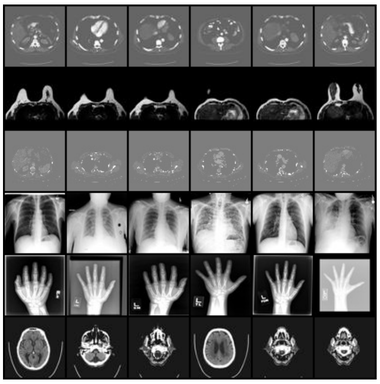
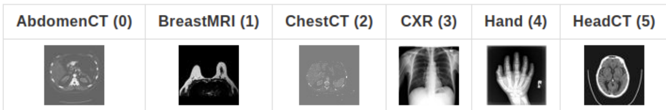
Deep Learning (Homework 1)

Due date : 2021/4/9 23:55:00 (Hard Deadline)

1 Feedforward Neural Network for Classification (60%)

You are given the dataset [1] of medical images (MedMNIST.zip). This dataset contains 6 classes. In this exercise, you need to implement a feedforward neural network (FNN) model by yourself to recognize radiological images, and use specified algorithms to update the parameters. Please use train.npz as training data and test.npz as test data.





Please follow the steps below to implement your program:

i. Understand how the "forward pass" and "backward pass" in FNN work in accordance with backpropagation algorithm.

ii. Please implement your code in one file. Your program needs to parse the neural network setting, initial weight and list of test images as follows: (The formats of three files are provided in the Appendix)

```
$ python3 <hw1_1_studentID>.py \
> --config <config>.json \
> --weight <weight>.npz \
> --imgfilelistname <imgfilelistname>.txt
```

iii. At the end of the process, the program needs to generate an "output.txt" file. The "output.txt" file is filled with the predicted class indices of test images. The format of output.txt file is provided in the Appendix.

iv. Both training and test images need to be normalized (divided by 255) as preprocessing.

v. Flatten images in row-major order.

vi. Use the cross entropy $J(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k=1}^{K} t_{nk} \log y_k(\mathbf{x}_n, \mathbf{w})$ as the objective function where t_{nk} is target value and $y_k(\mathbf{x}_n, \mathbf{w}_n)$ is the FNN output.

vii. DO NOT RESHUFFLE THE DATA AGAIN. We had already shuffled train.npz and test.npz. Reshuffle may let your "output.txt" file not match the model answer.

1. Design a FNN model architecture and perform the random initialization for model weights. Run backpropagation algorithm and use mini-batch SGD (stochastic gradient descent) $\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \frac{\eta}{N}\nabla J(\mathbf{w}^{(\tau)})$ to optimize the parameters, where η is the learning rate and N is the number of data in the batch:
   (a) **Plot** the learning curves of $J(\mathbf{w})$ and the accuracy of classification versus the number of iterations until convergence for training data as well as test data. (10%)
   (b) Repeat 1(a) by using different batch sizes. And **do some discussions**. (10%)
   (c) Repeat 1(a) by performing zero initialization for the model weights. And **do some discussions**. (10%)
2. Implement a flexible program that can parse the arguments to generate a specific FNN model but without bias for each neuron, and also need to run backpropagation algorithm and use mini-batch SGD to optimize the parameters:
   (a) TA will convert your submission <hw1_1_studentID>.ipynb to <hw1_1_studentID>.py. Then run <hw1_1_studentID>.py with 3 groups of arguments to check if the 3 "output.txt" files generated from your program match the model answer. (30%)

Hint: Be careful to assign the value to variable (mutable vs immutable object). Double check the dimensions of your matrices.

2 Convolutional Neural Network for Recognition (40%)

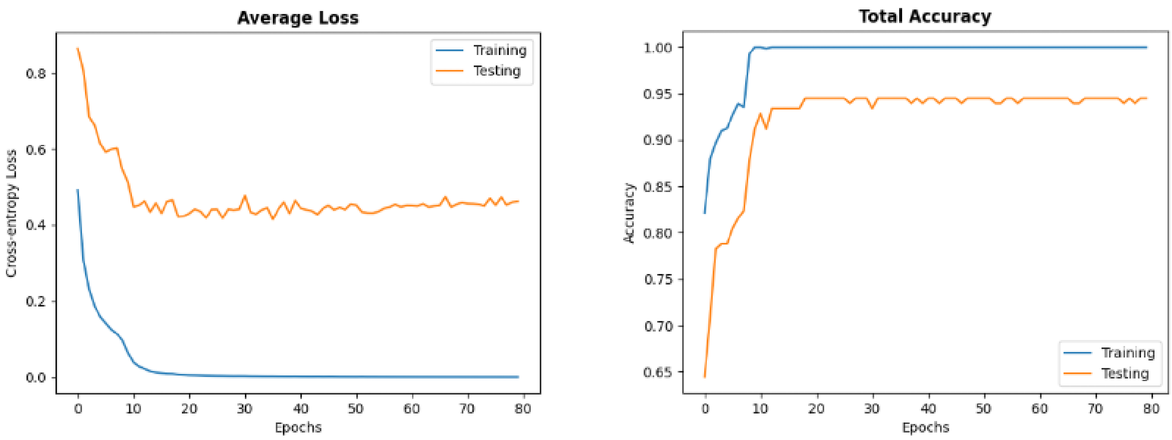
In this exercise, you will construct a convolutional neural network (CNN) for image recognition by using **Traffic Sign Recognition Dataset** (TSRD.zip). This dataset consists of 1770 traffic sign images from 7 categories. Each image is a zoomed view of a single traffic sign. Annotations provide image properties as well as traffic sign coordinates within image and category.

The files train.csv and test.csv contain the corresponding images and their **filename**, **width**, **height**, **bounding box** and **category**. The details of this dataset are shown below:

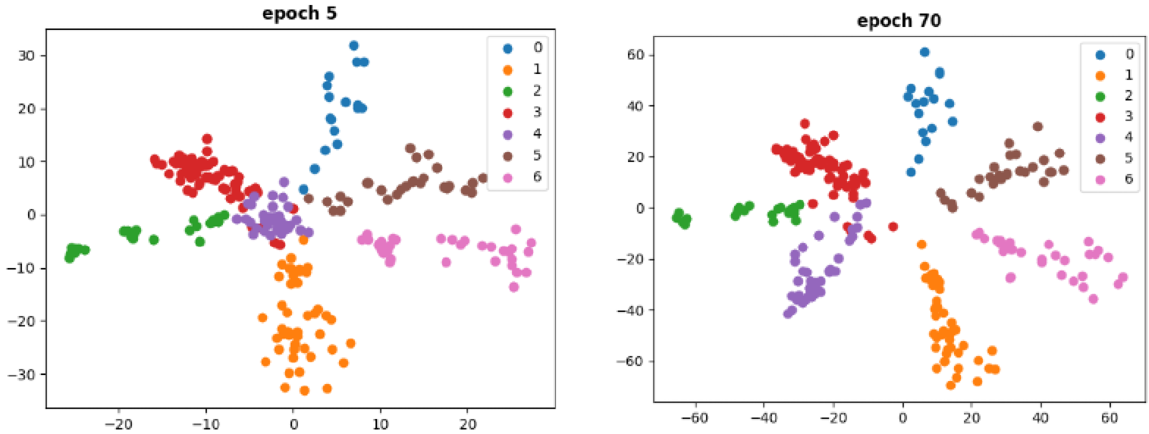
file_name	width	height	x1	y1	x2	y2	category
000_0001.png	134	128	19	7	120	117	0
001_0001.png	69	62	16	12	54	48	1
002_0001.png	79	82	16	18	67	69	2
003_0001.png	85	76	20	10	67	58	3
004_0001.png	107	101	17	11	91	87	4

1. Please implement a CNN for image recognition by using **Traffic Sign Recognition Dataset**. You need to design the network architecture with the layer of 2 nodes before the output layer.

   (a) **Plot** the learning curve and the accuracy rate of training and test data. (10%)



(b) **Plot** the latent feature distributions of test data at different training stages. For example, you may show the results when running at 5^{th} and 70^{th} learning epochs. (5%)



2. Please **preprocess** the images by cropping through the **bounding box**.

   (a) **Repeat** 1 and **Plot** the learning curve, accuracy rate and the latent feature distributions of test data. (20%)

   (b) Do some discussions on the results of the 1 and 2, please **describe** what you found. (5%)

3 Rule

- In your submission, you need to submit two files. And only the following file format is accepted:
  - **hw1_<ProblemNumber>_<StudentID>.ipynb** file which need to contain all the results, codes and reports for each exercise (e.g. **hw1_2_0123456.ipynb**).
- Implementation will be graded by
  - Completeness
  - Algorithm correctness
  - Description of model design
  - Discussion and analysis
- Only Python implementation is acceptable.
- For problem 1, any tools with automatic differentiation are forbidden, such as Tensorflow, PyTorch, Keras, etc. You should implement backpropagation algorithm by yourself.
- For problem 2, high-level API is forbidden, such as Keras, slim, TFLearn, etc. You should implement the forward computation by yourself. (Only this problem you can use PyTorch or Tensorflow).
- DO NOT PLAGIARISM. (We will check program similarity score.)

4 Appendix

1. <config>.json format

```
{
    "nn": {
        "layer1":{
            "input_dim": 1024,
            "output_dim": 2048
            "act": "relu"
        },
        "layer2":{
            "input_dim": 2048,
            "output_dim": 512
            "act": "relu"
        },
        "output":{
            "input_dim": 512,
            "output_dim": 6
            "act": "softmax"
        }
    },
    "epoch": 5,
    "lr": 0.001,
    "batch_size": 2048,
    "criterion": "cross_entropy"
}
```

2. <weight>.npz format
   Stored as a dictionary-like format. Take the above <config>.json as an example, the format of <weight>.npz is shown by:

```
{
    "layer1": numpy.ndarray
    "layer2": numpy.ndarray
    "output": numpy.ndarray
}
```

3. <imgfilelistname>.txt format
   There is an absolute path of an image be written at each line, and <imgfilelistname>.txt has the form of:

```
<absolute folder path>/004211.jpeg
<absolute folder path>/001531.jpeg
<absolute folder path>/002145.jpeg
<absolute folder path>/009839.jpeg
...
<absolute folder path>/005543.jpeg
```

4. <output>.txt format
   Directly store the the predicted class indices in one line, and <output>.txt has the form of:

```
021354213543213 21.....3
```

References

[1] apolanco3225, "Medical MNIST classification," https://github.com/apolanco3225/Medical-MNIST-Classification, 2017.