Q1 Augmentation Implementation

2 Points

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(),
])
```

```
train_tfm = transforms.Compose([
  #new type
  transforms.RandomResizedCrop((128,128)),
  transforms.RandomHorizontalFlip(),
  transforms.GaussianBlur(7,3),
  transforms.ToTensor(),
  transforms.Normalize([0.485, 0.456, 0.406],[0.229, 0.224, 0.225])
])
```

Q2 Residual_Model Implementation

2 Points

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
```

```
class Classifier(nn.Module):
  def __init__(self):
    super(Classifier, self).__init__()
    # torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride, padding)
    # torch.nn.MaxPool2d(kernel_size, stride, padding)
    # input 維度 [3, 128, 128]
    dRate = 0.25
    self.cnn = nn.Sequential(
      nn.Conv2d(3, 64, 3, 1, 1),  # [64, 128, 128]
      nn.BatchNorm2d(64),
      nn.ReLU(),
      nn.Dropout(dRate),
      nn.MaxPool2d(2, 2, 0),      # [64, 64, 64]

      nn.Conv2d(64, 128, 3, 1, 1), # [128, 64, 64]
      nn.BatchNorm2d(128),
      nn.ReLU(),
```

```
            nn.Dropout(dRate),
            nn.MaxPool2d(2, 2, 0),      # [128, 32, 32]

            nn.Conv2d(128, 256, 3, 1, 1), # [256, 32, 32]
            nn.BatchNorm2d(256),
            nn.ReLU(),
            nn.Dropout(dRate),
            nn.MaxPool2d(2, 2, 0),      # [256, 16, 16]

            nn.Conv2d(256, 512, 3, 1, 1), # [512, 16, 16]
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.Dropout(dRate),
            nn.MaxPool2d(2, 2, 0),       # [512, 8, 8]

            nn.Conv2d(512, 512, 3, 1, 1), # [512, 8, 8]
            nn.BatchNorm2d(512),
            nn.ReLU(),
            nn.Dropout(dRate),
            nn.MaxPool2d(2, 2, 0),       # [512, 4, 4]
        )
        self.fc = nn.Sequential(
            nn.Linear(512*4*4, 1024),
            nn.ReLU(),
            nn.Linear(1024, 512),
            nn.ReLU(),
            nn.Linear(512, 11)
        )

    def forward(self, x):
        out = self.cnn(x)
        origin = nn.Sequential(
            nn.Conv2d(3,512,3,1,1),
            nn.MaxPool2d(32, 32, 0))(x)
        out = out+origin
        out = out.view(out.size()[0], -1)
        return self.fc(out)
```

HW3

● GRADED

STUDENT
梁峻瑋

TOTAL POINTS
3 / 4 pts

QUESTION 1

Augmentation Implementation **2** / 2 pts

QUESTION 2

Residual_Model Implementation **1** / 2 pts