DL_LAB3_Report _ Leukemia Classification

學號：312554004　姓名：林垣志

(1) Introduction:

In this lab, there are three tasks I need to accomplish. First, I will need to analysis acute lymphoblastic leukemia dataset, design our own data preprocessing using our custom DataLoader within the PyTorch framework. Second, classify acute lymphoblastic leukemia via the ResNet18/50/152 architecture. Finally, we have to plot the confusion matrix and the accuracy curve to evaluate the performance, also upload prediction result to the Kaggle competition.

The dataset contains 10,661 images, TA divided dataset into 7995 training data, 1599 validating data, and 1067 testing data. The image resolution is 450 * 450 pixels.

(2) Implementation Details:

(a) The detail of your model (ResNet)

ResNet18 is composed of Basic block and ResNet50/152 are composed of Bottleneck block. The difference between Basic block and Bottleneck block is that there are 1x1 convolution in beginning and end of Bottleneck. This way can reduce the number of parameters. The “make_layer” function in the class is used to construct convolution layers, where “down_sample” is used for the transformation of channels, height, and width between convolution layers. The output feature of the last layer is set to the number of classes: 5.

The figure (code) below shows the Basic Block class, Bottleneck Block class, and ResNet model composed of these blocks.

1. Basic Block class:

```python
""" BasicBlock for ResNet18 """
class BasicBlock(nn.Module):
    """
    output = (channels, H, W) -> conv2d (3x3) -> (channels, H, W) -> conv2d (3x3) -> (channels, H, W) + (channels, H, W)
    """
    expansion = 1
    def __init__(self, filters, strides = 1, downsample = None):
        super(BasicBlock, self).__init__()

        in_channels, out_channels = filters

        # Both self.conv1 and self.downsample layers downsample the input when stride != 1
        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, (3, 3), stride = strides, padding = (1, 1), bias = False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace = True)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(out_channels, out_channels, (3, 3), padding = (1, 1), bias = False),
            nn.BatchNorm2d(out_channels)
        )
        self.relu = nn.ReLU(inplace = True)
        self.downsample = downsample

    def forward(self, x):
        out = self.conv1(x)
        out = self.conv2(out)

        identity = x
        if self.downsample is not None:
            identity = self.downsample(x)

        out = self.relu(identity + out)
        return out
```

2. Bottleneck Block class:

```python
""" BottleneckBlock for ResNet50 & 152 """
class BottleneckBlock(nn.Module):
    """
    output = (channels * 4, H, W) -> conv2d (1x1) -> (channels, H, W) -> conv2d (3x3) -> (channels, H, W)
             -> conv2d (1x1) -> (channels * 4, H, W) + (channels * 4, H, W)
    """
    expansion = 4
    def __init__(self, filters, strides = 1, downsample = None):
        super(BottleneckBlock, self).__init__()

        in_channels, mid_channels = filters
        out_channels = mid_channels * self.expansion

        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels, mid_channels, (1, 1), stride = strides, padding = 0, bias = False),
            nn.BatchNorm2d(mid_channels),
            nn.ReLU(inplace = True)
        )
        self.conv2 = nn.Sequential(
            nn.Conv2d(mid_channels, mid_channels, (3, 3), stride = 1, padding = (1, 1), bias = False),
            nn.BatchNorm2d(mid_channels),
            nn.ReLU(inplace = True)
        )
        self.conv3 = nn.Sequential(
            nn.Conv2d(mid_channels, out_channels, (1, 1), stride = 1, padding = 0, bias = False),
            nn.BatchNorm2d(out_channels)
        )
        self.relu = nn.ReLU(inplace = True)
        self.downsample = downsample

    def forward(self, x):
        out = self.conv1(x)
        out = self.conv2(out)
        out = self.conv3(out)

        identity = x
        if self.downsample is not None:
            identity = self.downsample(x)

        out = self.relu(identity + out)
        return out
```

3. ResNet model and call function of ResNet18/50/152:

```python
""" ResNet Model """
class ResNet(nn.Module):
    def __init__(self, block, layers, num_classes = 5):
        super(ResNet, self).__init__()

        self.in_channels = 64

        self.conv1 = nn.Sequential(
            nn.Conv2d(in_channels = 3, out_channels = self.in_channels, kernel_size = (7, 7), stride = (2, 2),
                      padding = (3, 3), bias = False),
            nn.BatchNorm2d(self.in_channels),
            nn.ReLU(inplace = True)
        )
        self.conv2 = nn.Sequential(
            nn.MaxPool2d(kernel_size = (3, 3), stride = (2, 2), padding = 1),
            self._make_layer(block, 64, layers[0])
        )
        self.conv3 = self._make_layer(block, 128, layers[1], strides = 2)
        self.conv4 = self._make_layer(block, 256, layers[2], strides = 2)
        self.conv5 = self._make_layer(block, 512, layers[3], strides = 2)
        self.avgpool = nn.AdaptiveAvgPool2d(output_size = (1, 1))
        self.fc = nn.Linear(512 * block.expansion, num_classes)

    def _make_layer(self, block, out_channels, num_blocks, strides = 1):
        downsample = None

        if strides != 1 or self.in_channels != out_channels * block.expansion:
            downsample = nn.Sequential(
                nn.Conv2d(self.in_channels, out_channels * block.expansion, (1, 1), stride = strides, bias = False),
                nn.BatchNorm2d(out_channels * block.expansion),
            )

        build_layer = []
        build_layer.append(block(filters = [self.in_channels, out_channels], strides = strides, downsample = downsample))

        self.in_channels = out_channels * block.expansion
        for _ in range(1, num_blocks):
            build_layer.append(block(filters = [self.in_channels, out_channels]))

        return nn.Sequential(*build_layer)

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.conv4(x)
        x = self.conv5(x)
        x = self.avgpool(x)

        x = x.view(x.shape[0], -1)
        out = self.fc(x)
        return out
```

```python
""" Get(return) ResNet18 model """
def ResNet18():
    return ResNet(block = BasicBlock, layers = [2, 2, 2, 2])

""" Get(return) ResNet50 model """
def ResNet50():
    return ResNet(block = BottleneckBlock, layers = [3, 4, 6, 3])

""" Get(return) ResNet152 model """
def ResNet152():
    return ResNet(block = BottleneckBlock, layers = [3, 8, 36, 3])
```

(b) The detail of your Dataloader:

The Dataloader inherits torch.utils.data.Dataset, so I need to implement __init__, __len__, __getitem__. In terms of data preprocessing, I'm using RandomRotation, RandomVerticalFlip, RandomHorizontalFlip, Resize(350 * 350), CenterCrop(224), and transforms to Tensor as data augmentation. However, during the experimental process, I found that the normalization technique didn't yield significant benefits, so I decided not to use it. In validating / testing mode, I just use Resize(350 * 350), CenterCrop(224), and transforms to Tensor as data augmentation. The figure (code) are shown below.

```python
def Data_transform(mode):
#     normalize = transforms.Normalize(
#         mean=[0.485, 0.456, 0.406],
#         std=[0.229, 0.224, 0.225]
#     )

    # Transform
    if mode == 'train':
        transformer = transforms.Compose([
            transforms.RandomRotation(degrees = 20),
            transforms.RandomHorizontalFlip(p = 0.5),
            transforms.RandomVerticalFlip(p = 0.5),
            transforms.Resize([350,350]),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
#             normalize
        ])
    else:
        transformer = transforms.Compose([
            transforms.Resize([350,350]),
            transforms.CenterCrop(224),
            transforms.ToTensor(),
#             normalize
        ])
def getData(mode):
    if mode == 'train':

    elif mode == "valid":

    elif mode == "resnet_18":

    elif mode == "resnet_50":

    elif mode == "resnet_152":

class LeukemiaLoader(data.Dataset):
    def __init__(self, root, mode):
        """
        self.root = root
        self.mode = mode
        if mode == 'train' or mode == 'valid':
            self.img_name, self.label = getData(mode)
        else:
            self.img_name = getData(mode)

        self.transform = Data_transform(mode = mode)
        print("> Found %d images..." % (len(self.img_name)))

    def __len__(self):
        """'return the size of dataset"""
        return len(self.img_name)

    def __getitem__(self, index):
        """
        path = os.path.join(self.root, f'{self.img_name[index]}')
        img = self.transform(Image.open(path))
        label = []
        if self.mode == 'train' or self.mode == 'valid':
            label = self.label[index]

        return img, label
```

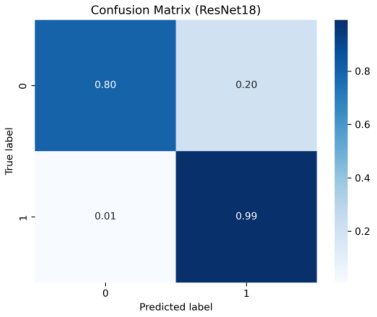
(c) Describing your evaluation through the confusion matrix:

The figure (code) on the left shows the implementation of my Confusion Matrix. Within the confusion matrix function, I set the hyperparameters including ground truth, predictions, title names, and filenames. Additionally, during the plotting process, I employ normalization to present the values in percentage form.

The figure on the right shows the ResNet18 evaluation result, we can observe that when the predicted label is 1 and the true label is also 1, I achieve a prediction accuracy close to 1. On the other hand, when the predicted label is 0 and the true label is also 0, the prediction accuracy is only 0.8. In other words, we have approximately 20% of images that are predicted incorrectly.

```
def plot_confusion_matrix(y_true, y_pred, classes=[0, 1], title = None,
                          cmap = plt.cm.Blues, filename = None):

    figure = confusion_matrix(y_true, y_pred, labels = classes, normalize = 'true')
    fig, ax = plt.subplots()
    sn.heatmap(figure, annot = True, ax = ax, cmap = cmap, fmt = '.2f')
    ax.set_xlabel('Predicted label')
    ax.set_ylabel('True label')
    plt.title(title)
    plt.savefig(filename, dpi = 300)
    plt.close()
```



(3) Data Preprocessing:

As mentioned above section 2.b, I employed various methods to implement data augmentation, including RandomRotation, RandomVerticalFlip, RandomHorizontalFlip, Resize(350 * 350), CenterCrop(224). Since the cells in the images are mostly located at the center and vary in size, so I first performed a resize operation to focus the images on the cells. Then, I use the CenterCrop method to extract a 224x224 image size from the center, ensuring that the cell remains in the center of the cropped image. Other transformations were mainly applied to generate various angles, enabling the machine to make more accurate judgments during subsequent recognition processes.

(4) Experimental results:

(a) The highest testing accuracy

The hyper parameters I use are shown below.

Batch_size	Learning rate	Epochs	Optimizer	Loss function
32	0.001	80	Adam	CrossEntropy

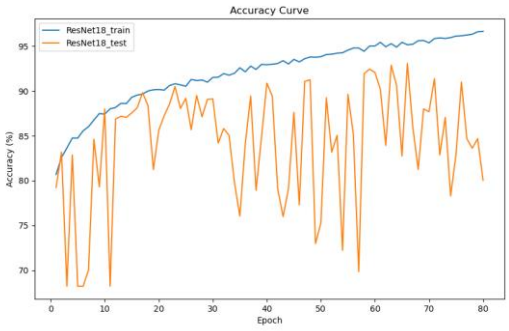
The highest testing accuracy are shown below.

- ResNet18: 93.058%
- ResNet50: 93.558%
- ResNet152: 91.807%

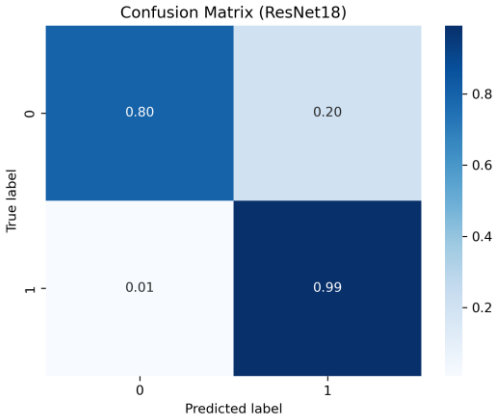
(b) Comparison figures

- **ResNet18:**

Accuracy Curve

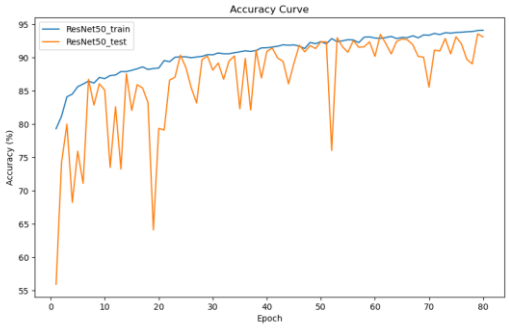


Confusion Matrix

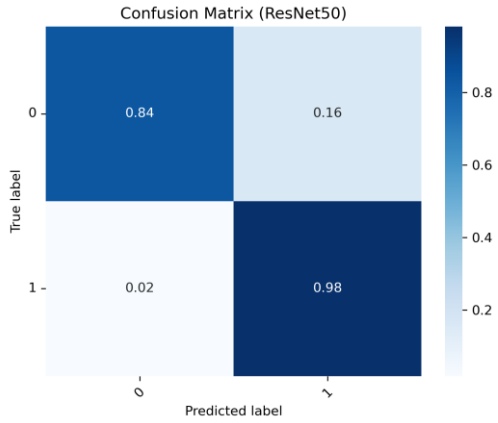


- **ResNet50:**

Accuracy Curve

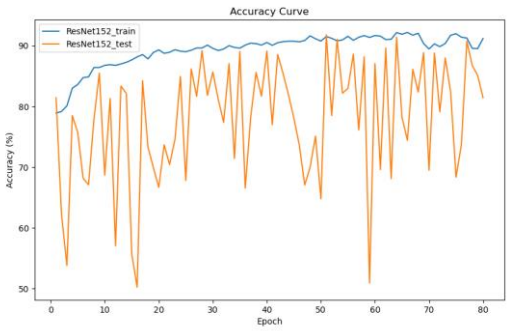


Confusion Matrix

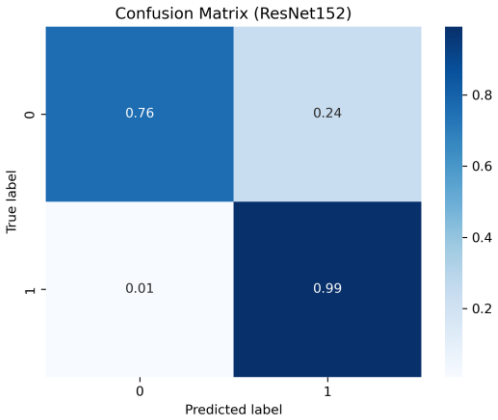


- **ResNet152:**

Accuracy Curve



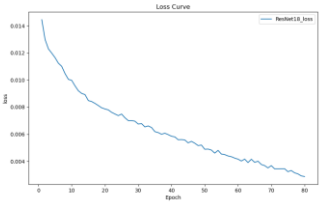
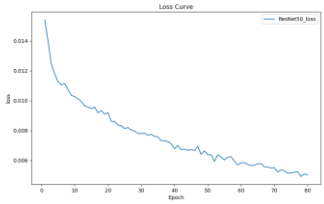
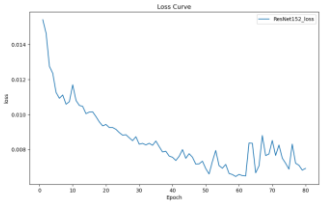
Confusion Matrix



(5) Discussion:

(a) The results of training loss

In the experiments, I observed that ResNet18's loss can achieve to 0.002, and it's smoothly than the other two model. As we can see that ResNet152's loss curve has slight oscillations, but I'm not sure why the loss is not consistently decreasing.

ResNet18	ResNet50	ResNet152
		

(b) Encountered Issue

- At first, the optimizer set "Sgd"，the number of epochs doesn't need to be set too high, around 60 epochs can achieve a training acc = 100%, resulting in a loss of almost below 0.001. However, the testing results are not as high. When I change to using Adam as the optimizer, the testing results improved compared to SGD. But the drawback is that I need to set a much higher number of epochs to converge the loss to 0.004. Additionally, there might be slight oscillations in the loss curve (liked ResNet152), and the overall training time for the model increased by 2 times.
- I'm tried to increase the batch_size of 64 during the experiments. However, due to the high-dimensional nature of the image data and the deep layer structure of ResNet152, I encountered "CUDA out of memory" errors. As a result, I could only set the batch size to a maximum of 32. Nevertheless, I experimented with batch_size of 64 when running the ResNet18 model. Surprisingly, the experimental results didn't outperform those achieved with batch_size of 32.