Q2. Weather Recognition

a. The design of dataloader

standardized input data: resize images to a consistent size and do normalization
defined labels: from image_name
whether shuffling was applied=True
chosen batch size = 16

b. Code screenshots related to (a)

```
batch_size = 16
train_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_size, shuffle=True)
```

```
class WeatherDataset(Dataset):
    def __init__(self, data_dir):
        self.data_dir = data_dir
        self.image_files = os.listdir(data_dir)
        self.transform = transforms.Compose([
            transforms.ToTensor(),
            transforms.Resize((224, 224)),  # Resize images to a consistent size
            transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))  # Normalize image data
        ])

        self.class_names = ['Sunny', 'Snowy', 'Cloudy', 'Rainy', 'Foggy']

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

    def __getitem__(self, idx):
        image_name = self.image_files[idx]
        image_path = os.path.join(self.data_dir, image_name)
        image = Image.open(image_path).convert('RGB')
        image = self.transform(image)

        # Extract label from image name
        label = re.match(r'^[A-Za-z]+', image_name).group(0)

        # Convert label to numerical value
        label_mapping = {
            'Sunny': 0,
            'Snowy': 1,
            'Cloudy': 2,
            'Rainy': 3,
            'Foggy': 4
        }
        label = label_mapping[label]

        # Convert label to one-hot encoding
        label_onehot = torch.nn.functional.one_hot(torch.tensor(label), num_classes=5)

        return image, label_onehot
```

c. A brief introduction to your model along with relevant code screenshots.

The code defines a neural network model named WeatherModel, which is built on the resnet architecture. The model includes ReLU activation function, Dropout layer, fully connected layer, and Softmax output layer.

The forward method of the model defines the forward propagation process of the data in the model. The input data passes through the resnet network and then goes through the ReLU activation function, Dropout layer, and fully connected layer for processing. Finally, it generates classification predictions through the Softmax output layer.

The model utilizes the cross-entropy loss function (nn.CrossEntropyLoss()) and the stochastic gradient descent optimizer (optim.SGD) for training.

```python
import torch
from torch import nn
import torchvision.models as models
# Load ResNet-50 with pretrained weights
resnet = models.resnet50(pretrained=True)
class WeatherModel(nn.Module):
        def __init__(self, net):
                super(WeatherModel, self).__init__()
                # resnet50
                self.net = resnet
                self.relu = nn.ReLU()
                self.dropout = nn.Dropout(0.1)
                self.fc = nn.Linear(1000, 8)
                self.output = nn.Softmax(dim=1)

        def forward(self, x):
                x = self.net(x)
                x = self.relu(x)
                x = self.dropout(x)
                x = self.fc(x)
                x = self.output(x)
                return x
model = WeatherModel(resnet)
```

d. Screenshots depicting the training process.

```python
# Set the device to GPU if available, otherwise use CPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# Set up the training loop
def train(model, train_loader, criterion, optimizer, num_epochs):
    model.to(device)
    model.train()  # Set the model to training mode

    for epoch in range(num_epochs):
        running_loss = 0.0

        for images, labels in train_loader:
            images = images.to(device)
            labels = labels.to(device)

            optimizer.zero_grad()

            # Forward pass
            outputs = model(images)
            loss = criterion(outputs, labels)

            # Backward pass and optimization
            loss.backward()
            optimizer.step()

            running_loss += loss.item()

        epoch_loss = running_loss / len(train_loader)
        print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {epoch_loss:.4f}")


# Set up the model, loss function, and optimizer

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=0.001, momentum=0.9)

# Train the model
num_epochs = 50
train(model, train_loader, criterion, optimizer, num_epochs)
```

e. Accuracy on the training set (with screenshots).

```python
def evaluate(model, test_loader):
    model.eval()
    correct = 0
    total = 0
    with torch.no_grad():
        for images, labels in test_loader:
            images = images.to(device)
            labels = labels.to(device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
    accuracy = correct / total * 100
    print(f"Accuracy: {accuracy:.2f}%")


```

```python
evaluate(model, train_loader)
```

```
Accuracy: 100.00%
```