深度学习实战

樊彬、胡艳艳、张利欣

智能科学与技术学院

教学安排

- 要求：设计并编程实现一个基于深度学习的系统
- 第1周：
  - 周一（4学时）：布置任务，分组
  - 周二（4学时）：选题，查找资料，下载数据库
  - 周三（4学时）：系统总体设计、分模块设计
  - 周四（4学时）：系统总体设计、分模块设计
  - 周五（4学时）：编程实现、调试、测试、记录实验数据
- 第2周：
  - 周一（4学时）：编程实现、调试、测试、记录实验数据
  - 周二（4学时）：编程实现、调试、测试、记录实验数据
  - 周三（4学时）：编程实现、调试、测试、准备答辩

考核要求

平时成绩：（占总分40%）

- 考勤（10分）
- 项目演示（30分）

考试成绩：（占总分60%）

- 答辩（30分）
- 报告（30分）（团队部分10分+个人部分20分）

实习地点：机电楼217（第5周周一~周五，第6周周一~周三）

因实验室场地有限，建议：

上午：智能211；下午：智能212、213

不要求必须在217，可在任意地方完成。

- 考勤要求
  - 签到时间：上午9:00-12:00，下午1:00-4:00
  - 签到地点：机电楼217签到
  - 上午：智能211
  - 下午：智能212、智能213
  - 每天上、下午只要签一次，多签无用，每天最多只算一次考勤
- 答辩

  答辩时间：清明节后，第7周

  答辩地点：机电信息楼217
- 报告（电子版，含答辩ppt、程序、数据、结果）

  重要：报告中要明确阐明个人在团队中的分工，以及具体开展的工作，无此部分，总分-20

  第10周周五之前交，按班级交

  智能211，　　交给：樊老师，智能学院306

  智能212，　　交给：胡老师，创新苑（10斋前2层小楼1楼）

  智能213，　　交给：张老师，机电楼217

答疑：机电楼217

- 深度学习环境配置
- 可选项目介绍

深度学习环境配置

1) OS：推荐在Linux系统下进行（Linux系统推荐Ubuntu18.04）,windows也可以

2) IDE： Visual Studio Code(推荐)、Pycharm、Juypter Notebook(有浏览器即可)

3) 编译器及依赖包：根据项目地址中的requirements.txt进行配置

4)GPU：如果使用英伟达显卡训练网络，应注意显卡兼容的CUDA、cuDNN、pytorch版本。

Visual Studio Code下载地址： https://code.visualstudio.com/

安装Ubuntu 18.04： https://zhuanlan.zhihu.com/p/38797088

深度学习环境配置

torch	torchvision	python
main / nightly	main / nightly	>=3.6, <=3.9
1.10.0	0.11.1	>=3.6, <=3.9
1.9.1	0.10.1	>=3.6, <=3.9
1.9.0	0.10.0	>=3.6, <=3.9
1.8.2	0.9.2	>=3.6, <=3.9
1.8.1	0.9.1	>=3.6, <=3.9
1.8.0	0.9.0	>=3.6, <=3.9
1.7.1	0.8.2	>=3.6, <=3.9
1.7.0	0.8.1	>=3.6, <=3.8
1.7.0	0.8.0	>=3.6, <=3.8
1.6.0	0.7.0	>=3.6, <=3.8
1.5.1	0.6.1	>=3.5, <=3.8
1.5.0	0.6.0	>=3.5, <=3.8
1.4.0	0.5.0	==2.7, >=3.5, <=3.8
1.3.1	0.4.2	==2.7, >=3.5, <=3.7
1.3.0	0.4.1	==2.7, >=3.5, <=3.7
1.2.0	0.4.0	==2.7, >=3.5, <=3.7
1.1.0	0.3.0	==2.7, >=3.5, <=3.7
<=1.0.1	0.2.2	==2.7, >=3.5, <=3.7

不同显卡需要的**CUDA**版本：https://www.jianshu.com/p/ac70300b598b

CUDA历史版本下载地址： https://developer.nvidia.com/cuda-toolkit-archive

CUDA与**cuDNN**版本对应： https://developer.nvidia.com/rdp/cudnn-archive

个人实验环境：**IDE**： Visual Studio Code

```
OS: CentOS 7.6.1810 Core
Kernel: x86_64 Linux 3.10.0-957.el7.x86_64
Uptime: up 4 weeks, 6 days, 22 hours, 15 minutes
Packages: 934
Shell: bash 4.2.46
CPU: Intel(R) Xeon(R) Gold 6248R CPU @ 3.00GHz
GPU: GeForce RTX 3090, GeForce RTX 3090, GeForce RTX 3090, GeForce RTX 3090
```

```
torch                   1.10.2+cu113
torchfile               0.1.0
torchvision             0.11.3+cu113
```

```
scikit-learn                  1.0.2
```

```
opencv-python                 4.5.4.60
```

实验1：YOLO V5 游戏人物检测

本项目采用深度学习中目标检测任务目前应用最为广泛，热度最高的网络YOLO，采用的版本为YOLO V5，支持在移动端部署，以及高速的实时检测。

本项主要对游戏中人物出现的位置进行检测，输出位置检测结果的同时输出人物的类别。目前网络出现的FPS类游戏的AI辅助基本均由YOLO V5实现。该项目可通过更换不同的数据集实现在不同场景下进行目标检测。

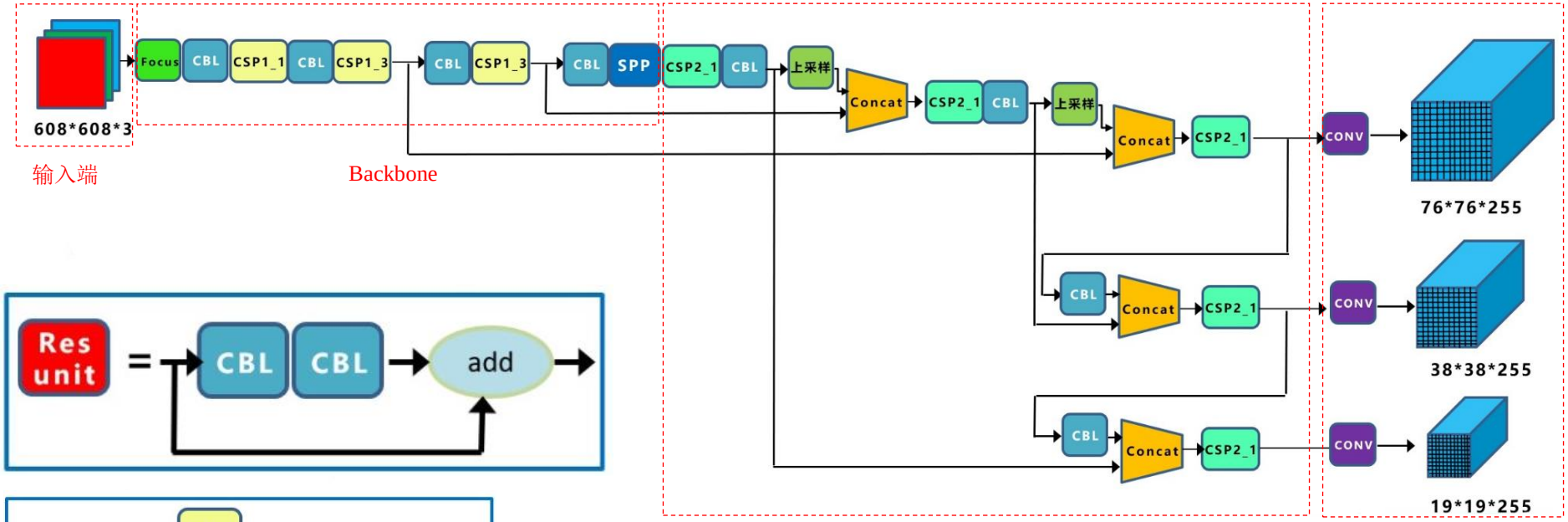
论文地址(YOLO)：https://arxiv.org/abs/1506.02640

原理：https://zhuanlan.zhihu.com/p/172121380

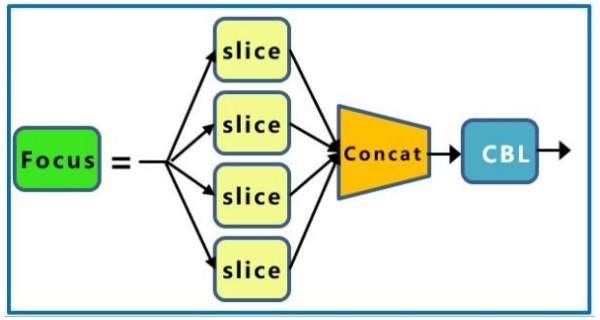
项目地址：https://github.com/ultralytics/yolov5

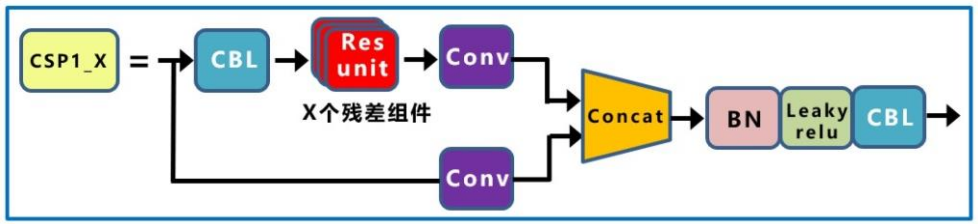
由于YOLO被用作军事袭击以及隐私窥测，目前YOLO之父于2020年已经宣布退出CV界，正确使用AI人人有责!!!

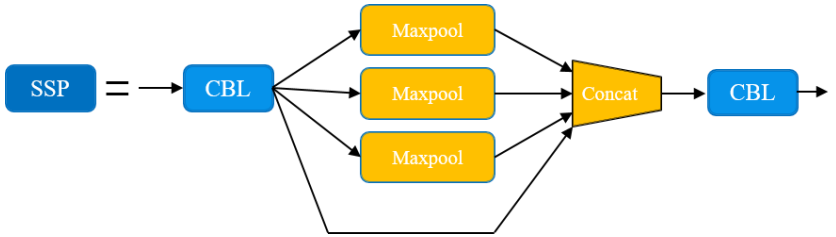
YOLO V5 网络结构

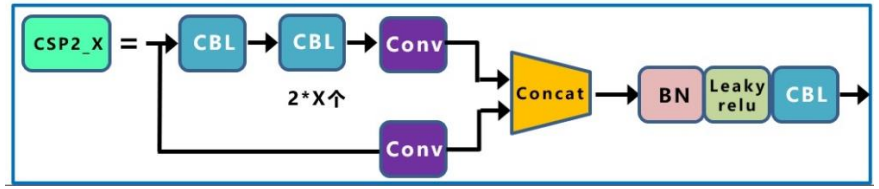


输入端　　Backbone　　Neck　　Predition









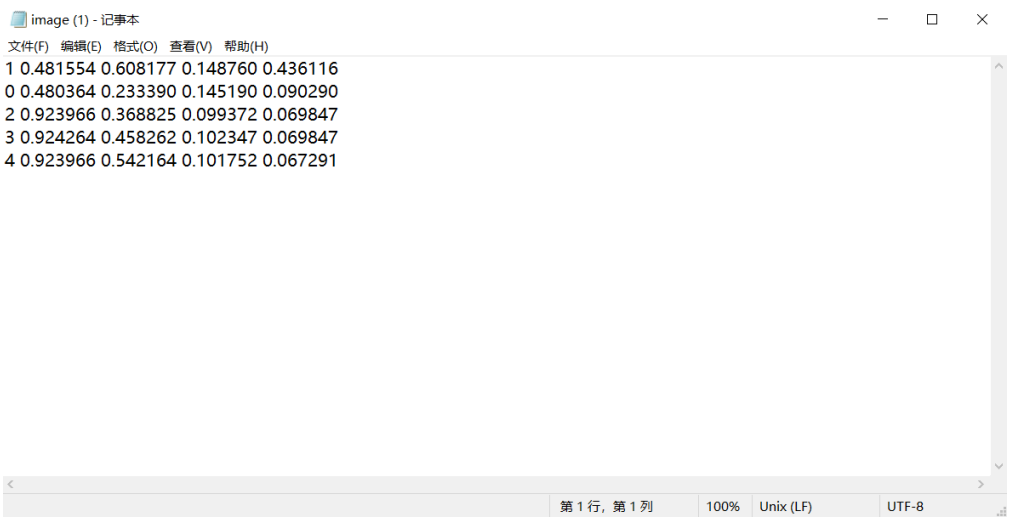
实验1：YOLO V5 游戏人物检测

1、深度学习环境配置

2、给数据集中的图片打上标签

1) 在进行目标检测网络的训练时，给模型输入的训练数据包含图像本身，以及需要检测的的目标所在位置以及目标的类别。打标签：标注目标在图像中的位置以及类别。

2) 打标签的项目地址: https://github.com/ultralytics/yolov5/wiki/Train-Custom-Data



实验1： YOLO V5 游戏人物检测

3、下载预训练的YOLO V5s模型

1) 为什么使用预训练模型：因为实验中给出的数据集太小等问题，不足以支撑训练出一个优秀的检测网络，需要使用别人在更大数据集上训练好的模型。

2) 模型下载地址：https://github.com/ultralytics/yolov5/releases

▾ Assets 12

yolov5l.pt	89.3 MB
yolov5l6.pt	147 MB
yolov5m.pt	40.8 MB
yolov5m6.pt	69 MB
yolov5n.pt	3.87 MB
yolov5n6.pt	6.86 MB
yolov5s.pt	14.1 MB
yolov5s6.pt	24.8 MB
yolov5x.pt	166 MB
yolov5x6.pt	270 MB
Source code (zip)	
Source code (tar.gz)	

实验1： YOLO V5 游戏人物检测

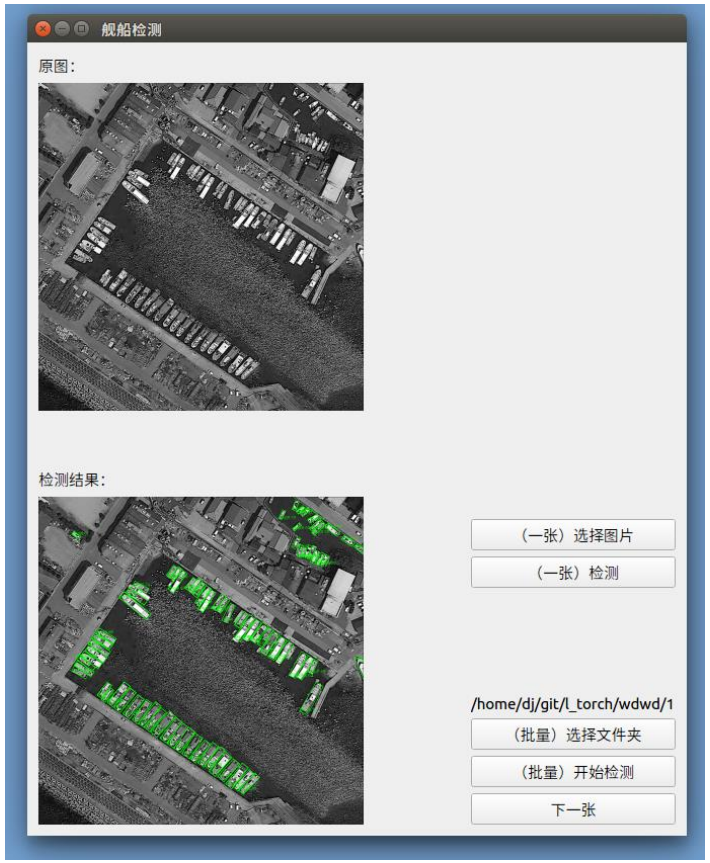
- 4、利用标注之后的图片对模型进行训练

1) 可以利用默认的参数进行训练，也可以通过更改参数进行训练。

实验1： YOLO V5 游戏人物检测

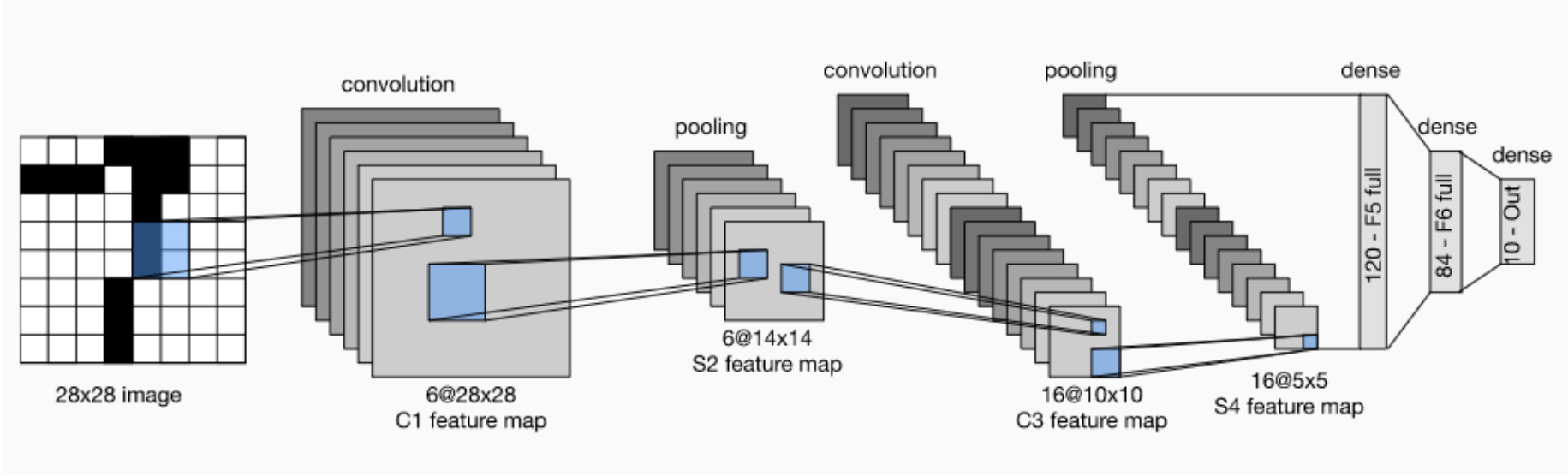
5、利用训练之后的模型对测试集进行检测，输出测试集的检测结果

6、制作GUI界面，实现在界面上一键运行程序，实现实时检测。(推荐pyqt5)

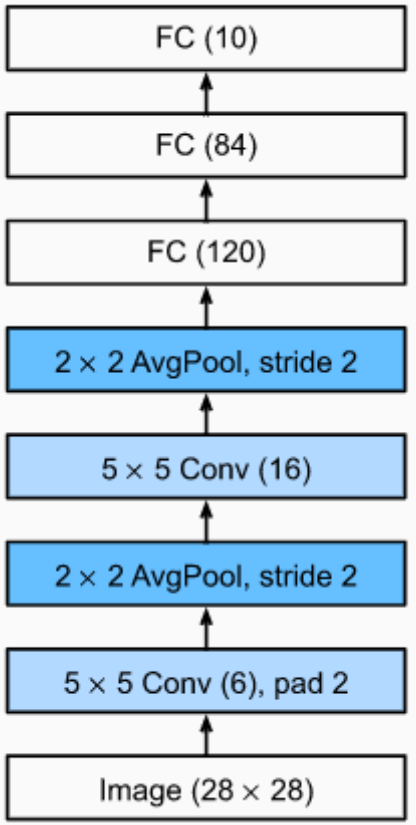


常见图像分类网络

LeNet5 (1994年)




```
net = nn.Sequential(
    nn.Conv2d(1, 6, kernel_size=5, padding=2), nn.Sigmoid(),
    nn.AvgPool2d(kernel_size=2, stride=2),
    nn.Conv2d(6, 16, kernel_size=5), nn.Sigmoid(),
    nn.AvgPool2d(kernel_size=2, stride=2),
    nn.Flatten(),
    nn.Linear(16 * 5 * 5, 120), nn.Sigmoid(),
    nn.Linear(120, 84), nn.Sigmoid(),
    nn.Linear(84, 10))
```



常见图像分类网络

AlexNet (2012年)

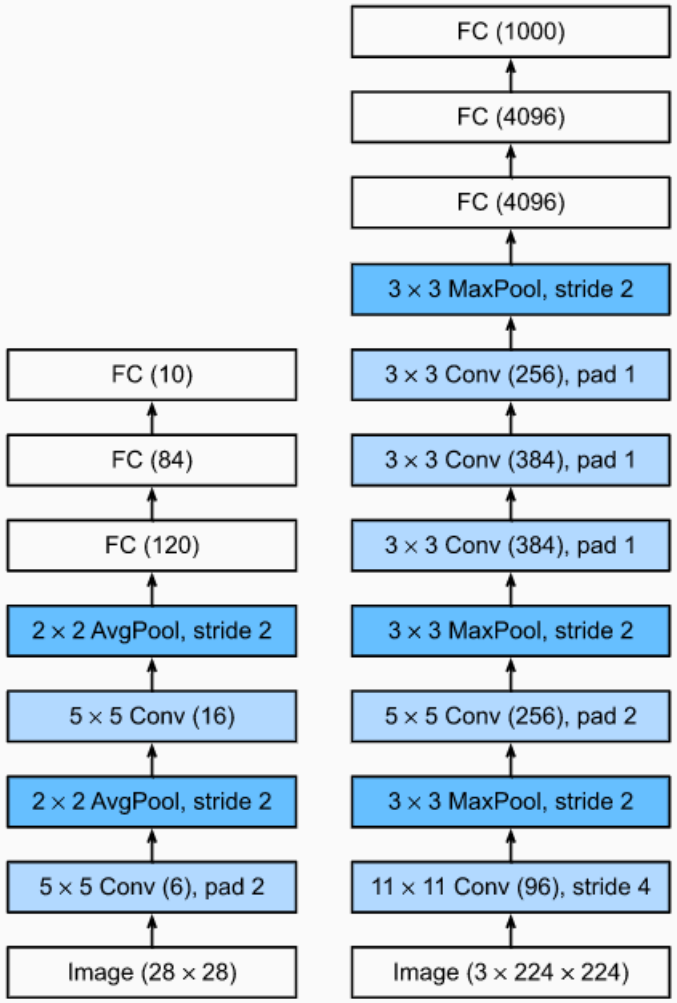
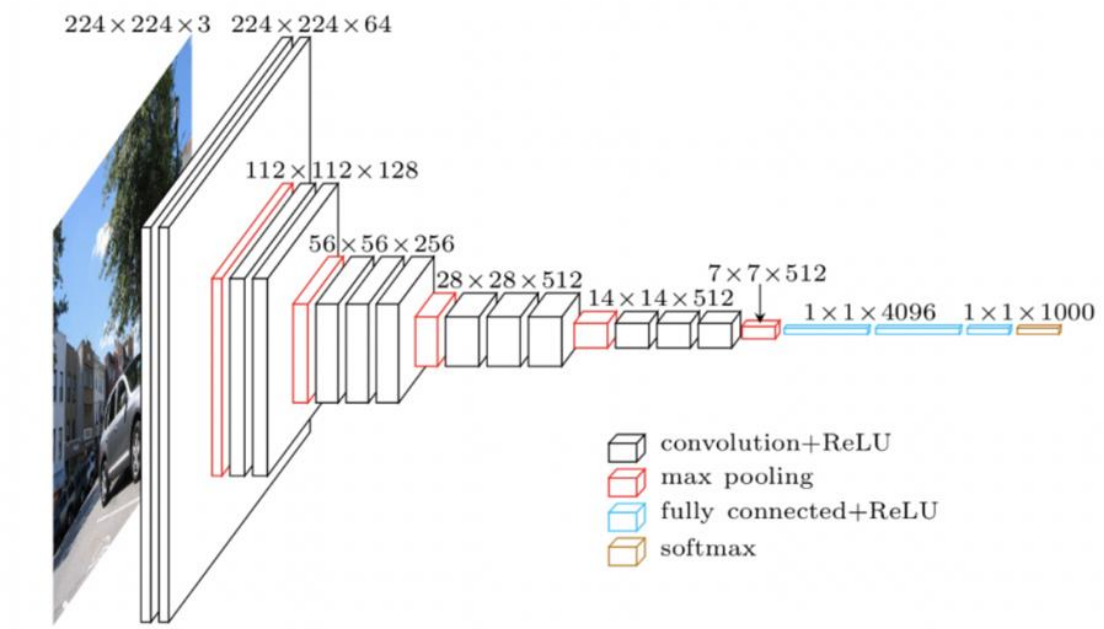


Fig. 7.1.2 From LeNet (left) to AlexNet (right).

```
net = nn.Sequential(
    # Here, we use a larger 11 x 11 window to capture objects. At the same
    # time, we use a stride of 4 to greatly reduce the height and width of the
    # output. Here, the number of output channels is much larger than that in
    # LeNet
    nn.Conv2d(1, 96, kernel_size=11, stride=4, padding=1), nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2),
    # Make the convolution window smaller, set padding to 2 for consistent
    # height and width across the input and output, and increase the number of
    # output channels
    nn.Conv2d(96, 256, kernel_size=5, padding=2), nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2),
    # Use three successive convolutional layers and a smaller convolution
    # window. Except for the final convolutional layer, the number of output
    # channels is further increased. Pooling layers are not used to reduce the
    # height and width of input after the first two convolutional layers
    nn.Conv2d(256, 384, kernel_size=3, padding=1), nn.ReLU(),
    nn.Conv2d(384, 384, kernel_size=3, padding=1), nn.ReLU(),
    nn.Conv2d(384, 256, kernel_size=3, padding=1), nn.ReLU(),
    nn.MaxPool2d(kernel_size=3, stride=2),
    nn.Flatten(),
    # Here, the number of outputs of the fully-connected layer is several
    # times larger than that in LeNet. Use the dropout layer to mitigate
    # overfitting
    nn.Linear(6400, 4096), nn.ReLU(),
    nn.Dropout(p=0.5),
    nn.Linear(4096, 4096), nn.ReLU(),
    nn.Dropout(p=0.5),
    # Output layer. Since we are using Fashion-MNIST, the number of classes is
    # 10, instead of 1000 as in the paper
    nn.Linear(4096, 10))
```

常见图像分类网络

VGG16 (2014年)



常见图像分类网络

GoogLeNet (2014年 ImageNet竞赛冠军)

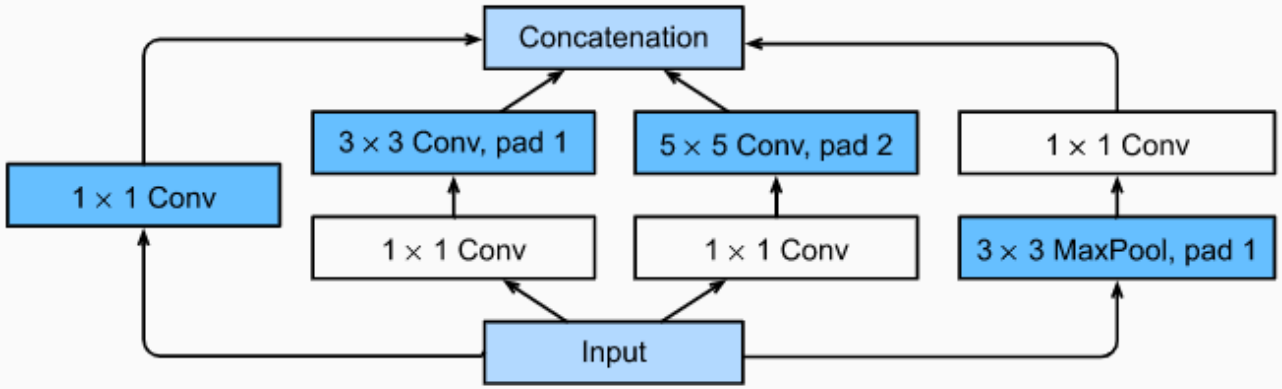
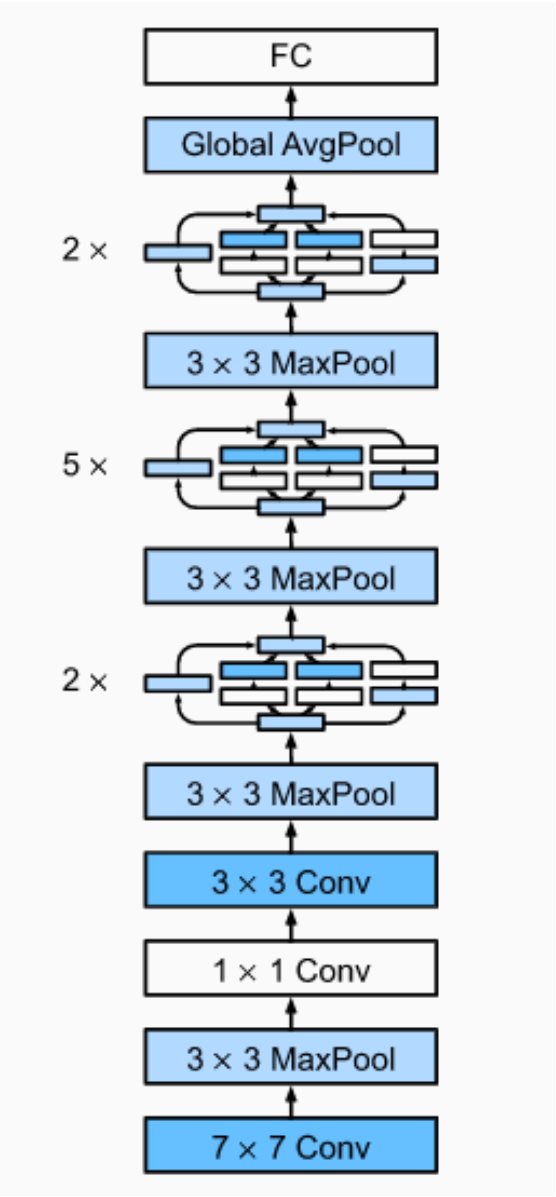


Fig. 7.4.1 Structure of the Inception block.



常见图像分类网络

ResNet (2016年 CVPR Best Paper)

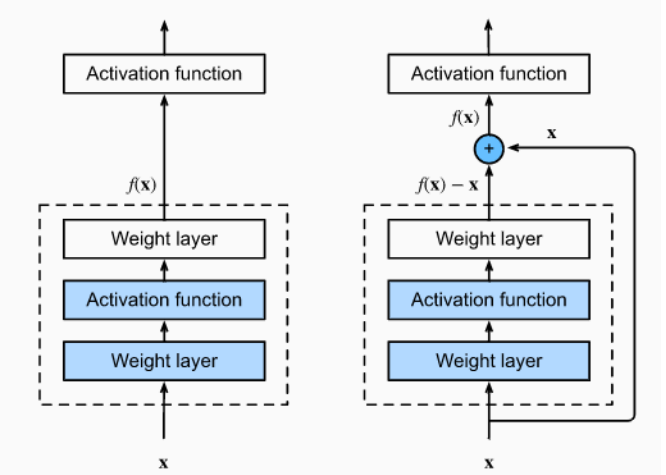


Fig. 7.6.2 A regular block (left) and a residual block (right).

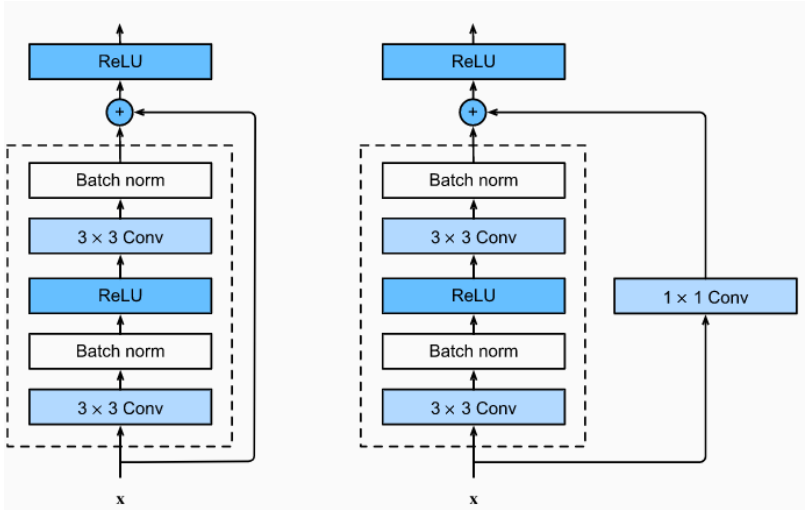


Fig. 7.6.3 ResNet block with and without 1×1 convolution.

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				

常见图像分类网络

DensenNet （2017年 CVPR Best Paper)

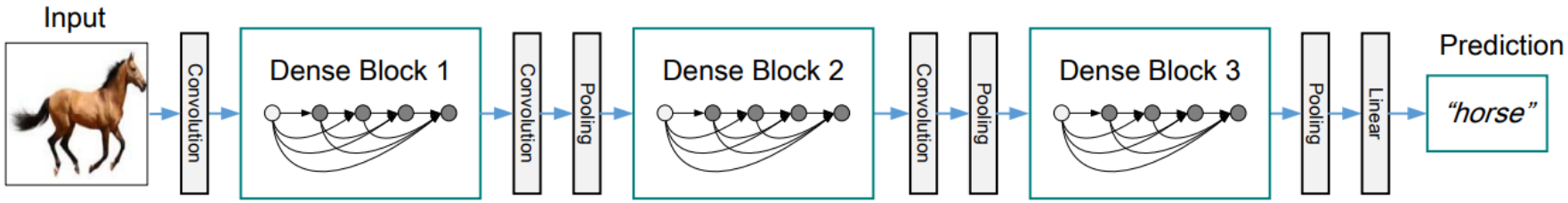


Figure 2: A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature-map sizes via convolution and pooling.

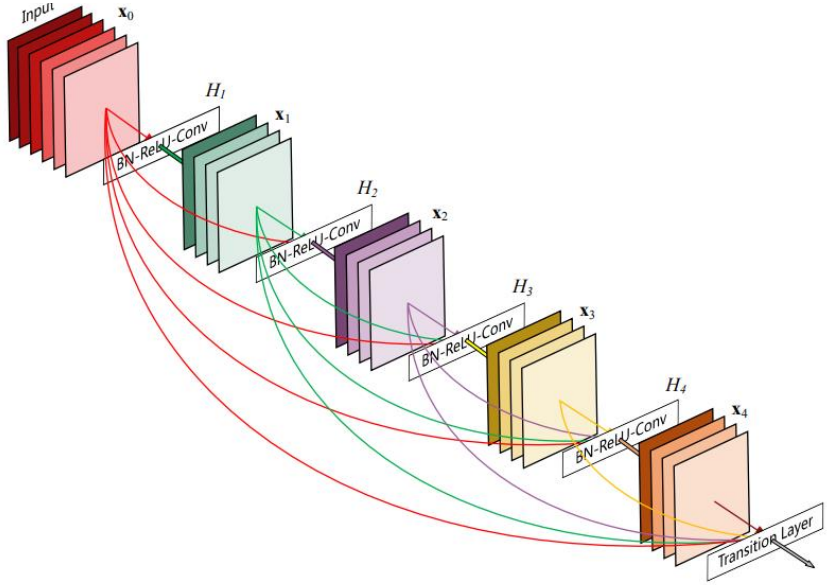


Figure 1: A 5-layer dense block with a growth rate of $k = 4$. Each layer takes all preceding feature-maps as input.

Layers	Output Size	DenseNet-121	DenseNet-169	DenseNet-201	DenseNet-264
Convolution	112 × 112	7 × 7 conv, stride 2			
Pooling	56 × 56	3 × 3 max pool, stride 2			
Dense Block (1)	56 × 56	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6	[1 × 1 conv; 3 × 3 conv] × 6
Transition Layer (1)	56 × 56	1 × 1 conv			
	28 × 28	2 × 2 average pool, stride 2			
Dense Block (2)	28 × 28	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12	[1 × 1 conv; 3 × 3 conv] × 12
Transition Layer (2)	28 × 28	1 × 1 conv			
	14 × 14	2 × 2 average pool, stride 2			
Dense Block (3)	14 × 14	[1 × 1 conv; 3 × 3 conv] × 24	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48	[1 × 1 conv; 3 × 3 conv] × 64
Transition Layer (3)	14 × 14	1 × 1 conv			
	7 × 7	2 × 2 average pool, stride 2			
Dense Block (4)	7 × 7	[1 × 1 conv; 3 × 3 conv] × 16	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 32	[1 × 1 conv; 3 × 3 conv] × 48
Classification Layer	1 × 1	7 × 7 global average pool			
		1000D fully-connected, softmax			

实验2：手写数字的识别

本项目为深度学习中图像分类任务的基础应用。采用卷积神经网络提取图像的特征，实现端到端的手写数字识别。

本项目采用的数据集为MNIST，该项目来源于最早的卷积神经网络之一LeNet5识别银行支票的手写数字。当年该项目的负责人为2018年图灵将得主Yann Lecun。完成该项目的学习之后，可以实现手写数字的识别，推荐使用Pytorch框架。

手写数据集介绍：http://yann.lecun.com/exdb/mnist/

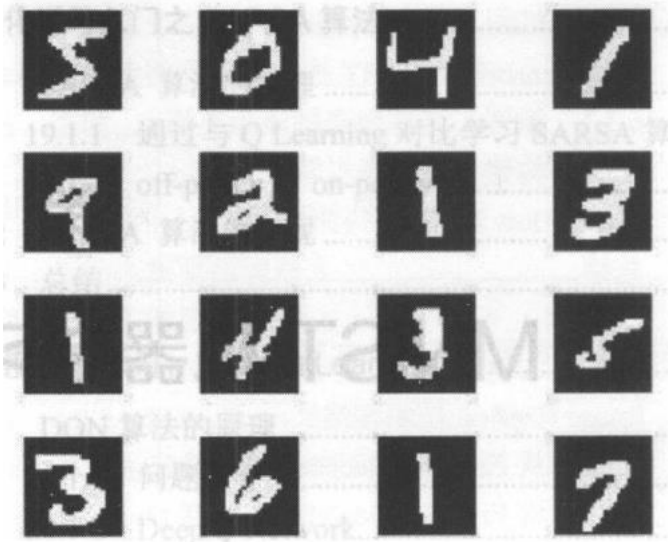
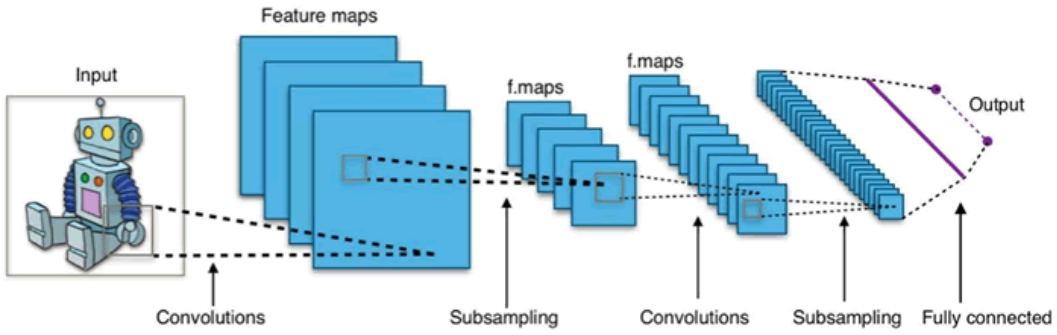


图 1-1　MNIST 数据集图片示例



实验2：手写数字的识别

1、深度学习环境配置

2、数据集的处理

1) 训练集和验证集的划分(在torchvision直接获取MNIST数据集)

MNIST

```
dset.MNIST(root, train=True, transform=None, target_transform=None, download=False)
```

参数说明： - root : `processed/training.pt` 和 `processed/test.pt` 的主目录 - train : `True` = 训练集, `False` = 测试集 - download : `True` = 从互联网上下载数据集，并把数据集放在 `root` 目录下. 如果数据集之前下载过，将处理过的数据（minist.py中有相关函数）放在 `processed` 文件夹下。

```
class torch.utils.data.DataLoader(dataset, batch_size=1, shuffle=False, sampler=None, num_workers=0, co
```

数据加载器。组合数据集和采样器，并在数据集上提供单进程或多进程迭代器。

参数:

- **dataset** (*Dataset*) – 加载数据的数据集。
- **batch_size** (*int*, optional) – 每个batch加载多少个样本(默认: 1)。
- **shuffle** (*bool*, optional) – 设置为 `True` 时会在每个epoch重新打乱数据(默认: False).
- **sampler** (*Sampler*, optional) – 定义从数据集中提取样本的策略。如果指定，则忽略 `shuffle` 参数。
- **num_workers** (*int*, optional) – 用多少个子进程加载数据。0表示数据将在主进程中加载(默认: 0)
- **collate_fn** (*callable*, optional) –
- **pin_memory** (*bool*, optional) –
- **drop_last** (*bool*, optional) – 如果数据集大小不能被batch size整除，则设置为True后可删除最后一个不完整的batch。如果设为False并且数据集的大小不能被batch size整除，则最后一个batch将更小。(默认: False)

2) 数据的增强

3) 利用torch.utils.data.DataLoader将训练集合验证集进行封装

实验2：手写数字的识别

3、网络的搭建，优化器、损失函数、学习率变化策略选择

1) 网络的搭建

```
net = nn.Sequential(
    nn.Conv2d(1, 6, kernel_size=5, padding=2), nn.Sigmoid(),
    nn.AvgPool2d(kernel_size=2, stride=2),
    nn.Conv2d(6, 16, kernel_size=5), nn.Sigmoid(),
    nn.AvgPool2d(kernel_size=2, stride=2),
    nn.Flatten(),
    nn.Linear(16 * 5 * 5, 120), nn.Sigmoid(),
    nn.Linear(120, 84), nn.Sigmoid(),
    nn.Linear(84, 10))
```

2) 优化器

```
optimizer = optim.SGD(model.parameters(), lr = 0.01, momentum=0.9)
optimizer = optim.Adam([var1, var2], lr = 0.0001)
```

```
optim.SGD([
                {'params': model.base.parameters()},
                {'params': model.classifier.parameters(), 'lr': 1e-3}
            ], lr=1e-2, momentum=0.9)
```

3)损失函数

class torch.nn.MSELoss(size_average=True)[source]

创建一个衡量输入 `x`（模型预测输出）和目标 `y` 之间均方误差标准。

$$loss(x, y) = 1/n \sum (x_i - y_i)^2$$

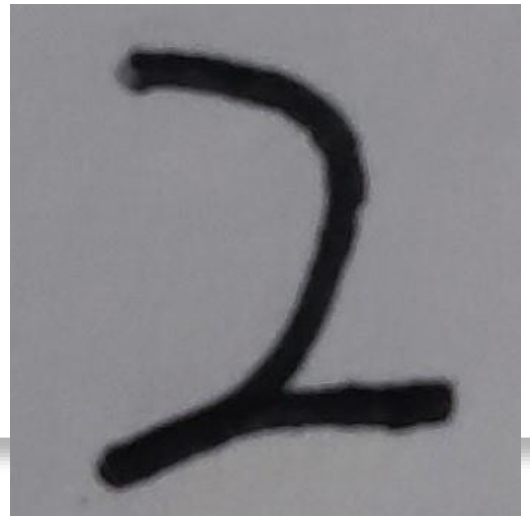
- `x` 和 `y` 可以是任意形状，每个包含 `n` 个元素。
- 对 `n` 个元素对应的差值的绝对值求和，得出来的结果除以 `n`。
- 如果在创建 `MSELoss` 实例的时候在构造函数中传入 `size_average=False`，那么求出来的平方和将不会除以 `n`

实验2：手写数字的识别

4、读取数据对构建的网络进行训练和验证

```
for i, (X, y) in enumerate(train_iter):
    timer.start()
    optimizer.zero_grad()
    X, y = X.to(device), y.to(device)
    y_hat = net(X)
    l = loss(y_hat, y)
    l.backward()
    optimizer.step()
```

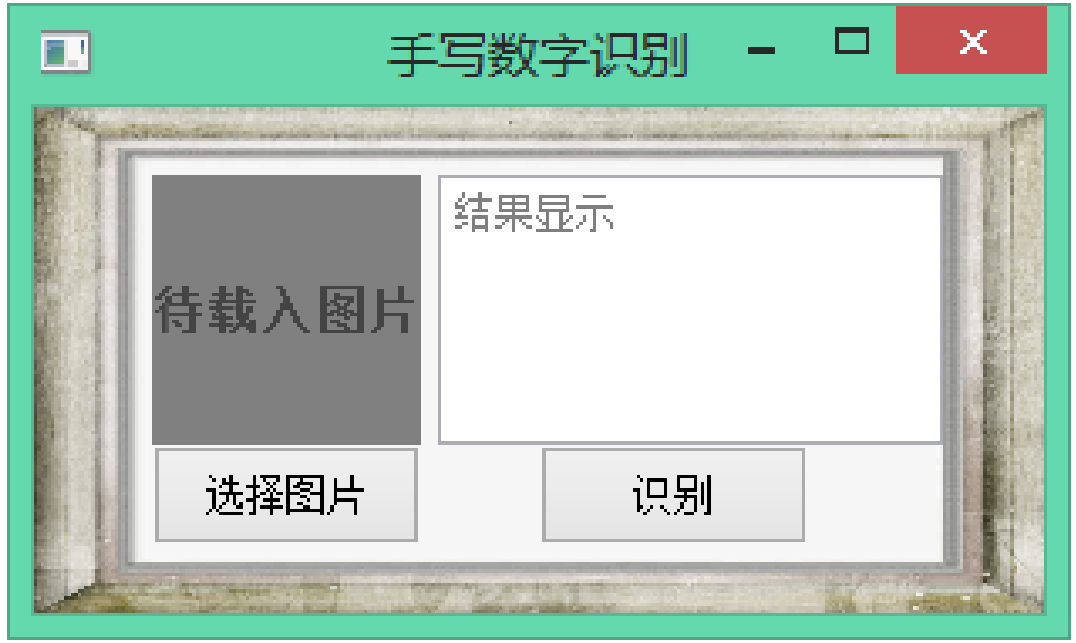
5、利用保存的网络结构实现对自己手写的数字的识别

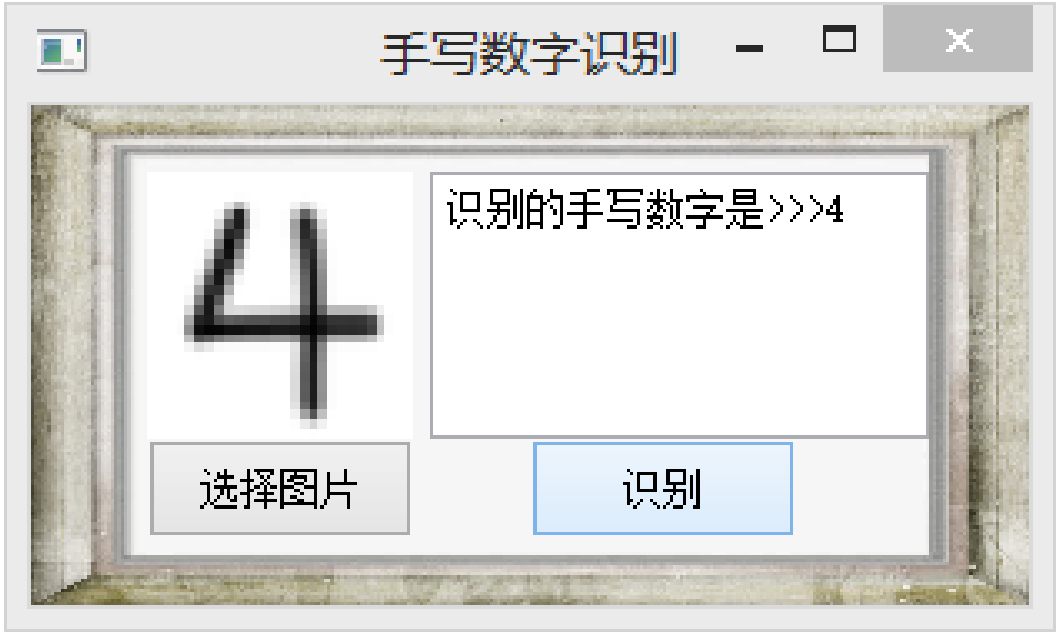


```
(base) [yongjie.chen@gpu-dev058 examples]$
(densenas) [yongjie.chen@gpu-dev058 example
en/.vscode-server/extensions/ms-python.pyth
ing/examples/mnist/my_data.py
tensor([[2]])
```

实验2：手写数字的识别

6、设计GUI界面实现一键运行程序，完成对自己手写数字识别

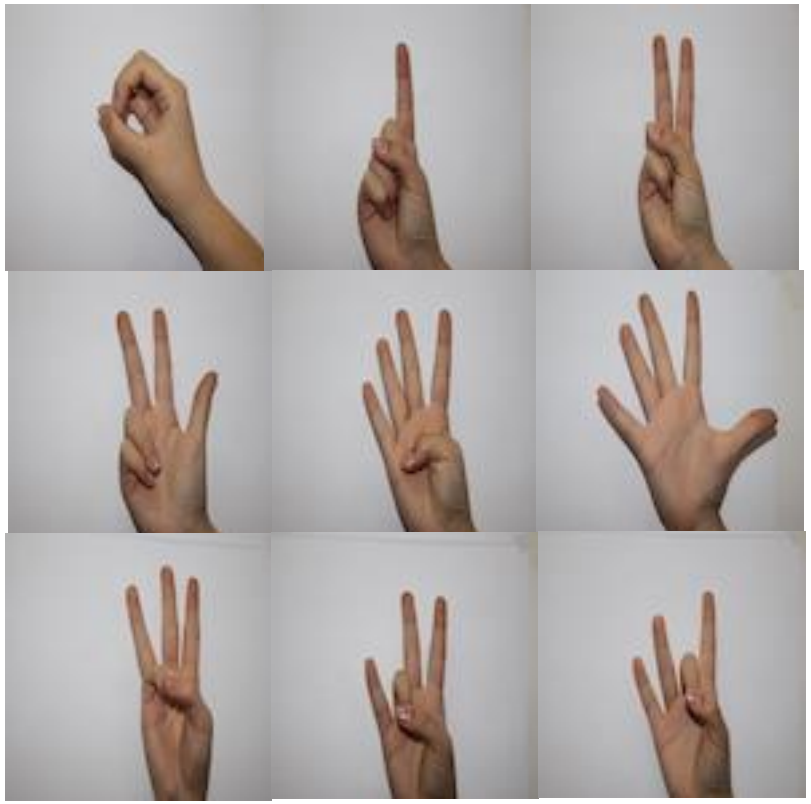




实验3：手势识别

本项目为深度学习中图像分类任务的基础应用。采用卷积神经网络提取图像的特征，实现端到端的手势识别。

本项目的目标输入手势图片，输出手势对应的数字。数据集中包含0-9这10个手势，每个手势存在一个文件夹。本项目来源于通过手势实现人机的交互，计算机通过检测操作人员的手势做出对应的行动。通过学习该项目，可以自定义手势含义，不局限于数字，电脑通过识别结果做出自定义的行动。

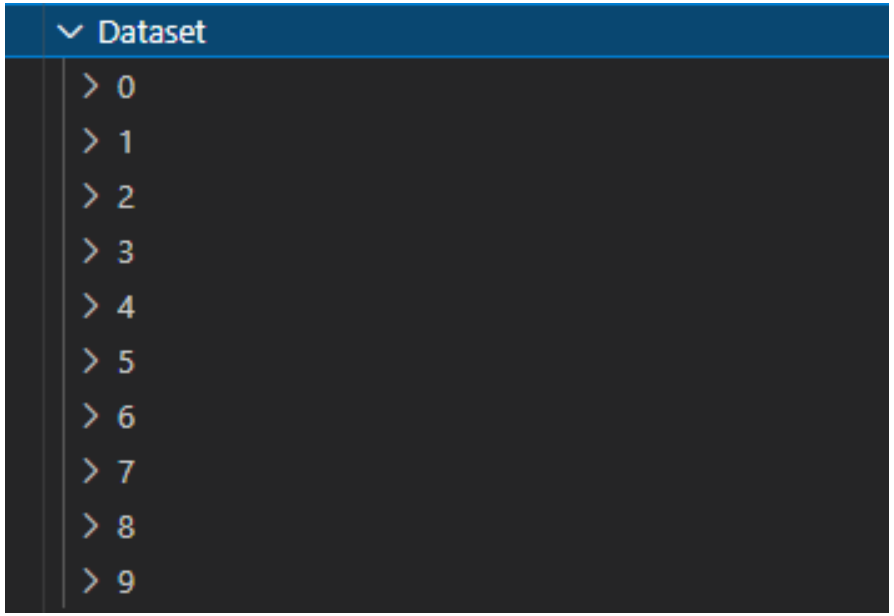


实验3：手势识别

1、深度学习环境配置

2、数据集的处理

1) 训练集和验证集的划分及Dataloader的构建




```python
import torch
import numpy as np


# 定义GetLoader类，继承Dataset方法，并重写__getitem__()和__len__()方法
class GetLoader(torch.utils.data.Dataset):
    # 初始化函数，得到数据
    def __init__(self, data_root, data_label):
        self.data = data_root
        self.label = data_label
    # index是根据batchsize划分数据后得到的索引，最后将data和对应的labels进行一起返回
    def __getitem__(self, index):
        data = self.data[index]
        labels = self.label[index]
        return data, labels
    # 该函数返回数据大小长度，目的是DataLoader方便划分，如果不知道大小，DataLoader会一脸懵逼
    def __len__(self):
        return len(self.data)
```

实验3：手势识别

2) 数据的增强:

地址：https://pytorch.org/vision/stable/transforms.html

3) 利用torch.utils.data.DataLoader，将训练集合验证集进行封装

```
class torch.utils.data.DataLoader(dataset, batch_size=1, shuffle=False, sampler=None, num_workers=0, co
```

数据加载器。组合数据集和采样器，并在数据集上提供单进程或多进程迭代器。

参数:

- **dataset** (*Dataset*) – 加载数据的数据集。
- **batch_size** (*int*, optional) – 每个batch加载多少个样本(默认: 1)。
- **shuffle** (*bool*, optional) – 设置为 `True` 时会在每个epoch重新打乱数据(默认: False).
- **sampler** (*Sampler*, optional) – 定义从数据集中提取样本的策略。如果指定，则忽略 `shuffle` 参数。
- **num_workers** (*int*, optional) – 用多少个子进程加载数据。0表示数据将在主进程中加载(默认: 0)
- **collate_fn** (*callable*, optional) –
- **pin_memory** (*bool*, optional) –
- **drop_last** (*bool*, optional) – 如果数据集大小不能被batch size整除，则设置为True后可删除最后一个不完整的batch。如果设为False并且数据集的大小不能被batch size整除，则最后一个batch将更小。(默认: False)

```
train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=args.batch_size,
    shuffle=(train_sampler is None),
    sampler=train_sampler,
    drop_last = getattr(args, 'drop_last', True),
    num_workers=args.data_loader_workers_per_gpu,
    pin_memory=True,
)
```

实验3：手势识别

3、网络的搭建，优化器、损失函数、学习率变化策略选择

1) 网络的搭建

```
net = nn.Sequential(
    nn.Conv2d(1, 6, kernel_size=5, padding=2), nn.Sigmoid(),
    nn.AvgPool2d(kernel_size=2, stride=2),
    nn.Conv2d(6, 16, kernel_size=5), nn.Sigmoid(),
    nn.AvgPool2d(kernel_size=2, stride=2),
    nn.Flatten(),
    nn.Linear(16 * 5 * 5, 120), nn.Sigmoid(),
    nn.Linear(120, 84), nn.Sigmoid(),
    nn.Linear(84, 10))
```

2) 优化器

```
optimizer = optim.SGD(model.parameters(), lr = 0.01, momentum=0.9)
optimizer = optim.Adam([var1, var2], lr = 0.0001)
```

```
optim.SGD([
                {'params': model.base.parameters()},
                {'params': model.classifier.parameters(), 'lr': 1e-3}
            ], lr=1e-2, momentum=0.9)
```

3)损失函数

class torch.nn.MSELoss(size_average=True)[source]

创建一个衡量输入 `x`（模型预测输出）和目标 `y` 之间均方误差标准。

$$loss(x, y) = 1/n \sum (x_i - y_i)^2$$

- `x` 和 `y` 可以是任意形状，每个包含 `n` 个元素。
- 对 `n` 个元素对应的差值的绝对值求和，得出来的结果除以 `n`。
- 如果在创建 `MSELoss` 实例的时候在构造函数中传入 `size_average=False`，那么求出来的平方和将不会除以 `n`

实验3：手势识别

4、读取数据对构建的网络进行训练和验证

```
for i, (X, y) in enumerate(train_iter):
    timer.start()
    optimizer.zero_grad()
    X, y = X.to(device), y.to(device)
    y_hat = net(X)
    l = loss(y_hat, y)
    l.backward()
    optimizer.step()
```

5、利用保存的网络结构实现对自己手势的识别

```
(densenas) [yongjie.che
/home/users/yongjie.che
tensor([[4]])
```

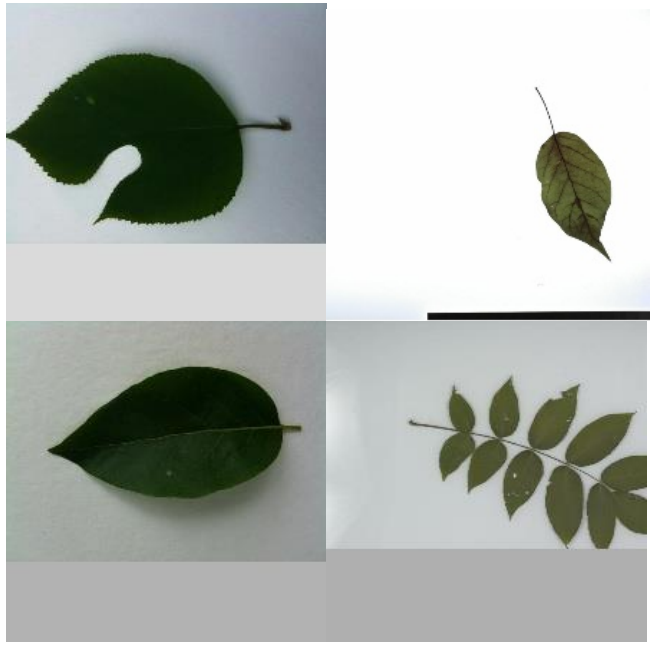
6、设计GUI界面实现一键运行程序，完成对自己手势的识别

实验4：树叶种类分类

本项目为深度学习中图像分类任务的基础应用。采用卷积神经网络提取图像的特征，实现端到端的树叶种类的识别。

本项目来自Kaggle上的一次竞赛，目标是通过给出的训练集训练网络，然后利用保存的模型对测试数据集进行分类。

比赛项目地址：https://www.kaggle.com/c/classify-leaves



实验4：树叶种类分类

1、深度学习环境配置

2、数据集的处理

1) 训练集和验证集的划分及Dataloader的构建


```python
import torch
import numpy as np


# 定义GetLoader类，继承Dataset方法，并重写__getitem__()和__len__()方法
class GetLoader(torch.utils.data.Dataset):
    # 初始化函数，得到数据
    def __init__(self, data_root, data_label):
        self.data = data_root
        self.label = data_label
    # index是根据batchsize划分数据后得到的索引，最后将data和对应的labels进行一起返回
    def __getitem__(self, index):
        data = self.data[index]
        labels = self.label[index]
        return data, labels
    # 该函数返回数据大小长度，目的是DataLoader方便划分，如果不知道大小，DataLoader会一脸懵逼
    def __len__(self):
        return len(self.data)
```

实验4：树叶种类分类

2) 数据的增强:

地址： https://pytorch.org/vision/stable/transforms.html

3) 利用torch.utils.data.DataLoader，将训练集合验证集进行封装

```
class torch.utils.data.DataLoader(dataset, batch_size=1, shuffle=False, sampler=None, num_workers=0, co
```

数据加载器。组合数据集和采样器，并在数据集上提供单进程或多进程迭代器。

参数:

- **dataset** (*Dataset*) – 加载数据的数据集。
- **batch_size** (*int*, optional) – 每个batch加载多少个样本(默认: 1)。
- **shuffle** (*bool*, optional) – 设置为 `True` 时会在每个epoch重新打乱数据(默认: False).
- **sampler** (*Sampler*, optional) – 定义从数据集中提取样本的策略。如果指定，则忽略 `shuffle` 参数。
- **num_workers** (*int*, optional) – 用多少个子进程加载数据。0表示数据将在主进程中加载(默认: 0)
- **collate_fn** (*callable*, optional) –
- **pin_memory** (*bool*, optional) –
- **drop_last** (*bool*, optional) – 如果数据集大小不能被batch size整除，则设置为True后可删除最后一个不完整的batch。如果设为False并且数据集的大小不能被batch size整除，则最后一个batch将更小。(默认: False)

```python
train_loader = torch.utils.data.DataLoader(
    train_dataset,
    batch_size=args.batch_size,
    shuffle=(train_sampler is None),
    sampler=train_sampler,
    drop_last = getattr(args, 'drop_last', True),
    num_workers=args.data_loader_workers_per_gpu,
    pin_memory=True,
)
```

实验4：树叶种类分类

3、网络的搭建，优化器、损失函数、学习率变化策略选择

1) 网络的搭建

```
net = nn.Sequential(
    nn.Conv2d(1, 6, kernel_size=5, padding=2), nn.Sigmoid(),
    nn.AvgPool2d(kernel_size=2, stride=2),
    nn.Conv2d(6, 16, kernel_size=5), nn.Sigmoid(),
    nn.AvgPool2d(kernel_size=2, stride=2),
    nn.Flatten(),
    nn.Linear(16 * 5 * 5, 120), nn.Sigmoid(),
    nn.Linear(120, 84), nn.Sigmoid(),
    nn.Linear(84, 10))
```

2) 优化器

```
optimizer = optim.SGD(model.parameters(), lr = 0.01, momentum=0.9)
optimizer = optim.Adam([var1, var2], lr = 0.0001)
```

```
optim.SGD([
                {'params': model.base.parameters()},
                {'params': model.classifier.parameters(), 'lr': 1e-3}
            ], lr=1e-2, momentum=0.9)
```

3)损失函数

class torch.nn.MSELoss(size_average=True)[source]

创建一个衡量输入 `x`（模型预测输出）和目标 `y` 之间均方误差标准。

$$loss(x, y) = 1/n \sum (x_i - y_i)^2$$

- `x` 和 `y` 可以是任意形状，每个包含 `n` 个元素。
- 对 `n` 个元素对应的差值的绝对值求和，得出来的结果除以 `n`。
- 如果在创建 `MSELoss` 实例的时候在构造函数中传入 `size_average=False`，那么求出来的平方和将不会除以 `n`

实验4：树叶种类分类

4、读取数据对构建的网络进行训练和验证

```
for i, (X, y) in enumerate(train_iter):
    timer.start()
    optimizer.zero_grad()
    X, y = X.to(device), y.to(device)
    y_hat = net(X)
    l = loss(y_hat, y)
    l.backward()
    optimizer.step()
```

5、利用保存的网络结构实现对测试集树叶种类的识别

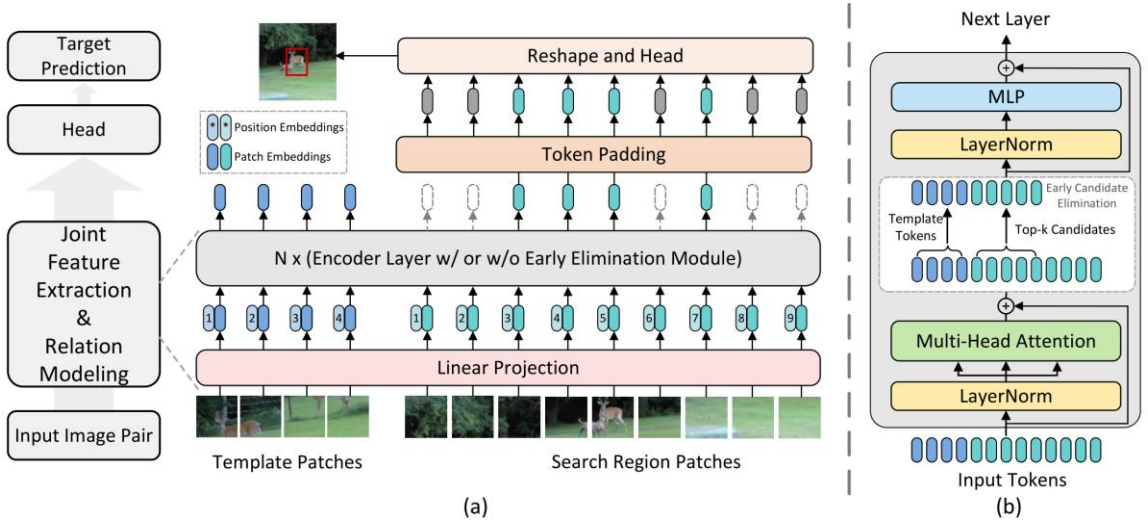
6、设计GUI界面实现一键运行程序，完成对测试集树叶种类的识别

实验5：OSTrack目标跟踪

本项目为深度学习中目标追踪任务的基础应用。采用Transformer提取图像的特征，实现端到端的目标追踪。

本项目来自2022年ECCV上发表的一篇论文。OSTrack框架简洁高效，基于自注意力操作进行联合特征学习和关系建模。该框架没有使用任何额外的时间信息，却在多个基准测试中实现了最先进的性能。该项目可通过更换不同的数据集实现在不同场景下进行目标追踪。

项目地址：https://github.com/botaoye/OSTrack
论文地址：https://arxiv.org/abs/2203.11991

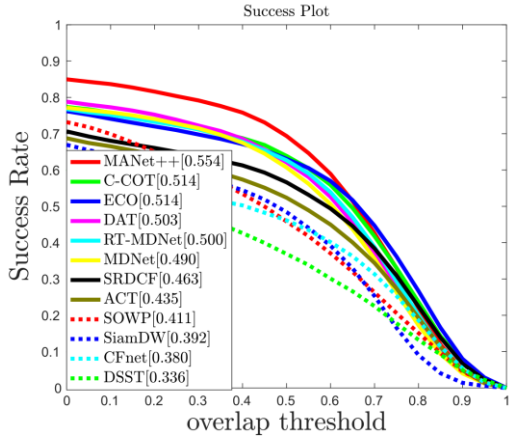


网络结构

实验5：OSTrack目标跟踪

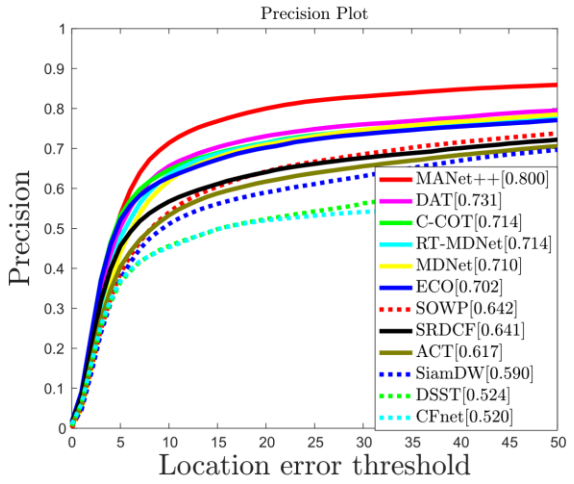
算法评价指标

①**AUC**，即模型输出的bbox 的准确度;一个视频在指定阈值下跟踪模型的成功跟踪率
中心位置之间的误差
计算步骤如下
1. 定一个阈值, 比如0.5
2. 计算 模型输出的bbox 和 实际gt bbox的iou(交并比, 就是交集 除上 并集)
3. 按第二个步骤计算整个序列的所有帧的 iou, 然后将iou高于0.5的累加计数,最后将累加和 除上 该视频总帧数 得到 阈值为0.5下的成功率
4. 一般会将上面的步骤1用在0-1的不同阈值,因此可以绘制出一条曲线，计算曲线面积得到最终的AUC



如左图所示，图例后面的数字(AUC)为不同算法对应的曲线的面积

②**Precision**，即模型输出的bbox 中心位置 和实际gt bbox 中心位置之间的误差
计算步骤如下
1. 设置一个阈值 ,一般为20
2. 计算模型输出的bounding box 的中心位置 x , 和ground truth box的中心位置y
3. x,y 计算欧式距离得到 中心位置像素误差
4. 然后计算整个视频序列的所有中心位置像素误差;将像素误差低于20的进行累加求和 上. 最后将累加求和的结果除以视频总帧数 得到 阈值为20的 precision .



如左图所示，图例后面的数字(Precision)为不同算法对应的曲线在横坐标为20处的值

③Normal Precision
考虑到Ground Truth框的尺度大小，将Precision 进行归一化，得到Norm. Prec。即判断预测框与Ground Truth框中心点的欧氏距离与Ground Truth框斜边的比例

实验5：OSTrack目标跟踪

程序主要部分

网络整体结构：backbone+检测头

```
model = OSTrack(
    backbone,
    box_head,
    aux_loss=False,
    head_type=cfg.MODEL.HEAD.TYPE,
)
```

优化器——AdamW

```
if cfg.TRAIN.OPTIMIZER == "ADAMW":
    optimizer = torch.optim.AdamW(param_dicts, lr=cfg.TRAIN.LR,
                                  weight_decay=cfg.TRAIN.WEIGHT_DECAY)
```

损失函数

```
# weighted sum
loss = self.loss_weight['giou'] * giou_loss + self.loss_weight['l1'] * l1_loss + self.loss_weight['focal'] * location_loss
```

Giou损失函数：GIoU（广义交并比）引入了一个“闭包”概念，即包含两个边界框的最小矩形框，以此来克服传统IOU在边界框不重叠时效用不足的问题。GIoU损失函数的目标是最小化真实边界框和预测边界框组成的闭包的面积与两者并集的面积之差在闭包面积中的比例。

L1损失函数：L1损失函数，也称为绝对误差损失函数，是深度学习中常用的一种回归损失函数。它衡量了预测值与真实值之间的绝对差异

Focal loss损失函数： Focal loss是基于二分类交叉熵改进的。它是一个动态缩放的交叉熵损失，通过一个动态缩放因子，可以动态降低训练过程中易区分样本的权重，从而将重心快速聚焦在那些难区分的样本

实验5：OSTrack目标跟踪

1、环境配置

按照项目所给配置文件install.sh或者ostrack_cuda113_env.yaml来配置环境

Install the environment

Option1: Use the Anaconda (CUDA 10.2)

```
conda create -n ostrack python=3.8
conda activate ostrack
bash install.sh
```

Option2: Use the Anaconda (CUDA 11.3)

```
conda env create -f ostrack_cuda113_env.yaml
```

Option3: Use the docker file

We provide the full docker file here.

实验5：OSTrack目标跟踪

2、训练

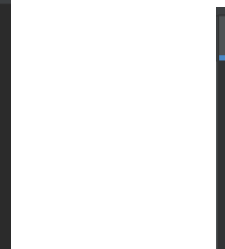
①创建 .py 文件用于后续设置环境变量：
命令行：python tracking/create_default_local_file.py --workspace_dir . --data_dir ./data --save_dir ./output

②模型下载地址：
https://dl.fbaipublicfiles.com/mae/pretrain/mae_pretrain_vit_base.pth
将模型放入以下目录“OSTrack/output/pretrained_models/“

③设置训练用的数据集，这里以GOT10K为例，实际训练中要求使用自己采集的数据集

vitb_256_mae_ce_32x4_ep300.yaml

```yaml
TRAIN:
  DATASETS_NAME:
  - LASOT
  - GOT10K_vottrain
  - COCO17
  - TRACKINGNET
  DATASETS_RATIO:
  - 1
  - 1
  - 1
  - 1
  SAMPLE_PER_EPOCH: 60000
VAL:
  DATASETS_NAME:
  - GOT10K_votval
  DATASETS_RATIO:
  - 1
  SAMPLE_PER_EPOCH: 10000
```



vitb_256_mae_ce_32x4_ep300.yaml

```yaml
TRAIN:
  DATASETS_NAME:
  - GOT10K_vottrain
  DATASETS_RATIO:
  - 1
  SAMPLE_PER_EPOCH: 60000
VAL:
  DATASETS_NAME:
  - GOT10K_votval
  DATASETS_RATIO:
  - 1
  SAMPLE_PER_EPOCH: 10000
```

实验5：OSTrack目标跟踪

④在lib/train/admin/local.py文件设置数据集路径

local.py

```
class EnvironmentSettings:
    def __init__(self):
        self.got10k_dir = '/xxx/xxx/GOT-10k/train'
        self.got10k_val_dir = '/xxx/xxx/GOT-10k/val'

        self.workspace_dir = '/xxx/xxx/proj/OSTrack'    # Base directory for saving network checkpoints.
        self.tensorboard_dir = '/xxx/xxx/proj/OSTrack/tensorboard'    # Directory for tensorboard files.
        self.pretrained_networks = '/xxx/xxx/proj/OSTrack/pretrained_networks'
```

设置数据集路径

⑤开始训练：

命令行： python tracking/train.py --script ostrack --config vitb_256_mae_ce_32x4_ep300 --save_dir ./output --mode single

实验5：OSTrack目标跟踪

⑥网络训练环节占用显存较大，建议降低batchsize或者冻结部分训练参数

冻结部分训练参数：

base_functions.py

```
164                 print(n)
165     else:
166         param_dicts = [
167             {"params": [p for n, p in net.named_parameters() if "backbone" not in n and p.requires_grad]},
168             {
169                 "params": [p for n, p in net.named_parameters() if "backbone" in n and p.requires_grad],
170                 "lr": cfg.TRAIN.LR * cfg.TRAIN.BACKBONE_MULTIPLIER,
171             },
172         ]
```

注释掉backbone部分的训练参数

base_functions.py

```
164                 print(n)
165     else:
166         param_dicts = [
167             {"params": [p for n, p in net.named_parameters() if "backbone" not in n and p.requires_grad]},
168             # {
169             #     "params": [p for n, p in net.named_parameters() if "backbone" in n and p.requires_grad],
170             #     "lr": cfg.TRAIN.LR * cfg.TRAIN.BACKBONE_MULTIPLIER,
171             # },
172         ]
```

降低batchsize：

vitb_256_mae_ce_32x4_ep300.yaml | train.py | run_training.py | train_script.py

```
54  TRAIN:
55    BACKBONE_MULTIPLIER: 0.1
56    DROP_PATH_RATE: 0.1
57    CE_START_EPOCH: 20  # candidate elimination start epoch
58    CE_WARM_EPOCH: 80  # candidate elimination warm up epoch
59    BATCH_SIZE: 4
60    EPOCH: 300
61    GIOU_WEIGHT: 2.0
62    L1_WEIGHT: 5.0
63    GRAD_CLIP_NORM: 0.1
```

实验5：OSTrack目标跟踪

3、测试

①模型下载地址(实际测试中用自己训练好的模型):
https://drive.google.com/drive/folders/1PS4inLS8bWNCecpYZ0W2fE5-A04DvTcd?usp=sharing
将模型放入以下位置
“OSTrack/output/checkpoints/train/ostrack/vitb_256_mae_ce_32x4_got10k_ep100/OSTrack_ep0100.pth.tar“

②开始测试:
命令行： python tracking/test.py ostrack vitb_256_mae_ce_32x4_got10k_ep100 --dataset got10k_test --threads 1 --num_gpus 1

③查看结果:
结果保存路径:
OSTrack/output/test/tracking_results/ostrack/vitb_256_mae_ce_32x4_got10k_ep100/got10k
以（x,y,w,h）格式保存对每一帧预测的bounding box，保存内容如右图所示

GOT-10k_Test_000001.txt

```
395 340 532 407
404 328 521 415
410 317 517 430
393 296 536 449
380 283 553 462
374 262 560 481
407 252 530 491
421 238 519 509
422 190 521 553
416 172 535 575
403 143 551 597
```

4、可视化

①设计GUI界面实现一键运行程序

②或者按照作者在github中的命令行，执行项目自带的可视化程序，如右图所示

实验5：OSTrack目标跟踪

5、数据集格式——以Got-10k数据集为例

数据集格式:

```
|-- GOT-10k/
    |-- train/
    |  |-- GOT-10k_Train_000001/
          |  |  001.jpg
          |  |  002.jpg
          |  |  ......
          |  |  groundtruth.txt
    |  |-- GOT-10k_Train_000002/
    |  |  ......
    |  |-- GOT-10k_Train_009335/
    |  |-- list.txt
    |-- val/
    |  |-- GOT-10k_Val_000001/
    |  |  ......
    |  |-- GOT-10k_Val_000180/
    |  |-- list.txt
    |-- test/
    |  |-- GOT-10k_Test_000001/
    |  |  ......
    |  |-- GOT-10k_Test_000180/
    |  |-- list.txt
```

数据集分为train/val/test三个部分，
GOT-10k_Train_000001为序列名，每个序列文件夹内包含图片和groundtruth.txt

groundtruth.txt记录每张图片的标注框坐标（x,y,w,h）
list.txt 统计所有序列名称，文件内容如下图所示

groundtruth.txt

```
882.0000,324.0000,529.0000,517.0000
882.0000,321.0000,520.0000,522.0000
866.0000,326.0000,533.0000,517.0000
854.0000,330.0000,536.0000,511.0000
843.0000,323.0000,536.0000,509.0000
857.0000,319.0000,522.0000,513.0000
848.0000,316.0000,515.0000,525.0000
```

list.txt

```
GOT-10k_Val_000001
GOT-10k_Val_000002
GOT-10k_Val_000003
GOT-10k_Val_000004
GOT-10k_Val_000005
GOT-10k_Val_000006
GOT-10k_Val_000007
```

按照Got-10k数据集的格式准备自己采集的数据集

各项目的补充说明

实验1：YOLO V5 游戏人物检测

(1) 数据集的图片中需要检测的类别为5类(shenli, xinqiu, zhongli, shenhe, qiuqiuren)

(2) 由于数据较少，验证集的数据仍然采用训练集的数据，最后以测试集的结果作为模型的评价指标。

建议使用yolov8，同一公司出品，技术更先进，效果更好

实验2：手写数字识别

(1) 网络训练的过程中需要输出数据读取时间、训练时间、学习率的变化、TOP-1准确率

(2) 不允许使用预训练模型，需利用数据集训练自己的网络模型

(3) 数据集可以在torchvision中获取到。提示：在测试自己的数据时，要使用OpenCV

可尝试ViT等基于Transformer的方法

各项目的补充说明

实验3：手势识别

(1) 网络训练的过程中需要输出数据读取时间、训练时间、学习率的变化、TOP-1准确率

(2)不允许使用预训练模型，需利用数据集训练自己的网络模型

(3) 由于数据集是自定义的，因此需要写一个封装的数据集类，传入torch.utils.data.DataLoader，实现数据读取

实验4：树叶种类分类

(1) 网络训练的过程中需要输出数据读取时间、训练时间、学习率的变化、TOP-1准确率

(2) 这个数据集比较小，单独训练效果并不是很好，允许使用预训练的模型。

(3) 同上，需要写一个封装的数据集类

(4) 如果验证集结果不如意，可以试用一下K折交叉验证

数据集地址：https://pan.baidu.com/s/1GJpok4YRfPQM6NjpXfj2AQ

提取码：a4nr

各项目的补充说明

实验5：OSTrack目标跟踪

(1) 使用预训练模型，并利用自己采集的数据对预训练模型进行微调

(2) 网络训练的过程中需要输出数据读取时间、训练时间、学习率的变化、训练集与验证集损失值

(3) 由于数据集是自定义的，因此需要写一个封装的数据集类，可以仿照“OSTrack/lib/train/dataset/got10k.py”文件中的类“Got10k”来写