北京大学

软件与微电子学院

《开源软件开发基础及实践》

实验报告

题目：源代码阅读

姓　　名：张领

学　　号：2001210705

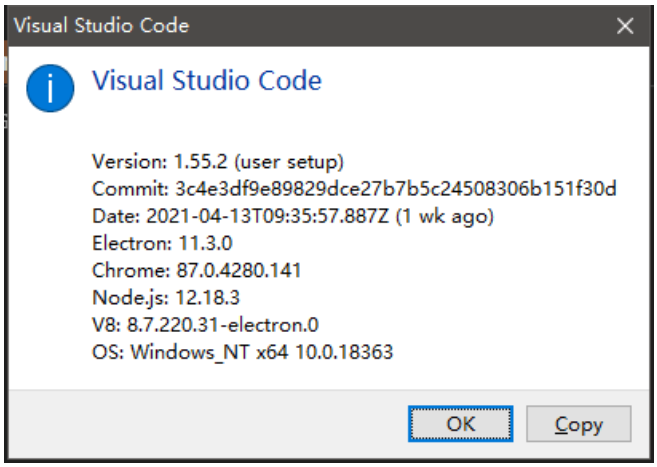
日　　期：2021.04.25

目录

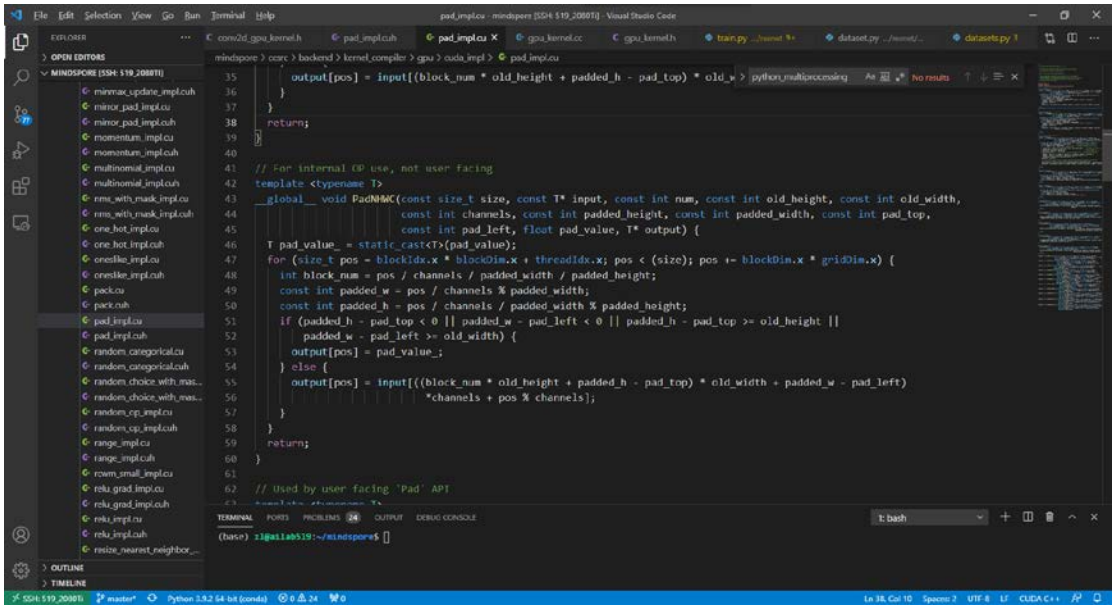
源代码阅读分析文档

使用的源代码阅读工具

使用 VSCODE 进行阅读源码，版本为 1.55.2，具体如下所示：



使用该工具的使用截图如下所示：



项目背景

MindSpore 是端边云全场景按需协同的华为自研 AI 计算框架，提供全场景统一 API，为全场景 AI 的模型开发、模型运行、模型部署提供端到端能力。

MindSpore 采用端-边-云按需协作分布式架构、微分原生编程新范式以及 AI Native 新执行模式，实现更好的资源效率、安全可信，同时降低行业 AI 开发门槛、释放昇腾芯片算力，助力普惠 AI。

MindSpore 具有开发态友好性：

- 自动微分，网络+算子统一编程，函数式/算法原生表达，反向网络算子自动生成
- 自动并行，模型自动切分实现最优效率的模型并行
- 自动调优，动态图+静态图同一套代码

MindSpore 具有运行态高效性：

- On-Device 执行，充分发挥昇腾大算力
- Pipeline 优化，最大化并行线性度
- 深度图优化，自适应 AI Core 算力和精度

MindSpore 具有部署态灵活性：

- 端-边-云按需协同计算，更好的保护隐私
- 端-边-云统一架构，实现一次开发，按需部署

使用 MindSpore 的优势：

- **简单的开发体验：**帮助开发者实现网络自动切分，只需串行表达就能实现并行训练，降低门槛，简化开发流程。
- **灵活的调试模式：**具备训练过程静态执行和动态调试能力，开发者通过变更一行代码即可切换模式，快速在线定位问题。
- **充分发挥硬件潜能：**最佳匹配昇腾处理器，最大程度地发挥硬件能力，帮助开发者缩短训练时间，提升推理性能。
- **全场景快速部署：**支持云、边缘和手机上的快速部署，实现更好的资源利用和隐私保护，让开发者专注于 AI 应用的创造。

代码结构

```
.
├── akg
├── cmake
│   └── external_libs
├── config
├── docker
│   ├── mindspore-cpu
│   │   ├── 0.1.0-alpha
│   │   ├── 0.2.0-alpha
│   │   ├── 0.3.0-alpha
│   │   ├── 0.5.0-beta
│   │   ├── 0.6.0-beta
│   │   ├── 0.7.0-beta
│   │   ├── 1.0.0
│   │   ├── 1.1.0
```

```
16. │   │   ├── devel
17. │   │   └── runtime
18. │   └── mindspore-gpu
19. │       ├── 0.1.0-alpha
20. │       ├── 0.2.0-alpha
21. │       ├── 0.3.0-alpha
22. │       ├── 0.5.0-beta
23. │       ├── 0.6.0-beta
24. │       ├── 0.7.0-beta
25. │       ├── 1.0.0
26. │       ├── 1.1.0
27. │       ├── devel
28. │       └── runtime
29. ├── docs
30. ├── graphengine
31. ├── include
32. │   └── api
33. │       └── ops
34. ├── mindspore
35. │   ├── ccsrc
36. │   │   ├── backend
37. │   │   ├── common
38. │   │   ├── cxx_api
39. │   │   ├── debug
40. │   │   ├── frontend
41. │   │   ├── minddata
42. │   │   ├── mindquantum
43. │   │   ├── pipeline
44. │   │   ├── profiler
45. │   │   ├── ps
46. │   │   ├── pybind_api
47. │   │   ├── runtime
48. │   │   ├── transform
49. │   │   ├── utils
50. │   │   └── vm
51. │   ├── common
52. │   ├── communication
53. │   ├── compression
54. │   │   ├── common
55. │   │   ├── export
56. │   │   └── quant
57. │   ├── core
58. │   │   ├── abstract
59. │   │   ├── base
```

```
60. |   |   ├── c_ops
61. |   |   ├── gvar
62. |   |   ├── ir
63. |   |   ├── load_mindir
64. |   |   ├── mindrt
65. |   |   ├── ops
66. |   |   └── utils
67. |   ├── dataset
68. |   |   ├── callback
69. |   |   ├── core
70. |   |   ├── datapreprocess
71. |   |   ├── engine
72. |   |   ├── text
73. |   |   ├── transforms
74. |   |   └── vision
75. |   ├── explainer
76. |   |   ├── benchmark
77. |   |   └── explanation
78. |   ├── _extends
79. |   |   ├── graph_kernel
80. |   |   ├── parallel_compile
81. |   |   ├── parse
82. |   |   └── remote
83. |   ├── graph_utils
84. |   |   └── python_pass
85. |   ├── lite
86. |   |   ├── examples
87. |   |   ├── include
88. |   |   ├── java
89. |   |   ├── micro
90. |   |   ├── minddata
91. |   |   ├── schema
92. |   |   ├── src
93. |   |   ├── test
94. |   |   └── tools
95. |   ├── mindrecord
96. |   |   ├── common
97. |   |   └── tools
98. |   ├── nn
99. |   |   ├── acc
100. |   |   ├── layer
101. |   |   ├── loss
102. |   |   ├── metrics
103. |   |   ├── optim
```

```
104. │   │   ├── probability
105. │   │   ├── sparse
106. │   │   └── wrap
107. │   ├── numpy
108. │   ├── offline_debug
109. │   ├── ops
110. │   │   ├── composite
111. │   │   ├── _grad
112. │   │   ├── operations
113. │   │   ├── _op_impl
114. │   │   └── _utils
115. │   ├── parallel
116. │   │   └── mpi
117. │   ├── profiler
118. │   │   ├── common
119. │   │   └── parser
120. │   ├── __pycache__
121. │   └── train
122. │       ├── callback
123. │       ├── summary
124. │       └── train_thor
125. ├── model_zoo
126. │   ├── community
127. │   ├── how_to_contribute
128. │   ├── official
129. │   │   ├── audio
130. │   │   ├── cv
131. │   │   ├── gnn
132. │   │   ├── lite
133. │   │   ├── nlp
134. │   │   ├── recommend
135. │   │   ├── rl
136. │   │   └── utils
137. │   ├── research
138. │   │   ├── audio
139. │   │   ├── cv
140. │   │   ├── hpc
141. │   │   ├── nlp
142. │   │   ├── recommend
143. │   │   └── rl
144. │   └── utils
145. │       ├── ascend_distributed_launcher
146. │       ├── cv_to_mindrecord
147. │       ├── graph_to_mindrecord
```

```
148. │       ├── hccl_tools
149. │       └── nlp_to_mindrecord
150. ├── scripts
151. │   └── map_dump_file_to_code
152. │       └── images
153. ├── test_homework
154. ├── tests
155. │   ├── mindspore_test_framework
156. │   │   ├── apps
157. │   │   ├── components
158. │   │   ├── pipeline
159. │   │   └── utils
160. │   ├── perf_test
161. │   │   ├── bert
162. │   │   ├── mind_expression_perf
163. │   │   └── mindrecord
164. │   ├── st
165. │   │   ├── auto_monad
166. │   │   ├── auto_parallel
167. │   │   ├── broadcast
168. │   │   ├── control
169. │   │   ├── cpp
170. │   │   ├── data_transfer
171. │   │   ├── dump
172. │   │   ├── dynamic_shape
173. │   │   ├── explainer
174. │   │   ├── export_and_load
175. │   │   ├── fusion
176. │   │   ├── gnn
177. │   │   ├── graph_kernel
178. │   │   ├── hcom
179. │   │   ├── heterogeneous_excutor
180. │   │   ├── high_grad
181. │   │   ├── host_device
182. │   │   ├── mem_reuse
183. │   │   ├── mix_precision
184. │   │   ├── model_zoo_tests
185. │   │   ├── nccl
186. │   │   ├── networks
187. │   │   ├── numpy_native
188. │   │   ├── ops
189. │   │   ├── probability
190. │   │   ├── profiler
191. │   │   ├── ps
```

```
192. │   │   ├── pynative
193. │   │   ├── quantization
194. │   │   ├── recompute
195. │   │   ├── summary
196. │   │   └── tbe_networks
197. │   ├── ut
198. │   │   ├── cpp
199. │   │   ├── data
200. │   │   └── python
201. │   └── vm_impl
202. └── third_party
203.     ├── patch
204.     │   ├── cppjieba
205.     │   ├── glog
206.     │   ├── grpc
207.     │   ├── icu4c
208.     │   ├── jpeg_turbo
209.     │   ├── projectq
210.     │   ├── sentencepiece
211.     │   └── sqlite
212.     └── securec
213.         ├── include
214.         └── src
215.
216. 228 directories
```

其中，docker 文件夹内存放的是已经搭建好的 MindSpore-Cpu，MindSpore-Gpu 的各种版本的 docker 环境；docs 内存放一些 MindSpore 的使用说明；include 内存放一些头文件，用来声明函数；scripts 内存放一些转换模型的脚本.sh 文件；model_zoo 内是 MindSpore 用的各种功能例程，比如 nlp 有 bert、bert_thor、lstm 等，cv 有 AlexNet、ResNet、yolo 等。值得一提的是，model_zoo 内部还有 MindSpore 在移动端的完整推理框架—MindSpore Lite；

MindSpore 的主体存放在其中的 mindspore 文件夹中，ccsrc 是 mindspore 底层的 c++源码，其中的 frontend 是用于并行计算、冗余节点消除的，backend 存放硬件算法的优化。其他文件则是基于 python 的，搭建了 MindSpore 的 python 库。

阅读源码的周边关系

本文所阅读的源码实现的功能为 ResNet 训练毒蘑菇数据集，该代码位于 model_zoo/offcial/cv/ResNet 内，它在该项目中充当一个完整的训练与推理功能的角色，它会通过各种上层 API（如 nn.conv2d，P.Conv2D，

PrimitiveWithInfer，Prim_attr_register）调用底层的算子（如conv2d_gpu_kernel.h），并构建计算图，同时通过底层的各种优化算法来优化自己前传和后传的速度。

本文首先会从该源码所在文件夹中的 resnet50_predict.py 以及 resnet50_train.py 开始讲解网络训练以及推理的全流程，再深入到其卷积层类的底层源码，解读底层 cuda 算子的建立与调用。

源代码分析

resnet50_predict.py

首先需要配置上下文：

```
context.set_context(mode=context.GRAPH_MODE, device_target="CPU", save_
graphs=False)
```

设置了静态图模式，设备为 CPU，并且设置不保存图。

接着创建模型，并加载其训练好的权重：

```
net = resnet50(class_num=class_num)
param_dict = load_checkpoint(local_ckpt_path)
load_param_into_net(net, param_dict)
net.set_train(False)
```

前三个函数都是调用了顶层的 API 来实现的，最后设置 net 为推理模式，以 freeze BN 层、dropout 层等。

最后从本地的文件夹中寻找需要推理预测的图像，并经过预处理，再输入网络得到预测结果。

```
    images = os.listdir(local_data_path)
    for image in images:
        img = data_preprocess(os.path.join(local_data_path, image))
        # predict model
        res = net(Tensor(img.reshape((1, 3, 224, 224)), mindspore.float
32)).asnumpy()
        predict_label = label_list[res[0].argmax()]
        print("预测的蘑菇标签为:\n\t"+predict_label+"\n")
```

其中，数据预处理的函数如下：

```
def data_preprocess(img_path):
    img = cv2.imread(img_path, 1)
    img = cv2.resize(img, (256, 256))
    img = _crop_center(img, 224, 224)
    mean = [0.485 * 255, 0.456 * 255, 0.406 * 255]
    std = [0.229 * 255, 0.224 * 255, 0.225 * 255]
    img = _normalize(img.astype(np.float32), np.asarray(mean), np.asarr
ay(std))
    img = img.transpose(2, 0, 1)

    return img
```

首先对图像做了一个缩放，将所有数据统一缩放成 256 x 256，然后对图像做中心裁剪，取图像中心部分，得到 224 x 224 大小的图像，最后做一个归一化，对 RGB 三个通道做的归一化的均值和方差是不同的。

resnet50_train.py

首先，同样的都需要配置上下文，不过 mindspore 目前不支持 cpu 训练，所以需要将 target_device 改成 gpu:

```
    context.set_context(mode=context.GRAPH_MODE, device_target="GPU", s
ave_graphs=False)
```

接着，需要加载并预处理训练需要的数据集，并获取整个数据集的大小：

```
train_dataset = create_dataset(dataset_path=local_data_path, do_train=T
rue,
                               repeat_num=epoch_size, batch_size=ba
tch_size)
    train_step_size = train_dataset.get_dataset_size()
```

其中，create_dataset 内首先会调用顶层 API，来将本地文件夹内按规定放好的数据集加载进来：

```
    if device_num == 1:
        ds = de.ImageFolderDataset(dataset_path, num_parallel_workers=1
, shuffle=True)
    else:
```

```
        ds = de.ImageFolderDataset(dataset_path, num_parallel_workers=8
, shuffle=True,
                                   num_shards=device_num, shard_id=rank
_id)
```

然后对数据集进行预处理：

```
        trans = [
            C.RandomCropDecodeResize(image_size, scale=(0.08, 1.0), rat
io=(0.75, 1.333)),
            C.RandomHorizontalFlip(prob=0.5),
            C.Normalize(mean=mean, std=std),
            C.HWC2CHW()
        ]
```

进行解码、随机裁剪、大小缩放、随机水平翻转等数据增强以及归一化和维度变换等操作。

然后创建模型、定义 loss 函数、学习率、优化器等，并指定网络训练时使用的混合精度模式：

```
    # create model
    net = resnet50(class_num=class_num)
    # reduction='mean' means that apply reduction of mean to loss
    loss = SoftmaxCrossEntropyWithLogits(sparse=True, reduction='mean')
    lr = Tensor(get_lr(global_step=0, total_epochs=epoch_size, steps_pe
r_epoch=train_step_size))
    opt = Momentum(net.trainable_params(), lr, momentum=0.9, weight_dec
ay=1e-4, loss_scale=loss_scale_num)
    loss_scale = FixedLossScaleManager(loss_scale_num, False)

    # amp_level="O2" means that the hybrid precision of O2 mode is used
 for training
    # the whole network except that batchnorm will be cast into float16
 format and dynamic loss scale will be used
    # 'keep_batchnorm_fp32 = False' means that use the float16 format
    model = Model(net, amp_level="O2", keep_batchnorm_fp32=False, loss_
fn=loss, optimizer=opt, loss_scale_manager=loss_scale, metrics={'acc'})
```

然后设置一些网络在训练时需要操作的东西，比如 print loss、保存模型参数、记录训练时间等，最后开始训练：

```
    time_cb = TimeMonitor(data_size=train_step_size)
    performance_cb = PerformanceCallback(batch_size)
    loss_cb = LossMonitor()
```

```
    cb = [time_cb, performance_cb, loss_cb]
    config_ck = CheckpointConfig(save_checkpoint_steps=cfg.save_checkpo
int_epochs * train_step_size,
                                 keep_checkpoint_max=cfg.keep_checkpoin
t_max)
    ckpt_cb = ModelCheckpoint(prefix="resnet", directory=local_ckpt_pat
h, config=config_ck)
    cb += [ckpt_cb]

    print(f'Start run training, total epoch: {epoch_size}.')
    #print(model)
    #assert False
    model.train(epoch_size, train_dataset, callbacks=cb)
```

nn.Conv

接着，分析卷积层的底层代码。首先，train 与 predict 会用以下代码调用：

```
import mindspore.nn as nn
```

然后查看 mindspore/nn 文件夹下的__init__.py 文件：

```
__all__.extend(layer.__all__)
```

可以发现它将内部的 layer.py 全部 import 进来了，再查看 layer.py：

```
__all__.extend(conv.__all__)
```

它将卷积部分代码全部 import 了进来，卷积内又将各种类型的卷积文件全部 import 进来，如下所示：

```
__all__ = ['Conv2d', 'Conv2dTranspose', 'Conv1d', 'Conv1dTrans
pose', 'Conv3d', 'Conv3dTranspose']
```

我们具体查看 Conv2d 的函数：

```
from mindspore.ops import operations as P
……
self.conv2d = P.Conv2D(out_channel=self.out_channels,
                                 kernel_size=self.kernel_size,
                                 mode=1,
                                 pad_mode=self.pad_mode,
                                 pad=self.padding,
                                 stride=self.stride,
```

```
                                    dilation=self.dilation,
                                    group=self.group,
                                    data_format=self.format)
```

可以发现 Conv2d 内仍然封装得很严，依旧是调用了 mindspore/ops 内的库。

继续查看 ops 内部的__init__.py 文件：

```
__all__.extend(operations.__all__)
```

然后查看 operations 内__init__.py 的代码：

```
from .nn_ops import (LSTM, SGD, Adam, FusedSparseAdam, FusedSparseLazyAdam, AdamNoUpdateParam, ApplyMomentum,
                     BiasAdd, Conv2D, Conv3D, Conv3DTranspose,
                     DepthwiseConv2dNative,
                     DropoutDoMask, Dropout, Dropout2D, Dropout3D, DropoutGenMask, Flatten,
                     InstanceNorm, BNTrainingReduce, BNTrainingUpdate,
                     GeLU, Gelu, FastGeLU, FastGelu, Elu,
                     GetNext, L2Normalize, LayerNorm, L2Loss, CTCLoss, CTCGreedyDecoder,
                     LogSoftmax, MaxPool3D,
                     MaxPool, DataFormatDimMap,
                     AvgPool, Conv2DBackpropInput, ComputeAccidentalHits,
                     MaxPoolWithArgmax, OneHot, Pad, MirrorPad, Mish, PReLU, ReLU, ReLU6, ReLUV2, HSwish, HSi
                     ResizeBilinear, Sigmoid, SeLU,
                     SigmoidCrossEntropyWithLogits, NLLLoss, BCEWithLogitsLoss,
                     SmoothL1Loss, Softmax, Softsign, Softplus, LRN, RNNTLoss, DynamicRNN, DynamicGRUV2,
                     SoftmaxCrossEntropyWithLogits, ROIAlign,
                     SparseSoftmaxCrossEntropyWithLogits, Tanh,
                     TopK, BinaryCrossEntropy, KLDivLoss, SparseApplyAdagrad, LARSUpdate, ApplyFtrl, SparseAp
                     ApplyProximalAdagrad, SparseApplyProximalAdagrad, SparseApplyAdagradV2, SparseApplyFtrlV
                     FusedSparseFtrl, FusedSparseProximalAdagrad,
                     ApplyAdaMax, ApplyAdadelta, ApplyAdagrad, ApplyAdagradV2,
                     ApplyAddSign, ApplyPowerSign, ApplyGradientDescent, ApplyProximalGradientDescent,
                     ApplyRMSProp, ApplyCenteredRMSProp, BasicLSTMCell, InTopK)
```

此处 import 了各种算子，然后我们针对 Conv2d 做详细解释：

```
@prim_attr_register
def __init__(self,
             out_channel,
             kernel_size,
             mode=1,
             pad_mode="valid",
             pad=0,
             stride=1,
             dilation=1,
             group=1,
             data_format="NCHW"):
    """Initialize Conv2D"""
    self.init_prim_io_names(inputs=['x', 'w'], outputs=['output'])
    self.kernel_size = _check_positive_int_or_tuple('kernel_size', kernel_size, self.
    self.stride = _check_positive_int_or_tuple('stride', stride, self.name, allow_fou
    self.add_prim_attr('stride', self.stride)
    self.dilation = _check_positive_int_or_tuple('dilation', dilation, self.name, all
    self.add_prim_attr('dilation', self.dilation)
    validator.check_value_type('pad', pad, (int, tuple), self.name)
    if isinstance(pad, int):
        pad = (pad,) * 4
    else:
        validator.check_equal_int(len(pad), 4, 'pad size', self.name)
    self.add_prim_attr("pad", pad)
    self.padding = pad
```

可以看到 Conv2d 在 init 自己之前先@prim_attr_register，先将自己的 init 函数输入给 prim_attr_register 函数，用来给底层代码传递自己的 init 参数，以此构建底层计算图。

```
from ..primitive import Primitive, PrimitiveWithInfer, PrimitiveWithCheck, prim_attr_register
```

上述代码是 Conv2d 用于调用该函数的代码。

接下来查看 Prim_attr_register 内的代码:

```
def deco(self, *args, **kwargs):
    class_name = self.__class__.__name__
    if hasattr(self.__class__, "substitute_name"):
        class_name = self.__class__.substitute_name
    if isinstance(self, PrimitiveWithInfer):
        PrimitiveWithInfer.__init__(self, class_name)
    elif isinstance(self, PrimitiveWithCheck):
        PrimitiveWithCheck.__init__(self, class_name)
    else:
        Primitive.__init__(self, self.__class__.__name__)
    bound_args = inspect.signature(fn).bind(self, *args, **kwargs)
    #print('bound_args@@@@')
    #print(bound_args)
    bound_args.apply_defaults()
    arguments = bound_args.arguments
    del arguments['self']
    del self.init_attrs['name']
    for name in arguments:
        value = arguments[name]
        self.add_prim_attr(name, value)
        self.init_attrs[name] = value
    fn(self, *args, **kwargs)
```

上述代码实际做的就是根据 Conv2d 的 init 函数构建各种算子与计算图。

同时，该代码中会用 c_expression 调用 C 中的真正算子：

```
from .._c_expression import Primitive_, real_run_op, prim_type
```

c_expression 是.so 文件，由源码内的 ccsrc 编译而成。接下来查看 ccsrc 内对应的代码，mindspore 的 c 语言比较奇怪，它在 conv2d_gpu_kernel.h 中进行类的编写，在 conv2d_gpu_kernel.cc 内进行算子的注册。

以下是 conv2d_gpu_kernel.cc 内的代码：

```
namespace mindspore {
```

```
namespace kernel {
MS_REG_GPU_KERNEL_ONE(
  Conv2D,
  KernelAttr().AddInputAttr(kNumberTypeFloat32).AddInputAttr(k
NumberTypeFloat32).AddOutputAttr(kNumberTypeFloat32),
  Conv2dGpuFwdKernel, float)
MS_REG_GPU_KERNEL_ONE(
  Conv2D,
  KernelAttr().AddInputAttr(kNumberTypeFloat16).AddInputAttr(k
NumberTypeFloat16).AddOutputAttr(kNumberTypeFloat16),
  Conv2dGpuFwdKernel, half)
}  // namespace kernel
}  // namespace mindspore
```

以下是 conv2d_gpu_kernel.h 内的代码：

```
class Conv2dGpuFwdKernel : public GpuKernel {
 public:
  Conv2dGpuFwdKernel() { ResetResource(); }
  ~Conv2dGpuFwdKernel() override { DestroyResource(); }
  const std::vector<size_t> &GetInputSizeList() const override
 { return input_size_list_; }
  const std::vector<size_t> &GetOutputSizeList() const overrid
e { return output_size_list_; }
  const std::vector<size_t> &GetWorkspaceSizeList() const over
ride { return workspace_size_list_; }

  bool Launch(const std::vector<AddressPtr> &inputs, const std
::vector<AddressPtr> &workspace,
              const std::vector<AddressPtr> &outputs, void *st
ream_ptr) override {
    if (is_null_input_) {
      return true;
    }
    T *input_addr = GetDeviceAddress<T>(inputs, 0);
    T *filter_addr = GetDeviceAddress<T>(inputs, 1);
    T *output_addr = GetDeviceAddress<T>(outputs, 0);
    T *workspace_addr = nullptr;
    if (workspace_size_ != 0) {
      workspace_addr = GetDeviceAddress<T>(workspace, 0);
    }
```

```
    const float alpha = 1;
    const float beta = 0;
```

上面是 Conv2d 前传初始化，比如构建构造函数与析构函数，得到输入输出的 shape，得到输入输出的地址，初始化 cudnn 的参数等等。

接着是 padding 处理：

```
    if (use_pad_) {
      T *padded_addr = GetDeviceAddress<T>(workspace, 1);
      if (data_format_ == kOpFormat_NHWC) {
        CalPadNHWC(padded_size_ / sizeof(T), input_addr, n_, old_height_, old_width_, c_, old_height_ + pad_height_,
                   old_width_ + pad_width_, pad_top_, pad_left_, pad_value_, padded_addr,
                   reinterpret_cast<cudaStream_t>(stream_ptr));
      } else {
        CalPad(padded_size_ / sizeof(T), input_addr, n_, c_, old_height_, old_width_, old_height_ + pad_height_,
               old_width_ + pad_width_, pad_top_, pad_left_, pad_value_, padded_addr,
               reinterpret_cast<cudaStream_t>(stream_ptr));
      }
```

在 padding 之后调用 cudnn 函数进行卷积操作：

```
      CHECK_CUDNN_RET_WITH_EXCEPT(
        kernel_node_,
        cudnnConvolutionForward(cudnn_handle_, &alpha, padded_desc_, padded_addr, filter_desc_, filter_addr, conv_desc_,
                                conv_algorithm_, workspace_addr, workspace_size_, &beta, output_desc_, output_addr),
        "cudnnConvolutionForward failed");
    } else {
      CHECK_CUDNN_RET_WITH_EXCEPT(
        kernel_node_,
        cudnnConvolutionForward(cudnn_handle_, &alpha, input_desc_, input_addr, filter_desc_, filter_addr, conv_desc_,
                                conv_algorithm_, workspace_addr, workspace_size_, &beta, output_desc_, output_addr),
        "cudnnConvolutionForward failed");
```

```
    }
```

实验部分

实验截图

resnet50_predict.py

```
removed in a future version, use 'Add' instead.
[WARNING] ME(65267:140452297619264,MainProcess):2021-04-25-16:22:28.363.345 [mi
removed in a future version, use 'Add' instead.
[WARNING] ME(65267:140452297619264,MainProcess):2021-04-25-16:22:28.421.908 [mi
removed in a future version, use 'Add' instead.
[WARNING] ME(65267:140452297619264,MainProcess):2021-04-25-16:22:28.710.139 [mi
removed in a future version, use 'Add' instead.
[WARNING] ME(65267:140452297619264,MainProcess):2021-04-25-16:22:28.913.604 [mi
removed in a future version, use 'Add' instead.
[WARNING] ME(65267:140452297619264,MainProcess):2021-04-25-16:22:29.116.347 [mi
removed in a future version, use 'Add' instead.
local_ckpt_path!!!!!!!!!
../ckpt_files/resnet-90_209.ckpt
image!!!!!!!!!!!!!!!!!
tum.jpg
[WARNING] DEBUG(65267,python):2021-04-25-16:22:29.722.649 [mindspore/ccsrc/debug
U.
预测的蘑菇标签为：
        Lactarius松乳菇,红菇目,红菇科,乳菇属,广泛分布于亚热带松林地,无毒

ResNet50 prediction success!
```

resnet50_train.py

```
epoch: 1 step: 2090, loss is 1.5089281
epoch time: 239883.486 ms, per step time: 114.777 ms
epoch 2 cost time = 236.72565817832947, train step num: 2090, one step time: 113.26586515709543 ms, train samples per second of c
luster: 282.5

epoch: 2 step: 2090, loss is 1.7359425
epoch time: 236731.509 ms, per step time: 113.269 ms
epoch 3 cost time = 236.8616440296173, train step num: 2090, one step time: 113.33093015771163 ms, train samples per second of cl
uster: 282.4

epoch: 3 step: 2090, loss is 1.4795192
epoch time: 236869.338 ms, per step time: 113.335 ms
epoch 4 cost time = 235.9153277873993, train step num: 2090, one step time: 112.87814726669822 ms, train samples per second of cl
uster: 283.5

epoch: 4 step: 2090, loss is 1.221369
epoch time: 235923.194 ms, per step time: 112.882 ms
epoch 5 cost time = 236.67205691337585, train step num: 2090, one step time: 113.24021861884012 ms, train samples per second of c
luster: 282.6

epoch: 5 step: 2090, loss is 1.1031483
epoch time: 237041.677 ms, per step time: 113.417 ms
epoch 6 cost time = 235.18902826309204, train step num: 2090, one step time: 112.53063553257992 ms, train samples per second of c
luster: 284.4

epoch: 6 step: 2090, loss is 0.57377076
epoch time: 235193.573 ms, per step time: 112.533 ms
epoch 7 cost time = 235.9086470603943, train step num: 2090, one step time: 112.87495074660013 ms, train samples per second of cl
uster: 283.5

epoch: 7 step: 2090, loss is 0.50989264
epoch time: 235915.511 ms, per step time: 112.878 ms
```

实验环境设置

由于 MindSpore 目前更关注于华为升腾芯片的训练与推理，对于 GPU 的支持不友好，所以不建议直接用自己的环境搭建 MindSpore 的训练和推理平台。我使用的是 docker，安装命令如下：

nvidia-container-toolkit 安装

```
# Acquire version of operating system version
DISTRIBUTION=$(. /etc/os-release; echo $ID$VERSION_ID)
curl -s -L https://nvidia.github.io/nvidia-docker/gpgkey |
apt-key add -
curl -s -L https://nvidia.github.io/nvidia-
docker/$DISTRIBUTION/nvidia-docker.list | tee
/etc/apt/sources.list.d/nvidia-docker.list

sudo apt-get update && sudo apt-get install -y nvidia-
container-toolkit nvidia-docker2
sudo systemctl restart docker
```

daemon.json 是 Docker 的配置文件，编辑文件 daemon.json 配置容器运行时，让 Docker 可以使用 nvidia-container-runtime:

```
$ vim /etc/docker/daemon.json
{
    "runtimes": {
        "nvidia": {
            "path": "nvidia-container-runtime",
            "runtimeArgs": []
        }
    }
}
```

再次重启 Docker:

```
sudo systemctl daemon-reload
sudo systemctl restart docker
```

获取 MindSpore 镜像

```
docker pull swr.cn-south-
1.myhuaweicloud.com/mindspore/mindspore-gpu:{tag}
```

运行 MindSpore 镜像

```
docker run -it -v /dev/shm:/dev/shm --runtime=nvidia --
privileged=true swr.cn-south-
1.myhuaweicloud.com/mindspore/mindspore-gpu:{tag} /bin/bash
```