神经网络可视化工具

TensorBoard 是一组用于数据可视化的工具。它包含在流行的开源机器学习库 Tensorflow 中。TensorBoard 的主要功能包括：

- 可视化模型的网络架构
- 跟踪模型指标，如损失和准确性等
- 检查机器学习工作流程中权重、偏差和其他组件的直方图
- 显示非表格数据，包括图像、文本和音频
- 将高维嵌入投影到低维空间

安装 TensorBoard

TensorBoard 包含在 TensorFlow 库中，所以如果我们成功安装了 TensorFlow，我们也可以使用 TensorBoard。 要单独安装 TensorBoard 可以使用如下命令：

```
pip install tensorboard
```

需要注意的是：因为TensorBoard 依赖Tensorflow ， 所以会自动安装Tensorflow的最新版

启动 TensorBoard

要启动 TensorBoard，打开终端或命令提示符并运行：

```
tensorboard --logdir=<directory_name> --port=
```

将 directory_name 标记替换为保存数据的目录。 默认是“logs”。

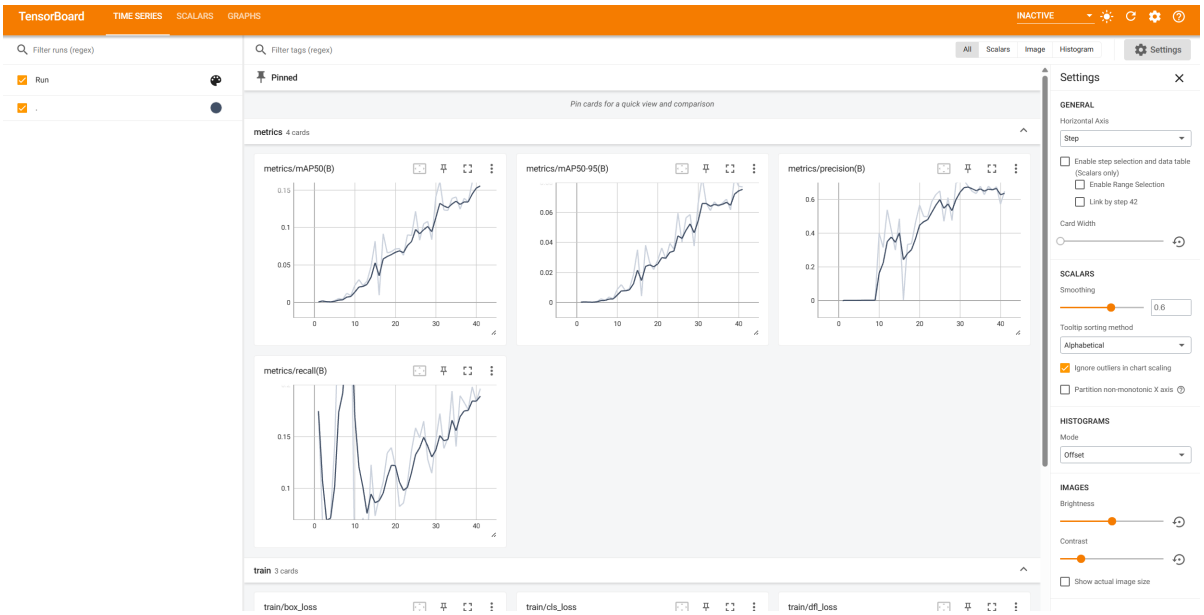
运行此命令后，我们将看到以下提示：

> Serving TensorBoard on localhost; to expose to the network, use a proxy or pass –bind_allTensorBoard 2.2.0 at http://localhost:6006/ (Press CTRL+C to quit)

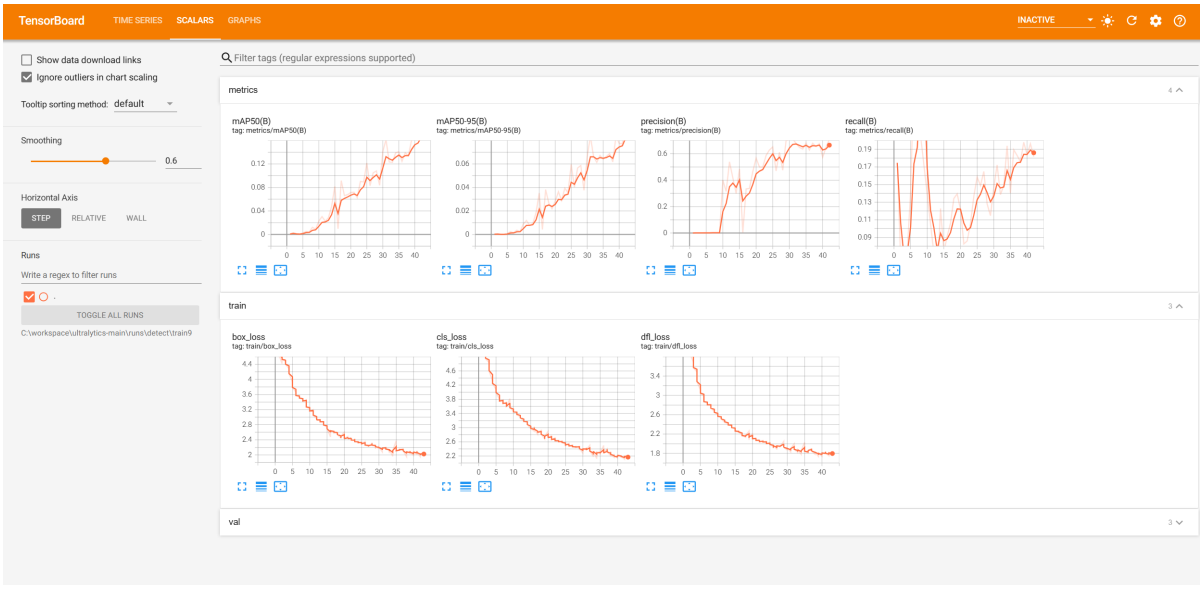
这说明 TensorBoard 已经成功上线。 我们可以用浏览器打开[http://localhost:6006/]查看。

TIME SERIES

主要用于将神经网络训练过程中的acc（训练集准确率）val_acc（验证集准确率），loss（损失值），weight（权重）等等变化情况绘制成折线图。

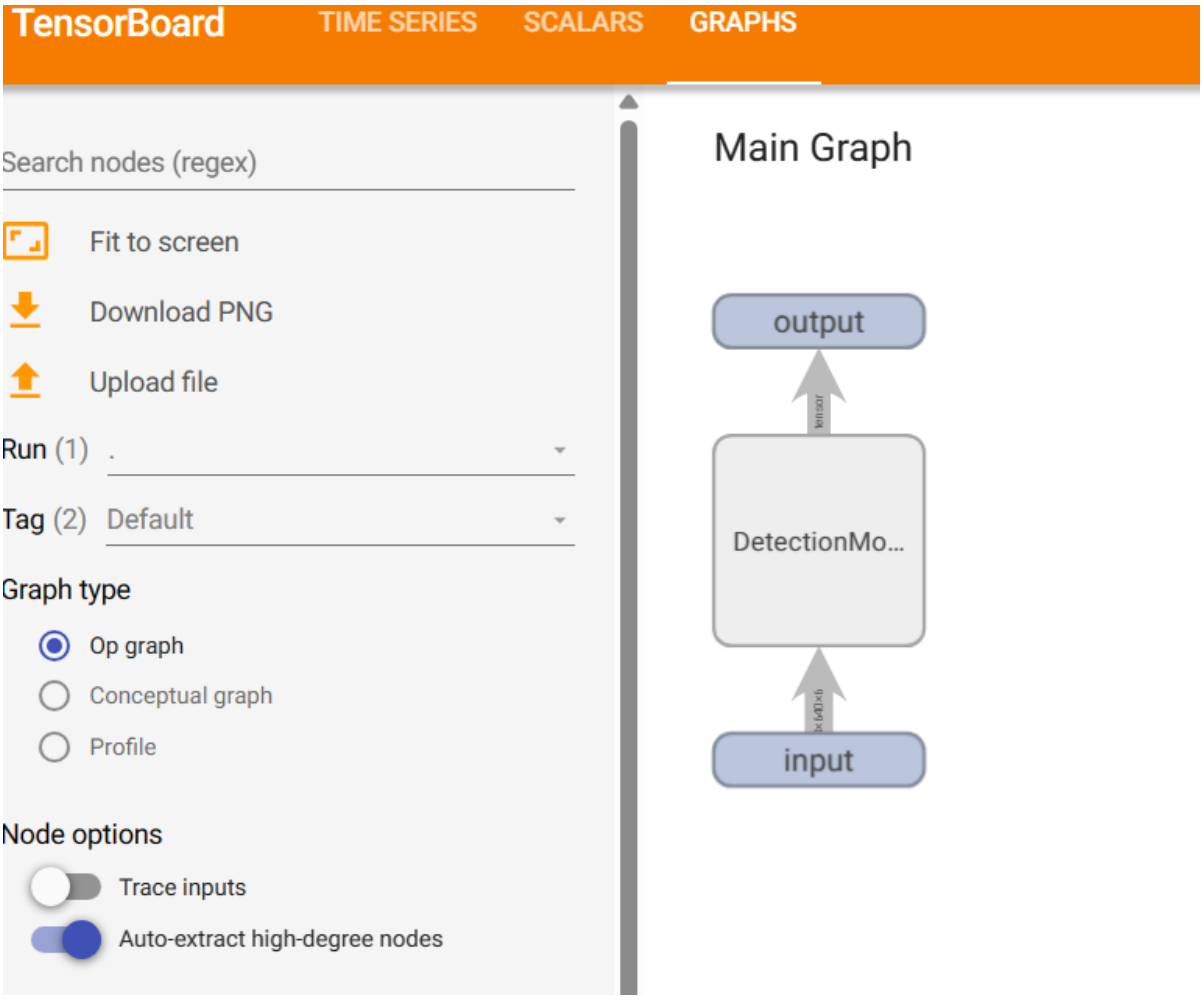


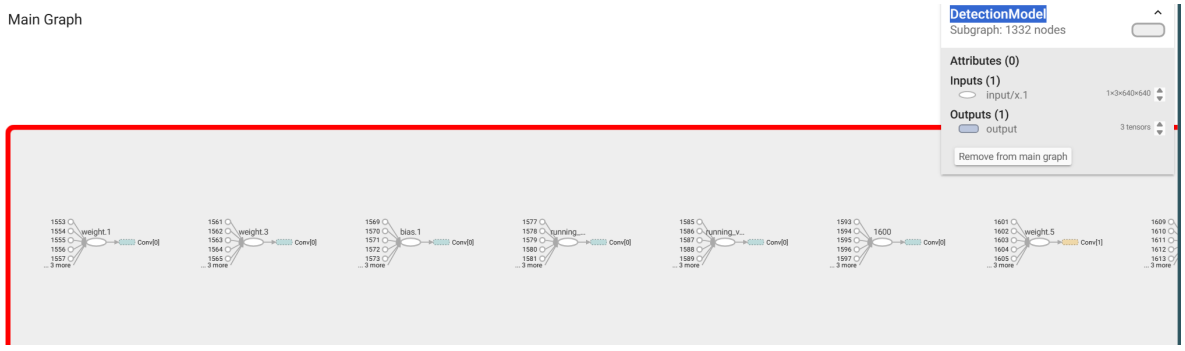
SCALARS



GRAPHS

可视化神经网络模型





手写数字识别

下载并处理数据集

数据集对于模型训练非常重要，好的数据集可以有效提高训练精度和效率。示例中用到的MNIST数据集是由10类28*28的灰度图片组成，训练数据集包含60000张图片，测试数据集包含10000张图片。

```
from mindvision.dataset import Mnist

# 下载并处理MNIST数据集
download_train = Mnist(path="../mnist", split="train", batch_size=32,
repeat_num=1, shuffle=True, resize=32, download=True)

download_eval = Mnist(path="../mnist", split="test", batch_size=32, resize=32,
download=True)

dataset_train = download_train.run()
dataset_eval = download_eval.run()
```

参数说明：

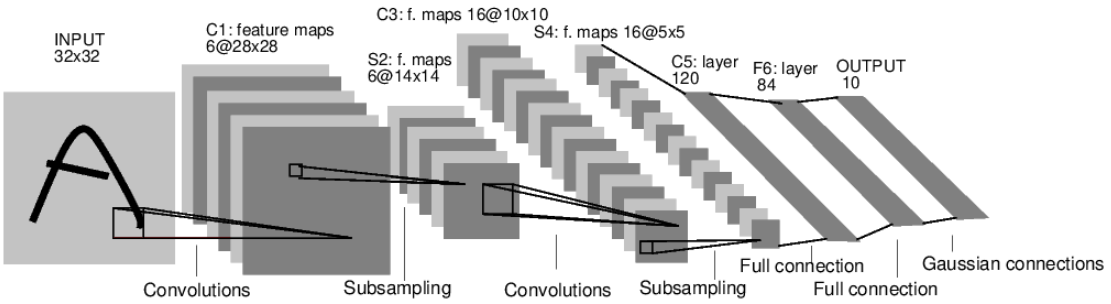
- path：数据集路径。
- split：数据集类型，支持train、 test、infer，默认为train。
- batch_size：每个训练批次设定的数据大小，默认为32。
- repeat_num：训练时遍历数据集的次数，默认为1。
- shuffle：是否需要将数据集随机打乱（可选参数）。
- resize：输出图像的图像大小，默认为32*32。
- download：是否需要下载数据集，默认为False。

下载的数据集文件的目录结构如下：

```
./mnist/
├── test
|   ├── t10k-images-idx3-ubyte
|   └── t10k-labels-idx1-ubyte
└── train
    ├── train-images-idx3-ubyte
    └── train-labels-idx1-ubyte
```

创建模型

按照LeNet的网络结构，LeNet除去输入层共有7层，其中有2个卷积层，2个子采样层，3个全连接层。



定义网络模型如下：

```
from mindvision.classification.models import lenet

network = lenet(num_classes=10, pretrained=False)
```

定义损失函数和优化器

要训练神经网络模型，需要定义损失函数和优化器函数。

- 损失函数这里使用交叉熵损失函数`SoftmaxCrossEntropyWithLogits`。
- 优化器这里使用`Momentum`。

```
import mindspore.nn as nn

# 定义损失函数
net_loss = nn.SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')

# 定义优化器函数
net_opt = nn.Momentum(network.trainable_params(), learning_rate=0.01,
momentum=0.9)
```

训练及保存模型

在开始训练之前，MindSpore需要提前声明网络模型在训练过程中是否需要保存中间过程和结果，因此使用`ModelCheckpoint`接口用于保存网络模型和参数，以便进行后续的Fine-tuning（微调）操作。

```
from mindspore.train.callback import ModelCheckpoint, CheckpointConfig

# 设置模型保存参数，模型训练保存参数的step为1875
config_ck = CheckpointConfig(save_checkpoint_steps=1875, keep_checkpoint_max=10)

# 应用模型保存参数
ckpoint = ModelCheckpoint(prefix="lenet", directory="./lenet", config=config_ck)
```

通过MindSpore提供的`model.train`接口可以方便地进行网络的训练，`LossMonitor`可以监控训练过程中`loss`值的变化。

```
from mindvision.engine.callback import LossMonitor
from mindspore.train import Model

# 初始化模型参数
model = Model(network, loss_fn=net_loss, optimizer=net_opt, metrics={'accuracy'})

# 训练网络模型，并保存为lenet-1_1875.ckpt文件
model.train(10, dataset_train, callbacks=[ckpoint, LossMonitor(0.01, 1875)])
```

```
model.train(10, dataset_train, callbacks=[ckpoint, LossMonitor(0.01, 1875)])
Executed at 2024.04.25 15:17:18 in 56s 733ms

Epoch:[  0/ 10], step:[ 1875/ 1875], loss:[0.287/0.409], time:5.266 ms, lr:0.01000
Epoch time: 5733.385 ms, per step time: 3.058 ms, avg loss: 0.409
Epoch:[  1/ 10], step:[ 1875/ 1875], loss:[0.292/0.064], time:5.933 ms, lr:0.01000
Epoch time: 5476.433 ms, per step time: 2.921 ms, avg loss: 0.064
Epoch:[  2/ 10], step:[ 1875/ 1875], loss:[0.002/0.044], time:5.571 ms, lr:0.01000
Epoch time: 5468.797 ms, per step time: 2.917 ms, avg loss: 0.044
Epoch:[  3/ 10], step:[ 1875/ 1875], loss:[0.044/0.034], time:5.601 ms, lr:0.01000
Epoch time: 5912.776 ms, per step time: 3.153 ms, avg loss: 0.034
Epoch:[  4/ 10], step:[ 1875/ 1875], loss:[0.000/0.027], time:5.773 ms, lr:0.01000
Epoch time: 6170.161 ms, per step time: 3.291 ms, avg loss: 0.027
Epoch:[  5/ 10], step:[ 1875/ 1875], loss:[0.027/0.023], time:5.398 ms, lr:0.01000
Epoch time: 5865.330 ms, per step time: 3.128 ms, avg loss: 0.023
Epoch:[  6/ 10], step:[ 1875/ 1875], loss:[0.001/0.021], time:5.702 ms, lr:0.01000
Epoch time: 5470.310 ms, per step time: 2.917 ms, avg loss: 0.021
Epoch:[  7/ 10], step:[ 1875/ 1875], loss:[0.000/0.018], time:5.484 ms, lr:0.01000
Epoch time: 5585.703 ms, per step time: 2.979 ms, avg loss: 0.018
Epoch:[  8/ 10], step:[ 1875/ 1875], loss:[0.027/0.017], time:5.813 ms, lr:0.01000
Epoch time: 5515.379 ms, per step time: 2.942 ms, avg loss: 0.017
Epoch:[  9/ 10], step:[ 1875/ 1875], loss:[0.001/0.014], time:4.692 ms, lr:0.01000
Epoch time: 5507.738 ms, per step time: 2.937 ms, avg loss: 0.014
```

训练过程中会打印loss值，loss值会波动，但总体来说loss值会逐步减小，精度逐步提高。每个人运行的loss值有一定随机性，不一定完全相同。

通过模型运行测试数据集得到的结果，验证模型的泛化能力：

1. 使用`model.eval`接口读入测试数据集。
2. 使用保存后的模型参数进行推理。

```
acc = model.eval(dataset_eval)

print("{}".format(acc))
```

```
Executed at 2024.04.25 15:18:12 in 502ms

{'accuracy': 0.9889823717948718}
```

可以在打印信息中看出模型精度数据，示例中精度数据达到95%以上，模型质量良好。随着网络迭代次数增加，模型精度会进一步提高。

加载模型

```
from mindspore import load_checkpoint, load_param_into_net

# 加载已经保存的用于测试的模型
param_dict = load_checkpoint("./lenet/lenet-1_1875.ckpt")
# 加载参数到网络中
load_param_into_net(network, param_dict)
```

```
Out 8    ([],
          ['global_step',
           'learning_rate',
           'momentum',
           'moments.backbone.conv1.weight',
           'moments.backbone.conv2.weight',
           'moments.backbone.fc1.weight',
           'moments.backbone.fc1.bias',
           'moments.backbone.fc2.weight',
           'moments.backbone.fc2.bias',
           'moments.backbone.fc3.weight',
           'moments.backbone.fc3.bias'])
```

验证模型

我们使用生成的模型进行单个图片数据的分类预测，具体步骤如下：

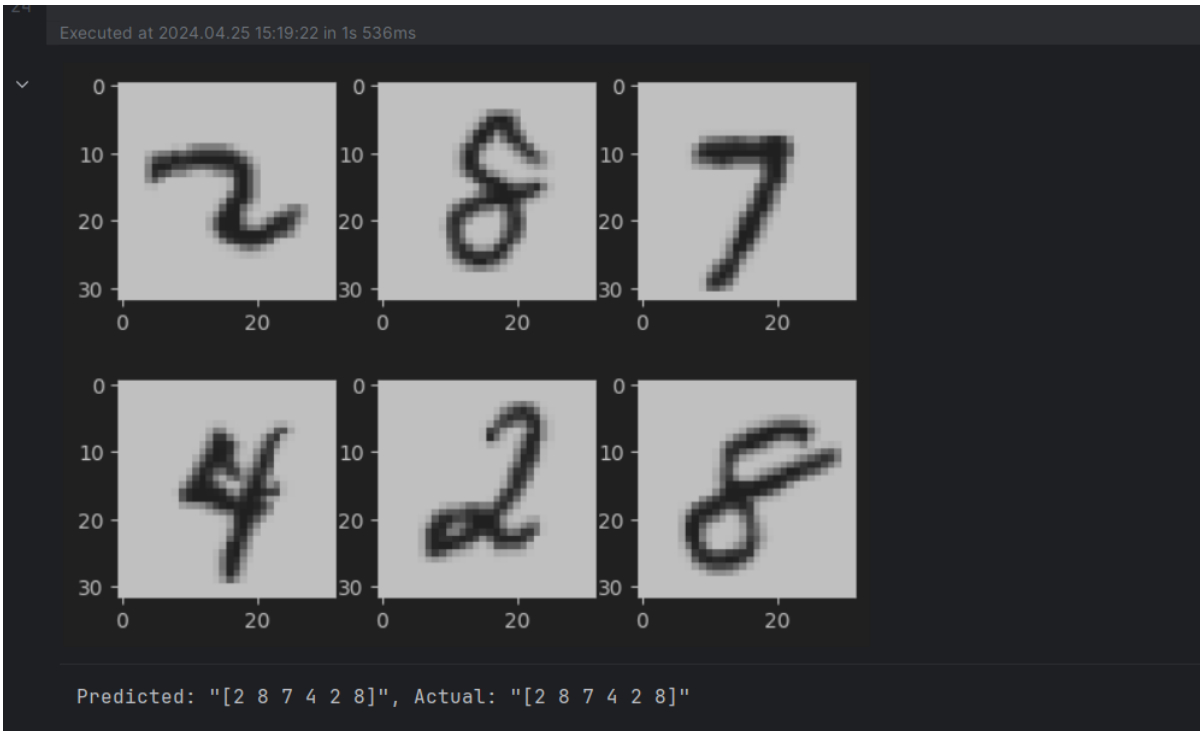
```
import numpy as np
from mindspore import Tensor
import matplotlib.pyplot as plt

mnist = Mnist("../mnist", split="train", batch_size=6, resize=32)
dataset_infer = mnist.run()
ds_test = dataset_infer.create_dict_iterator()
data = next(ds_test)
images = data["image"].asnumpy()
labels = data["label"].asnumpy()

plt.figure()
for i in range(1, 7):
    plt.subplot(2, 3, i)
    plt.imshow(images[i-1][0], interpolation="None", cmap="gray")
plt.show()

# 使用函数model.predict预测image对应分类
output = model.predict(Tensor(data['image']))
predicted = np.argmax(output.asnumpy(), axis=1)

# 输出预测分类与实际分类
print(f'Predicted: "{predicted}", Actual: "{labels}"')
```



从上面的打印结果可以看出，预测值与目标值完全一致。