SCHOOL OF SCIENCE, COMPUTING AND ENGINEERING TECHNOLOGIES

COS40007

Artificial Intelligence for Engineering

Portfolio Assessment-4: "Deep learning using Tensorflow and Keras"

Studio 5 – Portfolio 5

Students' Name: Duc Thinh Pham - 104169675

Lecturer: Dr. Trung Luu

Date: October 11th, 2024

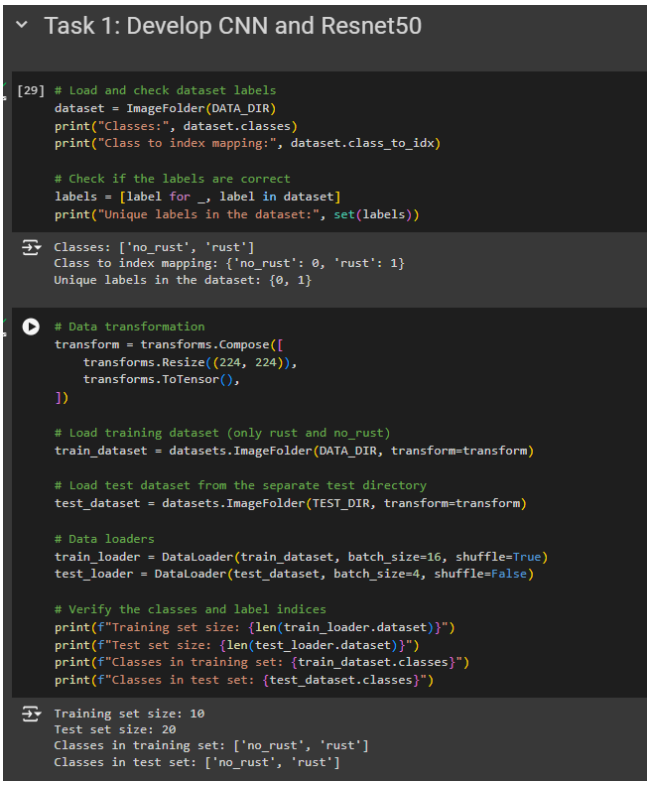
Contents

Abstract

This portfolio demonstrates the application of deep learning techniques for image classification and object detection using TensorFlow and Keras. The project comprises three primary tasks: developing and evaluating a Convolutional Neural Network (CNN) and ResNet50 model for classifying corrosion in structural assets, implementing a Mask R-CNN model for detecting wooden logs in images, and extending the labeling process to incorporate additional classes. The CNN models trained on a labeled corrosion dataset are evaluated for accuracy on a test set, comparing performance between a simple CNN and a more complex ResNet50 architecture. The Mask R-CNN model is trained on a labeled wooden log dataset to detect log objects, with a post-processing step to count detected logs in the test images. This portfolio also includes updated annotations to identify broken logs, highlighting the potential for extending detection capabilities to multi-class object recognition. The results are validated with accuracy metrics and visual outputs, demonstrating the effectiveness of these models in handling classification and detection tasks in engineering contexts.

Task 1: Develop CNN and Resnet50

1. Start by selecting 10 images of rust and 10 images of no rust at random to create a Test Set, ensuring these images are excluded from the training dataset.



Task 1: Develop CNN and Resnet50

```
[29] # 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))
```

```
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}")
```

```
Training set size: 10
Test set size: 20
Classes in training set: ['no_rust', 'rust']
Classes in test set: ['no_rust', 'rust']
```

2. Develop a simple CNN model similar to the MNIST classification architecture. Train this model using the corrosion dataset with two classes: "rust" and "no rust" (excluding the Test Set). After

training and saving the model, evaluate its accuracy by testing it with the 20 images in the Test Set, measuring the correct classifications.

Simple CNN

```
[ ]  # 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.")

SimpleCNN model initialized.
```

```
# Training the simple CNN model
num_epochs = 10
for epoch in range(num_epochs):
    simplecnn.train()
    running_loss = 0.0
    for images, labels in train_loader:
        optimizer.zero_grad()
        outputs = simplecnn(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 trained model
torch.save(simplecnn.state_dict(), 'Portfolio05/models/simple_cnn.pth')

print("Training complete. Model saved as 'Portfolio05/models/simple_cnn.pth'.")

Epoch 1/10, Loss: 0.6930
Epoch 2/10, Loss: 4.1532
Epoch 3/10, Loss: 0.5105
Epoch 4/10, Loss: 2.5327
Epoch 5/10, Loss: 1.6189
Epoch 6/10, Loss: 0.6919
Epoch 7/10, Loss: 0.6065
Epoch 8/10, Loss: 0.7250
Epoch 9/10, Loss: 0.7487
Epoch 10/10, Loss: 0.7008
Training complete. Model saved as 'Portfolio05/models/simple_cnn.pth'.
```

Result table:

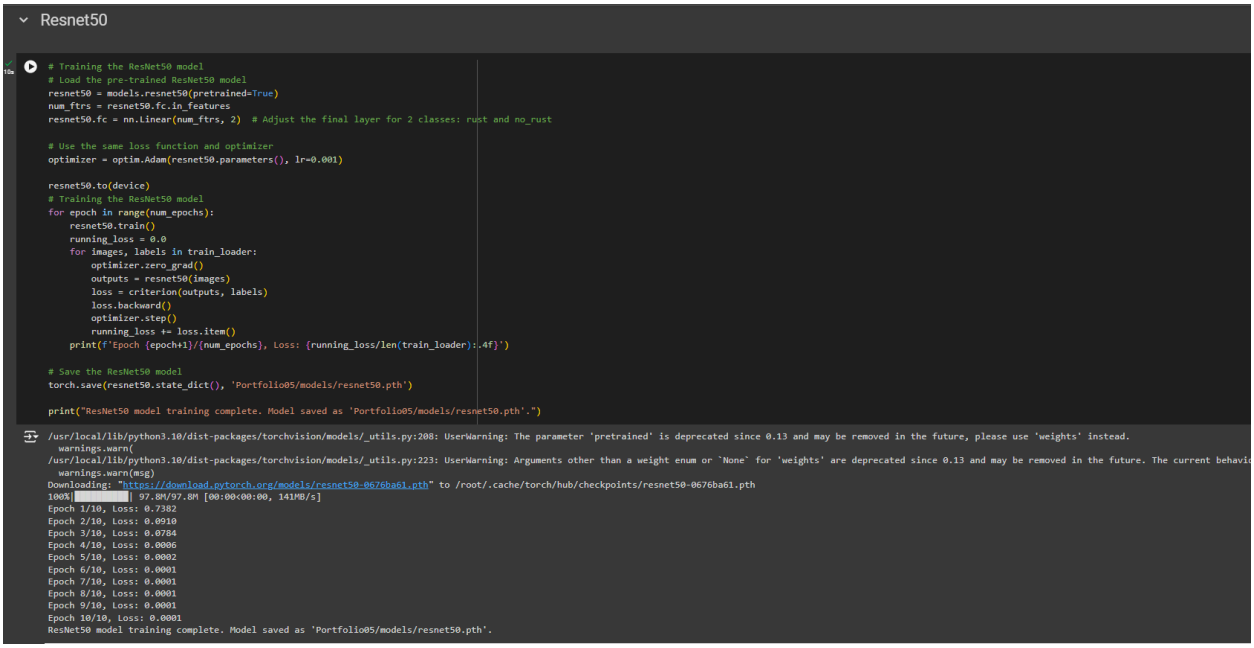
True Class	Predicted Class
0	0
0	0
0	0
0	0
0	0
0	0
0	0

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

Accuracy of the Simple CNN model on the test set: 55.00%

You can find the test results with images in the **cnn_test** folder.

3. Next, implement a more advanced CNN model using the ResNet50 architecture, training it with the same dataset as in step 2. Test this model with the same Test Set to measure its accuracy.



Result table:

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

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

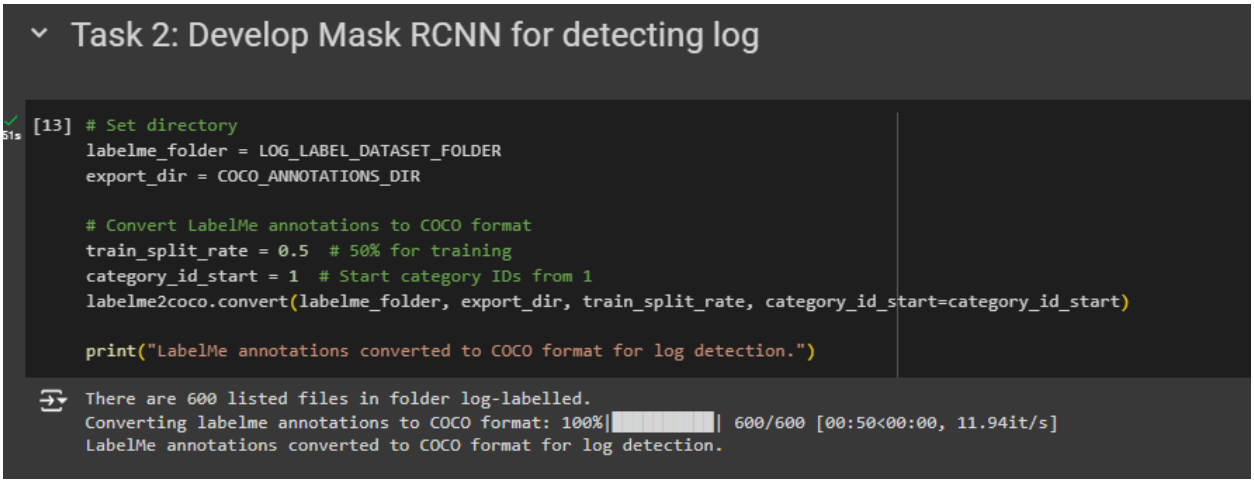
Accuracy of the ResNet50 model on the test set: 50.00%

You can find the test results with images in the **restnet50_test** folder.

Task 2: Develop Mask RCNN for Detecting Log

1. Begin by randomly selecting 10 images to form a Test Set.

Due to hardware limitations on my local machine, I split the training and testing datasets evenly using **labelme2coco** for this task.



2. Create a Mask R-CNN model and train it using the labeled log images, excluding those in the 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 = 'Portfolio05/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 = 10
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()
```

3. Evaluate the model on the Test Set, generating images that display detected log objects along with their confidence scores.
4. Write a python program that count number of detected logs in each output image (Log counting)

For question 3 and 4, generating both log counts and detected objects with confidence scores and segmentation, as illustrated in the portfolio requirements PDF. Furthermore, I included a threshold to filter out inaccurate or low-confidence detections.

Threshold: 80%
Log count: 11

Task 3: Extending Log Labelling to Another Class

I update the labels for those 10 images using labelme, replacing the label of any broken logs with "detected_log"? Then, save the updated labels for all 10 images.

Appendix

My GitHub repository: thinhpham1807/COS40007---Artificial-Intelligence-for-Engineering (github.com)

Data files:
https://drive.google.com/drive/folders/1fyCmJYQNspCvwO_vfwarsQs6x0MKW4cP?usp=sharing