模型框架转换

```
git clone --recursive https://github.com/Tencent/ncnn/
```

创建 build 文件夹，编译 ncnn 源码后，ncnn 内置模型转换工具位于 ncnn/build/tools 路径下。

编译方法参见 https://github.com/Tencent/ncnn/wiki/how-to-build

Darknet->ncnn

使用 ncnn/build/tools/darknet 路径下./darknet2ncnn 工具进行转换

Darknet2ncnn 用法：

./darknet2ncnn [*.cfg] [*.weights] [*.param] [*.bin] [merge_output]

参数说明：

[*.cfg]	输入 Darknet 模型的.cfg 文件
[*.weights]	输入 Darknet 模型的.weights 文件
[*.param]	输出 NCNN 模型的.param 文件
[*.bin]	输出 NCNN 模型的.bin 文件
[merge_output]	合并所有输出 yolo 层为一个，默认启用（default=1）

Caffe->ncnn

使用 ncnn/build/tools/caffe 路径下./caffe2ncnn 工具进行转换

Caffe2ncnn 用法：

./caffe2ncnn [*.prototxt] [*.caffemodel] [*.param] [*.bin]

参数说明：

[*.prototxt]	输入 Caffe 模型的.prototxt 文件
[*.caffemodel]	输入 Caffe 模型的.caffemodel 文件
[*.param]	输出 NCNN 模型的.param 文件
[*.bin]	输出 NCNN 模型的.bin 文件

**注意：ncnn 只支持转换新版的 caffe 模型，旧版 caffe 模型需要先转换成新版才能通过 caffe2ncnn 工具转换为 ncnn 支持的模型格式。

需要安装前置关联包：sudo apt install build-essential git cmake libprotobuf-dev protobuf-compiler libvulkan-dev vulkan-utils libopencv-dev 等。

Onnx->ncnn

1. 模型简化

安装 onnx-simplifier

pip3 install -U pip && pip3 install onnxsim

Onnxsim 用法：

python -m onnxsim [*.onnx] [* _sim.onnx]

参数说明：

[*.onnx]	输入 Onnx 模型的.onnx 文件
[* _sim.onnx]	输出 Onnx 模型的简化后.onnx 文件

2. 框架转换

使用 ncnn/build/tools/onnx 路径下./onnx2ncnn 工具进行转换

Onnx2ncnn 用法：

./onnx2ncnn [*_sim.onnx] [*.param] [*.bin]

参数说明：

[*_sim.onnx]	输入 Onnx 模型的简化后.onnx 文件
[*.param]	输出 NCNN 模型的.param 文件
[*.bin]	输出 NCNN 模型的.bin 文件

Mxnet2ncnn

使用 ncnn/build/tools/mxnet 路径下./mxnet2ncnn 工具进行转换

Mxnet2ncnn 用法：

./mxnet2ncnn [*.json] [*.params] [*.param] [*.bin]

参数说明：

[*.json]	输入 Mxnet 模型的.json 文件
[*.params]	输入 Mxnet 模型的.params 文件
[*.param]	输出 NCNN 模型的.param 文件
[*.bin]	输出 NCNN 模型的.bin 文件

Pytorch->ncnn

方法一：将 Pytorch 模型转换为 Onnx 格式输出，再使用 Onnxsim、Onnx2ncnn 工具进行转换

新版 Pytorch 已支持模型直接导出为 Onnx 框架格式，以 Pytroch 官方库中的 resnet18 推理模型为例，编写 export.py 如下：

```
import torch
import torchvision.models as models
import torch.onnx as onnx

# 加载预训练的 ResNet-18 模型
resnet = models.resnet18(pretrained=True)

# 将模型设置为评估模式
resnet.eval()

# 创建一个示例输入张量
dummy_input = torch.randn(1, 3, 224, 224)

# 使用 torch.onnx.export 函数导出模型为 ONNX 格式
onnx_file_path = "./resnet18.onnx"
onnx.export(resnet, dummy_input, onnx_file_path)

# 打印提示信息
print("ResNet-18 模型已成功导出为 ONNX 格式：", onnx_file_path)
```

在 Pytorch 环境下运行 export.py 即可在当前路径下生成.onnx 模型文件，参考上述 Onnx2ncnn 过程即可实现到 ncnn 框架格式转换。

方法二：使用 Pnnx 将 Pytorch 模型直接导出为 NCNN 格式

PyTorch Neural Network eXchange (PNNX)：是一种新型的 PyTorch 模型互用性开放标准。相比 ONNX，PNNX 具有可读且可编辑的表示形式，便于修改计算图或添加自定义操作符；PNNX 的操作符定义与 Pytorch 更加接近，避免了大量胶水操作，有助于提高推理效率。

1. 安装并编译 pnnx

在 ncnn/tools/pnnx 路径下新建 build 文件夹并进行编译或直接下载 pnnx 可执行文件（Releases · pnnx/pnnx (github.com)）。

2. Pytorch2libtorch

在 Pytorch 环境下将模型文件导出到 torchscript，以 resnet18 为例编写 export.py 如下：

```
import torch
import torchvision.models as models

# 加载预训练的 ResNet-18 模型
resnet = models.resnet18(pretrained=True)
resnet = resnet.eval()
```

```
# 创建一个示例输入张量
dummy_input = torch.rand(1, 3, 224, 224)

# 当跟踪引发错误时，可以尝试禁用检查
# mod = torch.jit.trace(net, x, check_trace=False)
mod = torch.jit.trace(net, x)

mod.save("resnet18.pt")
```

运行 export.py，得到 libtorch 框架下的模型文件 resnet18.pt。

3. Libtorch2ncnn

使用 pnnx 工具将 torchscript 模型转换为 ncnn 模型，命令如下：

./pnnx [*.pt] inputshape=[b, c, h, w]

参数说明：

[*.pt]	输入 Torchscript 模型的.pt 文件
[inputshape]	可选参数，用于解析模型图中的张量形状

使用 pnnx 工具后通常会生成以下 7 个文件：

*.pnnx.param	PNNX 图定义
*.pnnx.bin	PNNX 模型权重
*_pnnx.py	用于模型构建和权重初始化的 python 代码
*.pnnx.onnx	Onnx 格式的 PNNX 模型
*.ncnn.param	NCNN 图定义
*.ncnn.bin	NCNN 模型权重
*_ncnn.py	用于推理的 Pyncnn 代码