jpienaar Rename to underlying type rather than alias ... History

6 contributors

150 lines (125 sloc) | 5.92 KB ...

```
/* Copyright 2015 The TensorFlow Authors. All Rights Reserved.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
==============================================================================*/

// See docs in ../ops/nn_ops.cc.

#define EIGEN_USE_THREADS

#include "third_party/eigen3/unsupported/Eigen/CXX11/Tensor"
#include "tensorflow/core/framework/numeric_op.h"
#include "tensorflow/core/framework/op_kernel.h"
#include "tensorflow/core/framework/tensor.h"
#include "tensorflow/core/framework/tensor_shape.h"
#include "tensorflow/core/kernels/ops_util.h"
#include "tensorflow/core/kernels/pooling_ops_common.h"
#include "tensorflow/core/lib/core/errors.h"
#include "tensorflow/core/platform/logging.h"
#include "tensorflow/core/util/padding.h"
```

```
#include "tensorflow/core/util/tensor_format.h"

namespace tensorflow {

typedef Eigen::ThreadPoolDevice CPUDevice;

template <typename Device, typename T>
class QuantizedAvgPoolingOp : public OpKernel {
 public:
  explicit QuantizedAvgPoolingOp(OpKernelConstruction* context)
      : OpKernel(context) {
    OP_REQUIRES_OK(context, context->GetAttr("ksize", &ksize_));
    OP_REQUIRES(context, ksize_.size() == 4,
                errors::InvalidArgument("Sliding window ksize field must "
                                        "specify 4 dimensions"));
    OP_REQUIRES_OK(context, context->GetAttr("strides", &stride_));
    OP_REQUIRES(context, stride_.size() == 4,
                errors::InvalidArgument("Sliding window strides field must "
                                        "specify 4 dimensions"));
    OP_REQUIRES_OK(context, context->GetAttr("padding", &padding_));
    OP_REQUIRES(context, ksize_[0] == 1 && stride_[0] == 1,
                errors::Unimplemented(
                    "Pooling is not yet supported on the batch dimension."));
  }

  void Compute(OpKernelContext* context) override {
    const Tensor& tensor_in = context->input(0);
    PoolParameters params{context,
                          ksize_,
                          stride_,
                          padding_,
                          /*explicit_paddings=*/{},
                          FORMAT_NHWC,
                          tensor_in.shape()};
    if (!context->status().ok()) {
      return;
    }

    const float min_input = context->input(1).flat<float>()(0);
    const float max_input = context->input(2).flat<float>()(0);

    OP_REQUIRES(context, params.depth_window == 1,
                errors::Unimplemented("Non-spatial pooling is not "
                                      "yet supported. Volunteers? :)"));

    OP_REQUIRES(context, tensor_in.dims() == 4,
                errors::InvalidArgument("tensor_in must be 4-dimensional"));

    Tensor* output = nullptr;
```

```
    OP_REQUIRES_OK(context, context->allocate_output(
                                0, params.forward_output_shape(), &output));
    const int32_t highest = static_cast<int32>(Eigen::NumTraits<T>::highest());
    const int32_t lowest = static_cast<int32>(Eigen::NumTraits<T>::lowest());

    // TODO(vrv): Switch this to the Eigen::Tensor version of
    // SpatialAvgPooling once that version is running quickly.
    Tensor int32_output(DT_INT32, params.forward_output_shape());
    // Cast input to int32 tensor and call SpatialAvgPool.
    Tensor int32_input(DT_INT32, tensor_in.shape());
    int32_input.flat<int32>() = tensor_in.flat<T>().template cast<int32>();
    SpatialAvgPool<Device, int32>(context, &int32_output, int32_input, params,
                                  padding_);

    // Clamp the int32 output back into quantized space.
    output->flat<T>() = int32_output.flat<int32>()
                            .cwiseMax(lowest)
                            .cwiseMin(highest)
                            .template cast<T>();

    Tensor* output_min = nullptr;
    OP_REQUIRES_OK(context, context->allocate_output(1, {}, &output_min));
    output_min->flat<float>()(0) = min_input;
    Tensor* output_max = nullptr;
    OP_REQUIRES_OK(context, context->allocate_output(2, {}, &output_max));
    output_max->flat<float>()(0) = max_input;
  }

 private:
  std::vector<int32> ksize_;
  std::vector<int32> stride_;
  Padding padding_;
};

template <typename Device, typename T>
class QuantizedMaxPoolingOp : public MaxPoolingOp<Device, T> {
 public:
  explicit QuantizedMaxPoolingOp(OpKernelConstruction* context)
      : MaxPoolingOp<Device, T>(context) {}

  void Compute(OpKernelContext* context) override {
    const float min_input = context->input(1).flat<float>()(0);
    const float max_input = context->input(2).flat<float>()(0);
    MaxPoolingOp<Device, T>::Compute(context);
    Tensor* output_min = nullptr;
    OP_REQUIRES_OK(context, context->allocate_output(1, {}, &output_min));
    output_min->flat<float>()(0) = min_input;
    Tensor* output_max = nullptr;
    OP_REQUIRES_OK(context, context->allocate_output(2, {}, &output_max));
```

```
128     output_max->flat<float>()(0) = max_input;
129   }
130 };
131
132 REGISTER_KERNEL_BUILDER(
133     Name("QuantizedAvgPool").Device(DEVICE_CPU).TypeConstraint<quint8>("T"),
134     QuantizedAvgPoolingOp<CPUDevice, quint8>);
135
136 REGISTER_KERNEL_BUILDER(
137     Name("QuantizedMaxPool").Device(DEVICE_CPU).TypeConstraint<quint8>("T"),
138     QuantizedMaxPoolingOp<CPUDevice, quint8>);
139
140 #ifdef INTEL_MKL
141 REGISTER_KERNEL_BUILDER(
142     Name("QuantizedAvgPool").Device(DEVICE_CPU).TypeConstraint<qint8>("T"),
143     QuantizedAvgPoolingOp<CPUDevice, qint8>);
144
145 REGISTER_KERNEL_BUILDER(
146     Name("QuantizedMaxPool").Device(DEVICE_CPU).TypeConstraint<qint8>("T"),
147     QuantizedMaxPoolingOp<CPUDevice, qint8>);
148 #endif
149
150 }  // namespace tensorflow
```