Honey Bee Image Classification

DATS6203 Machine Learning 2, Individual Report

Yijia Chen

Introduction

This project goal is to use deep learning network to classify Bees Subspecies and Hive Health. The data is from Kaggle: https://www.kaggle.com/jenny18/honey-bee-annotated-images/kernels

In this project, we first explore the dataset and operate data preprocessing. Then we use 2 different neural network frameworks to classify Bee Subspecies and Hive Health. In this project, we use Gradient Descent Algorithm to optimize the network. In the end we evaluate the result by f1 score, AUC and ROC and test accuracy.

Description of Individual Work

The project is mainly separated in 4 parts. What I did is mainly the subspecies classification using Pytorch.

Portion

At the beginning, I get the data directory and write a function to show image.

In the modelling part, I have wrote a function to bring data into Pytorch. I have searched online to see how exactly to write a function for Pytorch Dataloader.

```
#define dataset

class honeybee(Dataset):
    def __init__(self, data, transform = None):
        self.data = data
        self.img_dir = '../input/bee_imgs'
        self.transform = transforms.Compose([
                         transforms.ToTensor(),
                         transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                 std=[0.229, 0.224, 0.225])
                         ])

    def __getitem__(self, index):
        img = os.path.join(self.img_dir, self.data.iloc[index,0])
        image = Image.open(img)
        image = image.resize((120,120))
        image = image.convert('RGB')
        image = self.transform(image)
        label = np.asarray(self.data.iloc[index, 5])  # type: object

        return image,label

    def __len__(self):
        return len(self.data)


# split train test
train, test = train_test_split(data, test_size=0.3)
train_data = honeybee(train)
test_data = honeybee(test)


epochs = 30
batch_size = 64
learning_rate = 0.001

#Data Loader
train_loader = torch.utils.data.DataLoader(dataset=train_data, batch_size=batch_size,shuffle=True)
test_loader = torch.utils.data.DataLoader(dataset=test_data, batch_size=batch_size, shuffle=False)
```

Then I create the CNN model and tried different layers and optimizers to get the best result.

```python
# CNN Model

class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()

        self.layer1 = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, stride= 1, padding=2),  # RGB image channel = 3, output channel = num_filter
            nn.ReLU(),
            nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=2),
            nn.MaxPool2d(kernel_size=2, stride=2),
            nn.Dropout(p=0.5)) #output size (16,61,61)

        self.layer2 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size=3, stride= 1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2, stride = 2),  #output size (32,32,32)
            nn.Conv2d(64, 64, kernel_size=3, stride=1, padding=2),
            nn.ReLU(),
            nn.Dropout(p=0.5)
            ) #output size (64,34,34)

        self.layer3 = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size=3, stride=1, padding=2),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size = 2, stride = 2),
            nn.Dropout(p=0.5))
            #output size(128,18,18)

        self.layer4 = nn.Sequential(
            nn.Linear(128*18*18, 256),
            nn.Linear(256, 128),
            nn.Linear(128, 7)
            )

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        out = out.view(out.size(0), -1)
        out = self.layer4(out)
        return out
```

To evaluate the process, I plot training loss function by creating a list in the training process.

Also using sklearn classification report package to get f1 score.

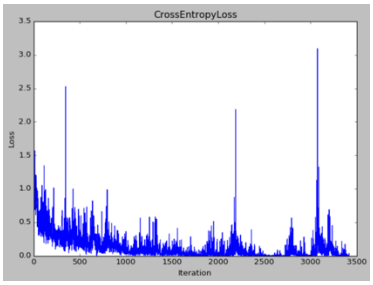
Results

Since we are classifying the image data, we decided to use CNN as our model.
The most important part of the CNN model in Pytorch is calculating the output size each time after we ran the convolution neural network. Here is the formula to calculate output size:

output_size = (in_size - kernel_size + 2*(padding) / stride) + 1

On each layer of the CNN model we show below, we calculated the output size and put in comment.
This is the experimental result without adding dropout node:

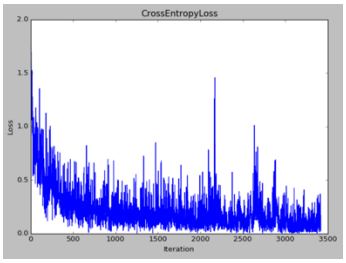
	precision	recall	f1-score	support
1 Mixed local stock 2	0.76	0.62	0.68	135
Western honey bee	0.89	0.80	0.84	10
Carniolan honey bee	0.99	1.00	0.99	134
VSH Italian honey bee	0.98	0.78	0.87	58
Italian honey bee	0.91	0.96	0.94	914
Russian honey bee	0.98	0.99	0.98	165
-1	0.91	0.82	0.86	136
avg / total	0.91	0.92	0.91	1552



This is the experimental result with dropout node:

Classfication report:

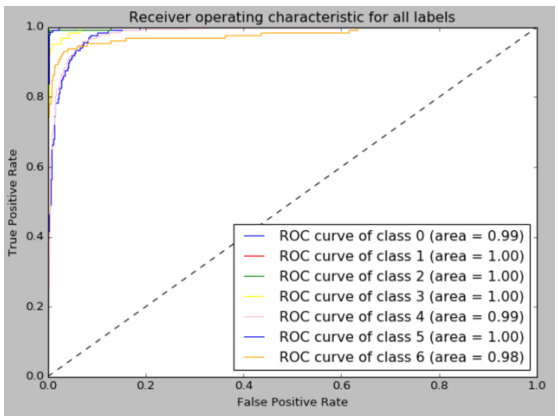
	precision	recall	f1-score	support
1 Mixed local stock 2	0.63	0.92	0.74	138
Western honey bee	0.75	1.00	0.86	6
Carniolan honey bee	0.93	0.99	0.96	142
VSH Italian honey bee	0.86	0.95	0.90	64
Italian honey bee	0.97	0.89	0.93	914
Russian honey bee	0.89	0.99	0.94	155
-1	0.97	0.76	0.85	133
avg / total	0.92	0.91	0.91	1552



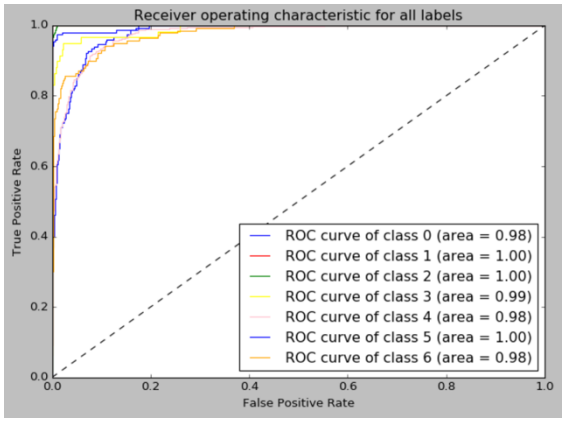
Surprisingly, the average f1-score remains the same, but adding dropout node made the f1 score of all classes over 0.7. So adding dropout node is a good choice for our model.

With dropout node, we tried using batch size 32 and batch size 64 to see if batch size would affect the model.

	Batch Size = 32	Batch Size = 64
Computational Time	423.86s	411.21
Test Accuracy	91%	82%
F1 Score	0.91	0.88



With batch size 32



With batch size 64

Summary and Conclusion

While developing the model, I had faced some difficulties to load the data into pytorch at the beginning. I have checked various reference (see below citations) and get a function. However, there is a bug inside the function so that it drives the whole model into a random mode. Accuracy was only around 50%, the model was not learning at all. After debugging for hours, I finally realize there is a 'self' missing in the class. In the future, the improvement would be care about every row and not rushing to run the model but ensure there is no mistake when loading the data.

After multiple experiments, we see the best model is with dropout node and batch size 32.

Percentage of the copied code = 48/246 * 100 = 19.5%.

Reference

Jha, Shekhar. “VGG16-Honey Bee Health Classification.” *RSNA Pneumonia Detection Challenge | Kaggle*, 2018, www.kaggle.com/xanthate/vgg16-honey-bee-health-classification.

Gage, Justin. “Convolutional Neural Nets in Pytorch.” *Algorithmia Blog*, Algorithmia, 10 Apr. 2018, blog.algorithmia.com/convolutional-neural-nets-in-pytorch/.

“Torch.nn.” *PyTorch*, pytorch.org/docs/stable/nn.html.

“Data Loading and Processing Tutorial.” *PyTorch*, pytorch.org/tutorials/beginner/data_loading_tutorial.html.

_Neo__. “Pytorch-学习记录 卷积操作--Tensor.size().” *为什么版本控制如此重要？ - CSDN 博客*, 28 Nov. 2017, blog.csdn.net/a132582/article/details/78658155.

Leigh. “PyTorch Tutorial: Dataset. Data Preparetion Stage.” *RSNA Pneumonia Detection Challenge | Kaggle*, www.kaggle.com/leighplt/pytorch-tutorial-dataset-data-preparetion-stage.

Santos, Leonardo Araujo dos. “DataLoader and DataSets.” *Deep Q Learning · Artificial Inteligence*, leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/pytorch/dataloader-and-datasets.html.

“Sklearn.metrics.f1_score.” *1.4. Support Vector Machines - Scikit-Learn 0.19.2 Documentation*, scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html.