Arthropod Taxonomy Orders Object Detection Dataset Report

Zewei Cai Simon Fraser University

1. Introduction

Species identification from an image is a complex problem. Image classification assumes there is only one species in the image. The ArTaxOr data set covers 2-class data, including Araneae and Coleoptera. In this project, binary classification would be implemented. Two different tools are used in this project, which are Keras and Pytorch.

2. Data splitting

In this project, given data includes per-class images and their corresponding .json files, which usually are used for training. But to test the performance of our models, we have to split data into training, validation and test sets. In fact, different implementation would use different splitting methods. For example, usually train_test_split() function would be used to split data of pandas dataframes, which can split data into 2 sets. And we can call it 2 times to get 3 sets.

For my Keras version, I just manually split data into training and test sets for convenience. Usually training set would be larger than test and validation set. However, it would run out of memory when loading if the training set is too large. Thus, I just evenly split data into 2 sets for each class, which are training and test sets. After training finished, training data would be deleted and test data would be loaded. Because there is an argument named validation_split in model.fit() for Keras, , which helps automatically get validation data from training data and test validation results during training, we don't have to manually get validation data. The input directory structure is shown in 3.1.

For my Pytorch version, I used datasets.ImageFolder to loads data from local machine. So, I manually evenly split data to 3 sets on my own in advance. The structure is shown in 4.1.

3. Classification by Keras (baseline)

3.1 Folder structure

ArTaxOr
----------Araneae
----------test
---------- ***.jpg
----------train
---------- ***.jpg
----------Coleoptera
----------test
---------- ***.jpg
----------train
---------- ***.jpg

3.2 Network Structure

The main structure of network used here is resnet50. In the end of the network, A fully connected layer named Dense would be used to obtain 2 outputs.

```
model = ResNet50(include_top = False, input_shape = (INPUT_HEIGHT, INPUT_WIDTH, 3), pooling = 'avg', classes = 2)

for layer in model.layers:
    layer.trainable = False

output = model.output
output = Dense(2, activation = 'tanh')(output)
model = Model(model.input, output)
model.compile(optimizer = 'adam', loss = "binary_crossentropy", metrics = ["accuracy"])
model.summary()
```

Figure 1

Hyperparameters

```
# Hyperparameters
EPOCH = 5
BATCH_SIZE = 64
INPUT_HEIGHT = 500
INPUT_WIDTH = 500
```

Figure 2

3.3 Results

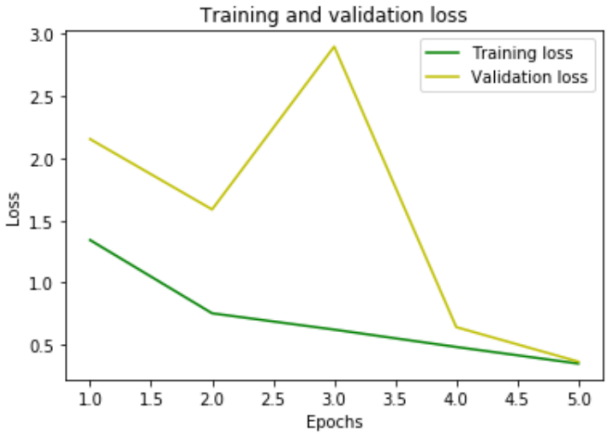

Figure 3

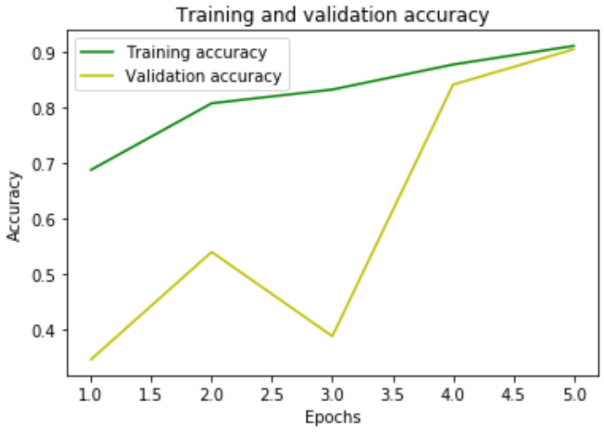

Figure 4

	Train time	Train loss	Train accuracy	Val loss	Val accuracy
Epoch 1	655s	1.3428	0.6878	2.1545	0.3458

Epoch 2	640s	0.7538	0.8078	1.5887	0.5396
Epoch 3	639s	0.6226	0.8326	2.8982	0.3877
Epoch 4	636s	0.4831	0.8778	0.6427	0.8414
Epoch 5	868s	0.3495	0.9113	0.3652	0.9053

Test results are as follow:

```
71/71 [==============================] - 634s 9s/step - loss: 0.4030 - accuracy: 0.8977

---------Test Data Evaluation---------
loss: 0.403005
accuracy: 0.897697
```

Figure 5

3.4 Visualization

```
Image path: ArTaxOr/Coleoptera/test/80aef02ad3af.jpg
Araneae Probability: -0.332911
Coleoptera Probability: 0.975294
Predicted Class: Coleoptera
```

Figure 6

```
Image path: ArTaxOr/Coleoptera/test/01a8549b844b.jpg
Araneae Probability: -0.705095
Coleoptera Probability: 0.993527
Predicted Class: Coleoptera
```

Figure 7

3.5 Improvements

I did not do further work on Keras version. Thus, there are some problems here. The number of epochs I set is 5, which is not sufficient. And also, learning rate should be initialized properly and be decayed with training going. Otherwise, we can not get a model with good performance. But for baseline, with 0.897 test accuracy, this model looks good enough.

4. Classification by Pytorch

4.1 Folder structure

data
-------test
---------- Araneae
--------------------------- ***.jpg
---------- Coleoptera
--------------------------- ***.jpg
-------train
---------- Araneae
--------------------------- ***.jpg
---------- Coleoptera
--------------------------- ***.jpg
-------validation
---------- Araneae
--------------------------- ***.jpg
---------- Coleoptera
--------------------------- ***.jpg

4.2 Network structure

Similarly, the main structure of this network is resnet18, which is pre-trained. And also we can set whether this pre-trained resnet model should be trainable. In the end of the network, a fully connected layer would be added and output 2 results.

```
class PreTrainedResNet(nn.Module):
    def __init__(self, num_classes, feature_extracting):
        super(PreTrainedResNet, self).__init__()

        # Load pre-trained ResNet Model
        self.resnet18 = models.resnet18(pretrained=True)

        # Set gradients to false
        if feature_extracting:
            for param in self.resnet18.parameters():
                param.requires_grad = False

        # Replace last fc layer
        num_feats = self.resnet18.fc.in_features

        # Replace fc layer in resnet to a linear layer of size (num_feats, num_classes)
        self.resnet18.fc = nn.Sequential(
            nn.Linear(num_feats, 512),
            nn.BatchNorm1d(512),
            nn.ReLU(True),
            nn.Linear(512, num_classes)
        )

    def forward(self, x):
        # Forward pass x through the model
        x = self.resnet18(x)
        return x
```

Figure 8

Hyperparameters are as follow:

```
# Hyperparams
NUM_EPOCHS = 20
LEARNING_RATE = 0.001
BATCH_SIZE = 32
RESNET_LAST_ONLY = True # Set to True to tune only the last layer. Set to False to tune entire network
AUGUMENTATION = False # Choose whether using data augumentation
```

Figure 9

Besides, I set a scheduler for learning rate. The initial learning rate is 0.001 and it would be decayed with each 10 epochs.

4.3 Results

4.3.1 With data augmentation

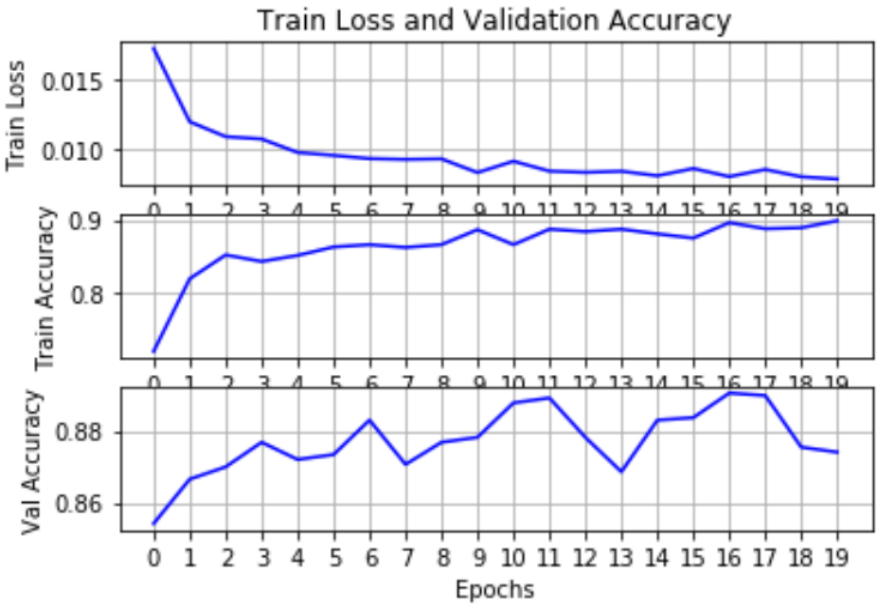

Figure 10

After training 20 epochs, I obtained evaluation as follows. Precision, recall and f1 score output per-class results.

```
Epoch 20/20 : Train Loss 0.0079; Train Accuracy 0.8989
100%|██████████| 20/20 [55:22<00:00, 166.15s/it]
Precision:  tensor([0.8662, 0.8835])
Recall:  tensor([0.9000, 0.8452])
F1 Score:  tensor([0.8662, 0.8835])
Validation Accuracy: 0.8741
Accuracy of Araneae : 0.8662
Accuracy of Coleoptera : 0.8835
```

Figure 11

Test results are as follow:

```
test(model, criterion)
```

```
Test Loss: 0.0077 Test Accuracy 0.8896
```

Figure 12

4.3.2 Without data augmentation

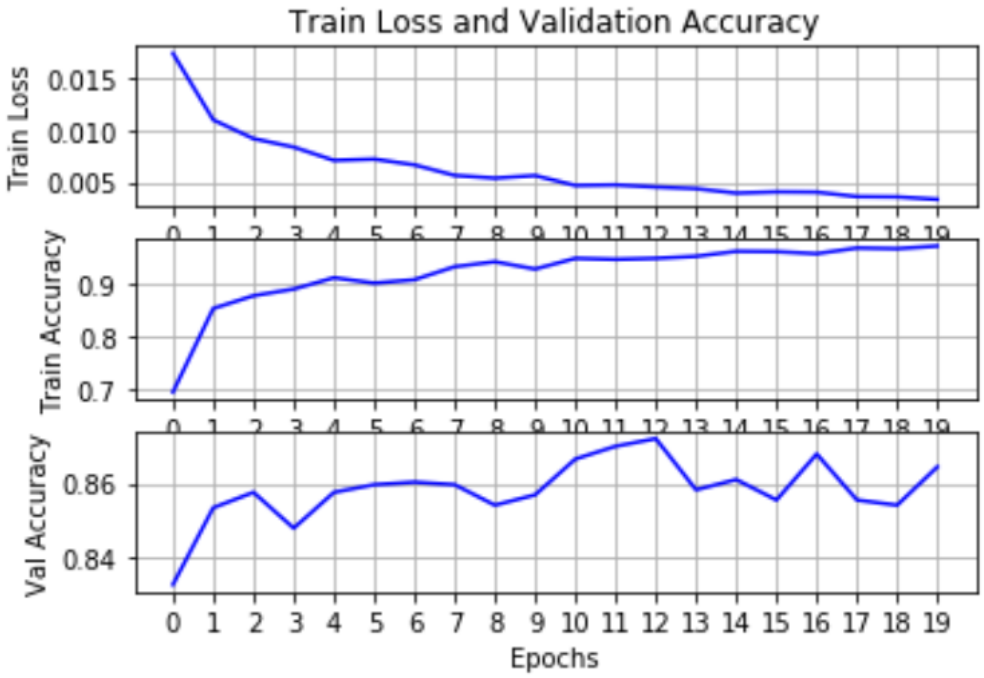

Figure 13

After training 20 epochs, I obtained evaluation as follows. Precision, recall and f1 score output per-class results.

```
Epoch 20/20 : Train Loss 0.0033; Train Accuracy 0.9738
100%|██████████| 20/20 [1:08:02<00:00, 204.11s/it]
Precision:  tensor([0.8675, 0.8608])
Recall:  tensor([0.8830, 0.8430])
F1 Score:  tensor([0.8675, 0.8608])
Validation Accuracy: 0.8645
Accuracy of Araneae : 0.8675
Accuracy of Coleoptera : 0.8608
```

Figure 14

Test results are as follow:

```
test(model, criterion)
```

```
Test Loss: 0.0097 Test Accuracy 0.8710
```

Figure 15

4.3.3 Conclusion

As we can see, data after augmentation would get little better results. Data augmentation would increase the amount of training data and increase noise data to improve the robustness of the model.

4.4 Visualization

After resizing image data, the samples would look smaller.

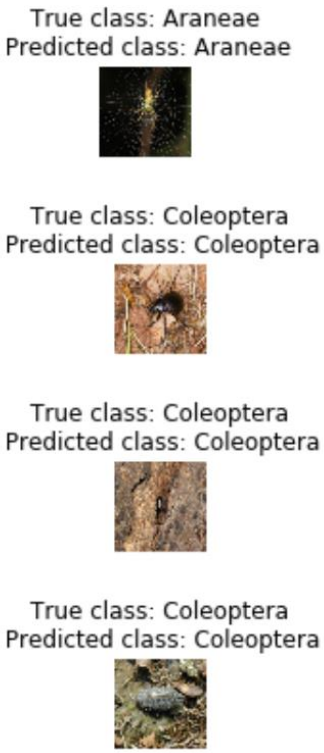

Figure 16

5. Future work

In this project, we only made use of image data and implemented binary classification. In fact, the original datasets include 7-class objects and also its .josn files. The limitation of image classification is that it assumes there is only one species in the images. In modern technology, object detection would be used for further classification, which would identify objects and also predict the labels. Some useful tools include Detectron2 and mmDetection. In the future, object detection could be implemented.

6. Reference

[1]https://towardsdatascience.com/machine-learning-recurrent-neural-networks-and-long-short-term-memory-lstm-python-keras-example-86001ceaaebc
[2] https://www.jianshu.com/p/f4952822e0e5