Q1 Augmentation Implementation
2 分數

Implement augmentation by finishing train_tfm in the code with image size of your choice. Copy your train_tfm code here, for example :

```
train_tfm = transforms.Compose([
    # Resize the image into a fixed shape (height = width = 128)
    transforms.Resize((128, 128)),
    # You need to add some transforms here.

    transforms.ToTensor(),
])
```

```
policies = [transforms.AutoAugmentPolicy.CIFAR10,
transforms.AutoAugmentPolicy.IMAGENET]
augmenters = [transforms.AutoAugment(policy) for policy in policies]

train_tfm = transforms.Compose([
    transforms.Resize((128, 128)),
    *augmenters,
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

Q2 Residual_Model Implementation
2 分數

Implement Residual Connections in the Residual_Model, following the graph in the slides.
Copy your Residual_Model code and paste it here, for example:

```
from torch import nn
class Residual_Network(nn.Module):
    def __init__(self):
        super(Residual_Network, self).__init__()

        self.cnn_layer1 = nn.Sequential(
            nn.Conv2d(3, 64, 3, 1, 1),
            nn.BatchNorm2d(64),
        )

        self.cnn_layer2 = nn.Sequential(
            nn.Conv2d(64, 64, 3, 1, 1),
            nn.BatchNorm2d(64),
        )
```

```
        self.cnn_layer3 = nn.Sequential(
            nn.Conv2d(64, 128, 3, 2, 1),
            nn.BatchNorm2d(128),
        )

        self.cnn_layer4 = nn.Sequential(
            nn.Conv2d(128, 128, 3, 1, 1),
            nn.BatchNorm2d(128),
        )
        self.cnn_layer5 = nn.Sequential(
            nn.Conv2d(128, 256, 3, 2, 1),
            nn.BatchNorm2d(256),
        )
        self.cnn_layer6 = nn.Sequential(
            nn.Conv2d(256, 256, 3, 1, 1),
            nn.BatchNorm2d(256),
        )
        self.fc_layer = nn.Sequential(
            nn.Linear(256* 32* 32, 256),
            nn.ReLU(),
            nn.Linear(256, 11)
        )
        self.relu = nn.ReLU()

    def forward(self, x):
        # input (x): [batch_size, 3, 128, 128]
        # output: [batch_size, 11]

        # Extract features by convolutional layers.
        x1 = self.cnn_layer1(x)

        x1 = self.relu(x1)

        x2 = self.cnn_layer2(x1)

        x2 = self.relu(x2)

        x3 = self.cnn_layer3(x2)

        x3 = self.relu(x3)

        x4 = self.cnn_layer4(x3)

        x4 = self.relu(x4)

        x5 = self.cnn_layer5(x4)

        x5 = self.relu(x5)

        x6 = self.cnn_layer6(x5)

        x6 = self.relu(x6)
```

```
        xout = x6.flatten(1)

        xout = self.fc_layer(xout)
        return xout

from torch import nn
class Residual_Network(nn.Module):
    def __init__(self, input_dim, output_dim, prob=0.2):
        super(Residual_Network, self).__init__()

        self.cnn_layer1 = nn.Sequential(
            nn.Conv2d(3, 64, 3, 1, 1),
            nn.BatchNorm2d(64),
        )

        self.cnn_layer2 = nn.Sequential(
            nn.Conv2d(64, 64, 3, 1, 1),
            nn.BatchNorm2d(64),
        )

        self.cnn_layer3 = nn.Sequential(
            nn.Conv2d(64, 128, 3, 2, 1),
            nn.BatchNorm2d(128),
        )

        self.cnn_layer4 = nn.Sequential(
            nn.Conv2d(128, 128, 3, 1, 1),
            nn.BatchNorm2d(128),
        )
        self.cnn_layer5 = nn.Sequential(
            nn.Conv2d(128, 256, 3, 2, 1),
            nn.BatchNorm2d(256),
        )
        self.cnn_layer6 = nn.Sequential(
            nn.Conv2d(256, 256, 3, 1, 1),
            nn.BatchNorm2d(256),
        )
        self.fc_layer = nn.Sequential(
            nn.Linear(256 * 32 * 32, 256),
            nn.ReLU(),
            nn.Linear(256, 11)
        )
        self.relu = nn.ReLU()

    def forward(self, x):
        x1 = self.cnn_layer1(x)
```

```
x1 = self.relu(x1)

x2 = self.cnn_layer2(x1)

x2 += x1

x2 = self.relu(x2)

x3 = self.cnn_layer3(x2)

x3 = self.relu(x3)

x4 = self.cnn_layer4(x3)

x4 += x3

x4 = self.relu(x4)

x5 = self.cnn_layer5(x4)

x5 = self.relu(x5)

x6 = self.cnn_layer6(x5)

x6 += x5

x6 = self.relu(x6)

xout = x6.flatten(1)

xout = self.fc_layer(xout)
return xout
```

總分

4 / 4 pts

問題 1

Augmentation Implementation 2 / 2 pts

問題 2

Residual_Model Implementation 2 / 2 pts