Q2. Weather Recognition

1. The design of dataloader

First we get the path of all JPG files in the folder, save them in a list which is not only make we easily load all the images but also can be seen as the index for the following operations.

```
# Get the path of all JPG files in the folder
folder_path = '../Q2_data/train_data'
file_list = os.listdir(folder_path)
jpg_files = [file for file in file_list if file.endswith('.jpg')]
print(len(jpg_files))
Executed at 2023.12.15 00:10:11 in 17ms

250
```

And then, we define a function to resize image and convert to tensor which is aimed to be trained. Using this function we get a tensor of 250*3*224*224.

```
# Define a global transformer to appropriately scale images and subsequently convert them to a
 Tensor
img_size = 224
loader = transforms.Compose([
  transforms.Resize(img_size),
  transforms.CenterCrop(img_size),
  transforms.ToTensor(),
])
def load_image(filename):
    image = PILImage.open(filename).convert('RGB')
    image_tensor = loader(image).float()
    image_var = Variable(image_tensor).unsqueeze(0)
    return image_var
Executed at 2023.12.15 00:10:17 in 4ms
```

```
# Convert the images in the folder to a tensor of 3 * 224 * 224
images_list = [load_image(folder_path+'/'+file) for file in jpg_files]
all_img = torch.cat(images_list, dim=0)
print(all_img.shape)
Executed at 2023.12.15 00:10:21 in 2s 284ms

torch.Size([250, 3, 224, 224])
```

Next step is converting weather labels into one-hot encoding tensor. We create these two tensors both by looping the list of images' name which we firstly defined, so they are one-to-one correspondence.

```python
# Convert weather labels into the form of one-hot encoding
label_list = []
weather_list = ['Cloudy', 'Foggy', 'Rainy', 'Snowy', 'Sunny']
for file in jpg_files:
    label = []
    for weather in weather_list:
        if weather in file:
            label.append(1)
        else:
            label.append(0)
    label_list.append(label)

all_labels = torch.Tensor(label_list)
print(all_labels.shape)
```
Executed at 2023.12.15 00:10:23 in 15ms

```
torch.Size([250, 5])
```

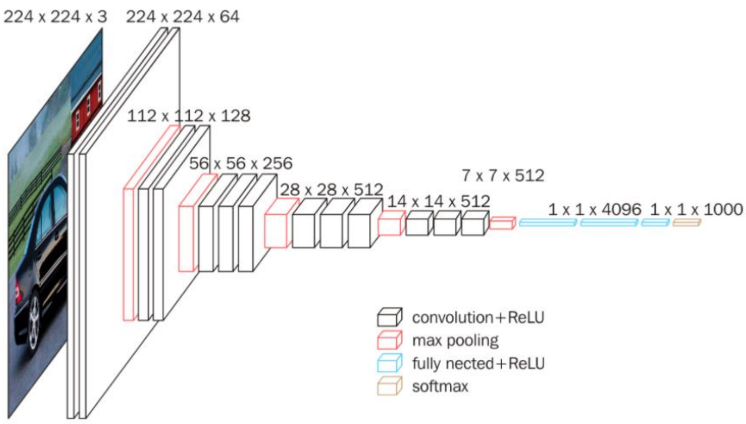
Finally, we use the TensorDataset, a existed Class of dataset in pytorch to encapsulate two tensors. And use Dataloader in pytorch to shuffle it and separate dataset by batch size of 25.

```python
# Encapsulate the data
train_dataset = TensorDataset(all_img, all_labels)
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
```

2. Introduction to model

The convolutional part uses the structure of vgg16, and the fully connected layer changes the final output to 5 dimensions.

VGG uses multiple convolutional layers with smaller kernels (3x3) instead of a convolutional layer with a larger kernel. On the one hand, it can reduce parameters, and on the other hand, it is equivalent to performing more nonlinear mapping, which can increase the network's fitting/expression ability.



The following is the model structure:

```python
# Construct a class for the model
class CNNModel(nn.Module):
    def __init__(self, num_classes=5):
        super(CNNModel, self).__init__()

        self.conv_layers = nn.Sequential(
            nn.Conv2d(3, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 64, kernel_size=3, padding=1),
            nn.BatchNorm2d(64),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),

            nn.Conv2d(64, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.Conv2d(128, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),

            nn.Conv2d(128, 256, kernel_size=3, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, kernel_size=3, padding=1),
            nn.BatchNorm2d(256),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),
```

```python
            nn.Conv2d(256, 512, kernel_size=3, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),

            nn.Conv2d(512, 512, kernel_size=3, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.Conv2d(512, 512, kernel_size=3, padding=1),
            nn.BatchNorm2d(512),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),

            )


        self.fc_layers = nn.Sequential(
            nn.Linear(512 * 7 * 7, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(0.1),

            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(0.1),
            nn.Linear(4096, 1000),
            nn.Linear(1000, 5),

            )
```

```python
        self.sigmoid = nn.Sigmoid()

    def forward(self,x):
        x = self.conv_layers(x)
        x = x.view(x.size(0), -1)
        x = self.fc_layers(x)
        # x = self.sigmoid(x)
        return x
Executed at 2023.12.15 00:18:50 in 18ms
```

3. Accuracy on the training set

Accuracy = 96.4%

```python
# Test

# Set the model to evaluation mode
model.eval()
correct = 0
with torch.no_grad():
    train_dataset = TensorDataset(all_img, all_labels)
    train_loader = DataLoader(train_dataset)
    for inputs, labels in train_loader:
        inputs, labels = inputs.to(device), labels.to(device)
        inputs = inputs.double()
        outputs = model(inputs.float())
        predicted = torch.max(outputs, 1)[1]
        label = torch.max(labels, 1)[1]
        correct += (predicted == label).sum()

# Calculate the accuracy
print('acc: %.2f %%' % (100 * correct / 250))
Executed at 2023.12.15 00:26:49 in 1s 39ms

acc: 96.40 %
```