Report of Q2
Weather Recognition

50015627 JIANG Zhuoyang

1. **Question Description:**

We need to train a five class classification model, with an input of an image and an output of a five dimensional category one hot vector. Therefore, we need to complete the following key steps:

- Building a dataloader
- Select methods and build models
- Model Training
- Model Evaluation as Required (On the whole Train set)

2. Building a dataloader

I designed a dataloader to load a dataset for weather recognition tasks. It first converts images into a format acceptable to neural networks, while assigning corresponding category labels to each image. And set the batch size to provide data in batches during training, while shuffling the data before each epoch to increase the randomness of model training.

```python
data_dir = 'Data_Q2/train_data'

# Transform Define
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
])

# DataLoader
class WeatherDataset(torch.utils.data.Dataset):
    def __init__(self, data_dir, transform=None):
        self.data_dir = data_dir
        self.transform = transform
        self.file_list = os.listdir(data_dir)

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

    def __getitem__(self, idx):
        img_name = self.file_list[idx]
        img_path = os.path.join(self.data_dir, img_name)
        image = Image.open(img_path).convert('RGB')

        if self.transform:
            image = self.transform(image)

        # 解析类别信息
        label_str = img_name.split('.')[0]  # 去除文件扩展名
        label = 0  # 默认类别索引
        if 'Sunny' in label_str:
            label = 0
        elif 'Snowy' in label_str:
            label = 1
        elif 'Cloudy' in label_str:
            label = 2
        elif 'Rainy' in label_str:
            label = 3
        elif 'Foggy' in label_str:
            label = 4

        return image, label

dataset = WeatherDataset(data_dir, transform=transform)
dataloader = torch.utils.data.DataLoader(dataset, batch_size=32, shuffle=True)
```

Figure.1 Dataloader

(1) Data loading and conversion

Data path: using data_ Dir='Data_ Q2/train_ Data 'specifies the data storage path.

Image conversion: using transforms Compose() defines the image conversion process, which includes uniformly adjusting the image to a size of (224, 224) and then converting it to tensor format.

(2) Dataset and label definitions:

Custom Dataset: The WeatherDataset class is used to define the dataset. The __init__ method handles initialization, __len__ method returns the dataset length, and __getitem__ method reads each sample.

Parsing Category Information: In the __getitem__ method, category labels for each image are determined by parsing keywords from the file names. For instance, "Sunny" corresponds to 0, "Snowy" corresponds to 1, "Cloudy" corresponds to 2, "Rainy" corresponds to 3, and "Foggy" corresponds to 4.

(3) Dataloader Configuration:

Batching and Shuffling: A dataloader object is created using torch.utils.data.DataLoader, where the defined WeatherDataset is passed in. It's configured with a batch size of 32, and shuffle=True ensures the data order is randomized before the start of each epoch.

(2) Visualization

```python
# Obtain data from a batch for visualization
dataset = WeatherDataset(data_dir, transform=transform)
sample_dataloader = torch.utils.data.DataLoader(dataset, batch_size=5, shuffle=True)
images, labels = next(iter(sample_dataloader))
images = images.numpy().transpose((0, 2, 3, 1))

# Define category names
class_names = ['Sunny', 'Snowy', 'Cloudy', 'Rainy', 'Foggy']

# Visualization
plt.figure(figsize=(15, 8))
for i in range(5):
    plt.subplot(1, 5, i+1)
    plt.imshow(images[i])
    plt.title(class_names[labels[i]])
    plt.axis('off')

plt.show()
```

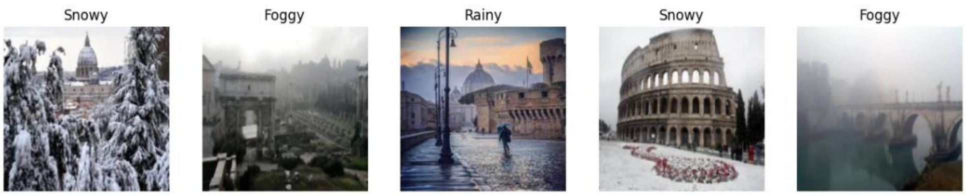

Figure.2 Visualization

3. Select methods and build models

I choose to use resnet18 as the backbone network for automatically extracting features from images. The last fully connected layer is replaced with a new linear layer (nn.Linear(num_ftrs, 5)) with an output size of 5, corresponding to the five weather categories.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		[3×3, 64; 3×3, 64] ×2	[3×3, 64; 3×3, 64] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3	[1×1, 64; 3×3, 64; 1×1, 256] ×3
conv3_x	28×28	[3×3, 128; 3×3, 128] ×2	[3×3, 128; 3×3, 128] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×4	[1×1, 128; 3×3, 128; 1×1, 512] ×8
conv4_x	14×14	[3×3, 256; 3×3, 256] ×2	[3×3, 256; 3×3, 256] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×6	[1×1, 256; 3×3, 256; 1×1, 1024] ×23	[1×1, 256; 3×3, 256; 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512; 3×3, 512] ×2	[3×3, 512; 3×3, 512] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3	[1×1, 512; 3×3, 512; 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figure.3 Backbone - Resnet18

4. Model Training

Then I conducted training for 50 epochs using a stochastic gradient descent optimizer with a learning rate of 0.001 and momentum of 0.9, employing cross-entropy loss, iterating through the dataset batches, and updating the model parameters iteratively while monitoring and printing the average loss per epoch.

The Training Process shows as below:

```
In [7]:
# Defining loss functions and optimizers
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

# Hyperparamer
num_epochs = 50 # batch_size = 32

for epoch in range(num_epochs):
    running_loss = 0.0
    for i, data in enumerate(tqdm(dataloader), 0):  # 在dataloader上添加进度条
        inputs, labels = data
        optimizer.zero_grad()

        outputs = model(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        running_loss += loss.item()

    print(f"Epoch {epoch+1}/{num_epochs} Loss: {running_loss/len(dataloader)}")

100%|██████████| 8/8 [00:20<00:00,  2.55s/it]
Epoch 47/50 Loss: 0.008789917163085192
100%|██████████| 8/8 [00:20<00:00,  2.56s/it]
Epoch 48/50 Loss: 0.0089359600096941
100%|██████████| 8/8 [00:21<00:00,  2.68s/it]
Epoch 49/50 Loss: 0.013098844501655549
100%|██████████| 8/8 [00:20<00:00,  2.54s/it]
Epoch 50/50 Loss: 0.007046281942166388
```

Figure.4 Training Process

5. Model Evaluation as Required (On the whole Train set)

After training the model, I tested the model on ALL the training data and report the accuracy. The Accuracy on the training set shows as below:

```
In [8]:
# Model Evaluation On the whole Train set
correct = 0
total = 0
with torch.no_grad():
    for data in tqdm(dataloader):
        images, labels = data
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = correct / total
print(f'Accuracy on training data: {accuracy}')

100%|██████████| 8/8 [00:09<00:00,  1.14s/it]
Accuracy on training data: 1.0
```

Figure.4 Accuracy on the training set