untitled10-1

January 19, 2024

[1]:
```
%tensorflow_version 2.x
```

```
Colab only includes TensorFlow 2.x; %tensorflow_version has no effect.
```

[2]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import tensorflow as tf
from keras.models import Model
from tensorflow.keras import datasets, layers, models
```

[3]:
```
TF_MODEL="cifar10"
TFL_MODEL_FILE="cifar10.tflite"
```

[4]:
```
(train_imgs, train_lbls), (test_imgs, test_lbls) = datasets.cifar10.load_data()
```

```
Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170498071/170498071 [==============================] - 2s 0us/step
```

[5]:
```
train_imgs = train_imgs / 255.0
test_imgs = test_imgs / 255.0
```

[6]:
```
def separable_conv(input, ch, idx):
    x = layers.DepthwiseConv2D((3,3), padding="same",␣
 ↪name='dwc0_dwsc'+str(idx))(input)
    x = layers.BatchNormalization( name='bn0_dwsc'+str(idx))(x)
    x = layers.Activation("relu", name='act0_dwsc'+str(idx))(x)
    x = layers.Conv2D(ch, (1,1), padding="same", name='conv0_dwsc'+str(idx))(x)
    x = layers.BatchNormalization(name='bn1_dwsc'+str(idx))(x)
    return layers.Activation("relu", name='act1_dwsc'+str(idx))(x)
```

[7]:
```
input = layers.Input((32,32,3))
x = layers.Conv2D(16, (3, 3), padding='same', name='conv1')(input)
x = layers.BatchNormalization(name='bn1')(x)
x = layers.Activation("relu", name='act1')(x)
x = separable_conv(x, 24, 2)
x = layers.MaxPooling2D((2, 2), name='pool1')(x)
x = separable_conv(x, 48, 3)
```

```
x = layers.MaxPooling2D((2, 2), name='pool2')(x)
x = separable_conv(x, 96, 4)
x = separable_conv(x, 192, 5)
x = layers.MaxPooling2D((2, 2), name='pool3')(x)
```

[8]:
```
x = layers.Flatten()(x)
x = layers.Dropout(0.2)(x)
x = layers.Dense(10)(x)
```

[9]:
```
model = Model(input, x)
model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]       0

 conv1 (Conv2D)              (None, 32, 32, 16)        448

 bn1 (BatchNormalization)    (None, 32, 32, 16)        64

 act1 (Activation)           (None, 32, 32, 16)        0

 dwc0_dwsc2 (DepthwiseConv2  (None, 32, 32, 16)        160
 D)

 bn0_dwsc2 (BatchNormalizat  (None, 32, 32, 16)        64
 ion)

 act0_dwsc2 (Activation)     (None, 32, 32, 16)        0

 conv0_dwsc2 (Conv2D)        (None, 32, 32, 24)        408

 bn1_dwsc2 (BatchNormalizat  (None, 32, 32, 24)        96
 ion)

 act1_dwsc2 (Activation)     (None, 32, 32, 24)        0

 pool1 (MaxPooling2D)        (None, 16, 16, 24)        0

 dwc0_dwsc3 (DepthwiseConv2  (None, 16, 16, 24)        240
 D)

 bn0_dwsc3 (BatchNormalizat  (None, 16, 16, 24)        96
 ion)
```

```
act0_dwsc3 (Activation)     (None, 16, 16, 24)        0

conv0_dwsc3 (Conv2D)        (None, 16, 16, 48)        1200

bn1_dwsc3 (BatchNormalizat  (None, 16, 16, 48)        192
ion)

act1_dwsc3 (Activation)     (None, 16, 16, 48)        0

pool2 (MaxPooling2D)        (None, 8, 8, 48)          0

dwc0_dwsc4 (DepthwiseConv2  (None, 8, 8, 48)          480
D)

bn0_dwsc4 (BatchNormalizat  (None, 8, 8, 48)          192
ion)

act0_dwsc4 (Activation)     (None, 8, 8, 48)          0

conv0_dwsc4 (Conv2D)        (None, 8, 8, 96)          4704

bn1_dwsc4 (BatchNormalizat  (None, 8, 8, 96)          384
ion)

act1_dwsc4 (Activation)     (None, 8, 8, 96)          0

dwc0_dwsc5 (DepthwiseConv2  (None, 8, 8, 96)          960
D)

bn0_dwsc5 (BatchNormalizat  (None, 8, 8, 96)          384
ion)

act0_dwsc5 (Activation)     (None, 8, 8, 96)          0

conv0_dwsc5 (Conv2D)        (None, 8, 8, 192)         18624

bn1_dwsc5 (BatchNormalizat  (None, 8, 8, 192)         768
ion)

act1_dwsc5 (Activation)     (None, 8, 8, 192)         0

pool3 (MaxPooling2D)        (None, 4, 4, 192)         0

flatten (Flatten)           (None, 3072)              0

dropout (Dropout)           (None, 3072)              0

dense (Dense)               (None, 10)                30730
```

```
=================================================================
Total params: 60194 (235.13 KB)
Trainable params: 59074 (230.76 KB)
Non-trainable params: 1120 (4.38 KB)
_________________________________________________________________
```

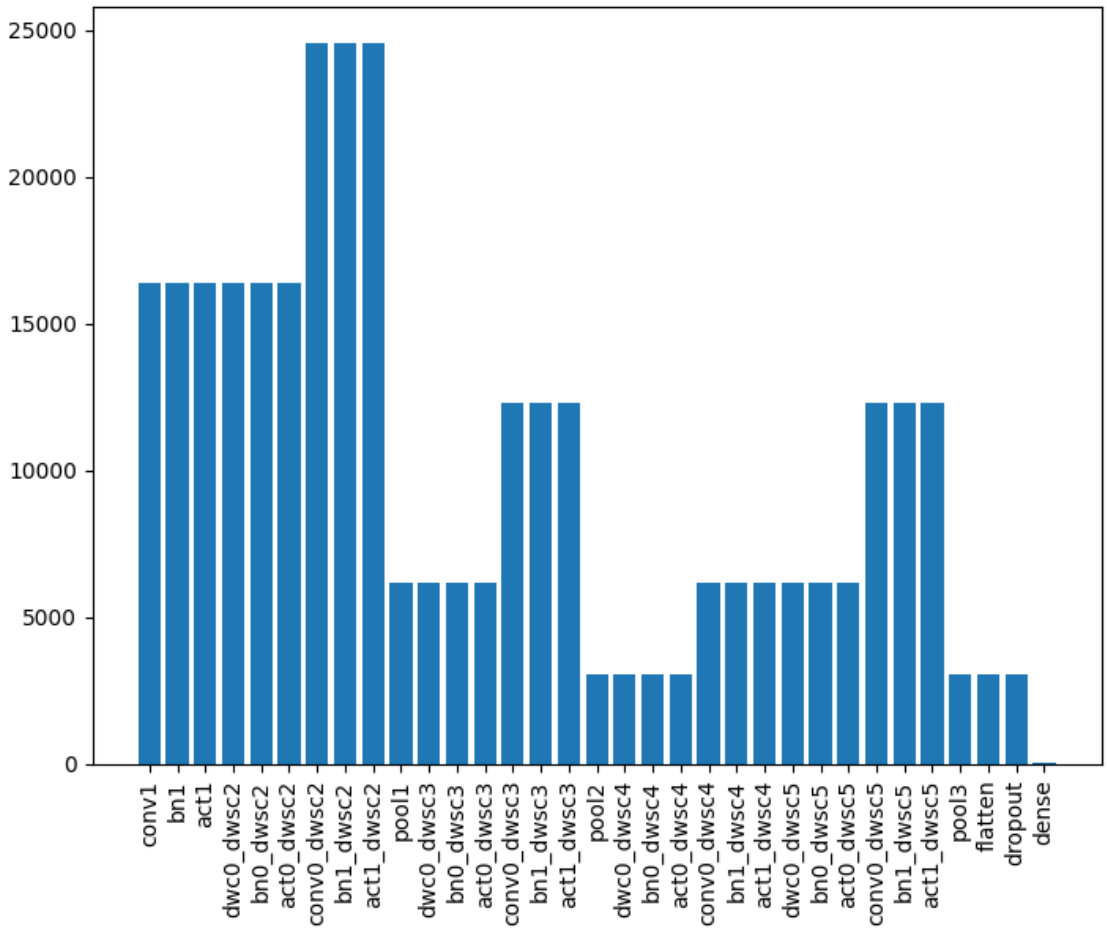
[10]:
```
fig = plt.figure(dpi=100)

ax = fig.add_axes([0,0,1,1])

l_idx   = []
l_sizes = []

for layer in model.layers[1:]:
  shape = layer.output_shape
  shape = np.delete(shape, 0)
  size  = np.prod(shape)
  l_idx   = np.append(l_idx, layer.name)
  l_sizes = np.append(l_sizes, size)

ax.bar(l_idx, l_sizes)
plt.xticks(rotation='vertical')
plt.show()
```



[11]:
```
model.compile(optimizer='adam',
              loss=tf.keras.losses.
 ↪SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

history = model.fit(train_imgs, train_lbls, epochs=10,
                    validation_data=(test_imgs, test_lbls))
```

```
Epoch 1/10
1563/1563 [==============================] - 208s 127ms/step - loss: 1.6333 -
accuracy: 0.4489 - val_loss: 1.2257 - val_accuracy: 0.5642
Epoch 2/10
1563/1563 [==============================] - 188s 120ms/step - loss: 1.1534 -
accuracy: 0.5958 - val_loss: 1.1242 - val_accuracy: 0.6049
Epoch 3/10
1563/1563 [==============================] - 191s 122ms/step - loss: 0.9885 -
accuracy: 0.6532 - val_loss: 1.1924 - val_accuracy: 0.5923
Epoch 4/10
```

```
1563/1563 [==============================] - 185s 119ms/step - loss: 0.8790 -
accuracy: 0.6935 - val_loss: 0.9522 - val_accuracy: 0.6727
Epoch 5/10
1563/1563 [==============================] - 186s 119ms/step - loss: 0.8008 -
accuracy: 0.7220 - val_loss: 0.9699 - val_accuracy: 0.6637
Epoch 6/10
1563/1563 [==============================] - 186s 119ms/step - loss: 0.7362 -
accuracy: 0.7427 - val_loss: 0.9088 - val_accuracy: 0.7026
Epoch 7/10
1563/1563 [==============================] - 186s 119ms/step - loss: 0.6906 -
accuracy: 0.7600 - val_loss: 0.8868 - val_accuracy: 0.7036
Epoch 8/10
1563/1563 [==============================] - 186s 119ms/step - loss: 0.6463 -
accuracy: 0.7749 - val_loss: 0.7992 - val_accuracy: 0.7326
Epoch 9/10
1563/1563 [==============================] - 190s 122ms/step - loss: 0.6122 -
accuracy: 0.7863 - val_loss: 0.8399 - val_accuracy: 0.7171
Epoch 10/10
1563/1563 [==============================] - 187s 119ms/step - loss: 0.5828 -
accuracy: 0.7943 - val_loss: 0.8768 - val_accuracy: 0.7116
```

[12]:
```
model.save(TF_MODEL)
```

[13]:
```
cifar_ds = tf.data.Dataset.from_tensor_slices(train_imgs).batch(1)
def representative_data_gen():
  for i_value in cifar_ds.take(100):
    i_value_f32 = tf.dtypes.cast(i_value, tf.float32)
    yield [i_value_f32]
```

[14]:
```
tfl_conv = tf.lite.TFLiteConverter.from_saved_model(TF_MODEL)
tfl_conv.representative_dataset = tf.lite.
 ↪RepresentativeDataset(representative_data_gen)
tfl_conv.optimizations = [tf.lite.Optimize.DEFAULT]
tfl_conv.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
tfl_conv.inference_input_type = tf.int8
tfl_conv.inference_output_type = tf.int8
```

[15]:
```
tfl_model = tfl_conv.convert()
open(TFL_MODEL_FILE, "wb").write(tfl_model)
```

[15]: 82152

[16]:
```
print(len(tfl_model))
```

```
82152
```

[17]:
```
tfl_inter = tf.lite.Interpreter(model_content=tfl_model)

# Allocate the tensors
tfl_inter.allocate_tensors()

# Get input/output layer information
i_details = tfl_inter.get_input_details()[0]
o_details = tfl_inter.get_output_details()[0]

i_quant = i_details["quantization_parameters"]
o_quant = o_details["quantization_parameters"]
i_scale      = i_quant['scales'][0]
i_zero_point = i_quant['zero_points'][0]
o_scale      = o_quant['scales'][0]
o_zero_point = o_quant['zero_points'][0]

def classify(i_data, o_value):
  input_data = i_value.reshape((1, 32, 32, 3))
  i_value_f32 = tf.dtypes.cast(input_data, tf.float32)

  # Quantize (float -> 8-bit) the input (check if input layer is 8-bit, first)
  i_value_f32 = i_value_f32 / i_scale + i_zero_point
  i_value_s8 = tf.cast(i_value_f32, dtype=tf.int8)

  tfl_inter.set_tensor(i_details["index"], i_value_s8)
  tfl_inter.invoke()
  o_pred = tfl_inter.get_tensor(o_details["index"])[0]

  return (o_pred - o_zero_point) * o_scale

num_correct_samples = 0
num_total_samples   = len(list(test_imgs))

for i_value, o_value in zip(test_imgs, test_lbls):
  o_pred_f32 = classify(i_value, o_value)
  if np.argmax(o_pred_f32) == o_value:
      num_correct_samples += 1
```

[18]:
```
print("Accuracy:", num_correct_samples/num_total_samples)
```

```
Accuracy: 0.7138
```

[19]:
```
!apt-get update && apt-get -qq install xxd
!xxd -i cifar10.tflite > model.h
```

```
Get:1 https://cloud.r-project.org/bin/linux/ubuntu jammy-cran40/ InRelease
[3,626 B]
Hit:2 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64
```

```
InRelease
Hit:3 http://archive.ubuntu.com/ubuntu jammy InRelease
Get:4 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [119 kB]
Get:5 http://security.ubuntu.com/ubuntu jammy-security InRelease [110 kB]
Get:6 http://archive.ubuntu.com/ubuntu jammy-backports InRelease [109 kB]
Get:7 http://archive.ubuntu.com/ubuntu jammy-updates/universe amd64 Packages
[1,321 kB]
Get:8 http://archive.ubuntu.com/ubuntu jammy-updates/main amd64 Packages [1,624
kB]
Get:9 http://archive.ubuntu.com/ubuntu jammy-updates/multiverse amd64 Packages
[50.4 kB]
Get:10 http://archive.ubuntu.com/ubuntu jammy-backports/main amd64 Packages
[50.4 kB]
Get:11 http://archive.ubuntu.com/ubuntu jammy-backports/universe amd64 Packages
[28.1 kB]
Get:12 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages
[1,344 kB]
Get:13 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages
[1,056 kB]
Hit:14 https://ppa.launchpadcontent.net/c2d4u.team/c2d4u4.0+/ubuntu jammy
InRelease
Hit:15 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy InRelease
Hit:16 https://ppa.launchpadcontent.net/graphics-drivers/ppa/ubuntu jammy
InRelease
Hit:17 https://ppa.launchpadcontent.net/ubuntugis/ppa/ubuntu jammy InRelease
Fetched 5,814 kB in 6s (940 kB/s)
Reading package lists… Done
```

[20]:
```python
def array_to_str(data):
    NUM_COLS = 12
    val_string = ''
    for i, val in enumerate(data):
        val_string += str(val)

        if (i + 1) < len(data):
            val_string += ','
        if (i + 1) % NUM_COLS == 0:
            val_string += '\n'
    return val_string
```

[21]:
```python
def gen_h_file(size, data, ilabel):
  str_out = f'int8_t g_test[] = '
  str_out += "\n{\n"
  str_out += f'{data}'
  str_out += '};\n'
  str_out += f"const int g_test_len = {size};\n"
  str_out += f"const int g_test_ilabel = {ilabel};\n"
```

```
  return str_out
```

[22]:
```
imgs = list(zip(test_imgs, test_lbls))
cols = ['Image', 'Label']
df = pd.DataFrame(imgs, columns = cols)
```

[23]:
```
cond = df['Label'] == 8
ship_samples = df[cond]
```

[24]:
```
c_code = ""

for index, row in ship_samples.iterrows():
  i_value = np.asarray(row['Image'].tolist())
  o_value = np.asarray(row['Label'].tolist())
  o_pred_f32 = classify(i_value, o_value)
  if np.argmax(o_pred_f32) == o_value:
    i_value_f32 = i_value / i_scale + i_zero_point
    i_value_s8  = i_value_f32.astype(dtype=np.uint8)
    i_value_s8  = i_value_s8.ravel()

    # Generate a string from NumPy array
    val_string = array_to_str(i_value_s8)

    c_code = gen_h_file(i_value_s8.size, val_string, "8")
    break
```

[25]:
```
with open("input.h", 'w') as file:
  file.write(c_code)
```