

Pytorch入门

环境配置

pytorch加载数据Dataset

TensorBoard

transforms

torchvision官方数据集使用

Dataloader使用

nn.Model

神经网络

卷积运算

卷积层-convolution

【卷积前后尺寸计算】：

池化层-pooling

非线性激活

归一化层 (Normalization Layers)

线性层-linear layers

Sequential

损失函数-Loss Functions

反向传播

优化器 optimizer

使用现有网络模型及修改

示例：vgg16 model

示例：CIFAR 10 Model

完整模型训练

GPU训练

完整模型验证

如何看开源项目

What I cannot create, I do not understand

不闻不若闻之，闻之不若见之，见之不若知之，知之不若行之

共勉

序号	资源名	资源地址
1	PyTorch深度学习快速入门教程（绝对通俗易懂！）【小土堆】	https://www.bilibili.com/video/BV1hE411t7RN/?spm_id_from=333.999.0.0
2	代码地址	https://github.com/huzibo-yyds/pytorch_learn
3	样例数据集-蚂蚁蜜蜂	https://download.pytorch.org/tutorial/hymenoptera_data.zip
4	pytorch官网	https://pytorch.org/
5	pytorch官方文档	https://pytorch.org/docs/stable/index.html
6	工具经验：关闭pycharm补全时大小写匹配	https://jingyan.baidu.com/article/8cdccae986787f705513cd45.html

【主要内容】

1. 环境配置

anaconda、pytorch、pycharm、jupyter

2. Pytorch加载数据（以蚂蚁图片为例）。

a. Dataset、Dataloader使用

b. 如何使用pytorch官方数据集

3. 数据预处理相关函数、库的使用

a. TensorBoard——工具，用于tensor的可视化

b. transform——对数据预处理

4. 神经网络 neural network

a. 输入层

b. 隐藏层

i. 卷积层、池化层、归一层、线性层

c. 输出层

5. 如何定义自己的神经网络

6. 如何使用现有网络，及其修改(网络迁移)

7. 完整神经网络训练过程

a. CPU

b. GPU

环境配置

- anaconda
- Cuda
- pytorch
- jupyter（在conda环境下再次安装）

```
1 # 创建虚拟环境
2 conda create -n pytorch_2.1.2_GPU python=3.9
3 # 启用环境
4 conda activate pytorch_2.1.2_GPU
5 # 下载pytorch|CUDA 11.8
6 conda install pytorch==2.1.2 torchvision==0.16.2 torchaudio==2.1.2 pytorch-
  cuda=11.8 -c pytorch -c nvidia
7
8 # 查看nvidia驱动版本
9 nvidia-smi
```

面对陌生库2个常用函数

```
1 # dir列出对象的所有属性及方法
2 dir(torch.cuda.is_available)
3 # help
4 help(torch.cuda.is_available)
```

pytorch加载数据Dataset

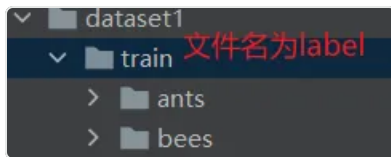
- Dataset
为数据，编号、label
- Dataloader
为网络提供不同的数据形式

【样例数据集-蚂蚁蜜蜂】

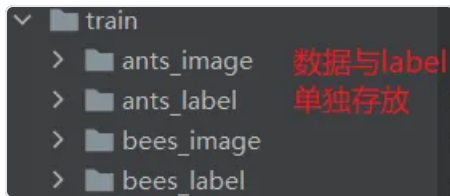
https://download.pytorch.org/tutorial/hymenoptera_data.zip

【数据与label的几种方式】

1. 文件名是label



2. 一个目录存放数据，另一个目录存放数据对应label



3. label直接标注在数据上

TensorBoard

训练模型时的**可视化工具** | [TensorBoard使用举例](#)

TensorBoard是一个由TensorFlow提供的**可视化工具**，用于帮助机器学习工程师和研究人员可视化、理解和调试他们的模型训练过程。它提供了各种可视化功能，可以帮助用户深入了解模型的性能、结构和训练过程。通过TensorBoard提供的这些可视化功能，用户可以更直观地了解模型的训练过程和性能，从而更好地调试模型、优化超参数，并作出更加合理的决策。

```
from torch.utils.tensorboard import SummaryWriter
```

【如何启动】

1. 确保安装TensorFlow
2. 在python脚本中使用`torch.utils.tensorboard.SummaryWriter`类来将数据写入到TensorBoard中
3. 在命令行中（此处为conda中），进入到你的Python脚本所在的目录，然后执行以下命令启动TensorBoard

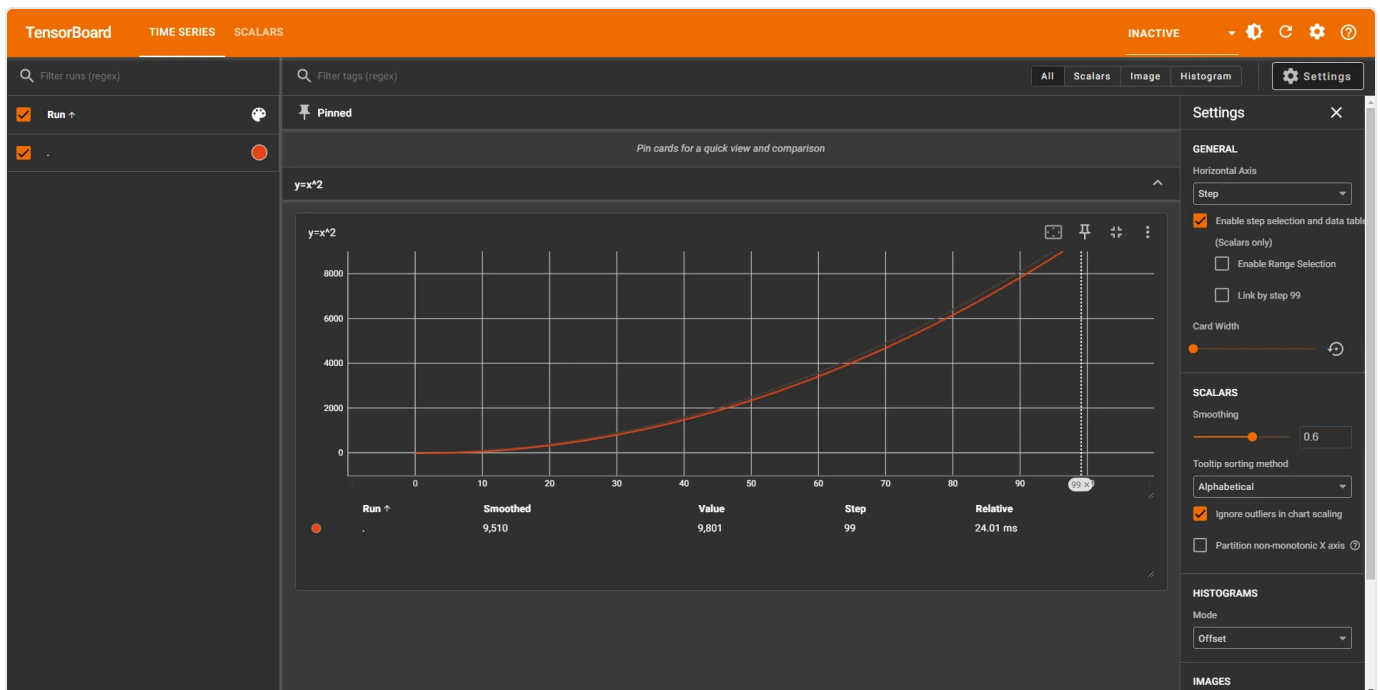
```
tensorboard --logdir=logs
```

4. 打开TensorBoard的Web界面，查看可视化信息

说明，同一tag下的数据被叠加

```
1 add_scalar(self,
2     tag,
3     scalar_value,
4     global_step=None,
5     walltime=None,
6     new_style=False,
7     double_precision=False,
8 )
```

【效果】



tensorboard 功能汇总

- Scalars：用于可视化训练过程中的标量数据，如损失函数值、准确率等。
- Graphs：用于可视化神经网络模型的计算图结构。
- Histograms：用于可视化张量的分布情况，如权重、偏置项等。
- Images：用于可视化图像数据。
- Embeddings：用于可视化高维数据的降维结果。
- Profiler：用于性能分析和调试。
- Projector：用于可视化嵌入向量。

【常用函数】

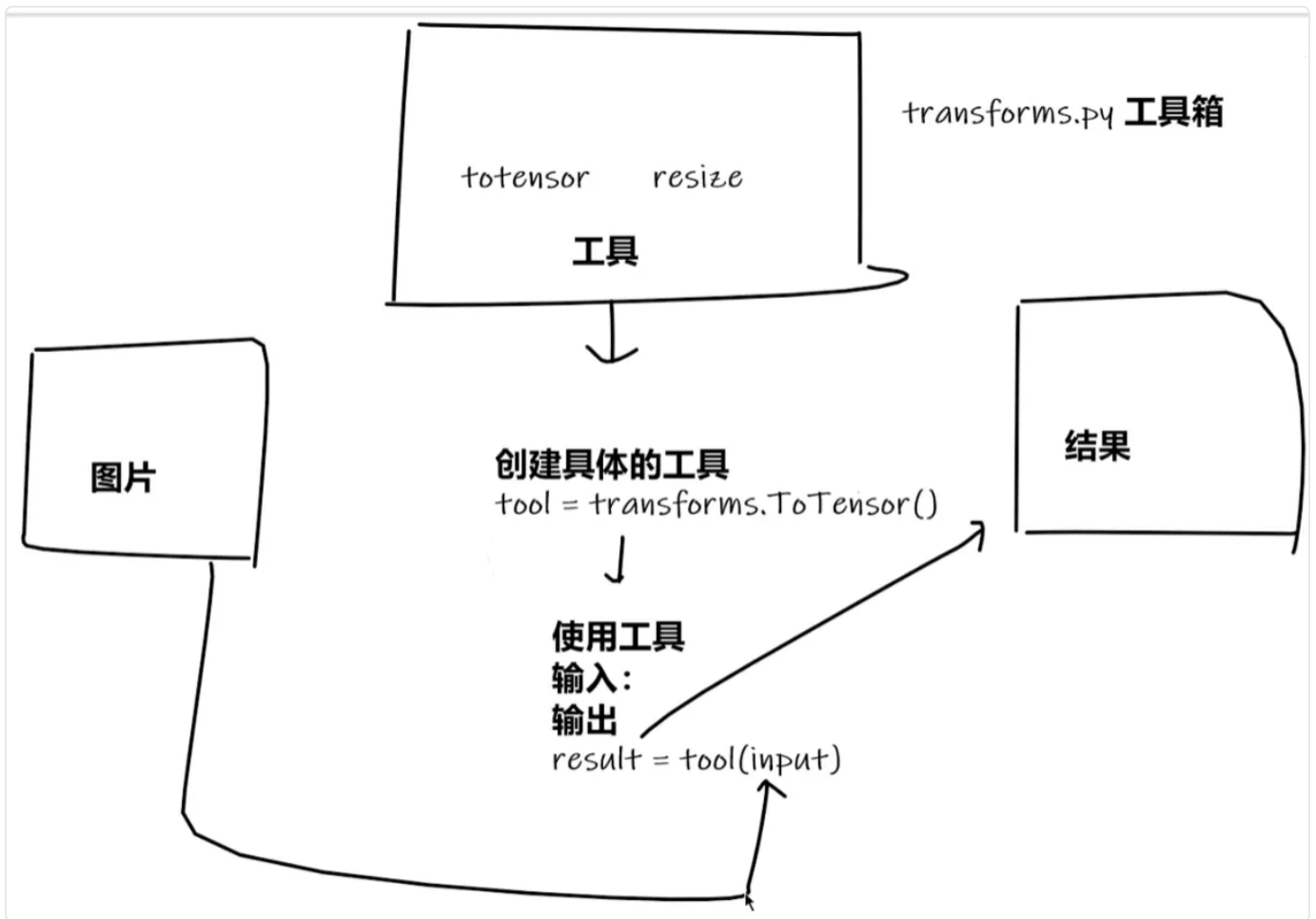
- `SummaryWriter`：用于创建一个 TensorBoard 日志文件，将需要可视化的数据写入到日志文件中。■
- `add_scalar`：用于添加标量数据到 TensorBoard 中，例如损失函数值、准确率等。■
- `add_graph`：用于添加计算图到 TensorBoard 中，可视化神经网络模型的结构。
- `add_histogram`：用于添加张量的直方图到 TensorBoard 中，可视化参数的分布情况。
- `add_image`：用于添加图像数据到 TensorBoard 中，可视化图像。■
- `add_embedding`：用于添加嵌入向量到 TensorBoard 中，进行降维可视化

transforms

用于对数据预处理■ | [transforms举例](#)

`transforms` 是一个用于对数据进行预处理和数据增强的模块。它包含了一系列用于处理图像、文本、音频等数据的转换函数，可以在数据加载过程中动态地对输入数据进行变换，以满足模型训练的需求。

在计算机视觉任务中，`transforms`模块通常用于对图像数据进行预处理和增强，包括裁剪、缩放、旋转、翻转、归一化等操作。通过对输入数据进行适当的变换，可以提高模型的鲁棒性和泛化能力，同时还可以减少模型的过拟合风险。



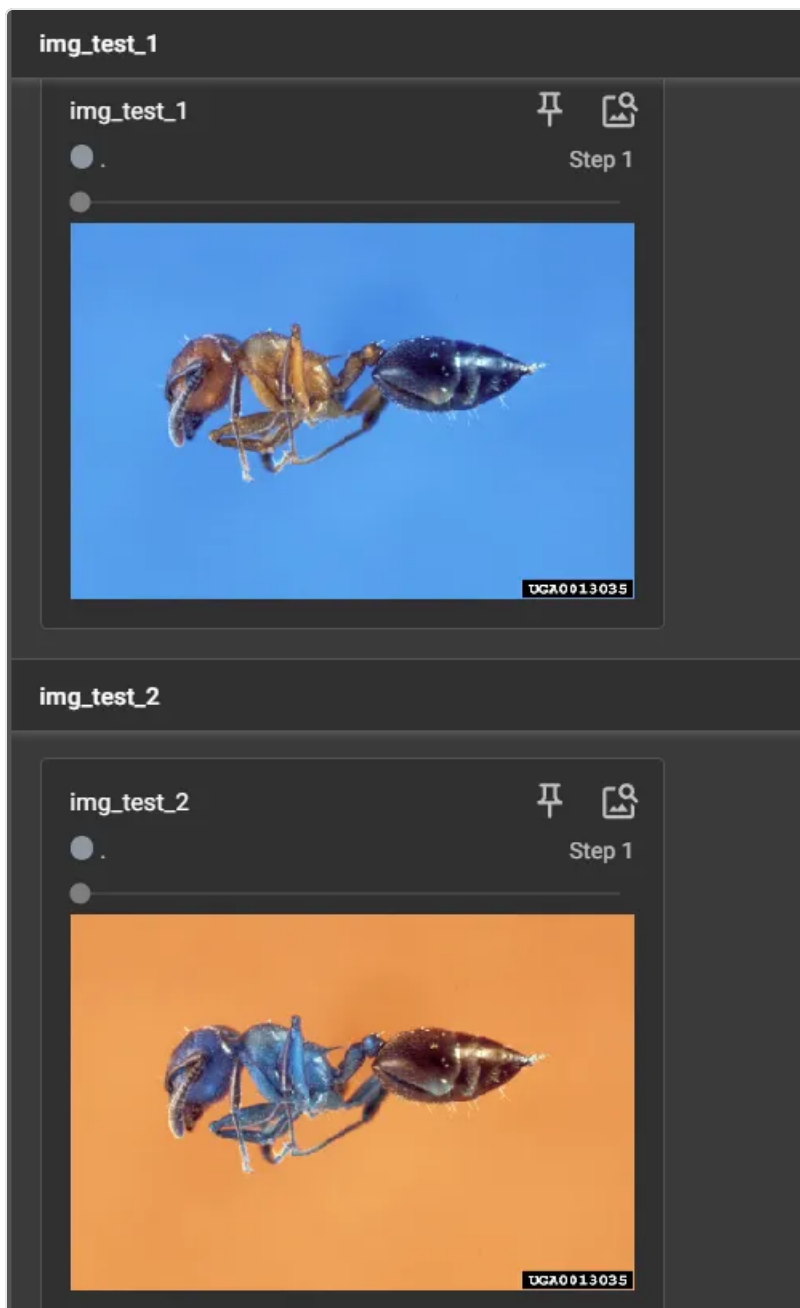
形象化transfroms使用

【举例，将PIL导入的img转换为tensor，并在tensorboard展示】


```

1  from PIL import Image
2  from torch.utils.tensorboard import SummaryWriter
3  from torchvision import transforms
4  import cv2
5
6  """对数据预处理"""
7
8  img_path = "dataset1/train/ants/0013035.jpg"
9  img = Image.open(img_path) #利用PIL
10
11
12  """transforms 用法"""
13  tensor_trans = transforms.ToTensor()
14  tensor_img = tensor_trans(img)
15  print(tensor_img)
16
17  """Tensor是神经网络所需的数据类型，包含需要的属性"""
18
19  cv_img = cv2.imread(img_path) #利用opencv
20  tensor_img_cv = tensor_trans(cv_img)
21  # 同一张图片，由不同格式转移到tensor
22
23
24  """使用Tensor类型传入 SummaryWriter展示"""
25  writer = SummaryWriter("logs_img")
26  writer.add_image("img_test_1", tensor_img, 1)
27  writer.add_image("img_test_2", tensor_img_cv, 1) # 变色
28
29  writer.close()

```



【transforms 常用类】

- ToTensor (将图片转换为tensor类型)
Convert a PIL Image or ndarray to **tensor** and scale the values accordingly.
- Normalize 归一化 (对每个通道上的像素变换)
$$\# \text{ output[channel]} = (\text{input[channel]} - \text{mean[channel]}) / \text{std[channel]}$$

torchvision官方数据集使用

- 视觉相关数据集

<https://pytorch.org/vision/stable/datasets.html#>

- CIFAR-10 数据集 示例

<https://pytorch.org/vision/stable/generated/torchvision.datasets.CIFAR10.html>

1. 获得官方数据集，并预处理为tensor
2. 使用tensorboard展示训练集中的图片

```
1  import torchvision
2  # 包含视觉相关的数据集
3  from torch.utils.tensorboard import SummaryWriter
4
5  dataset_transform=torchvision.transforms.Compose([
6      torchvision.transforms.ToTensor()
7  ])
8
9  """ https://pytorch.org/vision/stable/generated/torchvision.datasets.CIFAR
10  10.html """
11  train_dataset=torchvision.datasets.CIFAR10("./dataset_CIFAR10",train=True,
12      transform=dataset_transform,download=True)
13  test_dataset=torchvision.datasets.CIFAR10("./dataset_CIFAR10",train=False,
14      transform=dataset_transform,download=True)
15
16  # print(train_dataset[0])
17
18  wirter=SummaryWriter("p10")
19  for i in range(len(train_dataset)):
20      img,label=train_dataset[i]
21      wirter.add_image("train_dataset",img,i)
22  wirter.close()
```

Dataloader使用

<https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader>

? dataset与dataloader分别是什么？有什么区别？

1. Dataset

表示数据集，可以包含多个样本

- Dataset是一个抽象类，用于表示数据集。它定义了数据集的基本操作，包括数据的加载、预处理和索引等
- 你可以创建自定义的Dataset类，继承自torch.utils.data.Dataset，并实现 `__len__()` 和 `__getitem__()` 方法，以便能够对数据集进行索引和加载——需要重写2个方法

2. Dataloader

训练模型时，用于**加载**训练数据集

- DataLoader是一个用于批量加载数据的类，它封装了Dataset并提供了多线程 数据加载、数据打乱、数据批处理等功能。
- 你可以创建一个DataLoader对象，将Dataset作为参数传入，然后通过设置batch_size、shuffle等参数来对数据进行批处理和打乱。

```

1  import torchvision
2  from torch.utils.data import DataLoader
3
4  train_dataset = torchvision.datasets.CIFAR10("./dataset_CIFAR10", train=True,
5                                              transform=torchvision.transforms.ToTensor(), download=True)
6
7  """DataLoader参数说明 https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader"""
8  test_loader = DataLoader(dataset=train_dataset, batch_size=4, shuffle=True, num_workers=0, drop_last=False)
9
10 # DataLoader不可索引，仅可迭代
11 for data in test_loader:
12     imgs, targets = data
13     print(imgs.shape)
14     print(targets)
15
16 """DataLoader总结
17 相当于对DataSet进行了一步打包，和预处理
18 """

```

nn.Model

(Neural Network 简单介绍入门)

简单理解nn

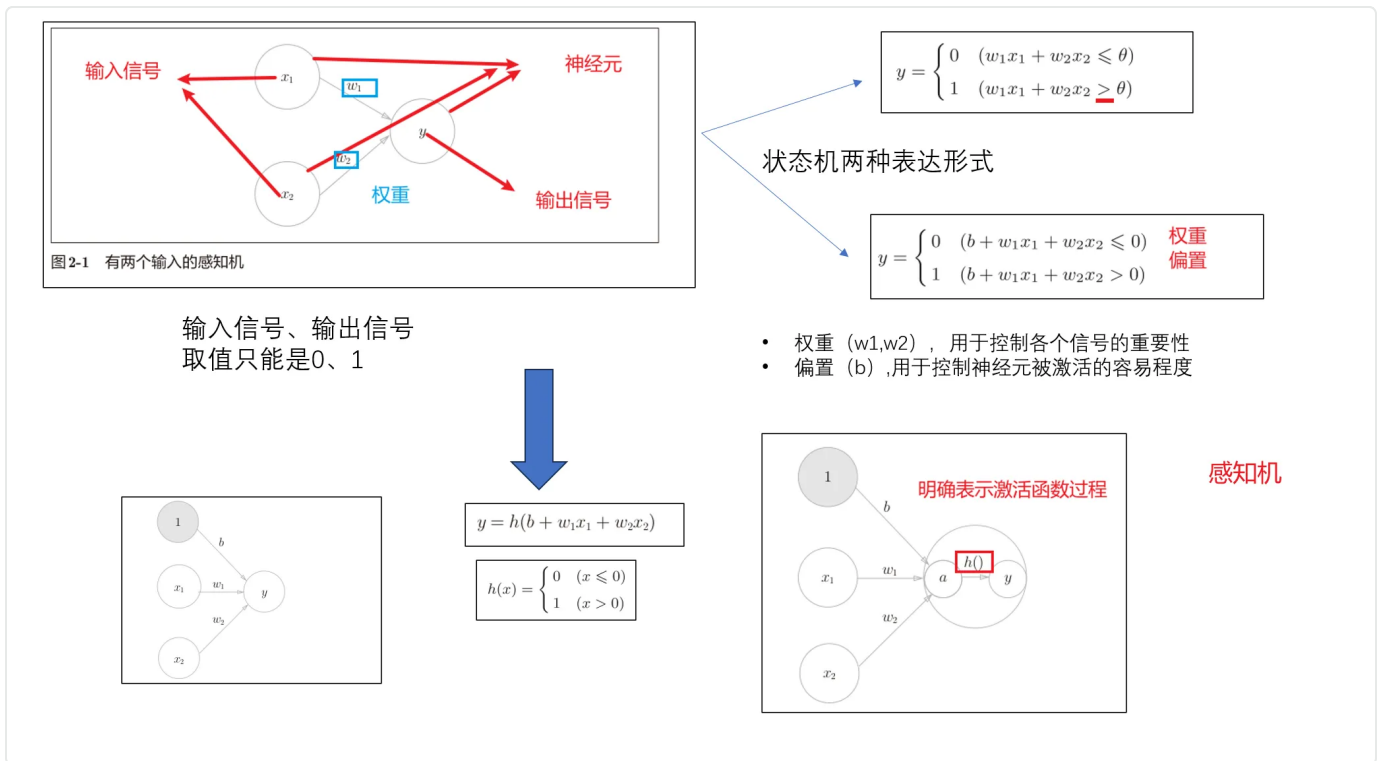
神经元(节点)，由感知机引入。

是一种由神经元（或称为节点）组成的计算模型，它受到人类大脑中神经元之间相互连接的启发。神经网络由多层神经元组成，分为输入层、隐藏层和输出层，每一层都由多个神经元组成。神经网络的基本单位是神经元，它接收来自前一层的输入信号，进行加权求和并通过激活函数处理，然后将输出传递给下一层。

【重要概念】

1. **输入层 (Input Layer)**：接收输入数据的层，每个输入神经元对应输入数据的一个特征。
2. **隐藏层 (Hidden Layer)**：位于输入层和输出层之间的层，用于提取输入数据的特征并进行非线性变换。神经网络可以包含多个隐藏层，每个隐藏层可以包含多个神经元。
3. **输出层 (Output Layer)**：产生神经网络的输出结果的层，通常对隐藏层的输出进行加权求和并通过激活函数处理，得到最终的输出结果。
4. **权重 (Weights)**：连接神经元之间的边上的参数，用于调整输入信号的重要性。
5. **偏置 (Bias)**：每个神经元都有一个偏置参数，用于调整神经元的激活阈值，影响神经元是否被激活。
6. **激活函数 (Activation Function)**：对神经元的输入进行非线性变换的函数，常见的激活函数包括Sigmoid、ReLU、Tanh等。
7. **损失函数 (Loss Function)**：衡量神经网络输出与真实标签之间的差异的函数，用于评估模型的性能。
8. **反向传播 (Backpropagation)**：一种训练神经网络的方法，通过计算损失函数对模型参数的梯度，并利用梯度下降算法来更新模型参数，使得模型的预测结果逐渐接近真实标签。

<https://pytorch.org/docs/stable/nn.html>——官方文档



神经网络

卷积运算

【卷积是什么】输出 = 输入 * 系统

卷积操作可以看作是一种滤波器在输入数据上滑动并与输入数据进行逐元素相乘后求和的过程。具体而言，对于二维输入数据（如图像），卷积操作可以通过一个称为卷积核（或滤波器）的小矩阵来实现，卷积核的大小通常是3x3或5x5等。卷积核在输入数据上滑动，对每个位置的局部区域与卷积核进行相乘并求和，得到输出数据的一个值。如下图：

1	2	0	3	1
0	1	2	3	1
1	2	1	0	1
5	2	3	1	0
2	1	0	2	1

输入图像
(5x5)

Stride=1

10	12	12
18	16	16
12	9	3

卷积后的输出

卷积核
(3x3)

遍历第一行结果

$$1+4+0+0+1+0+2+2+0=10$$

$$2+0+3+0+2+0+4+1=12$$

$$0+6+1+0+3+0+2+0+0=12$$


```

1 input = torch.tensor([[1, 2, 0, 3, 1],
2                        [0, 1, 2, 3, 1],
3                        [1, 2, 1, 0, 0],
4                        [5, 2, 3, 1, 1],
5                        [2, 1, 0, 1, 1]])
6
7 kernel = torch.tensor([[1, 2, 1],
8                        [0, 1, 0],
9                        [2, 1, 0]])
10 input = torch.reshape(input, (1, 1, 5, 5))
11 kernel = torch.reshape(kernel, (1, 1, 3, 3))
12
13 output = m.conv2d(input, kernel, stride=1)
14 print(output)
15
16 """
17 tensor([[[[10, 12, 12],
18          [18, 16, 16],
19          [12,  9,  3]]]])
20 """

```

【conv2d】对由多个输入平面组成的输入图像应用 2D 卷积

`torch.nn.functional.conv2d(input, weight, bias=None, stride=1, padding=0, dilation=1, groups=1)`

卷积操作的输入形式:

*input*输入张量的形状通常为 `(batch_size, channels, height, width)`

- `batch_size` 是批量大小，表示一次处理的样本数量。
- `channels` 是通道数，表示输入数据的通道数量，例如彩色图像有 RGB 三个通道。
- `height` 是图像的高度，表示图像的行数。
- `width` 是图像的宽度，表示图像的列数。

*weight*卷积核张量的形状通常为 `(out_channels, in_channels, kernel_height, kernel_width)`

- `out_channels` 是输出通道数，表示卷积核的个数。
- `in_channels` 是输入通道数，表示每个卷积核的输入通道数。
- `kernel_height` 是卷积核的高度，表示卷积核的行数。
- `kernel_width` 是卷积核的宽度，表示卷积核的列数。

卷积层-convolution

- [Convolution Layers](#)
- 【卷积运算——详细解释了stride、padding】

 [conv_arithmetic/README.md at master · vdumoulin/conv_arithmetic](#)

- 卷积操作的基本原理是将一个卷积核（或滤波器）与输入数据进行逐元素相乘，并将结果相加，得到输出特征图中的一个单个值。
 - 卷积核的参数是需要学习的，通过反向传播算法进行优化
- 卷积层，属于神经网络中（隐含层）的一种，用于提取输入数据中的局部特征，并保留空间结构

举例: `CLASS torch.nn.Conv2d`

神经网络中, 进行卷积计算, 只需要定义好卷积层即可。注意输入输出的数据尺寸要符合。

【卷积前后尺寸计算】: 

官方-`torch.nn.Conv2d`

Shape:

- Input: $(N, C_{in}, H_{in}, W_{in})$ or (C_{in}, H_{in}, W_{in})
- Output: $(N, C_{out}, H_{out}, W_{out})$ or $(C_{out}, H_{out}, W_{out})$, where

$$H_{out} = \left\lfloor \frac{H_{in} + 2 \times \text{padding}[0] - \text{dilation}[0] \times (\text{kernel_size}[0] - 1) - 1}{\text{stride}[0]} + 1 \right\rfloor$$

$$W_{out} = \left\lfloor \frac{W_{in} + 2 \times \text{padding}[1] - \text{dilation}[1] \times (\text{kernel_size}[1] - 1) - 1}{\text{stride}[1]} + 1 \right\rfloor$$

感性理解计算:

padding时, 是上, 下, 左, 右四个方向填充。根据卷积核的滑动, 来判断卷积后的尺寸。

池化层-pooling

- 官方-pooling layer

- 池化层, 与卷积层一样都属于隐藏层的一种
- 池化操作, 对输入特征图的局部进行聚合来降低空间维度

【池化】

Pooling定义：一种常用神经网络操作。通常用于减少特征图的空间维度，从而降低模型复杂度、减少计算量，并且提取特征的位置不变性。

池化操作通常在卷积神经网络的卷积层之后进行，它通过对输入特征图的局部区域进行聚合操作来降低特征图的空间维度。常用的池化操作有最大池化（Max Pooling）和平均池化（Average Pooling）两种。

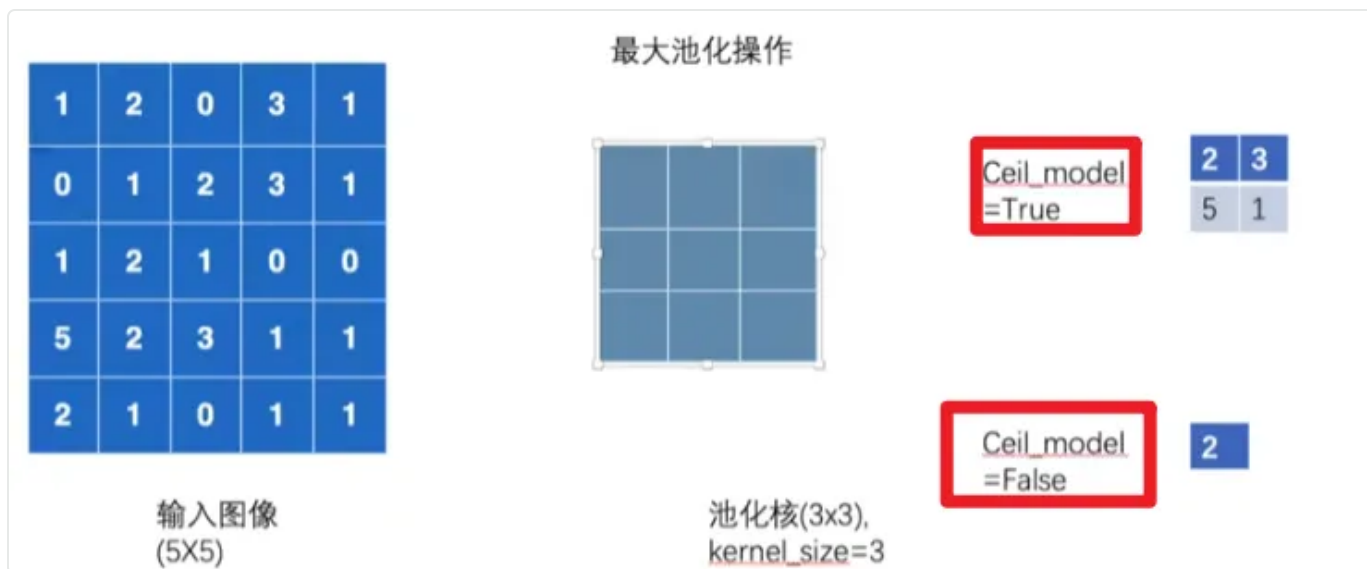
- 最大池化中，对于每个局部区域，取该区域内的最大值作为输出
- 平均池化中，对于每个局部区域，取该区域内的平均值作为输出

池化操作通常通过设置池化窗口大小和步幅来控制输出特征图的大小。

作用：降维、减少计算量、提取特征不变性

池化操作通常与卷积操作交替使用，用于逐步提取和降维输入数据的特征。

鲁棒性（Robustness）指的是系统在面对异常情况或不良输入时仍能保持稳定性和正确性的能力



非线性激活

- 【官方】non-linear-activations (weighted-sum、nonlinearity)

为什么是非线性？

非线性激活的

【定义】：

非线性激活函数（**Non-linear Activation Function**）是指一类函数，它们引入了非线性变换，使得神经网络能够学习和表示更加复杂的非线性关系。

在神经网络的每一层中，通常会在线性变换（如全连接层或卷积层）后添加一个非线性激活函数，以增加网络的表达能力。没有非线性激活函数的神经网络将由一系列线性变换组成，无法表示复杂的非线性关系，因此引入非线性激活函数可以让神经网络具备更强的表达能力。

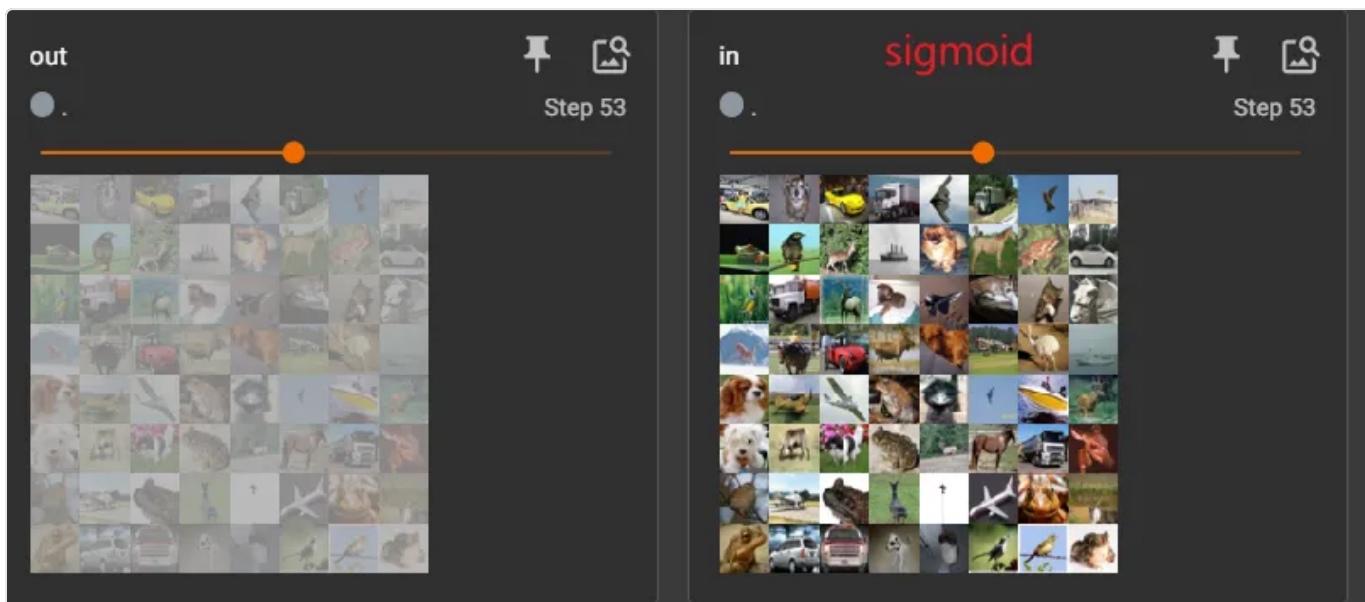
【常见】

- 1. Sigmoid 函数 (Sigmoid Function) :** $\sigma(x) = \frac{1}{1+e^{-x}}$
Sigmoid 函数将输入映射到 (0, 1) 的区间，适用于二分类问题，但存在梯度消失的问题。
- 2. 双曲正切函数 (Tanh Function) :** $\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
Tanh 函数将输入映射到 (-1, 1) 的区间，与 Sigmoid 函数类似但输出范围更广，也存在梯度消失的问题。
- 3. ReLU 函数 (Rectified Linear Unit) :** $\text{ReLU}(x) = \max(0, x)$
ReLU 函数在 x 大于 0 时返回 x ，否则返回 0，是目前应用最广泛的非线性激活函数之一，因为它可以加速收敛、减少梯度消失问题。
- 4. Leaky ReLU 函数:** $\text{LeakyReLU}(x) = \max(ax, x)$
Leaky ReLU 函数在 x 小于 0 时返回 $a \cdot x$ ，其中 a 是一个小的斜率，可以解决 ReLU 函数中负数部分导致的“神经元死亡”问题。
- 5. ELU 函数 (Exponential Linear Unit) :** $\text{ELU}(x) = \begin{cases} x & \text{if } x > 0 \\ \alpha(e^x - 1) & \text{if } x \leq 0 \end{cases}$
ELU 函数在 x 大于 0 时返回 x ，小于等于 0 时返回一个指数函数，有助于减少梯度消失问题，并且对于负值的响应更加平滑。

【作用】高了模型的表达能力和性能

【常见非线性激活函数】

- ReLU
- Sigmoid



归一化层 (Normalization Layers)

[normalization-layers](#)

【定义】

用于在神经网络中对输入数据或隐藏层的输出进行标准化处理，以使得数据的分布更稳定、更易于训练

通常在神经网络的隐藏层之后、激活函数之前使用。以确保输入数据或隐藏层的输出在经过激活函数之前已经进行了标准化处理

- 批量归一化 (Batch Normalization)
- 层归一化 (Layer Normalization)

【作用】加快训练速度、提高模型鲁棒性，改善激活函数效果

📌 归一化Normalization ≠ 正则化Regularization

- 归一化

- 对输入数据或隐藏层的输出进行标准化处理，使得数据的分布更稳定、更易于训练
- 正则化
 - 正则化主要是用于控制模型的复杂度，防止模型过拟合训练数据，提高模型的泛化能力

线性层-linear layers

linear-layers

$$\text{output} = \text{activation}(\text{input} \times \text{weight} + \text{bias})$$

CLASS torch.nn.Linear

📌 对过拟合的说明

过拟合（Overfitting）是指机器学习模型在训练过程中过度拟合训练数据，导致模型在训练集上表现良好，但在测试集或未见过的数据上表现较差的现象

Sequential

<https://pytorch.org/docs/stable/generated/torch.nn.Sequential.html#torch.nn.Sequential>

Sequential 是一个用于构建神经网络模型的容器。它允许你按顺序堆叠一系列的神经网络层，构建一个顺序模型。Sequential 容器提供了一个简单而直观的方式来构建神经网络，特别是对于那些由一系列层依次组成的简单模型。

——将一系列网络堆叠起来

损失函数-Loss Functions

简言之，计算预测结果与实际标签之间的差异

- Loss Functions

定义：于衡量模型预测结果与实际标签之间的差异或误差，并且是优化算法的目标函数之一。

常见损失函数

- 均方误差 (Mean Squared Error, MSE) / 平方差
- 交叉熵损失 (Cross-Entropy Loss)
- 对数损失 (Log Loss)
- Hinge损失

作用

评估模型性能、指导模型优化（优化算法目标是最小化损失函数）、指导模型学习（损失函数为模型提供反馈信号）

X:1, 2, 3
Y:1, 2, 5

$$L1loss = (0+0+2) / 3 = 0.6$$
$$MSE = (0+0+2^2) / 3 = 4/3 = 1.333$$

两种常见损失函数

output	target
选择 (10)	选择 (30)
填空 (10)	填空 (20)
解答 (20)	解答 (50)

$Loss = (30-10) + (20-10) + (50-10) = 70$

损失函数作用：

1. 计算实际输出和目标之间的差距
2. 为我们更新输出提供一定的依据（反向传播），grad

反向传播

定义：深度学习中用于训练神经网络的一种优化算法。它通过计算损失函数对模型参数的梯度，并使用梯度下降或其变种来更新参数，从而使得模型能够逐渐优化和拟合训练数据。

基本思想是利用链式法则 (Chain Rule) 来计算损失函数对模型参数的梯度

反向传播算法步骤

1. **前向传播 (Forward Propagation)**

从输入数据开始，通过神经网络前向计算，逐层计算出输出，并最终得到模型的预测结果

2. **计算损失函数 (compute Loss)**

将模型预测结果与真实标签值比较，计算损失函数，用于衡量模型预测结果与真实标签的差异

3. **反向传播 (Backward Propagation)**

从损失函数开始，利用链式法则计算损失函数对模型参数的梯度。通过逆向遍历神经网络计算图，将损失函数的梯度沿着网络的每一条边传播回去，并更新每个参数的值

4. **参数更新 (Update Parameters)**

根据计算得到的参数梯度，使用梯度下降或其变种的算法来更新模型的参数，使损失函数逐渐减小，模型逐渐优化和拟合训练数据。

关键：利用链式法则来计算梯度，从而实现了高效的参数更新和模型优化

链式法则 (Chain Rule)

微积分中的一个基本概念，它描述了复合函数的导数如何计算的规则。在数学上，如果一个函数是另一个函数的复合，那么它们的导数之间存在特定的关系

——微积分中的求导法则

$$h(x) = f(g(x)) \quad \text{复合函数导函数} \rightarrow \frac{dh}{dx} = \frac{dh}{du} \cdot \frac{du}{dx}$$

- 链式法则在反向传播算法中扮演了重要角色，通过计算损失函数对每个参数的梯度，并将梯度沿着网络反向传播，以便调整参数值以最小化损失函数
- 在神经网络中，每一层的输出都是通过对输入进行一系列的线性变换和非线性激活函数的计算得到的。通过链式法则，可以计算出损失函数对每一层输出的梯度，然后利用梯度下降法来更新网络参数，以减小损失函数
- 链式法则在反向传播算法中扮演了至关重要的角色，它使得我们能够有效地计算复杂网络结构中每个参数的梯度，并用于优化网络参数，从而提高神经网络的性能。

📌 **梯度grand**：梯度是损失函数相对于模型参数的偏导数

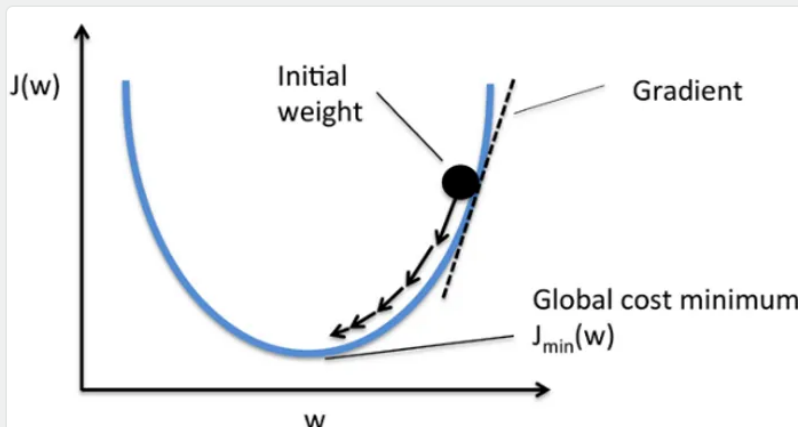
梯度告诉我们，如果我们稍微改变模型参数的值，损失函数会如何改变。梯度的方向指示了损失函数增加最快的方向，而梯度的大小则表示了损失函数增加的速率。

优化器 optimizer

- optimizer

定义：机器学习和深度学习中用于优化模型参数的算法

沿着损失函数的负梯度方向移动参数值，从而使得损失函数逐渐减小



作用：根据模型参数的梯度信息，更新模型参数，以最小化[损失函数](# 损失函数-Loss Functions)

工作流程：

1. 计算梯度

首先，使用反向传播算法 (backward) 计算损失函数相对于模型参数的**梯度**。

2. 更新参数

根据计算得到的梯度信息，使用**优化算法**来更新模型参数。

3. 重复迭代

重复以上步骤直到满足停止条件，例如达到一定的迭代次数或损失函数收敛到一个稳定值。

常见优化器：

- 随机梯度下降 (SGD)
- 动量优化 (Momentum)
- Adam
- Adagrad

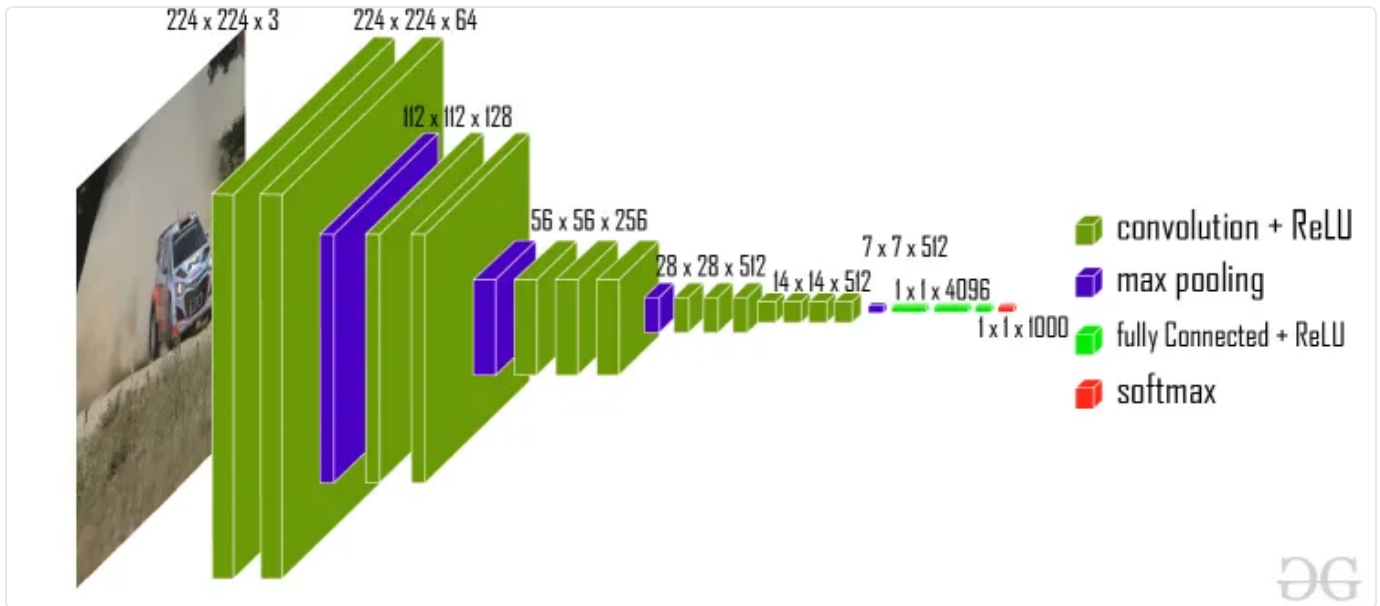
使用现有网络模型及修改

【B站视频】[使用+修改网络模型](#)

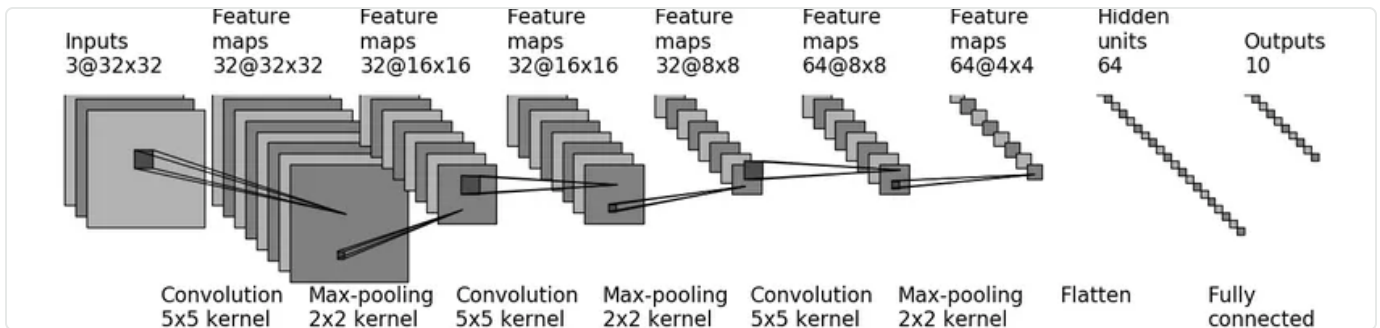
Downloading: "<https://download.pytorch.org/models/vgg16-397923af.pth>"

示例：vgg16 model

[computer-vision-vgg16](#)



示例：CIFAR 10 Model



完整模型训练

[\[demo\] train](#)

1. 准备数据集
2. 准备DataLoader
3. 创建网络模型
4. 损失函数

5. 优化器
6. 预处理-设置训练过程中的一些参数
7. 训练
 - a. 取数据
 - b. 前向传播计算
 - c. 计算loss
 - d. 反向传播
 - e. 更新参数-优化
8. 测试
9. 后处理-（保存模型|输出展示信息）

GPU训练

[\[demo\] trian_gpu](#)

【谷歌的服务，可以免费使用GPU训练，一周一定额度】

<https://colab.research.google.com/>

完整模型验证

使用训练好的模型来验证

如何看开源项目

[【up视频】看开源项目](#)

【完结撒花】

感谢up! activate love tuu love 我是土堆 BiliBili

感谢土堆 感谢老师，讲的太好了! 感谢up主 感谢土堆!!!

pytorch-CycleGAN-and-pix2pix

Code Issues (341) Pull requests (21) Actions Projects Wiki Security Insights

f13aab8148 pytorch-CycleGAN-and-pix2pix / options / base_options.py / </> Jump to

kagudkovArt separate save train and test options Latest commit c85b38e on 1 Apr 2019 History

9 contributors

136 lines (118 sloc) 7.87 KB

```
1 import argparse
2 import os
3 from util import util
4 import torch
5 import models
6 import data
7
8
9 class BaseOptions():
10     """This class defines options used during both training and test time.
11
12     It also implements several helper functions such as parsing, printing, and saving the options.
13     It also gathers additional options defined in <modify_commandline_options> functions in both dataset class and model class.
14 """
15
16 def __init__(self):
17     """Reset the class; indicates the class hasn't been initialized"""
18     self.initialized = False
19
20 def initialize(self, parser):
21     """Define the common options that are used in both train and test scripts.
22     # basic parameters
23     parser.add_argument('--dataroot', required=True, help='path to data root (should have subfolders trainA, trainB, valA, valB, etc)')
24     parser.add_argument('--name', type=str, default='experiment_name', help='name of the experiment. It decides where to store samples and models')
25
```

2024年3月6日 16点47分

完结撒花，感谢

完结撒花!!!