Summer 2023 DLP Lab3

312552017 董家鳴

1. Introduction (10%)

In this lab, we need to implement a top-rated Convolutional Neural Network (CNN) architecture, ResNet, to analyze acute lymphoblastic leukemia in PyTorch. We used the three datasets that TA provided on Kaggle for our data analysis. Each image in this dataset is labeled either 0 for a normal cell or 1 for a leukemia blast. Our task is to create an automated analysis system that judges whether the cell is a Leukemia blast or not.

The requirements in this lab are listed below:

1. Implement the ResNet18, ResNet50, ResNet152 architecture on your own; do not call the model from any library.
2. Train the model from scratch, do not load parameters from any pre-trained model.
3. Compare and visualize the accuracy trend between the 3 architectures. Plot each epoch's accuracy (not loss) during the training phase and validation phase.
4. Implement a custom DataLoader.
5. Design a data preprocessing method.
6. Calculate the confusion matrix and plotting.

2. Implementation Details (30%)

A. The details of your model (ResNet)

- **Basic Blocks and Bottleneck** The ResNet architecture with different layers(e.g., 18-layer, 34-layer, or 50-layer) would consist of different blocks. According to figure 5 in the paper [1], the ResNet18/34 consists of a basic block, while the ResNet-50/101/152 consist of a bottleneck block.
- **ResNet** According to the different layers depicted in Table 1 [1], we can follow the numbers of blocks stacked and channels needed to construct a convolution block. Eventually, we can build a ResNet. For simplicity, Here I just show the conv3_x for ResNet18 and two blocks stacked of the conv3_x for ResNet50. Notice that when we're doing the implementation of the shortcut, we need to use the convolution when the input and output channel is not the same.

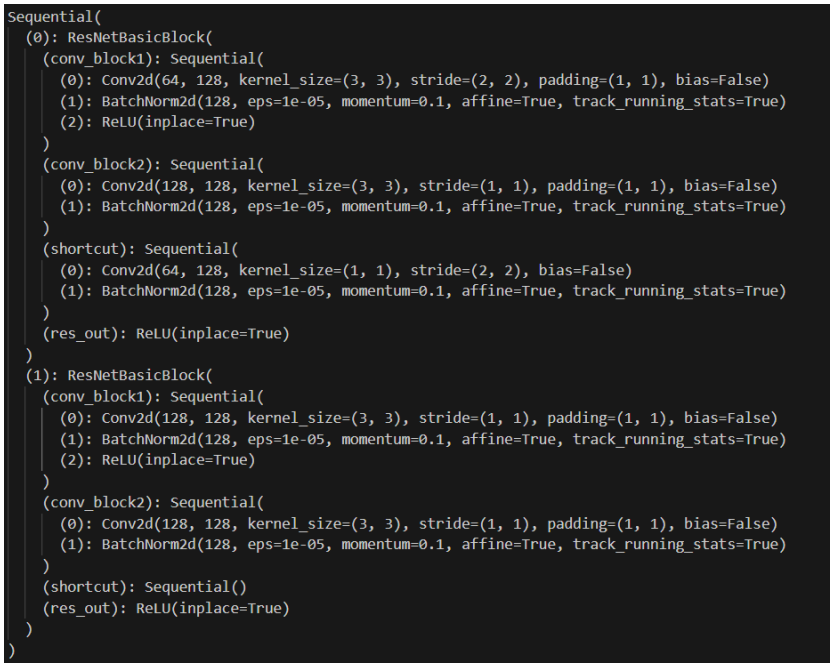


Figure 1: conv3_x used for ResNet18

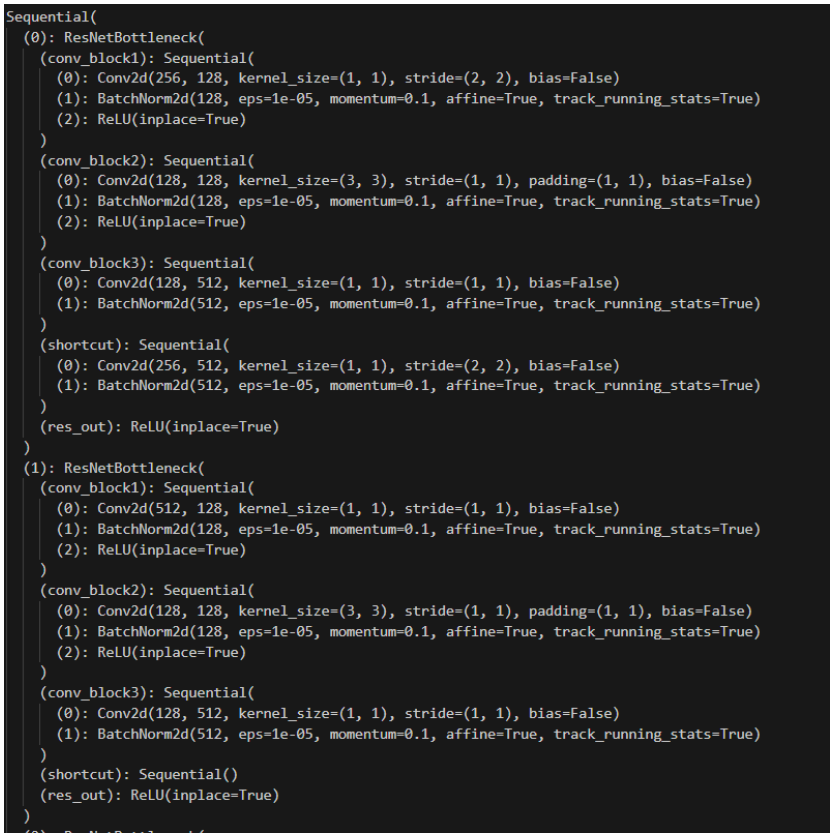


Figure 2: Part of conv3_x used for ResNet50

B. The details of your Dataloader

- **___init___** Dataloader would be used to load the different datasets like training or

testing, so we need to define two modes for different kinds of usage in the constructor.

- **___getitem___** In an abstract class of the Dataset, this special method is used to retrieve the item at a specific index from the dataset. Therefore, we need to open the images and perform the transformation on them.
- **___len___** This special method is used to get the number of items in a dataset object, so we can simply return the size of the dataset.
- **getData** This function is used for getting the image path and its label. In my Data pre-processing method, I additionally use my hand-craft training and validation dataset based on the original dataset, so other CSV files are used here.
- **transformer** In this function, I only transform the image to Tensor and then normalize it. Other data augmentation skills like center cropping, random flipping, and rotation are handled in the data pre-processing.

C. Describing your evaluation through the confusion matrix

- **Confusion matrix** The function in figure 3 is used to visualize the performance of a classification model. It will receive prediction results from the model, and compare it to the ground truth. Besides, the labels is a list containing classes. In this lab, the labels will be 0, and 1. Each row of my matrix represents the instances in a predicted class, while each column represents the instances in an actual class. By normalizing, we can find out what percentage of errors are present and what kind of types.

```python
@staticmethod
def plot_confusion_matrix(y_true, y_pred, labels, model_name):
    cm = confusion_matrix(y_true=y_true, y_pred=y_pred, labels=labels)
    cm_normalized = np.round(cm / np.sum(cm, axis=1).reshape(-1, 1), decimals=2)
    sns.heatmap(cm_normalized, annot=True, cmap='Blues')
    plt.title(f'{model_name} Normalized Confusion Matrix')
    plt.xlabel('Predicted label')
    plt.ylabel('True label')
    plt.savefig( model_name + '_confusion_matrix' + '.png')
    plt.show()
```

Figure 3: Codes of plotting confusion matrix

3. Data Preprocessing (20%)

A. How you preprocessed your data?

- **Central crop** The cropping technique can get rid of the black border in the original image. In order to have a fixed resolution, we need to resize the image by scaling to the pixel that we want, so that the shorter side will be length 224. Since the rescale may not make the longer size be the length 224, we need to do the center crop to get the square, which is the 224 * 224 image.
- **Random flip** The Random flip is kind of a data augmentation skill, which can increase the diversity and amount of training data without actually collecting new data. The horizontal flip, vertical flip, and random rotation are applied in my pre-processing.

- **Preprocessing method flow** The method can be divided into training, validation, and testing dataset pre-processing. For the training data, firstly, I perform a center crop on the original data, then perform random flips on images labeled as 0, generating new images to balance the number of 0s and 1s. Finally, I perform random flips on the entire dataset and generate new images again. So in the end, there are a total of 21074 images for training. For the validation data, firstly, I perform a center crop on the original data, then perform random flips on them to generate new images for training. So in this part, the validation set is divided into two parts, one for training the model and the other for fine-tuning the model's hyperparameters. Finally, for the testing dataset, I only performed a center crop.

B. What makes your method special?

- **Center crop 224** The reason for doing a 224 center crop is mainly because, in section 3.4 of the ResNet Paper [1], it is mentioned that they first resize the image to 224 * 224 pixels before inputting it to the model.
- **More training data** ResNet is a large architecture, and I think the original training data was too limited. Therefore, I used random flips and rotations to generate more data. Moreover, in order for the model to learn how to distinguish the validation dataset without just using the original one, I generated an additional dataset from the validation data for the model to learn from.
- **Balance Training data** Because there are more labels of 1 in the training images, I am afraid that the model will only determine 1, so I generate data with label 0 to balance it.
- **Reducing the loading time** After cropping the image to a smaller size, I save new the image to reduce the loading time each time we're doing the training.

4. Experimental results (10%)

A. The highest testing accuracy is as depicted in the figure 4. The accuracies of ResNet 18/50/152 are 0.96, 0.94, and 0.93 respectively. For all the models, I use the same hyper-parameter setting in table 1.

Figure 4: Highest Accuracy of ResNet18/50/152

B. Comparison figures is depicted in figures 5. For each model, I trained for 40 epochs. From figure 5, we can see a significant drop in the validation curve. I believe this is because

Batch size	Epochs	Learning rate	Optimizer	Loss function
16	40	1e-3	SGD	CrossEntropy

Table 1: Hyperparameters of the models

the loss functions for training and validation are different. Therefore, although the training accuracy can almost always improve, the validation accuracy may fluctuate continuously. Figure 6 to 8 is the confusion matrix for three models. From the graph, the performance of all three models is similar, with ResNet18 performing the best.

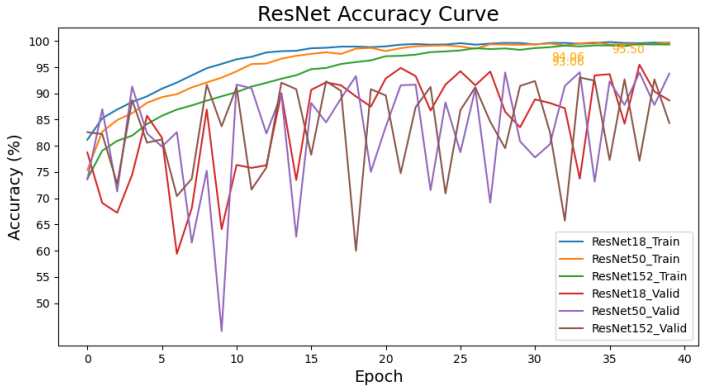


Figure 5: Result Comparison of accuracy

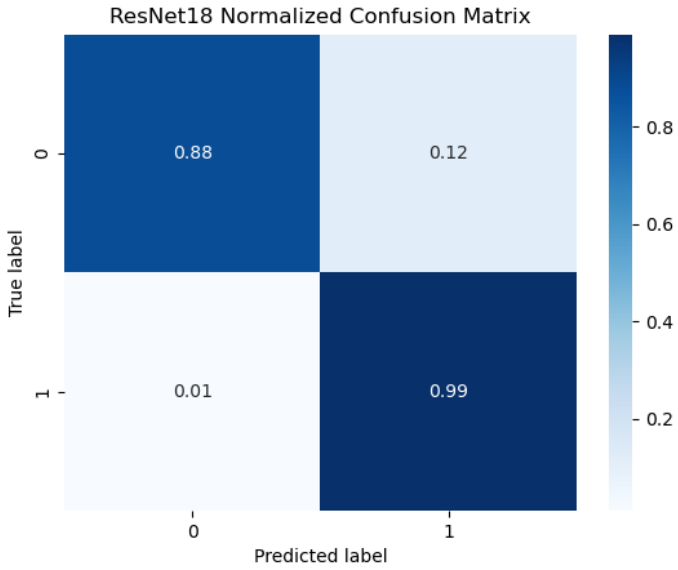


Figure 6: ResNet18 confusion matrix

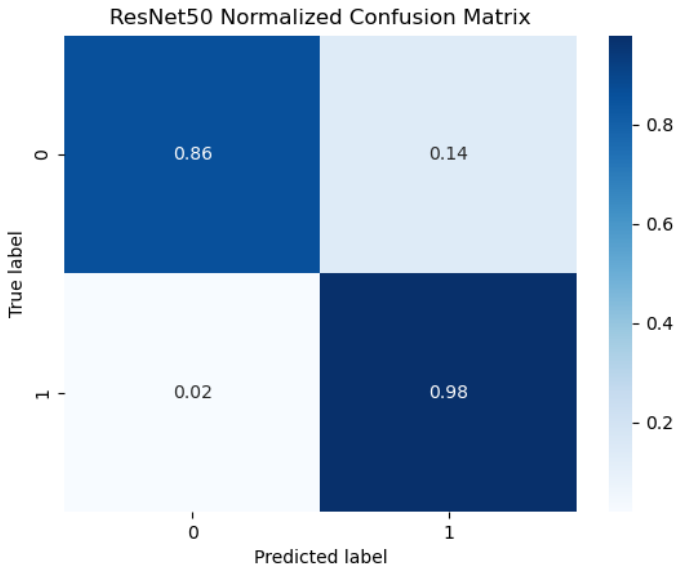


Figure 7: ResNet50 confusion matrix

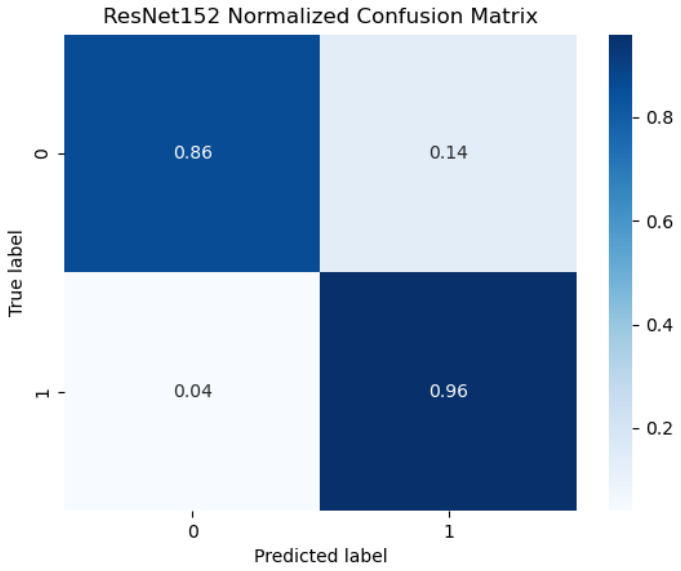


Figure 8: ResNet152 confusion matrix

5. Discussion (30%)

A. Validation accuracy fluctuates

- In section 4 experimental results, figure 5, the validation accuracy is continuously fluctuating. I think it's because the loss of training data and validation data is different since the data distribution is different. The following figure 9 in [2] explains what I'm saying.
- Another reason may be the learning rate. If the learning rate is set too high, the model

might skip the optimal solution during training. Instead of steadily converging towards the minimum loss, the model might bounce around, resulting in fluctuating validation accuracy.

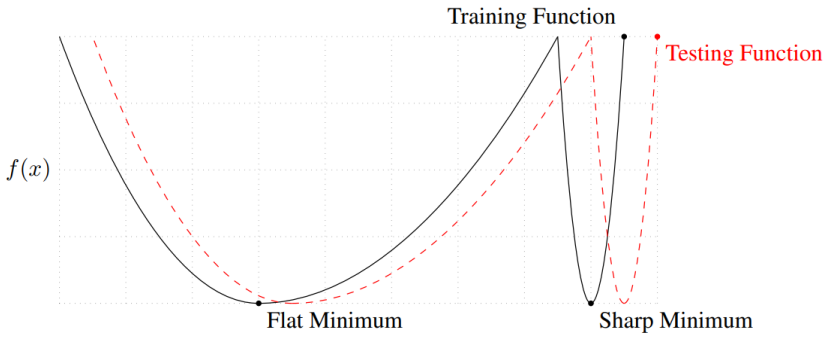


Figure 9: Loss of training and validation data

B. Small batch size is better on testing

- When we are training, we optimize through gradient descent based on the loss. A smaller batch size means more gradient descents, while a larger batch size means fewer. Suppose the loss of the testing data is slightly to the right of the loss of the training data like in figure 9. In this case, the performance of flat minima on the testing data will be similar to the training data, while the performance of sharp minima will be greatly different.
- The directions of parameter updates in small batch sizes are numerous and fast, so when a small batch encounters a small valley (sharp), it may jump out of it quickly, so it usually occurs in flat minima. The situation with large batches is the opposite.
- To prove this concept, I also trained a ResNet18 model with 16 and 64 batch sizes which are depicted in figure 10 and 11. From the figures, We can see that the smaller batch size oscillates more severely because the parameters update more quickly, while the larger one is more stable.

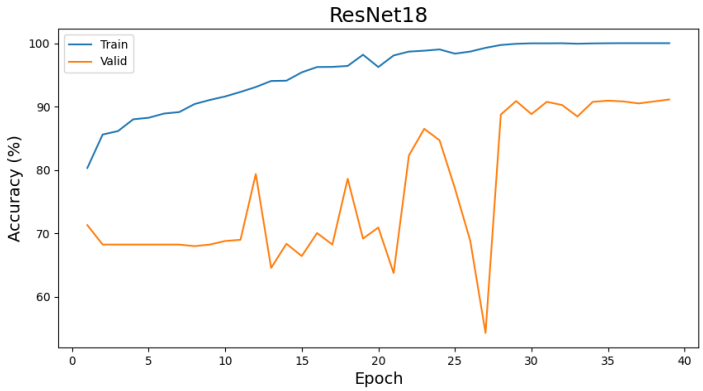


Figure 10: Large batch size

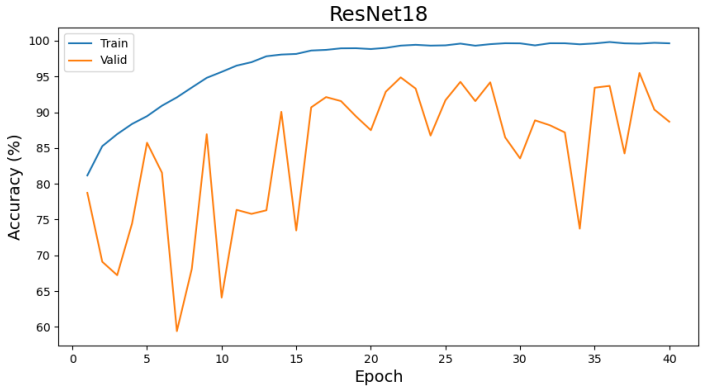
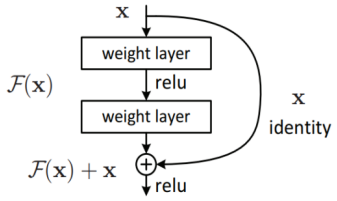


Figure 11: Small batch size

C. ResNet can avoid vanishing gradient problem

- When the model gets deeper, the vanishing gradient problem becomes more serious. ResNet can use shortcuts to add gradients from shallow layers, making the current layer's gradient larger. Figure 12 in TA's provided slide shows the problem.



$$\mathbf{x}_L = \mathbf{x}_l + \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{x}_i, \mathcal{W}_i),$$

$$\frac{\partial \mathcal{E}}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}\frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L}\left(1 + \frac{\partial}{\partial \mathbf{x}_l}\sum_{i=l}^{L-1}\mathcal{F}(\mathbf{x}_i, \mathcal{W}_i)\right).$$

Figure 12: ResNet use shortcuts to avoid vanishing gradient problem

- To prove that ResNet indeed solves the vanishing gradient problem, let's look at the testing loss of ResNet50 and ResNet152. From figures 13 and 14, we can see that on average, the testing loss of ResNet152 is lower than that of ResNet50. This indicates that after using shortcuts, the performance of deeper layers indeed does not deteriorate

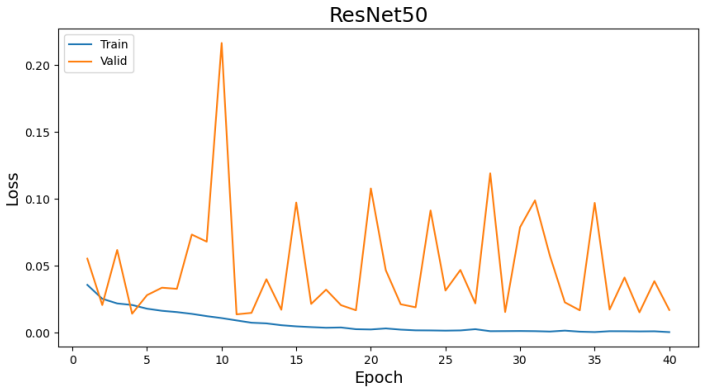


Figure 13: ResNet50 loss curve

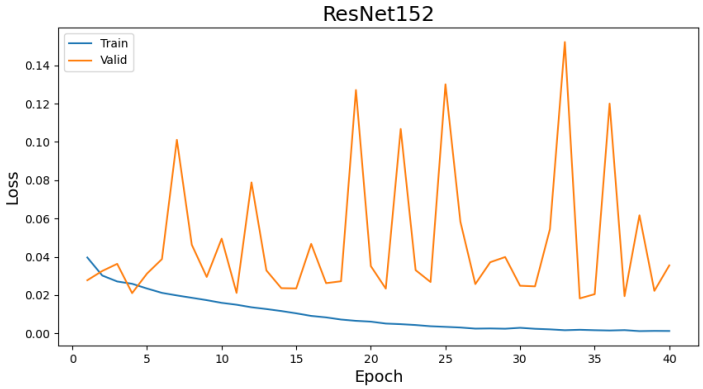


Figure 14: ResNet152 loss curve

References

[1] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 770–778. DOI: 10.1109/CVPR.2016.90. URL: https://ieeexplore.ieee.org/abstract/document/7780459.

[2] Nitish Shirish Keskar et al. *On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima*. 2017. arXiv: 1609.04836 [cs.LG].