# C1_W4_Lab_2_resnet-example

December 23, 2020

## 1 Ungraded Lab: Implementing ResNet

In this lab, you will continue exploring Model subclassing by building a more complex architecture.

Residual Networks make use of skip connections to make deep models easier to train. - There are branches as well as many repeating blocks of layers in this type of network. - You can define a model class to help organize this more complex code, and to make it easier to re-use your code when building the model. - As before, you will inherit from the Model class so that you can make use of the other built-in methods that Keras provides.

### 1.1 Imports

[1]:
```
try:
    # %tensorflow_version only exists in Colab.
    %tensorflow_version 2.x
except Exception:
    pass

import tensorflow as tf
import tensorflow_datasets as tfds
from tensorflow.keras.layers import Layer
```

### 1.2 Implement Model subclasses

As shown in the lectures, you will first implement the Identity Block which contains the skip connections (i.e. the `add()` operation below. This will also inherit the Model class and implement the `__init__()` and `call()` methods.

[2]:
```
class IdentityBlock(tf.keras.Model):
    def __init__(self, filters, kernel_size):
        super(IdentityBlock, self).__init__(name='')

        self.conv1 = tf.keras.layers.Conv2D(filters, kernel_size,␣
↪padding='same')
        self.bn1 = tf.keras.layers.BatchNormalization()
```

```
        self.conv2 = tf.keras.layers.Conv2D(filters, kernel_size,␣
↪padding='same')
        self.bn2 = tf.keras.layers.BatchNormalization()

        self.act = tf.keras.layers.Activation('relu')
        self.add = tf.keras.layers.Add()

    def call(self, input_tensor):
        x = self.conv1(input_tensor)
        x = self.bn1(x)
        x = self.act(x)

        x = self.conv2(x)
        x = self.bn2(x)
        x = self.act(x)

        x = self.add([x, input_tensor])
        x = self.act(x)
        return x
```

From there, you can build the rest of the ResNet model. - You will call your `IdentityBlock` class two times below and that takes care of inserting those blocks of layers into this network.

[3]:
```
class ResNet(tf.keras.Model):
    def __init__(self, num_classes):
        super(ResNet, self).__init__()
        self.conv = tf.keras.layers.Conv2D(64, 7, padding='same')
        self.bn = tf.keras.layers.BatchNormalization()
        self.act = tf.keras.layers.Activation('relu')
        self.max_pool = tf.keras.layers.MaxPool2D((3, 3))

        # Use the Identity blocks that you just defined
        self.id1a = IdentityBlock(64, 3)
        self.id1b = IdentityBlock(64, 3)

        self.global_pool = tf.keras.layers.GlobalAveragePooling2D()
        self.classifier = tf.keras.layers.Dense(num_classes,␣
↪activation='softmax')

    def call(self, inputs):
        x = self.conv(inputs)
        x = self.bn(x)
        x = self.act(x)
        x = self.max_pool(x)

        # insert the identity blocks in the middle of the network
        x = self.id1a(x)
```

```
        x = self.id1b(x)

        x = self.global_pool(x)
        return self.classifier(x)
```

### 1.3 Training the Model

As mentioned before, inheriting the Model class allows you to make use of the other APIs that Keras provides, such as: - training - serialization - evaluation

You can instantiate a Resnet object and train it as usual like below:

**Note**: If you have issues with training in the Coursera lab environment, you can also run this in Colab using the "open in colab" badge link.

[4]:
```
# utility function to normalize the images and return (image, label) pairs.
def preprocess(features):
    return tf.cast(features['image'], tf.float32) / 255., features['label']

# create a ResNet instance with 10 output units for MNIST
resnet = ResNet(10)
resnet.compile(optimizer='adam', loss='sparse_categorical_crossentropy',␣
 ↪metrics=['accuracy'])

# load and preprocess the dataset
dataset = tfds.load('mnist', split=tfds.Split.TRAIN, data_dir='./data')
dataset = dataset.map(preprocess).batch(32)

# train the model.
resnet.fit(dataset, epochs=1)
```

```
   214/Unknown - 30s 141ms/step - loss: 0.5611 - accuracy: 0.8656


      ␣
↪---------------------------------------------------------------------------

       KeyboardInterrupt                         Traceback (most recent call␣
↪last)

       <ipython-input-4-2f518416899f> in <module>
        12
        13 # train the model.
   ---> 14 resnet.fit(dataset, epochs=1)
```

```
      /opt/conda/lib/python3.7/site-packages/tensorflow_core/python/keras/
↪engine/training.py in fit(self, x, y, batch_size, epochs, verbose, callbacks,␣
↪validation_split, validation_data, shuffle, class_weight, sample_weight,␣
↪initial_epoch, steps_per_epoch, validation_steps, validation_freq,␣
↪max_queue_size, workers, use_multiprocessing, **kwargs)
      817         max_queue_size=max_queue_size,
      818         workers=workers,
  --> 819         use_multiprocessing=use_multiprocessing)
      820
      821   def evaluate(self,


      /opt/conda/lib/python3.7/site-packages/tensorflow_core/python/keras/
↪engine/training_v2.py in fit(self, model, x, y, batch_size, epochs, verbose,␣
↪callbacks, validation_split, validation_data, shuffle, class_weight,␣
↪sample_weight, initial_epoch, steps_per_epoch, validation_steps,␣
↪validation_freq, max_queue_size, workers, use_multiprocessing, **kwargs)
      340                 mode=ModeKeys.TRAIN,
      341                 training_context=training_context,
  --> 342                 total_epochs=epochs)
      343             cbks.make_logs(model, epoch_logs, training_result,␣
↪ModeKeys.TRAIN)
      344


      /opt/conda/lib/python3.7/site-packages/tensorflow_core/python/keras/
↪engine/training_v2.py in run_one_epoch(model, iterator, execution_function,␣
↪dataset_size, batch_size, strategy, steps_per_epoch, num_samples, mode,␣
↪training_context, total_epochs)
      126         step=step, mode=mode, size=current_batch_size) as batch_logs:
      127       try:
  --> 128         batch_outs = execution_function(iterator)
      129       except (StopIteration, errors.OutOfRangeError):
      130         # TODO(kaftan): File bug about tf function and errors.
↪OutOfRangeError?


      /opt/conda/lib/python3.7/site-packages/tensorflow_core/python/keras/
↪engine/training_v2_utils.py in execution_function(input_fn)
       96     # `numpy` translates Tensors to values in Eager mode.
       97     return nest.map_structure(_non_none_constant_value,
  ---> 98                               distributed_function(input_fn))
       99
      100   return execution_function
```

```
      /opt/conda/lib/python3.7/site-packages/tensorflow_core/python/eager/
↪def_function.py in __call__(self, *args, **kwds)
      566         xla_context.Exit()
      567     else:
  --> 568       result = self._call(*args, **kwds)
      569
      570     if tracing_count == self._get_tracing_count():


      /opt/conda/lib/python3.7/site-packages/tensorflow_core/python/eager/
↪def_function.py in _call(self, *args, **kwds)
      597       # In this case we have created variables on the first call, so␣
↪we run the
      598       # defunned version which is guaranteed to never create␣
↪variables.
  --> 599       return self._stateless_fn(*args, **kwds)  # pylint:␣
↪disable=not-callable
      600     elif self._stateful_fn is not None:
      601       # Release the lock early so that multiple threads can perform␣
↪the call


      /opt/conda/lib/python3.7/site-packages/tensorflow_core/python/eager/
↪function.py in __call__(self, *args, **kwargs)
     2361     with self._lock:
     2362       graph_function, args, kwargs = self.
↪_maybe_define_function(args, kwargs)
  -> 2363     return graph_function._filtered_call(args, kwargs)  # pylint:␣
↪disable=protected-access
     2364
     2365   @property


      /opt/conda/lib/python3.7/site-packages/tensorflow_core/python/eager/
↪function.py in _filtered_call(self, args, kwargs)
     1609          if isinstance(t, (ops.Tensor,
     1610                            resource_variable_ops.
↪BaseResourceVariable))),
  -> 1611         self.captured_inputs)
     1612
     1613   def _call_flat(self, args, captured_inputs,␣
↪cancellation_manager=None):


      /opt/conda/lib/python3.7/site-packages/tensorflow_core/python/eager/
↪function.py in _call_flat(self, args, captured_inputs, cancellation_manager)
     1690       # No tape is watching; skip to running the function.
```

```
   1691       return self._build_call_outputs(self._inference_function.call(
-> 1692           ctx, args, cancellation_manager=cancellation_manager))
   1693     forward_backward = self._select_forward_and_backward_functions(
   1694         args,


/opt/conda/lib/python3.7/site-packages/tensorflow_core/python/eager/function.py in call(self, ctx, args, cancellation_manager)
    543               inputs=args,
    544               attrs=("executor_type", executor_type, "config_proto", config),
--> 545               ctx=ctx)
    546         else:
    547           outputs = execute.execute_with_cancellation(


/opt/conda/lib/python3.7/site-packages/tensorflow_core/python/eager/execute.py in quick_execute(op_name, num_outputs, inputs, attrs, ctx, name)
     59     tensors = pywrap_tensorflow.TFE_Py_Execute(ctx._handle, device_name,
     60                                                op_name, inputs, attrs,
---> 61                                                num_outputs)
     62   except core._NotOkStatusException as e:
     63     if name is not None:


KeyboardInterrupt:
```

[ ]:

[ ]: