

Lab4: Diabetic Retinopathy Detection

TA 劉子齊 Jonathan

Lab Objective:

In this lab, you will need to analysis diabetic retinopathy (糖尿病所引發視網膜病變) in the following three steps. First, you need to write your own custom DataLoader through

PyTorch framework. Second, you need to classify diabetic retinopathy grading via the ResNet architecture [1]. Finally, you have to calculate the confusion matrix to evaluate the performance.

Important Date:

1. Experiment Report Submission Deadline: 4/25 (Tue) 11:55 a.m.
2. Demo date: 4/25 (Tue)

Turn in Experiment Report (.pdf) & Source code (.py)

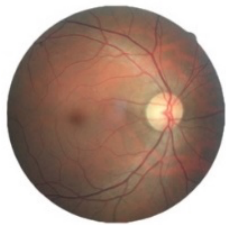
Notice: zip all files in one file and name it like 「DLP_LAB4_YourStudentID_name.zip」, ex: 「DLP_LAB4_311605004_劉子齊.zip」

Requirements:

1. Implement the **ResNet18** 、 **ResNet50** architecture and **load parameters** from a pretrained model
2. Compare and **visualize** the accuracy trend between the **pretrained model and without pretraining** in same architectures, you need to plot each **epoch accuracy (not loss)** during training phase and testing phase.
3. Implement your own custom **DataLoader**
4. Design **your own data preprocessing method**
5. Calculate the **confusion matrix** and **plotting**

Dataset- Diabetic Retinopathy Detection (kaggle):

Diabetic retinopathy is the leading cause of blindness in the working-age population of the developed world. It is estimated to affect over 93 million people. This dataset provided with a large set of high-resolution retina images taken under a variety of imaging conditions.



Format: .jpeg

Reference :

<https://www.kaggle.com/c/diabetic-retinopathy-detection#description>

Implementation Details:

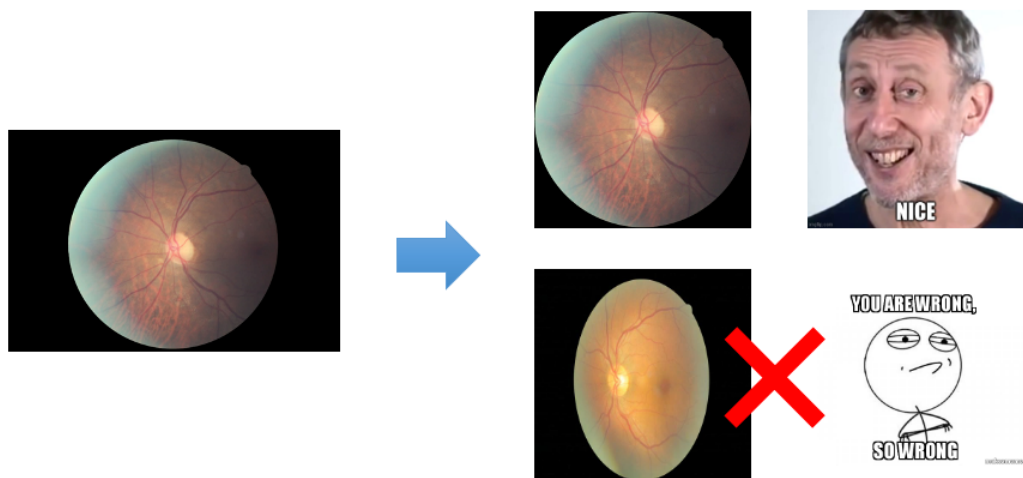
Prepare Data

The dataset contains 35124 images, we divided dataset into 28,100 training data and 7,026 testing data. The images' resolutions are different and are required to be preprocessed into the same resolution, which is 512 * 512.

Download link:

https://drive.google.com/drive/folders/1Kh2KI9-BqP4kEk9n59mxOw0Z1pJy0wuv?usp=share_link

Please note, when preprocessing data, as shown in the figure below, do not change the original eyeballs into an oval shape to avoid affecting the recognition results.



Custom Dataloader

- This is the skeleton that you have to fill to have a custom dataset, refer to “dataloader.py”

```
class RetinopathyLoader(data.Dataset):
    def __init__(self, root, mode):
        """ ...

        self.root = root
        self.img_name, self.label = getData(mode)
        self.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"""

        """ ...

        return img, label
```

- The `__init__` function is where the initial logic happens like reading a csv, assigning transforms etc.
- The `__getitem__` function returns the data and labels and you can process the data like loading image, preprocessing, transforming image before returns.
- The index parameter where in the `__getitem__` function, it is the nth data/image you are going to return.
- Reference:
https://pytorch.org/tutorials/beginner/data_loading_tutorial.html
<https://github.com/utkuozbulak/pytorch-custom-dataset-examples>

- You can use `getData` function to read all images name and ground truth.

```
def getData(mode):
    if mode == 'train':
        img = pd.read_csv('train_img.csv')
        label = pd.read_csv('train_label.csv')
        return np.squeeze(img.values), np.squeeze(label.values)
    else:
        img = pd.read_csv('test_img.csv')
        label = pd.read_csv('test_label.csv')
        return np.squeeze(img.values), np.squeeze(label.values)
```

ResNet

ResNet (Residual Network) is the Winner of ILSVRC 2015 in image classification, detection, and localization, as well as Winner of MS COCO 2015 detection, and segmentation [2].

- Degradation problem:** the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly. Not overfitting, it's the vanishing/ exploding gradient problem.

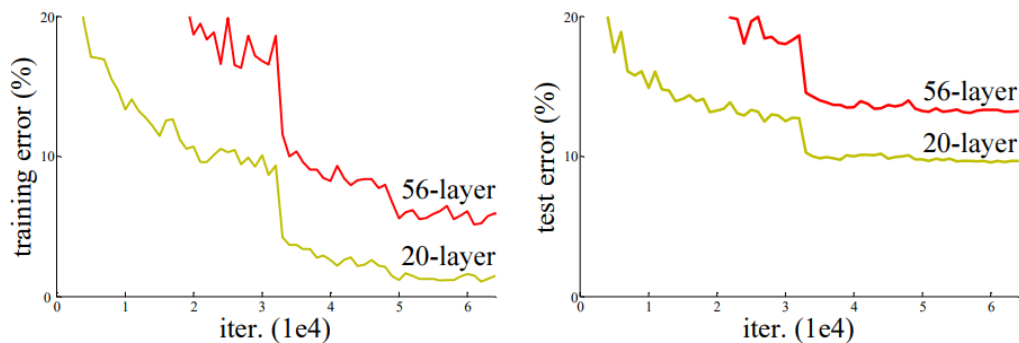
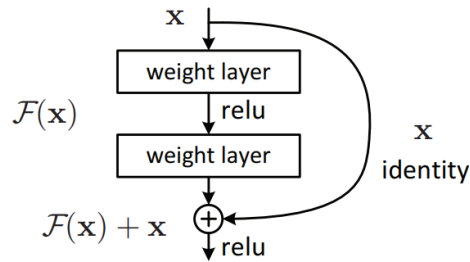


Figure. Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer “plain” networks. The deeper network has higher training error, and thus test error.

- Skip/ Shortcut connection:**
To solve the problem of vanishing/exploding gradients, a skip/ shortcut connection is added to add the input x to the output after few weight layers as below [2]

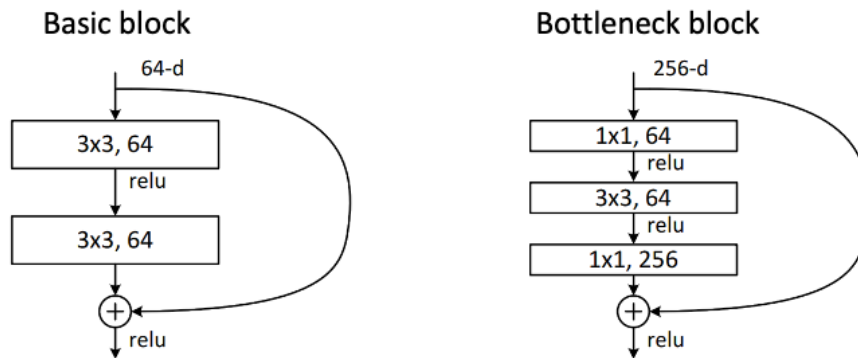


- Why ResNet can avoid vanishing gradient problem?

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

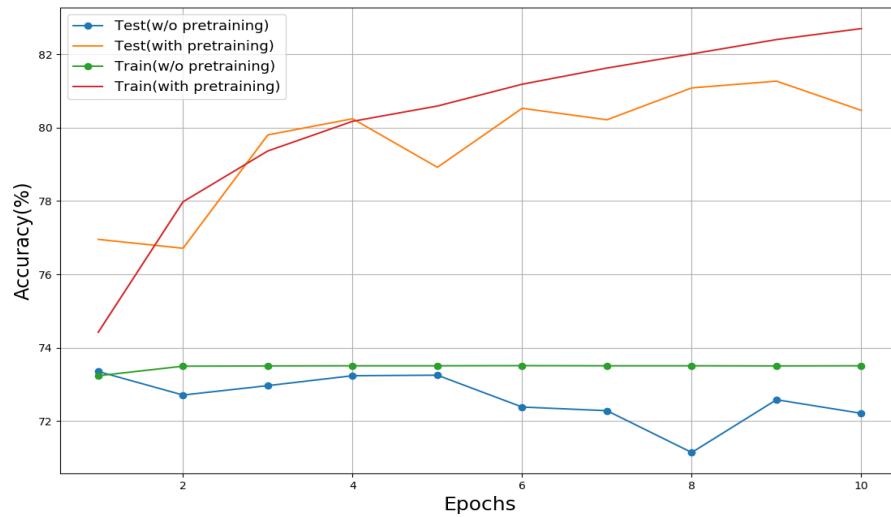
$$\frac{\partial \mathcal{E}}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L} \frac{\partial \mathbf{x}_L}{\partial \mathbf{x}_l} = \frac{\partial \mathcal{E}}{\partial \mathbf{x}_L} \left(1 + \frac{\partial}{\partial \mathbf{x}_l} \sum_{i=l}^{L-1} \mathcal{F}(\mathbf{x}_i,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

- Building residual basic block and bottleneck block:



- Using pretrained model and reinitialize the specific layers
https://pytorch.org/tutorials/beginner/finetuning_torchvision_models_tutorial.html
- Your visualization figure of accuracy should like example as below.

Result Comparison(ResNet18)



Confusion matrix

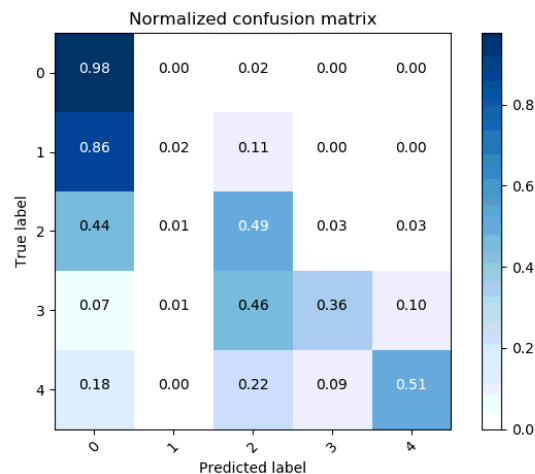
A confusion matrix is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known. It is extremely useful for measuring Recall, Precision, Specificity, Accuracy and most importantly AUC-ROC curve [3].

- **Sample code:**

You can use the following example and library to conduct this section.

https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py

- The confusion matrix will be similar as the following example:



Hyper Parameters

- Batch size= 4
- Learning rate = $1e-3$
- Epochs for ResNet-18 = 10
- Epochs for ResNet-50 = 5
- Optimizer: SGD
- Momentum = 0.9
- Weight Decay = $5e-4$
- Loss function: Cross Entropy Loss

You can adjust the hyper-parameters according to your own ideas.

Report Spec

1. Introduction (20%)
2. Experiment settings (20%)
 - A. The details of your model (ResNet)
 - B. The details of your DataLoader
 - C. Describing your evaluation through the confusion matrix
3. Data Preprocessing (20%)
 - A. How you preprocessed your data!
 - B. What makes your method special?
4. Experimental results (20%)
 - A. The final validation accuracy
 - Screenshot
 - Anything you want to present
 - B. Comparison figures
 - Plotting the comparison figures
 - (ResNet18/50, with/without pretraining)
5. Discussion (20%)
 - A. Anything you want to share

----- Criterion of result (30%) -----

- Accuracy $\geq 82\%$ = 100 pts
- Accuracy 80 - 82% = 90 pts
- Accuracy 75 - 80% = 80 pts
- Accuracy $< 75\%$ = 70 pts

Score: 30% experimental results + 70% (report + demo score)

P.S If the zip file name or the report spec have format error, it will be penalty (-5).

In demo phase, you need to load trained model and evaluate.

Reference:

[1] He, Kaiming, et al. "Deep residual learning for image recognition." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.

[2] Review: ResNet

<https://towardsdatascience.com/review-resnet-winner-of-ilsvrc-2015-image-classification-localization-detection-e39402bfa5d8>

[3] Understanding Confusion Matrix

<https://towardsdatascience.com/understanding-confusion-matrix-a9ad42dcfd62>

[4] Confusion Matrix

http://www2.cs.uregina.ca/~dbd/cs831/notes/confusion_matrix/confusion_matrix.html