上海交通大學
SHANGHAI JIAO TONG UNIVERSITY

手写计算器
Mnist Calculator

自定义张量计算
Custom Tensor Op

基于 Qlib 的案例创新
Qlib CNN

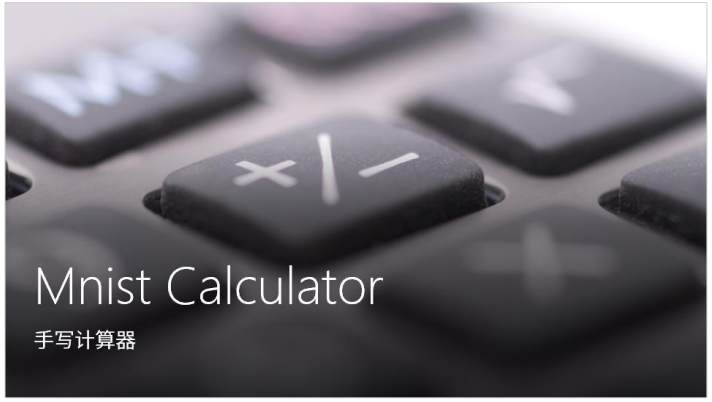
项目汇报

2021年秋季微软实践空间站

李子龙

LogCreative 

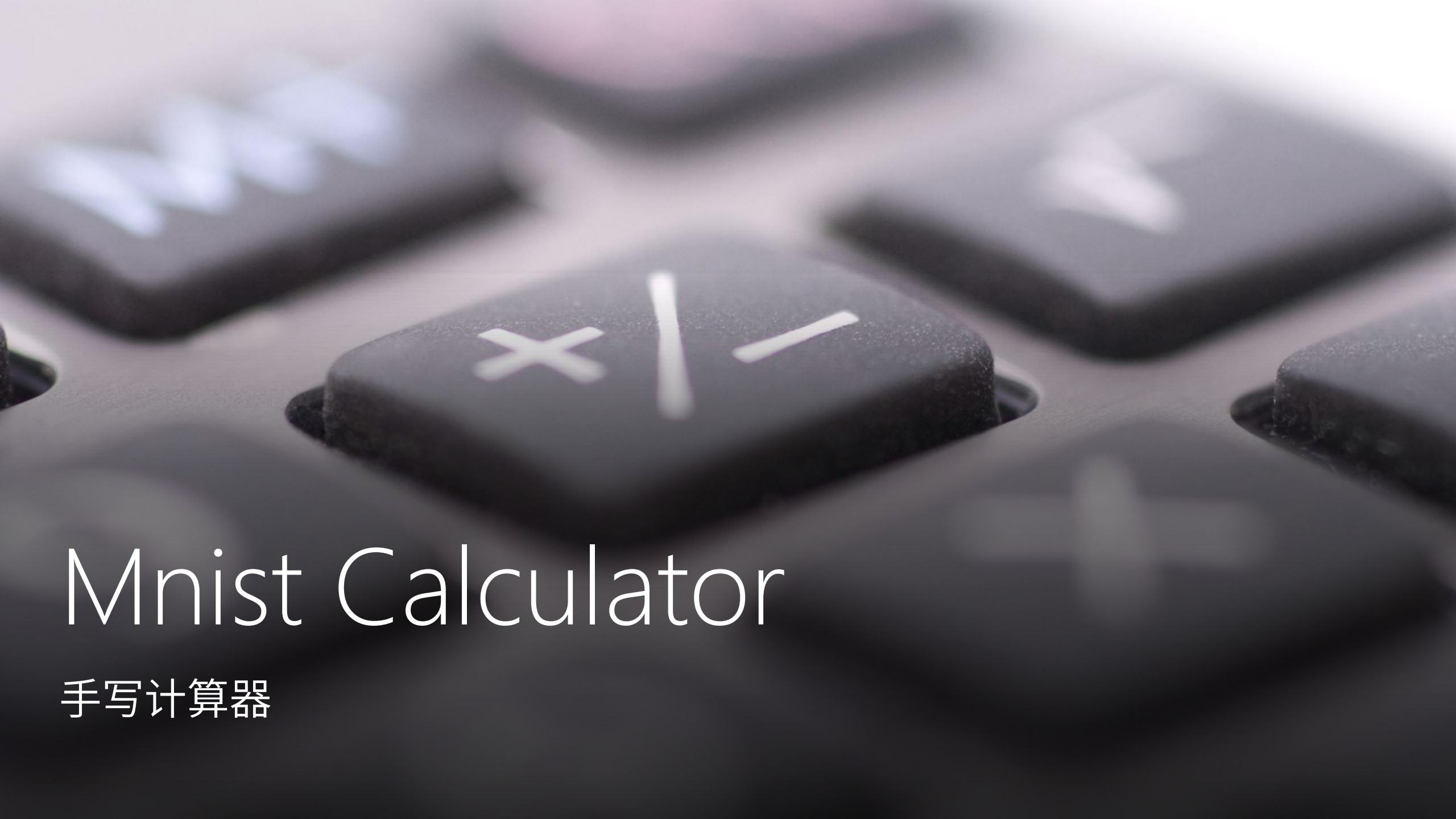
目录



LogCreative/mnist-calculator

LogCreative/custom-tensor-op

LogCreative/qlib-CNN



Mnist Calculator

手写计算器

手写计算器

任务列表

- [x] 使用 C# 复现两个项目
- [x] 使用 Python 基于桌面 .NET Framework 环境实现“手写数字识别应用”
- [x] 使用 Python 实现“识别手写数学表达式并对其进行计算”

编译环境

Visual Studio 2017

python™ 3.6.6

TensorFlow™ 1.5.0

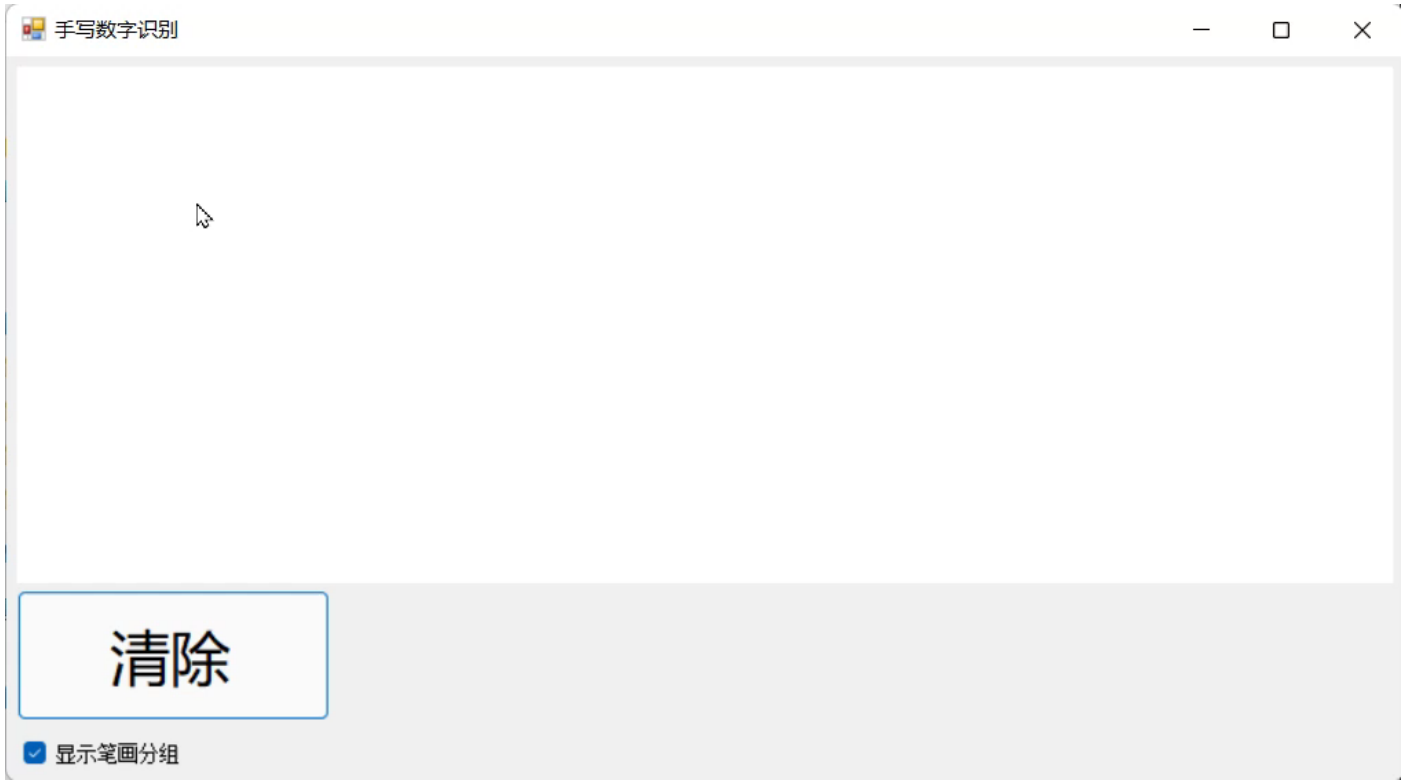
▲B07 手写数字识别复现

依赖

AI Tools for VS 需要在 Visual Studio 2017 中安装 Python 3.6.6 工作负荷后，手动安装插件。

TensorFlow 环境按照 samples-for-ai 进行配置。

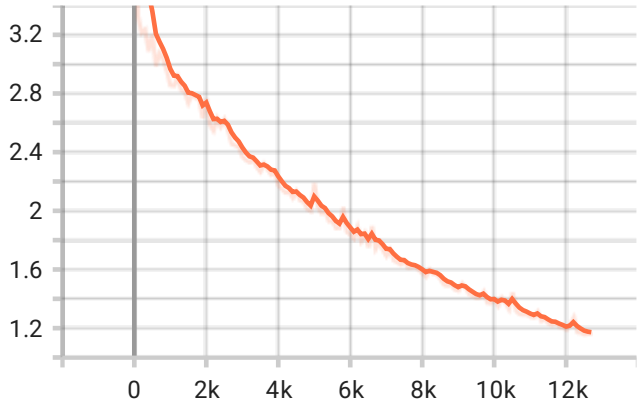
符号拓展数据集的数据源使用了开源数据（周雄，钟宏宇，杨帆，吴沛刚，2018）。



▲B09 识别手写数学表达式并对其进行计算复现

拓展数据集训练指标

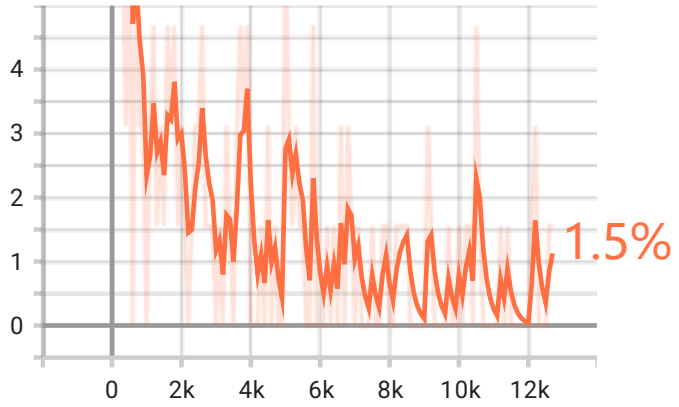
在该数据集上使用 TensorBoard 可视化训练指标结果如下：



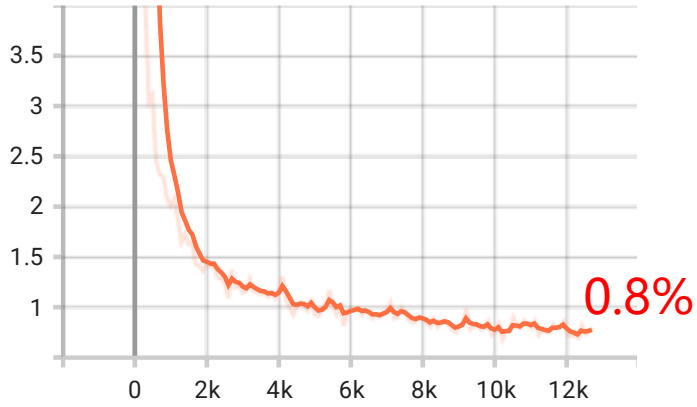
$$\texttt{loss} = exploss + \lambda \times L_2 loss$$

$$exploss = logits + cross\text{-}entropy$$

$$\lambda = 5 \times 10^{-4}$$



Minibatch Error Rate (%)



Validation Error Rate (%)

Test Error Rate	0.6%

Python.NET

基于 Python 的 .NET Framework 一般有两种选择。

Python.NET	IronPython
✓ 对 Python 3 支持较好。	✗ 支持有限，并不能完整支持 Python 3。
✓ 直接在 Python 中调用 .NET 运行时。	! 调用需要通过 C# 包装。
	! Visual Studio 已经不再内置该工作负荷。

迁移问题

选定框架后，仍然需要解决许多兼容问题，因为 Python 和 C# 语法在某些方面并不是完全等价的。

✓ 导入 .NET 运行时

✓ 使用指定的动态链接库 (DLL)

✓ 绕过 Python.NET 在 IEnumerator<T> 上的 bug*

```
inferResult.First().First()
```

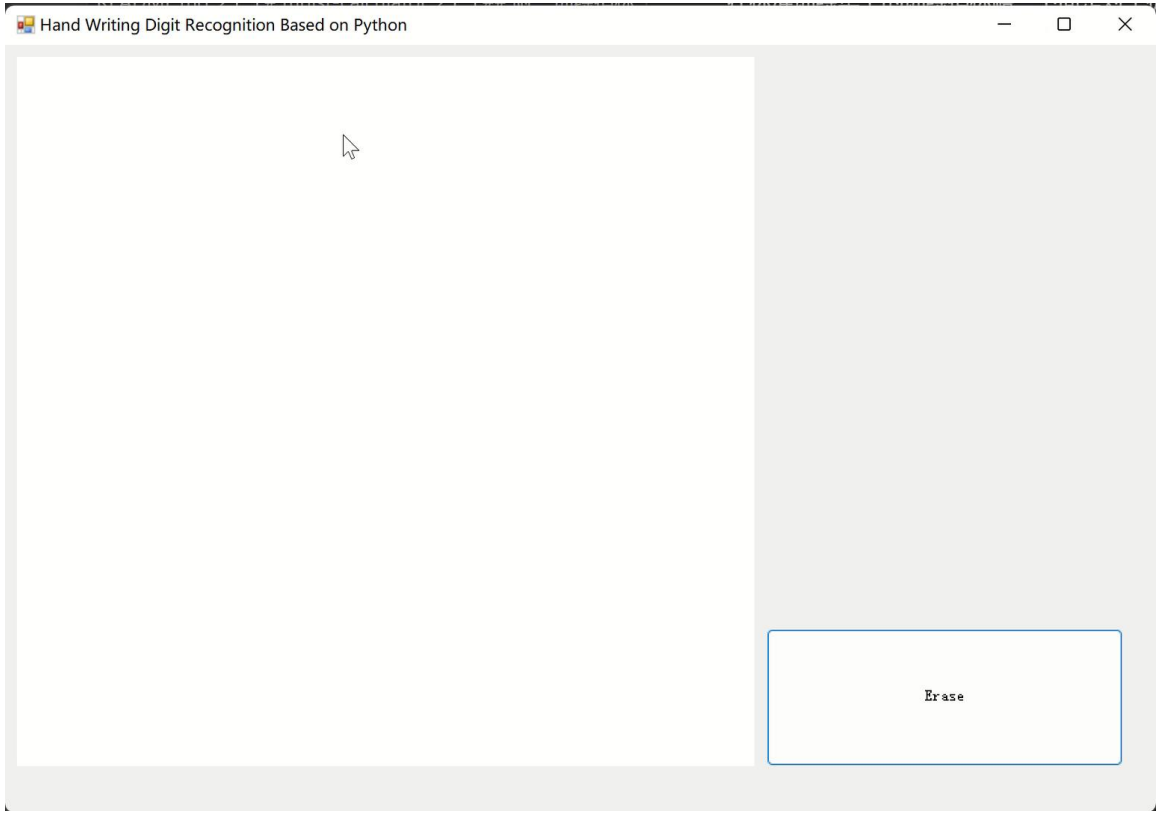
```
# Get the first of the first in IEnumerator<IEnumerator<T>>
enumer = inferResult.GetEnumerator()
enumer.MoveNext()
enumer = enumer.Current.GetEnumerator()
enumer.MoveNext()
self.outputText.Text = Convert.ToString(enumer.Current)
```

* 应当视作是 Python.NET 在 Collections 上的实现问题。

实现效果

这个 bug 大概是该挑战最困难的一点。

虽然在 Python 中直接调用 .NET 运行时仍然有一些 bug，也是思考了很久，但是实现出来后展现了 AI Tools for VS 的结果也可以通过这种方法在 Python 中得到调用，展示了该插件一定的应用潜力。



▲B07 基于 Python.NET 的实现

tkinter

本任务要求完全适用 Python 实现手写计算器计算功能。

这里使用了轻量级 GUI 框架 tkinter，类似于 C++ 中的 FLTK (Fast Light Toolkit)，可以方便地在原生语言中完成简单的界面功能。

当然也存在架构迁移的风险，有些功能可能需要换种方法解决，甚至是重新造轮子。

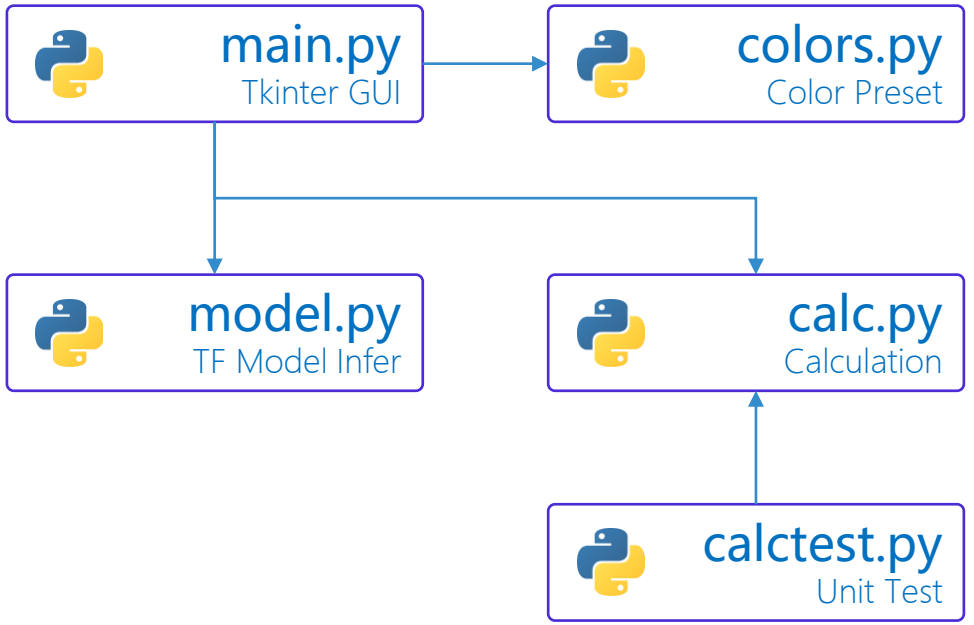
▲B09 基于 Python 的实现

实现架构

- AI Tools for VS 载入为 C# 文件
- 需要使用 Python 通用方法载入模型文件

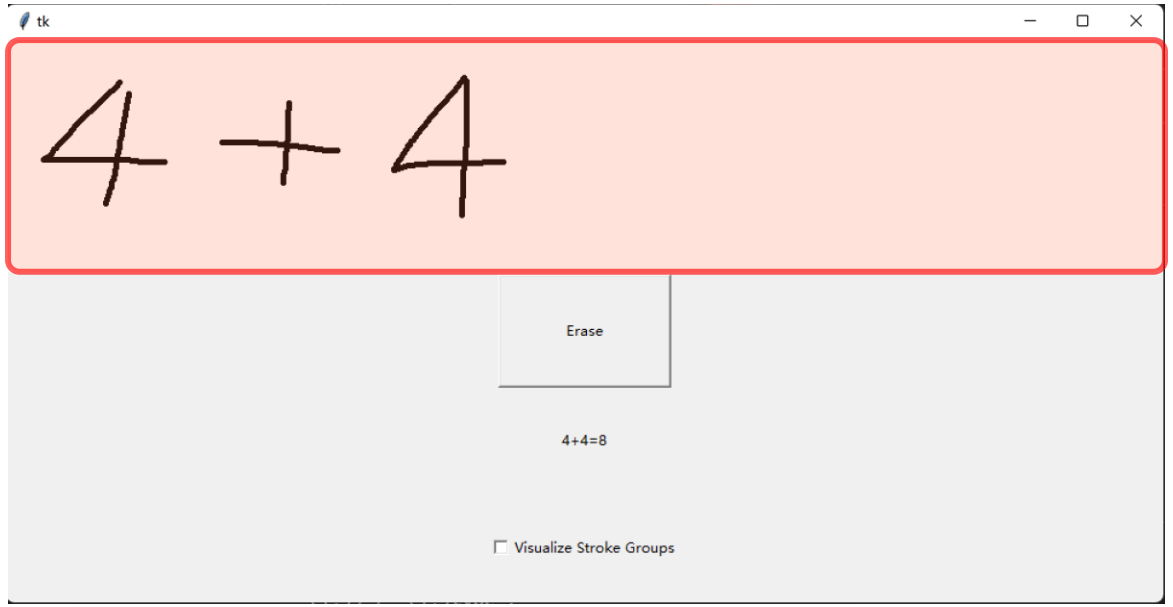
- 使用 .NET 内置的 DataTable 运算表达式
- 需要使用 Python 实现表达式计算算法

- 添加单元测试以避免各种边界情况

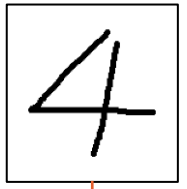


手写计算器 · 任务三

推理细节



main.py
Tkinter GUI



识别笔画分组
在内存的虚拟图中重绘
在 28x28 的画布中居中

model.py
TF Model Infer

反色输入模型
推理结果

calc.py
Calculation

4+4=8 计算表达式

手写计算器

总结

Windows 7 推出了数学输入面板实用程序，十多年后的今天，通过这些任务，已经可以通过机器学习方法和几百行 Python 代码就实现它的一些基础功能，从中便可窥见机器学习让一些传统问题有了全新的思路。

从这个过程中也体会到，在模型落地的过程中，很多的时间，并不是在机器学习的核心模型代码上下的功夫，而是花在如何将用户的输入转化为被模型所识别输入上。后者也是很重要的，因为现实有很多不可控因素，可能就没有理论那么理想，但也会直接影响推理结果的好坏。

▲墨迹公式

Custom Tensor Op

自定义张量计算

任务列表

- [x] 实现线性层的自定义张量计算
- [x] 基于 Python API 实现卷积层的自定义张量计算
- [x] 基于 C++ API 实现卷积层的自定义张量计算
- [x] 使用 Profiler 比较网络性能

卷积层

MNIST 图像识别的 LeNet 实现网络如右。

- [x] 实现线性层的自定义张量计算

  参考代码给出了对应的例子。

- [ ] 实现卷积层的自定义张量计算

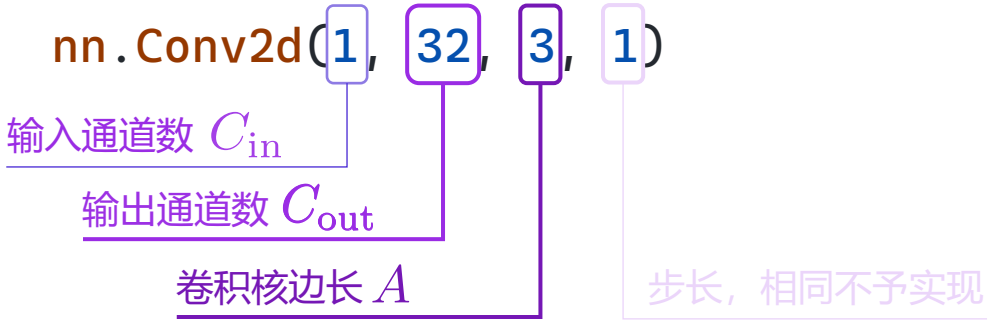
  主要尝试实现这一部分，替换原有的实现。

网络结构▼

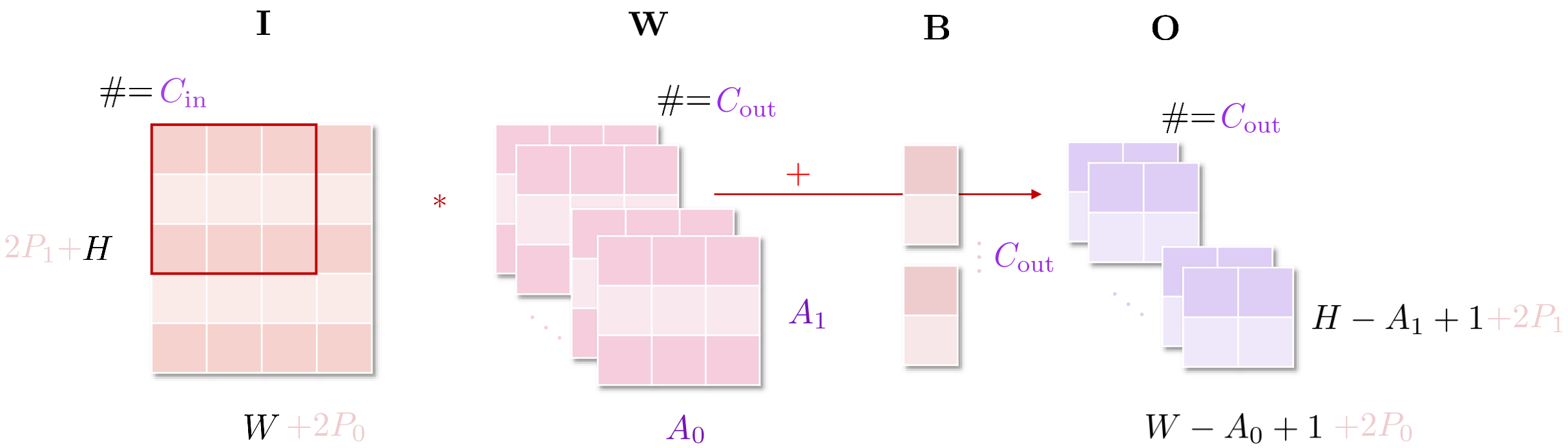
```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, 3, 1)
        self.conv2 = nn.Conv2d(32, 64, 3, 1)
        self.dropout1 = nn.Dropout2d(0.25)
        self.dropout2 = nn.Dropout2d(0.5)
        self.fc1 = nn.Linear(9216, 128)
        self.fc2 = nn.Linear(128, 10)

    def forward(self, x):
        x = self.conv1(x)
        x = F.relu(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = F.max_pool2d(x, 2)
        x = self.dropout1(x)
        x = torch.flatten(x, 1)
        x = self.fc1(x)
        x = F.relu(x)
        x = self.dropout2(x)
        x = self.fc2(x)
        output = F.log_softmax(x, dim=1)
        return output
```

卷积层的前向和后向传播

简化起见，将会只实现这几个参数的自定义卷积层。



卷积层的前向和后向传播



▲卷积计算概要

卷积层的前向和后向传播

卷积运算前向公式

$$\mathbf{O}_{i,j} = \sum_{k=0}^{C_{\text{in}}-1} \mathbf{W}_{j,k} * \mathbf{I}_{i,k} + \mathbf{B}_j$$

```python
def conv2dbasis(input, kernal, padding=(0,0)):
    h,w = list(input.size())
    kh,kw = list(kernal.size())
    oh,ow = h-kh+2*padding[1]+1,w-kw+2*padding[0]+1
    output = torch.Tensor(oh,ow)
    input_ = F.pad(input,
             (padding[0],padding[0],padding[1],
             padding[1]), "constant", 0)
    for i in range(oh):
        for j in range(ow):
            output[i,j] =
                input_[i:i+kh,j:j+kw].mul(kernal).sum()
    return output # imm
```

▲CPU 版本卷积运算实现

✔ 实现了 CPU 版本的 `conv2dbasis(input, kernel, padding)`

Python 精度有限，会造成一定的误差。

☑ Pytorch 内置 `F.conv2d(input, weight, bias)`

适用于 GPU，并有一定的优化。

卷积层的前向和后向传播

卷积运算前向公式

$$\mathbf{O}_{i,j} = \sum_{k=0}^{C_{\text{in}}-1} \mathbf{W}_{j,k} * \mathbf{I}_{i,k} + \mathbf{B}_j$$

卷积运算后向公式

$$\delta_{i,k}^{\text{in}} = \mathbf{W}_{j,k}^{rot180} * \delta_{i,j}^{\text{out}}$$

$$\delta_{j,k}^{w} = \delta_{i,j}^{\text{out}} * \mathbf{I}_{i,k}$$

$$\delta_j^b = \sum_{i=0}^{N-1} \sum_{a_0=0}^{A_0-1} \sum_{a_1=0}^{A_1-1} \delta_{i,j}^{\text{out}}[a_0, a_1]$$

! GPU 实现只有 `F.conv2d` 无法传递下标

```
for i in range(batch_size):
   for j in range(out_channels):
      for k in range(in_channels):
         grad_input[i,k] += conv2dbasis(grad_output[i,j],
                  torch.Tensor.rot90(weight[j,k], 2),
                  padding=(kernel_width-1,kernel_height-1))
         grad_weight[j,k] += conv2dbasis(input[i,k],
                  grad_output[i,j])
      grad_bias[j] += grad_output[i,j].sum()
```

▲CPU版本反向传播实现

卷积层的前向和后向传播

卷积运算前向公式

$$\mathbf{O}_{i,j} = \sum_{k=0}^{C_{\text{in}}-1} \mathbf{W}_{j,k} * \mathbf{I}_{i,k} + \mathbf{B}_j$$

卷积运算后向公式

$$\delta_{i,k}^{\text{in}} = \mathbf{W}_{k,j}^{rot180} * \delta_{i,j}^{\text{out}}$$

$$\delta_{k,j}^{w} = \delta_{j,i}^{\text{out}} * \mathbf{I}_{k,i}$$

$$\delta_j^b = \sum_{i=0}^{N-1} \sum_{a_0=0}^{A_0-1} \sum_{a_1=0}^{A_1-1} \delta_{i,j}^{\text{out}}[a_0, a_1]$$

☑ Pytorch 内置实现 `F.conv2d`
交换下标的技巧

```
grad_input = F.conv2d(grad_output,
    torch.Tensor.rot90(weight,2,[2,3]).transpose(0,1)
    , padding=(kernel_width-1,kernel_height-1))
grad_weight = F.conv2d(input.transpose(0,1),
    grad_output.transpose(0,1)).transpose(0,1)
    grad_bias = grad_output.sum([0,2,3])
```

▲GPU版本反向传播实现

语言迁移

仿照例子，可以很方便地写出对应的 C++ 版本实现。

❗ 可选参数在 Pytorch C++ API 中被实现为对应的 Options 结构。

```python
grad_input = F.conv2d(grad_output,
    torch.Tensor.rot90(weight,2,[2,3]).transpose(0,1)
    , padding=(kernel_width-1,kernel_height-1))
grad_weight = F.conv2d(input.transpose(0,1),
    grad_output.transpose(0,1)).transpose(0,1)
    grad_bias = grad_output.sum([0,2,3])
```

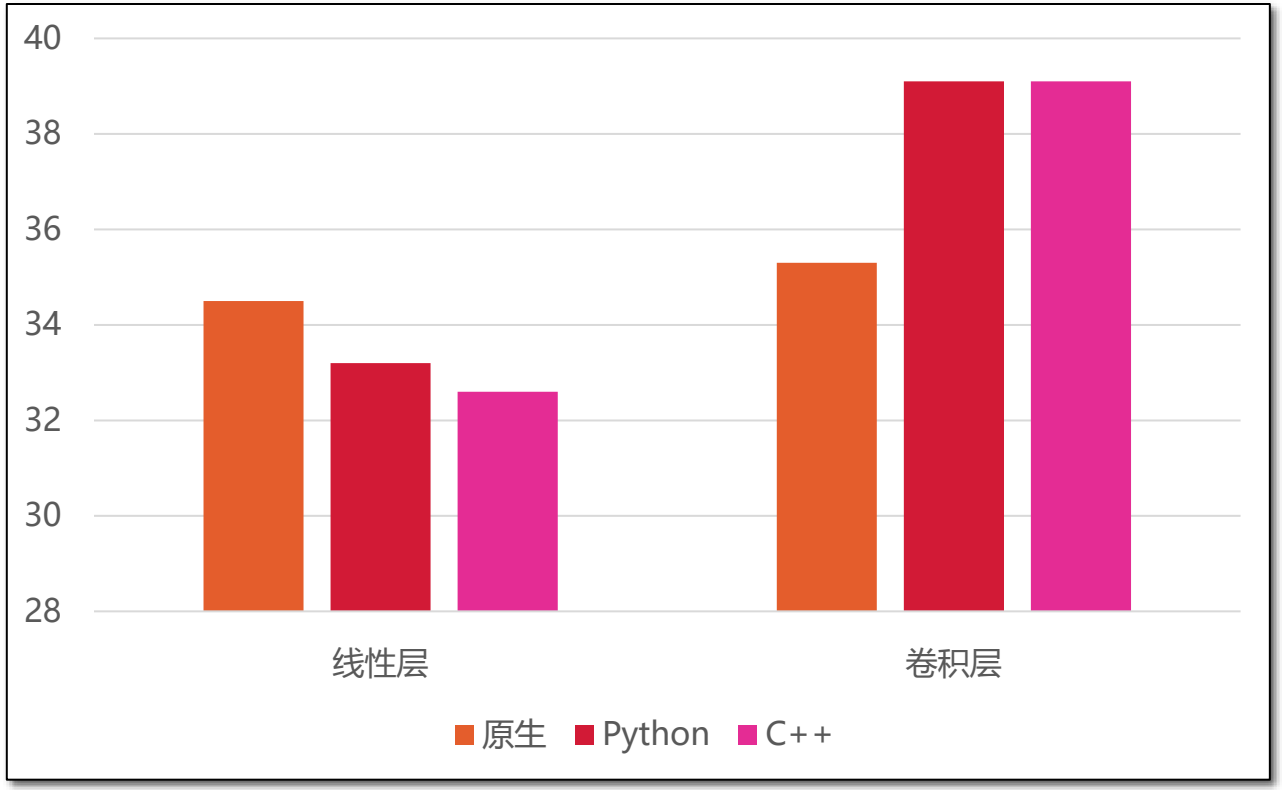
▲GPU版本反向传播实现(Python API)

```cpp
std::vector<torch::Tensor> myconv2d_backward(
   torch::Tensor grad_output,
   torch::Tensor input,
   torch::Tensor weight,
   torch::Tensor bias
) {
   auto kernal_height = weight.size(2);
   auto kernal_width = weight.size(3);
   auto grad_input = F::conv2d(
      grad_output,
      weight.rot90(2, { (2),(3) }).transpose(0, 1),
      F::Conv2dFuncOptions().padding({ kernal_width
- 1, kernal_height - 1 })
   );
   auto grad_weight = F::conv2d(
      input.transpose(0, 1),
      grad_output.transpose(0, 1)
   ).transpose(0, 1);
   auto grad_bias = grad_output.sum({ 0,2,3 });
   return { grad_input, grad_weight, grad_bias };
}
```

(C++ API) ▲

运行时间

- ⏫ 线性层时间有一定提升。
- ⏬ 卷积层自定义实现有性能下降。



▲Profiler 得到的 CUDA 时间比较（单位：秒）

* Profiler 粗略时间测量，更加精细的测量在后面补充设计了另一个实验。

分阶段测量

设计纯净测试，测量前向和后向时间，分别在 CPU 和 GPU 进行5000次测试取平均（单位：秒）。

〰 前向上基本一致，因为只是都是调用自身内置的卷积操作。

➰ 加上后向上会有差距，说明主要问题在于卷积层后向的实现细节上。Forward+Backward 在 CPU 上慢了 33%，在 CUDA 上慢了 141%。

CPU 测试结果	Forward	Forward+Backward
native	0.006395	0.014766
pyver	0.006491	0.019503
cppver	0.006585	0.019705

CUDA 测试结果	Forward	Forward+Backward
native	0.001082	0.001365
pyver	0.001043	0.003303
cppver	0.001108	0.003151

▲对于卷积层的纯净测试结果

调用次数

调查 Profiler 函数调用。

发现原生使用 CudnnConvolutionBackward0 实现反向传播，内置函数直接执行了与前向传播基本一致的时间。

而 Python API 实现导致卷积函数调用次数变为原来的3倍。

▼ 原生 Profiler 运行情况

#	Name	CUDA time total	CUDA time avg	# of Calls
5	aten::conv2d	3.858s	2.035ms	1896
6	aten::convolution	3.832s	2.021ms	1896
7	aten::_convolution	3.802s	2.005ms	1896
14	CudnnConvolutionBackward0	3.015s	1.607ms	1896

#	Name	CUDA time total	CUDA time avg	# of Calls
5	aten::conv2d	8.579s	1.519ms	5648
6	aten::convolution	8.518s	1.508ms	5648
7	aten::_convolution	8.397s	1.487ms	5648
9	myConv2dFunctionBackward	5.411s	2.884ms	1876

▲Python API Profiler 运行情况

内部优化

这也就意味着自动图运算生成的反向传播算法可以减少卷积顶层调用的次数，类似于编译优化，虽然手动编写反向传播是可行的，但是对于专用运算器使用专用的方法进行优化会更好。

猜测有 GPU 在数据并行性上可以接受更大量的数据同时计算，将输入矩阵拼接起来可以一次等价于使用两次卷积运算，减少了时钟周期数的占用，从而节约了时间。

事实上 CPU 的时间也是可以被优化的，使用 TVM 对循环结构进行调整，减少数据依赖性，提高硬件利用率，提高卷积操作 60% 左右的效率。细节详见个人之前的大作业。

LogCreative/CompilerPrinciple
使用 TVM 对卷积运算进行优化▶

优化结果

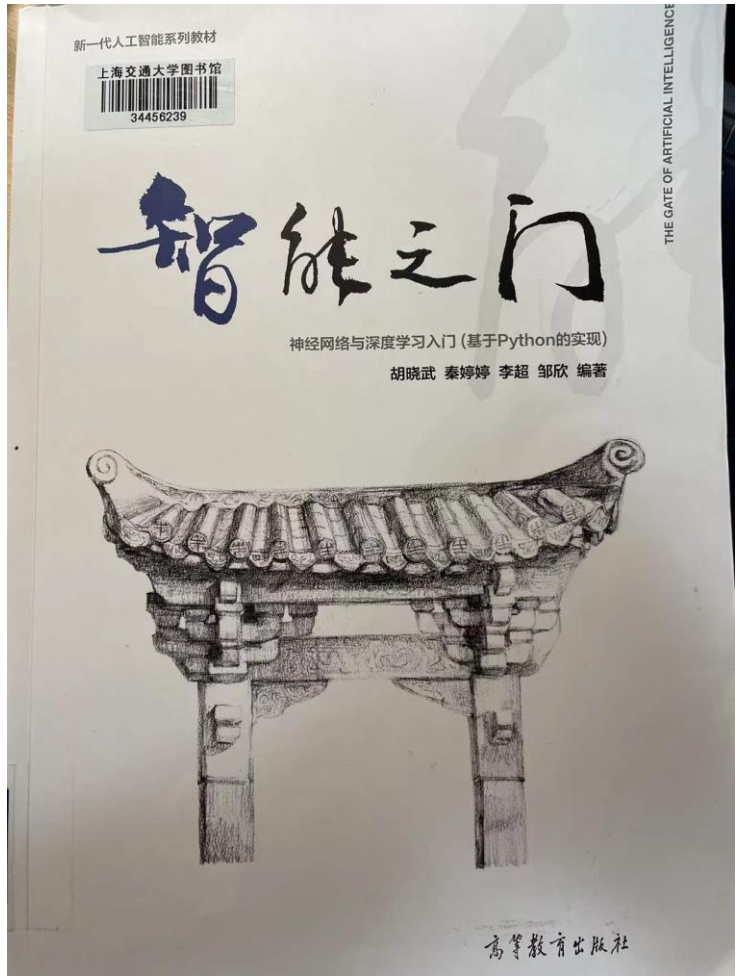
输入大小和输出大小	优化前	优化后	提升效率
n, ic, ih, iw = 1, 3, 32, 32 oc, kh, kw = 32, 3, 3	0.165312 ms	0.057781 ms	65.0%
n, ic, ih, iw = 100, 512, 32, 32 oc, kh, kw = 1024, 3, 3	493806.119851 ms	160967.332992 ms	67.4%

自定义张量运算

总结

这个过程展现出如果想要实现自定义卷积层运算，就需要从底层知识出发。这种底层知识在我这学期学的人工智能课上是没有的，后来也是看到了 ai-edu 的相关章节，然后从图书馆借了本《智能之门》把这一部分详细地看了一遍，才对这些实现有了些思路。对于我学习自己的课程也有了更深的理解。

虽然最后卷积层的性能实现没有太好的提升，但是学到这些知识让我对实现它的每个细节都有了了解，解释一些现象才会有些思路。



▲ai-edu 配套教材《智能之门》

Qlib CNN

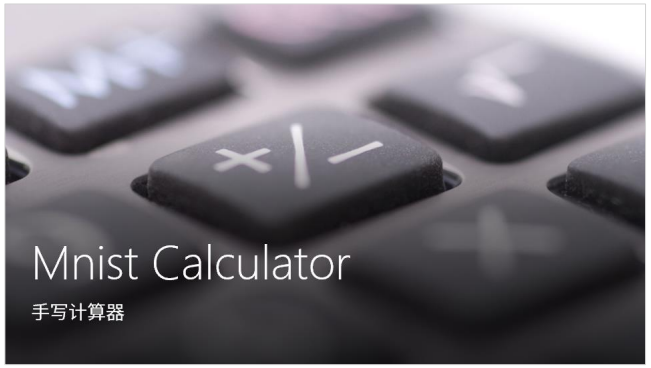
基于 Qlib 的案例创新

任务列表

- [x] 跑通程序
- [x] 实现基础版本的 CNN
- [x] 使用 Qlib 体验量化交易流程

卷积网络

由于另外两个项目都分别复现了基础版本 CNN 实现 MNIST 手写数字识别，在此就从略。作为补充，尝试从二维卷积网络出发，迁移实现 Qlib 模型。



TensorFlow™

PyTorch

二维卷积网络

基础版本的二维卷积网络结构如右。

卷积层

池化层

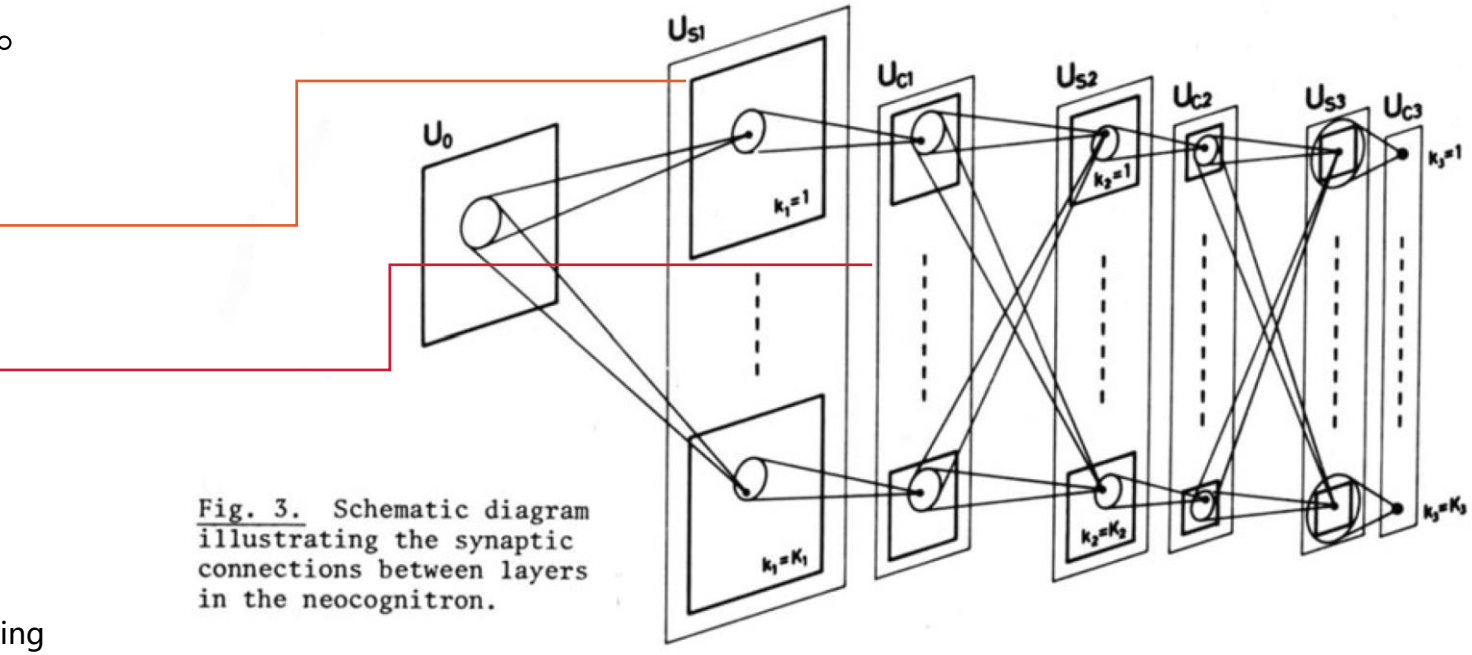
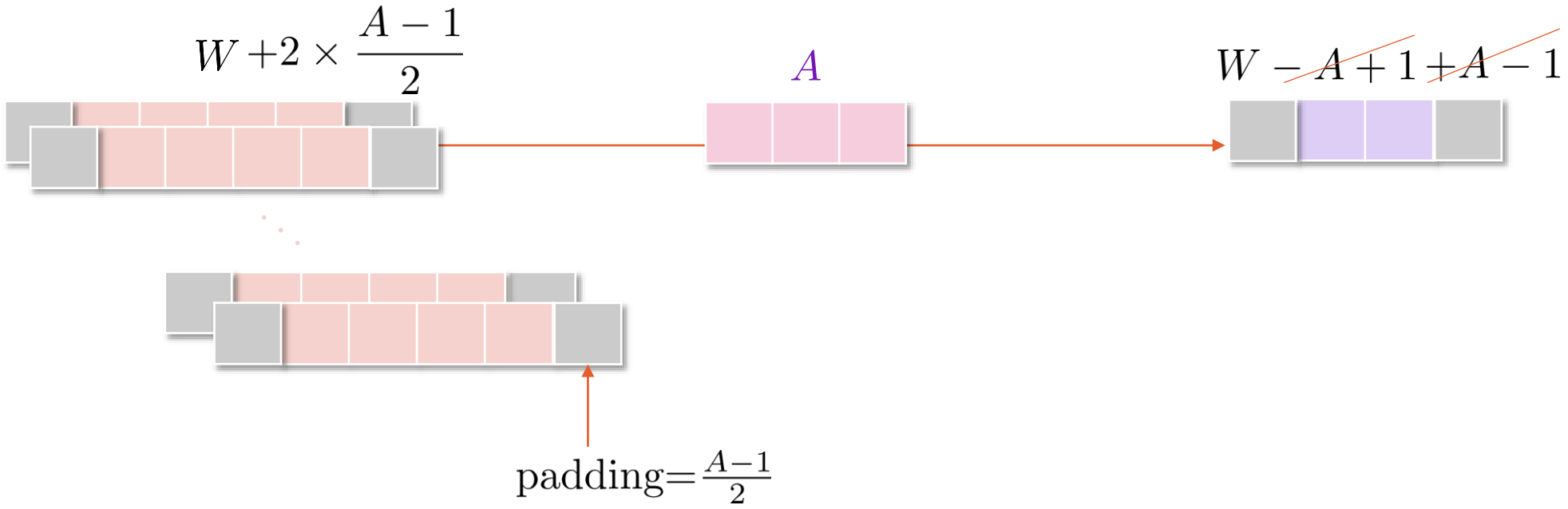


Fig. 3. Schematic diagram illustrating the synaptic connections between layers in the neocognitron.

* Fukushima, K. (1980). Neocognitron: A self organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biological Cybernetics, 36(4), 193-202.

一维卷积网络

尝试迁移应用到一维卷积网络，但是我们不希望更改数据横向维度，受到后文 TCN 的启发，就添加 padding 填充。池化层会影响维度，所以也不会包含池化层。



Qlib 框架下实现

这种方法是能跑通的，但是我这里的实现仍然没有太好的效果，还是 TCN 更有可解释性一些。

```
[751:MainThread](2022-01-22 22:38:50,288) INFO - qlib.workflow - [record_temp.py:496] -
as the artifact of the Experiment 1
'The following are analysis results of benchmark return(1day).'
                       risk
mean               0.000477
std                0.012295
annualized_return  0.113561
information_ratio  0.598699
max_drawdown      -0.370479
'The following are analysis results of the excess return without cost(1day).'
                       risk
mean              -0.000271
std                0.003295
annualized_return -0.064382
information_ratio -1.266677
max_drawdown      -0.256159
'The following are analysis results of the excess return with cost(1day).'
                       risk
mean              -0.000305
std                0.003295
annualized_return -0.072584
information_ratio -1.427900
max_drawdown      -0.284228
[751:MainThread](2022-01-22 22:38:50,312) INFO - qlib.workflow - [record_temp.py:521] -
aved as the artifact of the Experiment 1
'The following are analysis results of indicators(1day).'
     value
ffr    1.0
pa     0.0
pos    0.0
```

基础版本的 CNN 实现▼

```python
class Net(nn.Module):
    def __init__(self, input_dim, output_dim, kernel_size,
layers=(256,64,16)):
        super(Net, self).__init__()
        layers = [input_dim] + list(layers)
        cnn_layers = []
        for i, (input_channel, output_channel) in enumerate(zip(layers[:-
1],layers[1:])):
            conv = nn.Conv1d(input_channel, output_channel, kernel_size, 1,
int((kernel_size-1)/2))
            activation = nn.ReLU(inplace=False)
            conv.weight.data.normal_(0, 0.01)  # init weight
            seq = nn.Sequential(conv, activation)
            cnn_layers.append(seq)
        drop_input = nn.Dropout(0.05)
        cnn_layers.append(drop_input)
        self.cnn_layers = nn.ModuleList(cnn_layers)
        self.linear = nn.Linear(layers[-1],output_dim)

    def forward(self, x):
        cur_output = x.transpose(0,1).unsqueeze(0)
        for i, now_layer in enumerate(self.cnn_layers):
            cur_output = now_layer(cur_output)
        cur_output = self.linear(cur_output[:,:,-1])
        cur_output = cur_output.squeeze().unsqueeze(0)
        return cur_output
```

时序卷积网络

Temporal Convolution Network

时序卷积网络（TCN）采用了编码—解码结构，用来匹配池化操作，并精炼语义。

Qlib 官方库在 2021 年 11 月加入了对 TCN 的支持，实现方式是后来一篇论文的方法，它不减少时序数据维度，与前面的一维卷积网络关系更为密切。

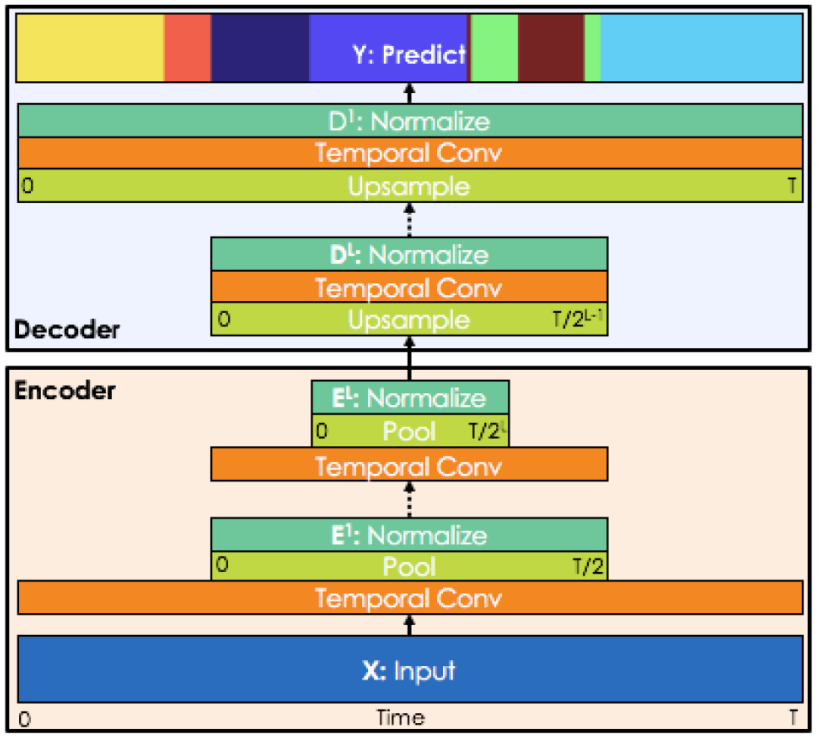
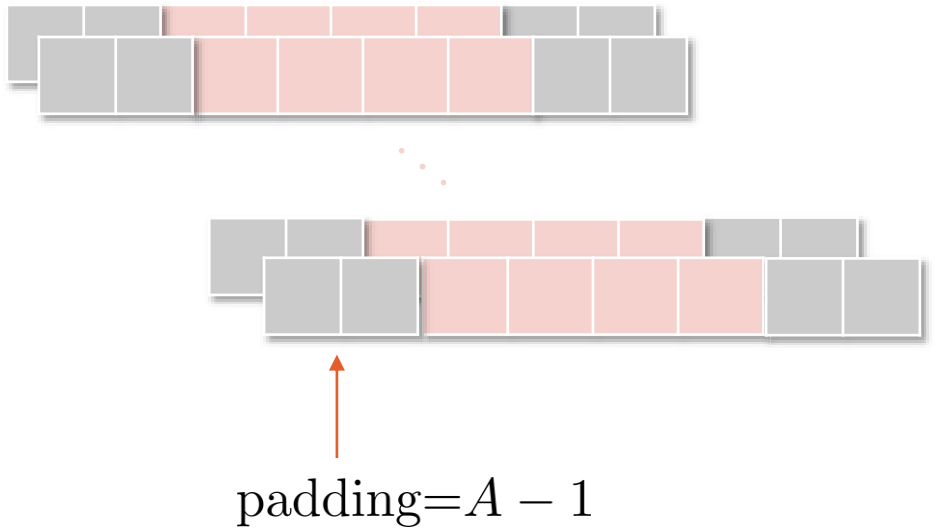


Figure 1. Our temporal encoder-decoder network hierarchically models actions from video or other time-series data.

* Lea, C., Vidal, R., Reiter, A., & Hager, G. (2016). Temporal Convolutional Networks: A Unified Approach to Action Segmentation. In Computer Vision – ECCV 2016 Workshops (Lecture Notes in Computer Science, pp. 47-54). Cham: Springer International Publishing.

TCN 结构

官方库的实现结构如图所示。考虑到只能向前看，不能向后看，会使用 Chomp1d 直接将后面多余的部分删除。




```
TCNModel(
  (tcn): TemporalConvNet(
    (network): Sequential(
      (0): TemporalBlock(
        (conv1): Conv1d(20, 32, kernel_size=(7,), stride=(1,), padding=(6,))
        (chomp1): Chomp1d()
        (relu1): ReLU()
        (dropout1): Dropout(p=0.5, inplace=False)
        (conv2): Conv1d(32, 32, kernel_size=(7,), stride=(1,), padding=(6,))
        (chomp2): Chomp1d()
        (relu2): ReLU()
        (dropout2): Dropout(p=0.5, inplace=False)
        (net): Sequential(
          (0): Conv1d(20, 32, kernel_size=(7,), stride=(1,), padding=(6,))
          (1): Chomp1d()
          (2): ReLU()
          (3): Dropout(p=0.5, inplace=False)
          (4): Conv1d(32, 32, kernel_size=(7,), stride=(1,), padding=(6,))
          (5): Chomp1d()
          (6): ReLU()
          (7): Dropout(p=0.5, inplace=False)
        )
        (downsample): Conv1d(20, 32, kernel_size=(1,), stride=(1,))
        (relu): ReLU()
      )
      (1): TemporalBlock(
        (conv1): Conv1d(32, 32, kernel_size=(7,), stride=(1,), padding=(12,), dilation=(2,))
        (chomp1): Chomp1d()
        (relu1): ReLU()
        (dropout1): Dropout(p=0.5, inplace=False)
        (conv2): Conv1d(32, 32, kernel_size=(7,), stride=(1,), padding=(12,), dilation=(2,))
        (chomp2): Chomp1d()
        (relu2): ReLU()
        (dropout2): Dropout(p=0.5, inplace=False)
        (net): Sequential(
          (0): Conv1d(32, 32, kernel_size=(7,), stride=(1,), padding=(12,), dilation=(2,))
          (1): Chomp1d()
          (2): ReLU()
```

* Bai, S., Kolter, J., & Koltun, V. (2018). An Empirical Evaluation of Generic Convolutional and Recurrent Networks for Sequence Modeling.

量化交易分析流程

参考：https://finance.sina.com.cn/stock/stockzmt/2021-02-23/doc-ikftpnny9226595.shtml

- 构造数据
- 特征分析
- 特征预处理
- 训练模型
- 回测模型
- 回测结果分析
- 模型上线

按照 Qlib 官方库的方法获取内置数据。

股票池：沪深300成分股。

```
import qlib
from qlib.data import D
qlib.init(provider_uri='~/.qlib/data/cn_data')
instruments = D.instruments(market='csi300')
D.list_instruments(instruments=instruments, start_time='2010-01-01', end_time='2017-12-31', as_list=True)[:20]
```

▲获取数据

```
qlib_init:
    provider_uri: "~/.qlib/qlib_data/cn_data"
    region: cn
market: &market csi300
benchmark: &benchmark SH000300
data_handler_config: &data_handler_config
    start_time: 2008-01-01
    end_time: 2020-08-01
    fit_start_time: 2008-01-01
    fit_end_time: 2014-12-31
    instruments: *market
    infer_processors:
        - class: RobustZScoreNorm
          kwargs:
              fields_group: feature
              clip_outlier: true
        - class: Fillna
          kwargs:
              fields_group: feature
    learn_processors:
        - class: DropnaLabel
        - class: CSRankNorm
          kwargs:
              fields_group: label
    label: ["Ref($close, -2) / Ref($close, -1) - 1"]
```

量化交易分析流程

- 构造数据
- 特征分析
- 特征预处理
- 训练模型
- 回测模型
- 回测结果分析
- 模型上线

特征采用 Qlib 内置的因子库 Alpha158 中的 158 个因子特征。

```
    kwargs:
        loss: mse
        input_dim: 158
        output_dim: 1
        layers: [64,16,4]
        kernel_size: 3
        lr: 0.002
        lr_decay: 0.96
        lr_decay_steps: 100
        optimizer: adam
        max_steps: 4000
        batch_size: 8196
        GPU: 0
        weight_decay: 0.0002
        loss_type: mse
dataset:
    class: DatasetH
    module_path: qlib.data.dataset
    kwargs:
        handler:
            class: Alpha158
            module_path: qlib.contrib.data.handler
            kwargs: *data_handler_config
        segments:
            train: [2008-01-01, 2014-12-31]
            valid: [2015-01-01, 2016-12-31]
            test: [2017-01-01, 2020-08-01]
record:
    - class: SignalRecord
      module_path: qlib.workflow.record_temp
      kwargs:
        model: <MODEL>
        dataset: <DATASET>
    - class: SigAnaRecord
      module_path: qlib.workflow.record_temp
      kwargs:
        ana_long_short: False
        ann_scaler: 252
    - class: PortAnaRecord
      module_path: qlib.workflow.record_temp
      kwargs:
```

量化交易分析流程

- 构造数据
- 特征分析
- 特征预处理
- 训练模型
- 回测模型
- 回测结果分析
- 模型上线

训练与验证集：
DropnaLabel 用于去除缺失值，
CSRrankNorm 用于截面标准化。

测试集：

RobustZScoreNorm 用来归一化数据，
Fillna 用来去除缺失值。

```yaml
qlib_init:
   provider_uri: "~/.qlib/qlib_data/cn_data"
   region: cn
market: &market csi300
benchmark: &benchmark SH000300
data_handler_config: &data_handler_config
   start_time: 2008-01-01
   end_time: 2020-08-01
   fit_start_time: 2008-01-01
   fit_end_time: 2014-12-31
   instruments: *market
   infer_processors:
      - class: RobustZScoreNorm
        kwargs:
           fields_group: feature
           clip_outlier: true
      - class: Fillna
        kwargs:
           fields_group: feature
   learn_processors:
      - class: DropnaLabel
      - class: CSRankNorm
        kwargs:
           fields_group: label
   label: ["Ref($close, -2) / Ref($close, -1) - 1"]
port_analysis_config: &port_analysis_config
   strategy:
      class: TopkDropoutStrategy
      module_path: qlib.contrib.strategy
      kwargs:
         signal:
            - <MODEL>
            - <DATASET>
         topk: 50
         n_drop: 5
   backtest:
      start_time: 2017-01-01
```

量化交易分析流程

- 构造数据
- 特征分析
- 特征预处理
- 训练模型
- 回测模型
- 回测结果分析
- 模型上线

这个 workflow 使用了自己的模型，也可以改为 TCN 等其他模型。

```yaml
            - <DATASET>
          topk: 50
          n_drop: 5
    backtest:
      start_time: 2017-01-01
      end_time: 2020-08-01
      account: 100000000
      benchmark: *benchmark
      exchange_kwargs:
        limit_threshold: 0.095
        deal_price: close
        open_cost: 0.0005
        close_cost: 0.0015
        min_cost: 5
task:
    model:
      class: CNNModelPytorch
      module_path: qlib.contrib.model.pytorch_cnn
      kwargs:
        loss: mse
        input_dim: 158
        output_dim: 1
        layers: [64,16,4]
        kernel_size: 3
        lr: 0.002
        lr_decay: 0.96
        lr_decay_steps: 100
        optimizer: adam
        max_steps: 4000
        batch_size: 8196
        GPU: 0
        weight_decay: 0.0002
        loss_type: mse
    dataset:
      class: DatasetH
      module_path: qlib.data.dataset
      kwargs:
        handler:
          class: Alpha158
          module_path: qlib.contrib.data.handler
          kwargs: *data_handler_config
```

量化交易分析流程

- 构造数据
- 特征分析
- 特征预处理
- 训练模型
- 回测模型
- 回测结果分析
- 模型上线

得到最优参数后，进行模型预测得到测试集中每一天股票的预测收益率。使用 TopkDropout 策略。

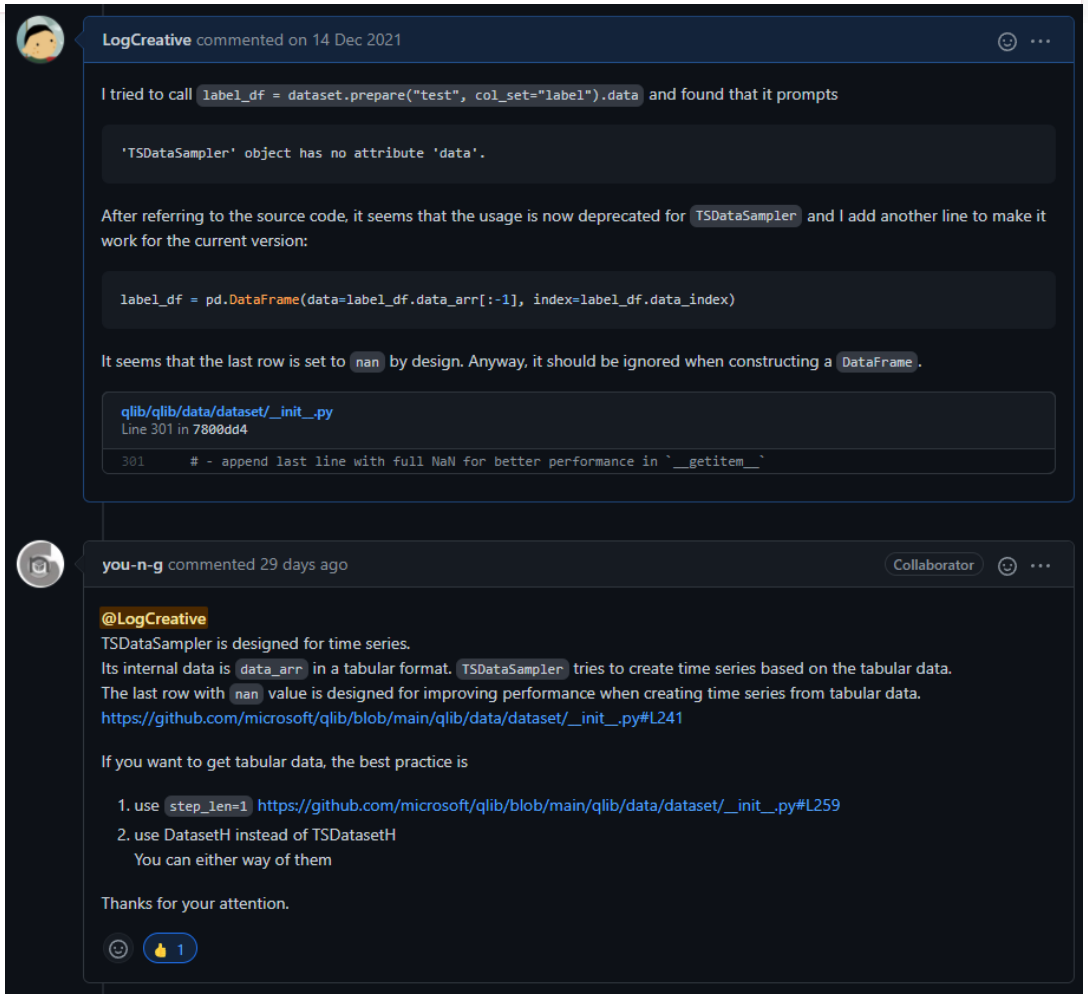
```
   infer_processors:
      - class: RobustZScoreNorm
        kwargs:
           fields_group: feature
           clip_outlier: true
      - class: Fillna
        kwargs:
           fields_group: feature
   learn_processors:
      - class: DropnaLabel
      - class: CSRankNorm
        kwargs:
           fields_group: label
   label: ["Ref($close, -2) / Ref($close, -1) - 1"]
port_analysis_config: &port_analysis_config
   strategy:
      class: TopkDropoutStrategy
      module_path: qlib.contrib.strategy
      kwargs:
         signal:
            - <MODEL>
            - <DATASET>
         topk: 50
         n_drop: 5
   backtest:
      start_time: 2017-01-01
      end_time: 2020-08-01
      account: 100000000
      benchmark: *benchmark
      exchange_kwargs:
         limit_threshold: 0.095
         deal_price: close
         open_cost: 0.0005
         close_cost: 0.0015
         min_cost: 5
task:
   model:
      class: CNNModelPytorch
      module_path: qlib.contrib.model.pytorch_cnn
      kwargs:
         loss: mse
```

量化交易分析流程

- 构造数据
- 特征分析
- 特征预处理
- 训练模型
- 回测模型
- 回测结果分析
- 模型上线

尝试将 TCN 适配进 workflow_by_code.ipynb 中可视化结果。发现由于 TCN 实现使用了 TSDatasetSampler，最近有 API 变动，适配了一些代码以绘制出图像。对此在 Qlib 中进行了 Issue 评论。

关于 TSDatasetSampler 可视化上的 Issue 评论▶



LogCreative commented on 14 Dec 2021

I tried to call `label_df = dataset.prepare("test", col_set="label").data` and found that it prompts

```
'TSDataSampler' object has no attribute 'data'.
```

After referring to the source code, it seems that the usage is now deprecated for `TSDataSampler` and I add another line to make it work for the current version:

```
label_df = pd.DataFrame(data=label_df.data_arr[:-1], index=label_df.data_index)
```

It seems that the last row is set to `nan` by design. Anyway, it should be ignored when constructing a `DataFrame`.

qlib/qlib/data/dataset/__init__.py
Line 301 in 7800dd4

```
301     # - append last line with full NaN for better performance in `__getitem__`
```

you-n-g commented 29 days ago — Collaborator

@LogCreative
TSDataSampler is designed for time series.
Its internal data is `data_arr` in a tabular format. `TSDataSampler` tries to create time series based on the tabular data.
The last row with `nan` value is designed for improving performance when creating time series from tabular data.
https://github.com/microsoft/qlib/blob/main/qlib/data/dataset/__init__.py#L241

If you want to get tabular data, the best practice is

1. use `step_len=1` https://github.com/microsoft/qlib/blob/main/qlib/data/dataset/__init__.py#L259
2. use DatasetH instead of TSDatasetH
   You can either way of them

Thanks for your attention.

1

量化交易分析流程

构造数据

特征分析

特征预处理

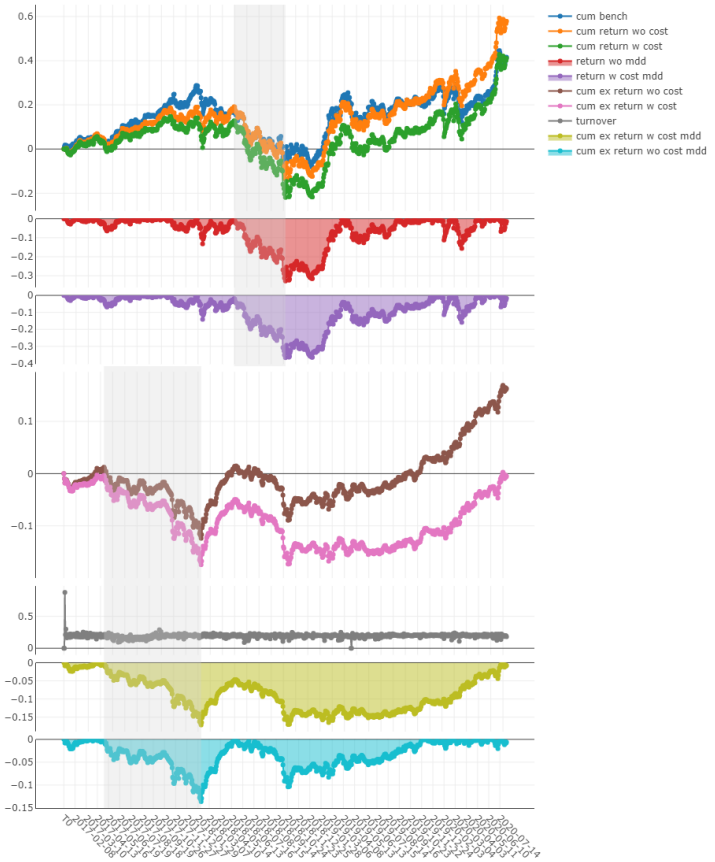
训练模型

回测模型

回测结果分析

模型上线

现在关于自行实现的 CNN 模型上线还不现实，效果还不好。而现在的 TCN 模型在普通的 GPU （GTX 1050 Ti）上会因为显存不足的原因无法运行，轻量级的 CNN 模型还有待探索。



TCN 模型的结果可视化▶

基于Qlib的案例创新
总结

本项目尝试将二维卷积迁移到一维卷积，尝试实现到 Qlib 中，尽管最后效果一般，但也能从这一思维过程体会到 TCN 模型的相关情况。

Qlib 为量化分析者提供了一个非常方便的与 AI 集成的平台，只需要一个 YAML 文件就能配置好很多的实验。统一的框架也使得像我这样对量化知识没有特别多了解的人，也能够做出一些自己的模型用于提升预测效果。

▲机器学习在生产环境中的应用 — TFX 1.0 (Google Developer Summit China 2021) 提到用于建模的代码实际上仅占整个解决方案的很少一部分。