Lab 3 Leukemia Classification

林穎志 411551007

1. Introduction

This is a binary classification task using deep learning approach on acute lymphoblastic leukemia data. We implemented the 3 classic ResNet models: ResNet18, ResNet50 and ResNet152. Furthermore, we tried our ultimate to seek for the best performance and plotted the figures containing training/testing accuracy and loss of different model. Confusion matrix heat maps are given.

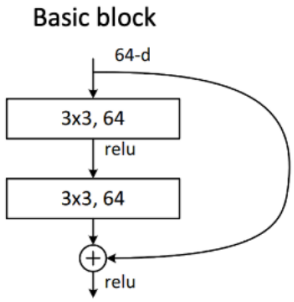
2. Implementation Details

A. The details of your model (ResNet)

- The main innovation of ResNet is the use of **residual blocks**, which enable training a very deep neural networks (hundreds of layers) by addressing the **vanishing gradient problem(VGP)**. The key behind residual blocks is to introduce a skip connection, allowing the network to learn the residual between the input and output of each block. This way, the network can focus on learning the small incremental changes in the features, making it easier to train deeper networks effectively.
- ResNet uses strided convolutions and pooling operations to perform **downsampling** in the network. This reduces the spatial dimensions of feature maps and increases the receptive field of higher layers.
- ResNet uses **global average pooling** to aggregate features spatially. This approach reduces the number of parameters and mitigates overfitting.

a. ResNet18

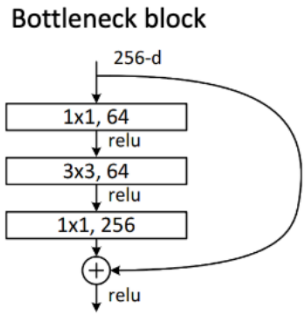
ResNet-18 consists of 18 layers, including 4 convolutional blocks consists of 2 groups of 2 convolutional layers (4*2*2 layers) for each, and 1 initial convolutional layer and 1 fully connected layer.



Basic block

b. ResNet50

ResNet-50 consists of 50 layers, making it a deep neural network. It contains multiple residual blocks, each containing 3 convolutional layers with shortcut connections which is called the **Bottleneck block**. The Bottleneck block consists of three convolutional layers: 1x1 convolution, 3x3 convolution, and another 1x1 convolution. These layers help reduce the computational cost while maintaining a deeper architecture.



Bottleneck block

c. ResNet152

ResNet152 is a much deeper model compared to ResNet-50, containing 152 layers. Same as ResNet50, ResNet152 mainly built from the 3-layer Bottleneck block. What the different is ResNet152 composites of more Bottleneck blocks, which goes very deep. The comparison of the ResNets is shown as below:

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Comparision of ResNets

B. The details of your Dataloader

In PyTorch, `DataLoader` is a utility that provides an efficient way to load and iterate over datasets during training and validation in deep learning models. We build the dataloader following the steps from spec.

step1. Get the image path from 'self.img_name' and load it.

step2. Get the ground truth label from self.label

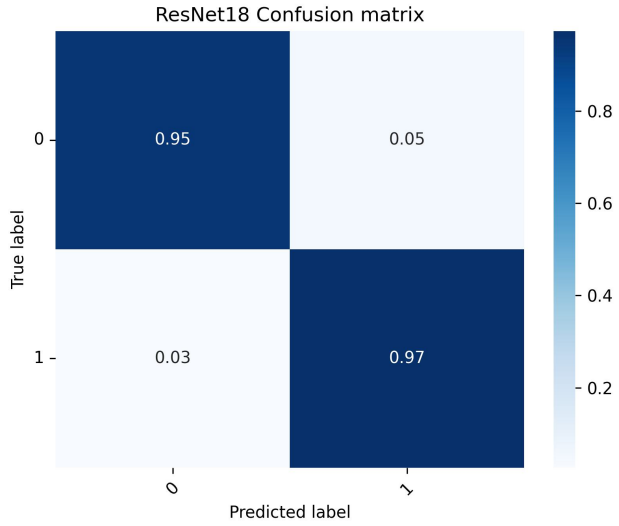
step3. Preprocessing of the data by different modes (train, validation/ test)

step4. Return processed image and label

```
class LeukemiaLoader(data.Dataset):
    def __init__(self, args, root, mode):
        """
        Args:
            mode : Indicate procedure status(training or testing)

            self.img_name (string list): String list that store all image names.
            self.label (int or float list): Numerical list that store all ground truth label values.
        """
        self.root = root
        if mode == 'test':
            self.img_name = getData(args, mode)
        else:
            self.img_name, self.label = getData(args, mode)

        self.mode = mode
        self.transform = transform(args, 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):
        """something you should implement here"""

        if self.mode != 'test':

            f_name = self.img_name[index]
            img_path = os.path.join(f_name)
            img = Image.open(img_path)
            label = self.label[index]
            img = self.transform(img)

            return img, label

        else:
            f_name = self.img_name[index]
            img_path = os.path.join(f_name)
            img = Image.open(img_path)
            img = self.transform(img)

            return img
```

C. Describing your evaluation through the confusion matrix

In the confusion matrix, we can see that the value in each block has been normalized. For those cases which label = 1, we get 97% of predictions corrected, which means our model sensitivity is 97%. For those cases which label = 0, we get 95% of predictions corrected, which means our model specificity is 95%. Our model shows high accuracy of ability on classifying leukemia.



Confusion matrix of ResNet18

3. Data Preprocessing

A. How you preprocessed your data

We transform the images according to different mode(train/ validation , test). During the training phase, we do **resizing, random flipping(vertical, horizontal), rotation and normalization**. In the testing and validation phase, we do **resizing and normalizing**.

```
def transform(args, mode ='train'):
    size = (args.img_x, args.img_y)

    if mode == 'train':
        transform =  transforms.Compose([
            transforms.ToTensor(),
            transforms.Resize(size),
            transforms.RandomHorizontalFlip(),
            transforms.RandomVerticalFlip(),
            transforms.RandomRotation(degrees=20),
        ])
    else:
        if args.five_crop:

        else:
            transform = transforms.Compose([
                        transforms.ToTensor(),
                        transforms.Resize(size),
        ])

    return transform
```

B. What makes your method special

a. Overlapping multi-crop evaluation

We do “**overlapping multi-crop evaluation**” which as known as **5-crop validation** or **10-crop validation**. The 5 crops are the center crop and the crops from the 4 corners of the image; while the 10 crops are the 5 crops with its horizontally flipped image (5+5). We first crop each patch in the size of 384x384, then giving the model to do prediction. The result will be shown in discussion.

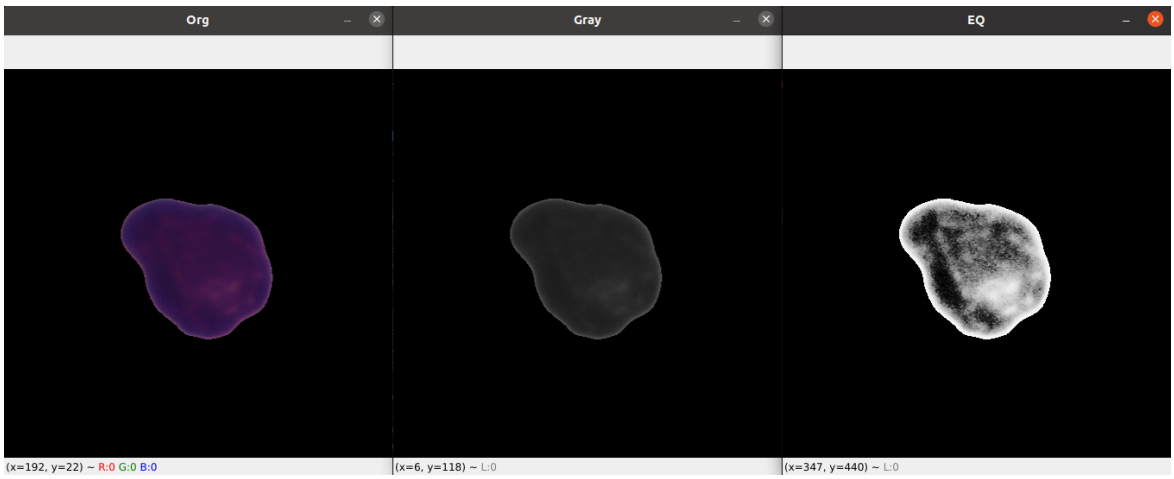
```
def transform(args, mode ='train'):
    size = (args.img_x, args.img_y)
```

```
    if mode == 'train':

    else:
        if args.five_crop:
            transform = transforms.Compose([
                        transforms.Resize(size),
                        transforms.FiveCrop((384, 384)),
                        transforms.Lambda(lambda crops: torch.stack([transforms.ToTensor()(crop) for crop in crops]))

                        ])
        else:

    return transform
```

b. Gray scale histogram equalization

We certainly change the image into **gray scale** then do **histogram equalization** before training. All the datasets are preprocessed using same flow. Further experiment results will be shown in discussion. The example images are shown as below:

```
img = cv2.imread(img_path, 0)
img_eq = cv2.equalizeHist(img)
```



Gray scale and histogram equalization

c. Center crop

We assume most of the features are in the middle of the image. We further crop all the images in the datasets from 450*450 to 384*384. The experiment results will be shown in discussion.

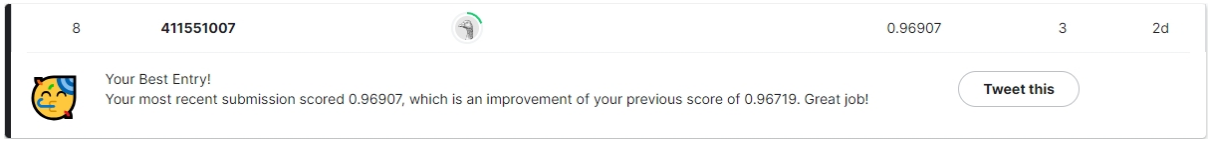
```
transform =  transforms.Compose([
            transforms.ToTensor(),
            transforms.RandomHorizontalFlip(),
            transforms.RandomVerticalFlip(),
            transforms.RandomRotation(degrees=20),
            transforms.CenterCrop((384, 384))
        ])
```

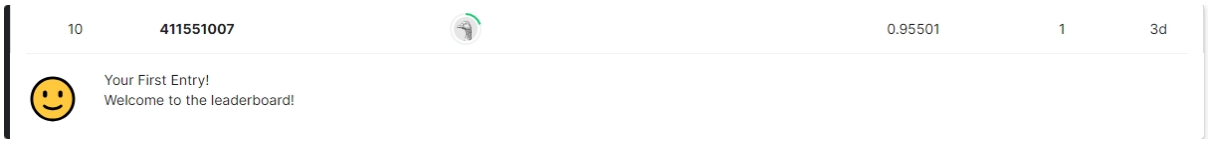
4. Experiment results

A. The highest testing accuracy(Screenshot)

Model	**Accuracy**	Optimizer	Learning rate	Loss	Batch size	Epoch
ResNet18	96.91%	Adam	0.001	BCE	64	300

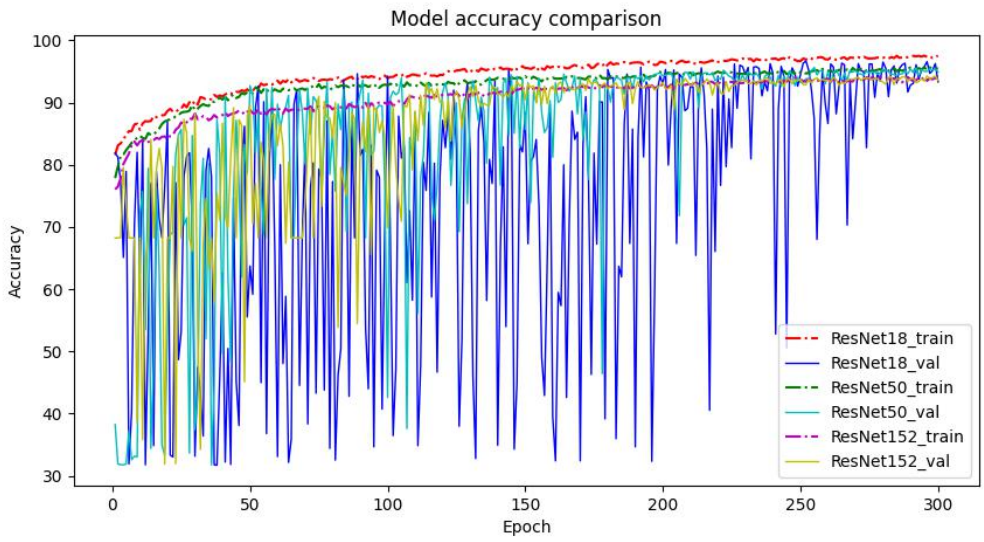
ResNet50	95.50%	Adam	0.001	BCE	32	300
ResNet152	93.44%	Adam	0.001	BCE	16	300





B. Comparison figures

We can see that in this case, ResNet18 gets the highest accuracy at training and validation phase. The trend of training and validation accuracy show the model is converging and without overfitting.



Comparison of model accuracy

5. Discussion

A. Anything you want to share

We had done some experiments to seek for the best performance as we can do. The results show below are the validation accuracy.

a. Overlapping multi-crop evaluation

We do **overlapping multi-crop evaluation** while doing validation and inference. The main motivation behind 5-crop validation is to improve the model's performance and robustness by considering multiple views of the same image during evaluation[1]. Each crop provides a slightly different perspective of the image, which can help the model make more accurate predictions, especially in cases where the object of interest is not centered or when the image contains

multiple objects. Each crop is predicted by the model, and class scores (probabilities) are obtained. The final prediction will be the average of the 5 or 10 crops of the image. The result shows that in our task, performing 5-crop or 10-crop is not promising to the growth of accuracy.

Model	Baseline	5-crop validation	10-crop validation
ResNet18	**96.69%**	96.19%	96.19%
ResNet50	95.62%	**95.75%**	95.56%
ResNet152	**94.25%**	93.06%	92.93%

b. Gray scale histogram equalization

It has been known that doing **histogram equalization** on medical image at preprocessing introduces a better performance[2]. We do experiments to observe whether this method can show the same trend of performance on modeling compared to the papers. The result shows that in this task, performing gray scale and histogram equalization may not be a good choice.

Model	Baseline	Histogram equalization
ResNet18	**96.69%**	86.43%

c. Center crop

For validating our assumption, we do center cropping for all the images. While there are no futher performance improvement, we infer that the main feature may not always locates on the middle of the image.

Model	Baseline	Center crop
ResNet18	**96.69%**	96.25%

Reference

[1]Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, *25*.

[2]Latif, J., Xiao, C., Imran, A., & Tu, S. (2019, January). Medical imaging using machine learning and deep learning algorithms: a review. In *2019 2nd International conference on computing, mathematics and engineering technologies (iCoMET)* (pp. 1-5). IEEE.