Final Project Report: Bird Species Classification

Introduction to Machine Learning, NYCU

110550107 傅莉妮

Before report: How to inference the predictions

Link to download model weights: https://drive.google.com/file/d/1urRBLI58yyXMdwv9-QHFXiRzvzcXm0u4/view?usp=sharing

You can save the model into 110550107_weight.pth or use other names.

In my code, I assume the directory structure is like this:

```
/submission
    /training
        main.py
        dataset.py
        model.py
    110550107_inference.py
    110550107_weight.pth ← empty, please download with link
    /data
        /train
            /001.Black_footed_Albatross …
        /test_head ← added outer folder for preprocessing
            /test ← only one class: “test”
                test_img1.jpg …
```

To inference, you should use the following prompts to call the inference file:

```
python 110550107_inference.py --mp <path-to-downloaded-weight
```

e.g.: `python 110550107_inference.py —-mp 110550107_weight.pth`

Environment Details

python=3.11, using miniconda env with cuda=12.1

GPU driver version: 537.13 (run on my PC)

framework: pytorch 2.1.2

installation prompt for pytorch-cuda:

```
conda install pytorch torchvision torchaudio
pytorch-cuda=12.1 -c pytorch -c nvidia
```

Other packages' version details are in the requirements.txt file.

Implementation Details

Dataset

I set 10% of the training dataset as valid dataset with random seed=42. I also added a random mask for the images.

The transform strategy is like this:

```
class RandomMask:
    def __init__(self, min_mask_size, max_mask_size):
        self.min_mask_size = min_mask_size
        self.max_mask_size = max_mask_size

    def __call__(self, img):
        if not isinstance(img, Image.Image):
            raise TypeError('Input must be a PIL image')

        draw = ImageDraw.Draw(img)
        width, height = img.size
        mask_width = random.randint(self.min_mask_size,
                                    self.max_mask_size)
        mask_height = random.randint(self.min_mask_size,
                                    self.max_mask_size)
        x1 = random.randint(0, width - mask_width)
        y1 = random.randint(0, height - mask_height)
        x2 = x1 + mask_width
        y2 = y1 + mask_height
        draw.rectangle([x1, y1, x2, y2], fill=0)
        return img

transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.RandomHorizontalFlip(),
    transforms.RandomRotation(180),
    RandomMask(min_mask_size=10, max_mask_size=50),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])

valid_transform = transforms.Compose([
    transforms.Resize((224,224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406],
                         std=[0.229, 0.224, 0.225])
])
```

`transform` is for training dataset, and `valid_transform` is for validation dataset and testing dataset.

Model

I used **Resnet50** pretrained model (pretrained weight: ResNet50_Weights.DEFAULT) to complete this task.

I set the pretrained weights trainable and replaced the final layer of the model with the following settings:

```
self.resnet.fc = nn.Sequential(
    nn.Linear(in_features, 1024),
    nn.ReLU(),
    nn.BatchNorm1d(1024),
    nn.Dropout(0.3),
    nn.Linear(1024, 512),
    nn.ReLU(),
    nn.BatchNorm1d(512),
    nn.Dropout(0.3),
    nn.Linear(512, num_classes)
)
```

other parameters and strategies:

- learning rate=0.0003
- lr scheduler=ExponentialLR(gamma=0.95)
- optimizer=Adam
- epoch=10

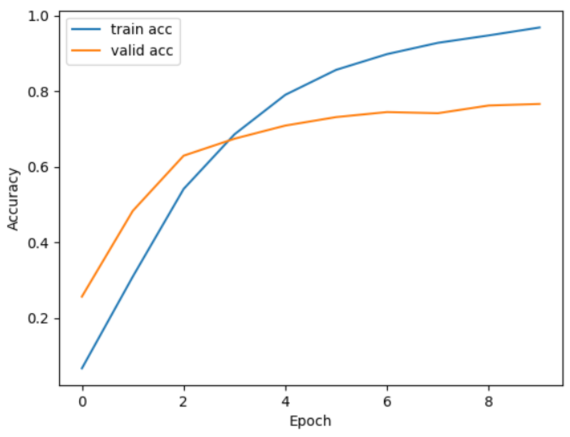
Experimental Results

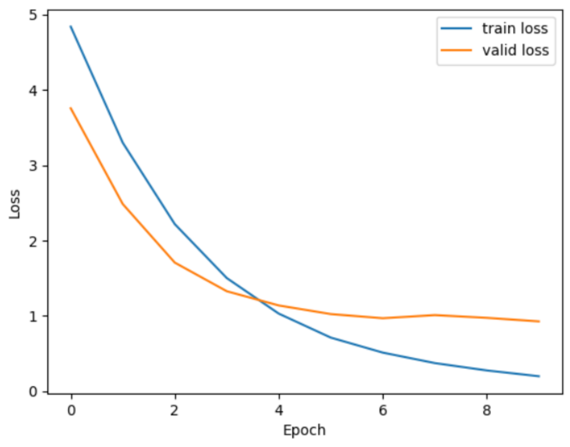
Evaluation Metrics

I calculate the number of correct predictions on training dataset and validation dataset as my evaluation metrics.

Accuracy & Loss Training Curve

It can reach at most 78% of accuracy on my validation dataset, and the validation loss is roughly 1.

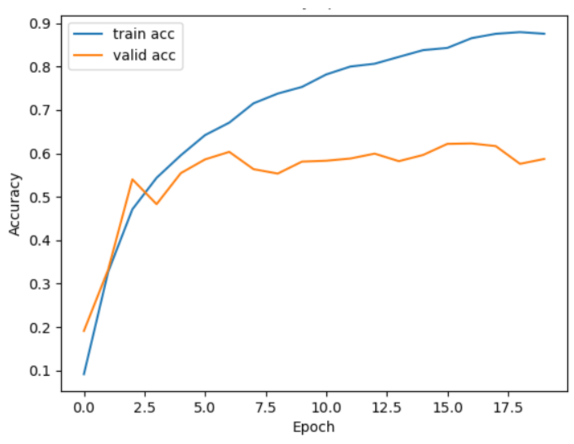


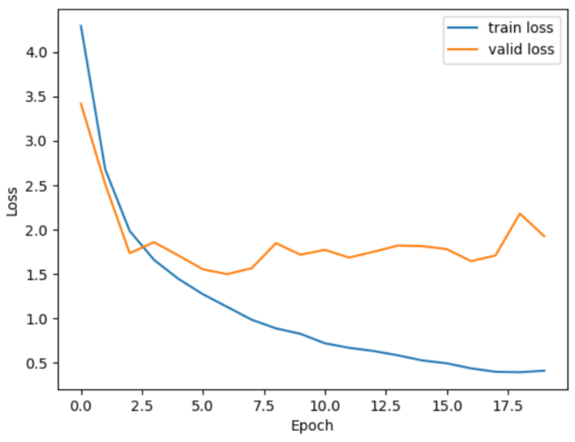


Ablation Study

I made the model to freeze the pretrained weights at first, but this way it makes the model harder to get a better performance. The graphs shown below are the results that the pretrained weights are frozen. The highest accuracy it can get on my validation dataset is only 60%, way lower than it can achieve when all the weights are being trained.

(The difference of epoch numbers is because I thought adding more epochs can make it reach a better performance, however it didn't.)





Bonus: Introduction to Resnet50

ResNet, also named as Residual Networks, is a type of deep neural network architecture. Resnet50 has 50 layers in its structure. In this part, I am going to briefly introduce the main concepts for Resnet50, the structure of blocks and variations of different resnet models.

We already know that the "deepness" for a neural network is crucial. However, there are plenty of problems in deeper models, such as gradient vanishing(as the depth increases, the gradients can become very small, making it difficult for the network to learn effectively) or the degradation problem (adding more layers doesn't improve its accuracy). ResNet addresses the degradation problem through the use of residual blocks and skip connections. Here are the introduction to the basic structure of resnet model:

Residual Blocks:

- Basic Block

  A basic residual block consists of two 3x3 conv layers, with a shortcut connection that adds the input to the output of the conv layers. Batch normalization and ReLU are also applied after each conv layers.

- Bottleneck Block

A bottleneck block consists of 1x1, 3x3, and 1x1 three conv layers. This architecture helps reducing the number of parameters and therefore reduce the complexity.

Resnet is composed of a stack of residual blocks. Shortcut (skip) connections can enable the gradient to flow easily, and solve the degradation problem and gradient vanishing problems. The final layer is a global average pooling layer, which can represent a fixed-size feature map for further classification.

There are plenty of variations of ResNet models, including Resnet18, Resnet101, which are different in the number of layers. There are also some models such as pre-activation resnet, which put the batch normalization and ReLU before each convolution.