DRSN-1D

This note will show you my understanding of how the data is been processed and how the model should work based on the code. This project consists of 3 main components, and let me try to explain it as detail as i can.

1. data_process.py

In a nutshell, this code file will do the data pre-processing before it is fed to the model.

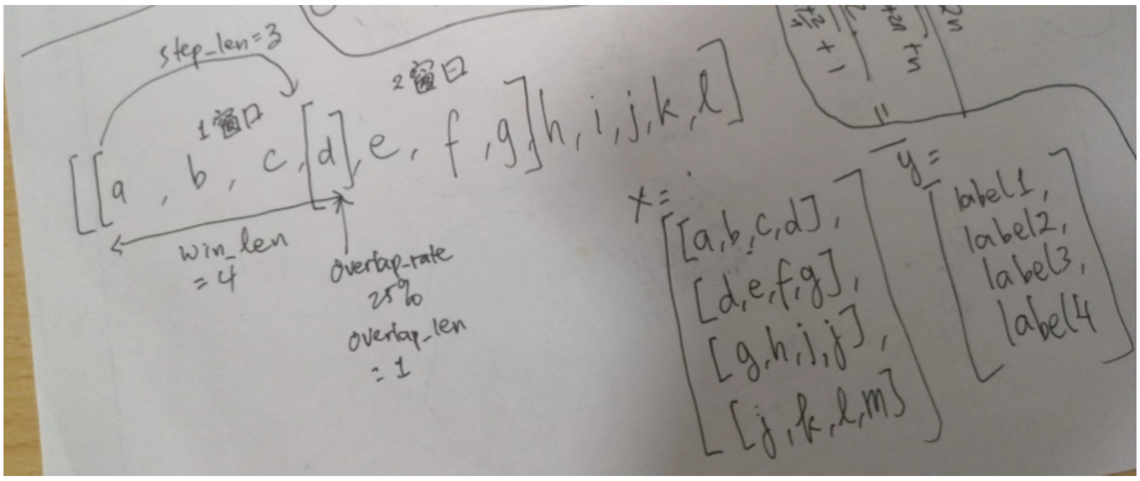
- `def iterator_raw_data(data_path)`
  - *data_path*: the path of our data (this case in .txt format). We use data in "DRSN-1D/test_data/data_snr-5"
  - return a (yield) *label, data*

  for all .txt data in that path, convert it into *ndarray* called **data** and paired with its corresponding label **label** (the label is determined by the filename i guess?).

- `def data_augment(fs, win_tlen, overlap_rate, data_iteror)`
  - *fs*: 原始数据的采样频率
  - *win_tlen*: 滑动窗口的时间长度
  - *overlap_rate*: 重叠部分比例
  - *data_iteror*: 原始数据的生成器格式
  - 返回 ndarrays *X* 数据 and *y* (标签)

  把1维时域信号切割城长度为`win_len = (win_tlen * fs)`的多个样本

  各个参数具体的含义以及滑动窗口采样过程见下图：

  
  step_len=3
2窗口
1窗口
[[a, b, c, [d], e, f, g] h, i, j, k, l]
win_len =4
overlap-rate 25%
overlap_len =1
x=
[[a,b,c,d], [d,e,f,g], [g,h,i,j], [j,k,l,m]]
y=
[label1, label2, label3, label4]

- `def preprocess(path, data_mark, fs, win_tlen, overlap_rate, random_seed, **kargs)`

  此函数将调用前2个函数，总结：
  - 从某个目录里读取相关的.txt文件，把里面的数据放在ndarray并写出数据相应的标签
  - 将1维ndarray进行滑动窗口采样切割成多个样本数据

2. rsnet.py

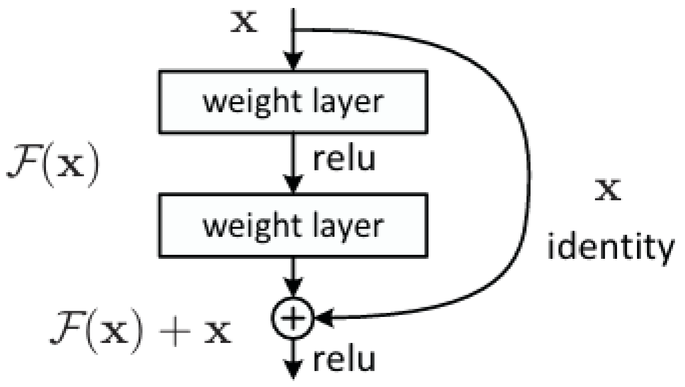
按照我对RSNet的理解来解释一下其概念：

此文件将设计项目里的深度残差收缩网络（Residual Shrinkage Network）模型。

简单看了 *Deep Residual Learning for Image Recognition* 这本论文，发现RSNet将解决的问题：Is learning better networks as easy as stacking more layers? The paper said, training and testing error become worse when we put more CNN layer on our model.

The idea: A deeper model should produce no higher training or testing error than the shallow one.

So they reformulate the layers as learning **residual functions** with reference to the input layer.

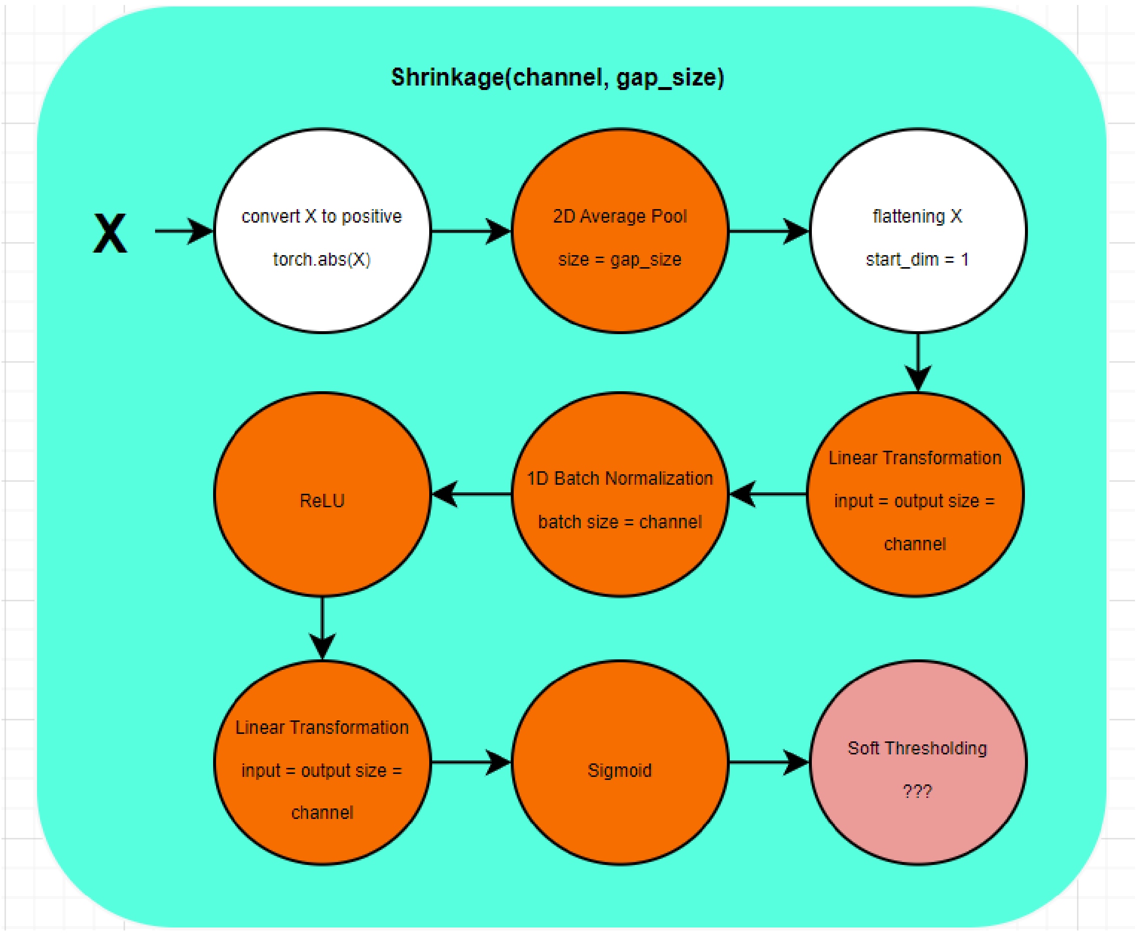


x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2. Residual learning: a building block.

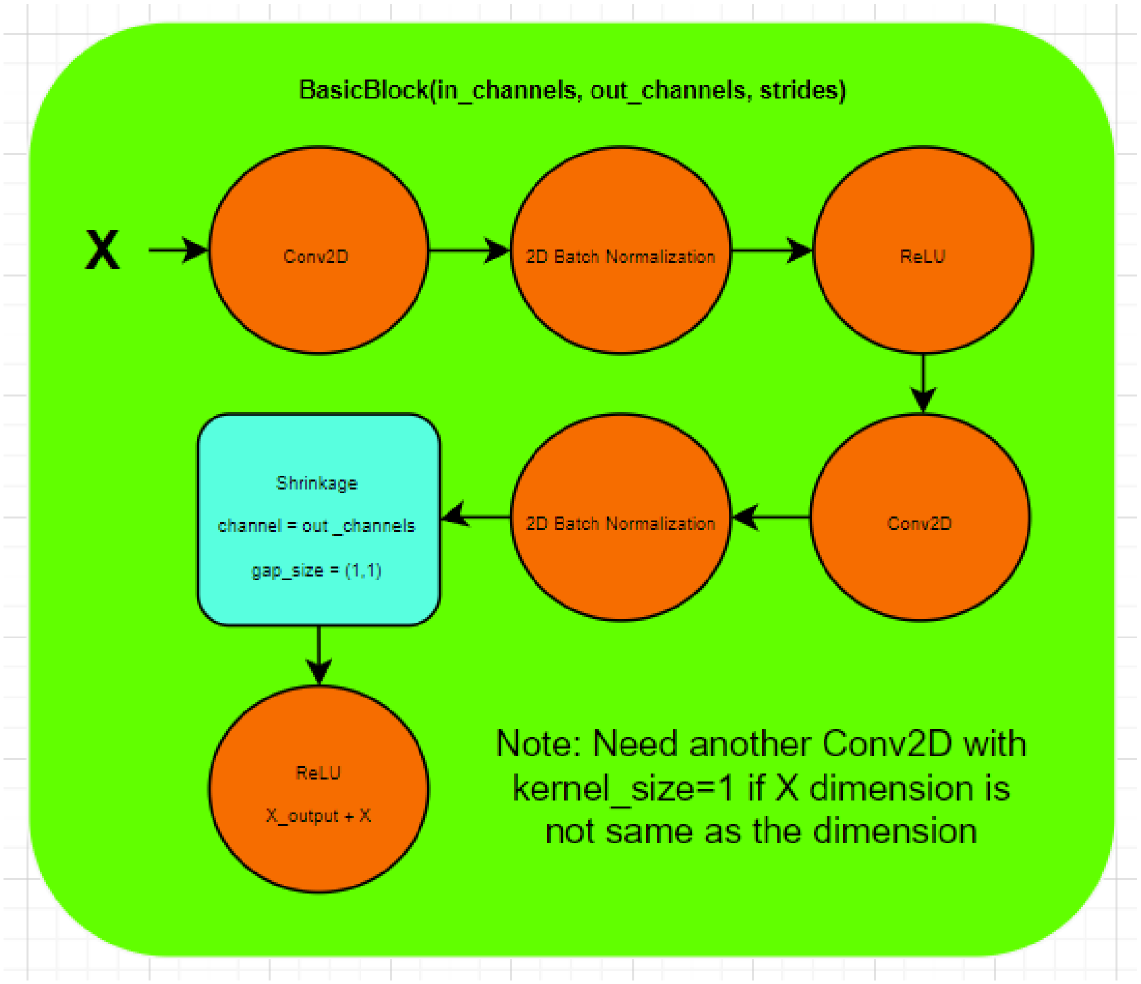
It creates some sort of skip relation among the layer. Suppose we want to squeeze the layer down to zero, and it will be much easier to do in this framework, rather than stacking the identity mapping.

- Shrinkage



更详细的解释见代码注释

- BasicBlock


BasicBlock(in_channels, out_channels, strides)
X
Conv2D
2D Batch Normalization
ReLU
Conv2D
2D Batch Normalization
Shrinkage
channel = out _channels
gap_size = (1,1)
ReLU
X_output + X
Note: Need another Conv2D with kernel_size=1 if X dimension is not same as the dimension

- RSNet

  Graph under construction...

3. train_1.py

代码里是训练及测试模型的整个过程。给学长们总结下：

- 数据采样参数：
  - 采样频率 12000
  - 滑动窗口的时间长度 2048 / 12000
  - 重叠百分比 0%
- 超参数：
  - batch size：16
  - 迭代次数epochs：10次
- 做个Dataset：
  - `preprocess()` 返回数据*train_data*及标签*train_label*
  - 把X与y从ndarray换成torch.Tensor
  - *train_data*与*train_label*放在一个TensorDataset，*train_dataset*

    - 利用`torch.utils.data.random_split()`把数据分成训练数据*train_dataset*（70%）及测试数据*test_dataset*（30%）
    - train_dataset与test_dataset各个传递给2个DataLoader，*train_data_loader*与*test_data_loader*，batch_size为16，每次迭代会对数据进行shuffle，忽略最后非灌满的batch。
- 模型定义：`net = rsnet34()`, 把模型放在CPU。
- use Stochastic Gradient Descent for optimizer algorithm
    - *learning rate: hyperparam, default = 1e-3, adjust training speed(lr * step_size). too large cause instability (converge too fast), too small cause learning process too slow*
    - *momentum: hyperparam, change (increase) lr in each epochs when the error cost gradient is heading in the same direction for a long time and avoid local minima by rolling over small bumps*
    - *weight_decay: hyperparam, change (reduce) lr in each epochs when a too high lr makes the learning jump back and forth over a minimum penalty*
- use Cross Entropy Loss for loss function (we'll explore more later)
    - Commonly used for measuring *classification model whose output is in range (0,1)*
    - Formula:

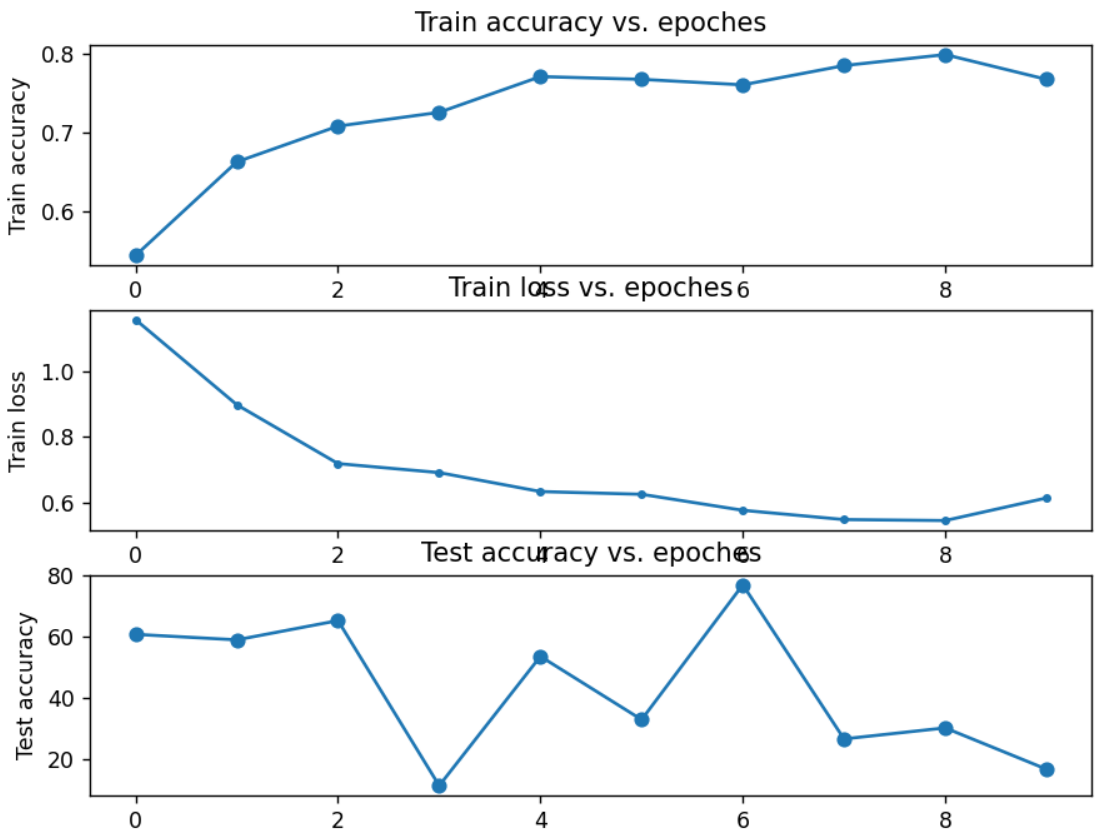
$$H(p,q) = -\sum_{x\in\mathcal{X}} p(x)\,\log q(x)$$

- 总共迭代10次，每次迭代：
    - 进行训练
        - `net.train()` Activate train mode
        - X数据类型设置为torch.float64, y数据类型设置为torch.int32
        - `optimizer.zero_grad()` Reset the gradients of all optimized tensors
        - Make a prediction!!! `outputs = net(X)` calls the forward method inside the model
        - `loss.backward()` Calculate the loss
        - `optimizer.step()` Update all parameters in the model (weight and bias)
        - Keep track of total loss and average accuracy in each epoch
    - 进行测试
        - `net.eval()` Deactivate train mode
        - `torch.no_grad()` won't keep track the gradient
        - Make a prediction!!! `outputs = net(X)` calls the forward method inside the model
        - From the prediction, choose the highest probability. That should be our answer
        - Calculate how many correct predictions we made, keep track for data visualization
    - 记录所有loss与accuracy

- 利用matplotlib进行数据可视化
    - 显出Train accuracy vs. epoches
    - 显出Train loss vs. epoches

- 显出Test accuracy vs. epoches

4. 运行代码结果



我的结论是我训练过的模型可能有点overfitting，因为训练精确度越来越高，但是测试精确度很差，通过调整超参数才能提高模型的精确度。