

- Labelled Log Dataset: A collection of 10 annotated images created using the LabelMe tool, along with their corresponding JSON files.

- CNN Model: The results from testing a basic CNN model, trained to classify images into "rust" and "no rust." Images and results are stored in the "cnn_test" folder.

- ResNet50 Model: The outcomes of testing the ResNet50 model on the same classification task, with results saved in the "resnet50_test" folder.

- Mask R-CNN Model: This model is designed for detecting log objects in images, producing results with bounding boxes, confidence scores, and segmentation masks. The results are in the "rcnn_test" folder.

- Source Code: All source code for the models is organized in the "code" folder.

Table of Contents

Task 1: Develop CNN and Resnet50

To create the test set, first choose 10 images with rust and 10 images without rust at random. Thus, these 20 photos will not be included in the training set.

```
# 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.
```

```
# 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}
```

```
# 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']
```

Similar to the MNIST classification model, a simple CNN model was created and trained using the corrosion dataset with the labels "rust" and "no rust." After training, the model was assessed using the test set, and its accuracy was determined by how well the 20 images in the test set were classified.

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!

1. 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

```python
# 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 results of the test, including the output images, can be found in the resnet50_test folder.

Task 2: Develop Mask RCNN for Detecting Log

Selecting Test Images:

Rather than choosing 10 images by hand for testing, I decided to use the labelme2coco tool to divide the dataset equally into training and testing portions due to some hardware limitations on my MacBook.

```python
# Set directory
labelme_folder = LOG_LABEL_DATASET_FOLDER
export_dir = COCO_ANNOTATIONS_DIR

# Convert LabelMe annotations to COCO format
train_split_rate = 0.5  # 50% for training
category_id_start = 1  # Start category IDs from 1
labelme2coco.convert(labelme_folder, export_dir, train_split_rate, category_id_start=category_id_start)

print("LabelMe annotations converted to COCO format for log detection.")
```

[28] MagicPython

Training the Mask RCNN Model:

I trained the model for roughly 17 epochs at first, during which time the training loss dropped to about 2.6. Upon retraining the model for twenty more epochs, I was able to reduce the training loss to about 0.6. This represents a noteworthy enhancement compared to the initial loss of 2.6. The training process took about six or seven hours in total.

```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
```

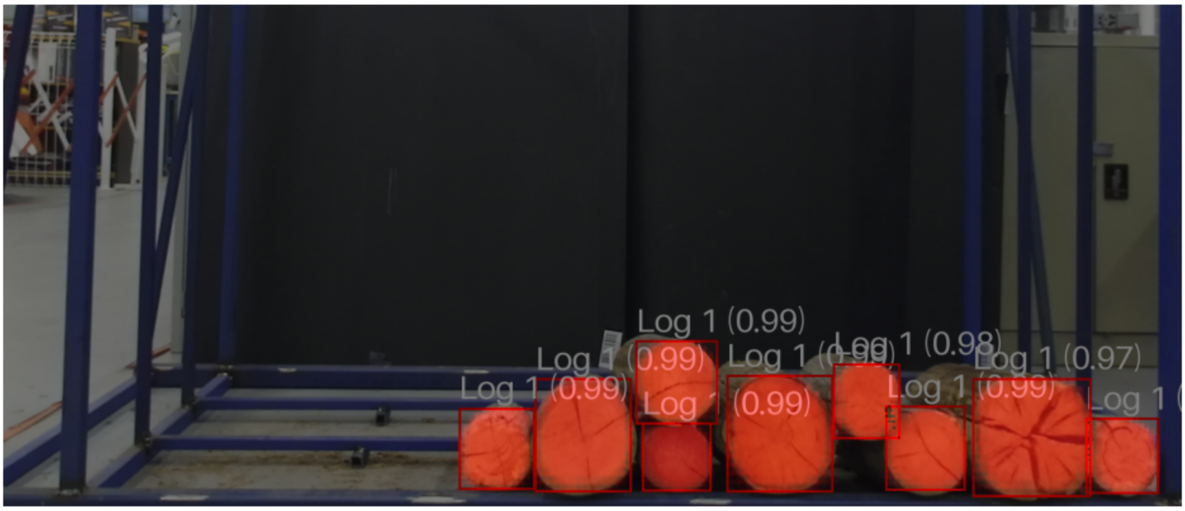
Testing the Model and Generating Output:

I created images with detected log objects and their confidence scores after evaluating the model with the test set. I used OpenCV to generate these images and display the outcomes, which included object segmentation.

Log Detection and Counting:

I wrote a Python script that integrated the log counting and detection features. As specified in the portfolio requirements, the output consists of segmentation masks, confidence scores, and the count of detected logs. I also used a threshold to eliminate detections that were erroneous or of low confidence. The rcnn_test folder contains the test results, along with the processed images.

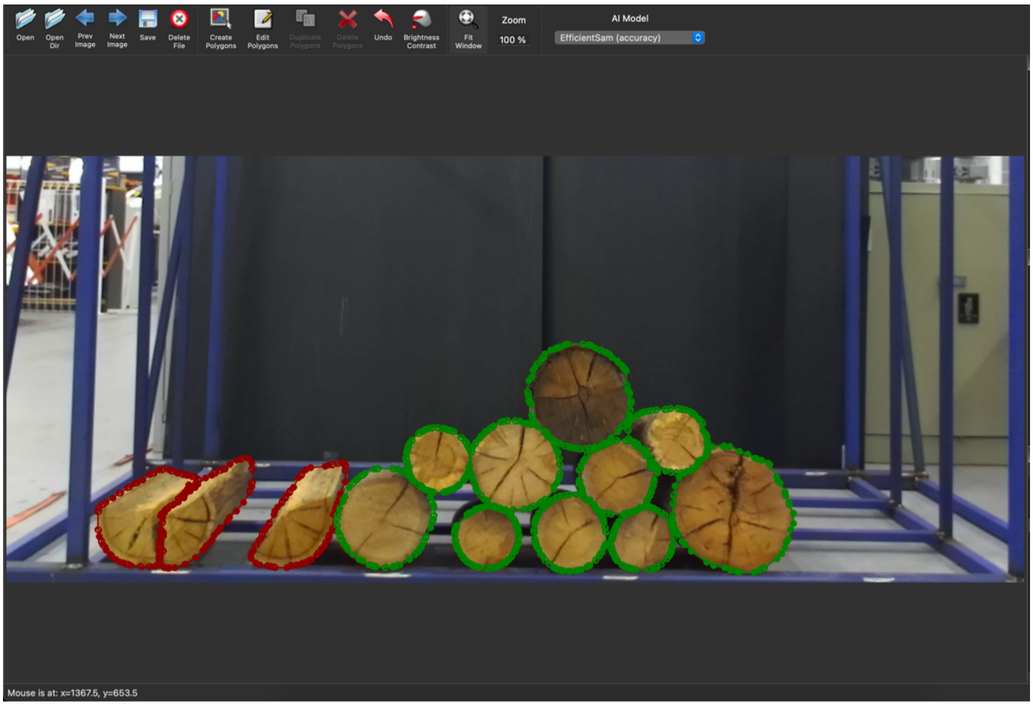
Threshold: 80%
Log count: 9



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

Task 3: Expanding Log Labelling to Include an Additional Class

The task results are located in the **my_coco_annotations** folder, and **dataset.json** is the file that must be submitted.


Open
Open Dir
Prev Image
Next Image
Save
Delete File
Create Polygons
Edit Polygons
Duplicate Polygons
Delete Polygons
Undo
Brightness Contrast
Fit Window
Zoom
100 %
AI Model
EfficientSam (accuracy)
Mouse is at: x=1367.5, y=653.5