

# C2\_W4\_Lab\_4\_one-device-strategy

February 5, 2025

## 1 One Device Strategy

In this ungraded lab, you'll learn how to set up a [One Device Strategy](#). This is typically used to deliberately test your code on a single device. This can be used before switching to a different strategy that distributes across multiple devices. Please click on the **Open in Colab** badge above so you can download the datasets and use a GPU-enabled lab environment.

### 1.1 Imports

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

import tensorflow as tf
import tensorflow_hub as hub
import tensorflow_datasets as tfds

tfds.disable_progress_bar()
```

### 1.2 Define the Distribution Strategy

You can list available devices in your machine and specify a device type. This allows you to verify the device name to pass in `tf.distribute.OneDeviceStrategy()`.

```
[ ]: # choose a device type such as CPU or GPU
devices = tf.config.list_physical_devices('GPU')
print(devices[0])

# You'll see that the name will look something like "/physical_device:GPU:0"
# Just take the GPU:0 part and use that as the name
gpu_name = "GPU:0"

# define the strategy and pass in the device name
```

```
one_strategy = tf.distribute.OneDeviceStrategy(device=gpu_name)
```

### 1.3 Parameters

We'll define a few global variables for setting up the model and dataset.

```
[ ]: pixels = 224
MODULE_HANDLE = 'https://tfhub.dev/tensorflow/resnet_50/feature_vector/1'
IMAGE_SIZE = (pixels, pixels)
BATCH_SIZE = 32

print("Using {} with input size {}".format(MODULE_HANDLE, IMAGE_SIZE))
```

### 1.4 Download and Prepare the Dataset

We will use the [Cats vs Dogs](#) dataset and we will fetch it via TFDS.

```
[ ]: splits = ['train[:80%]', 'train[80%:90%]', 'train[90%:]']

(train_examples, validation_examples, test_examples), info = tfds.
    ↳load('cats_vs_dogs', with_info=True, as_supervised=True, split=splits)

num_examples = info.splits['train'].num_examples
num_classes = info.features['label'].num_classes
```

```
[ ]: # resize the image and normalize pixel values
def format_image(image, label):
    image = tf.image.resize(image, IMAGE_SIZE) / 255.0
    return image, label
```

```
[ ]: # prepare batches
train_batches = train_examples.shuffle(num_examples // 4).map(format_image).
    ↳batch(BATCH_SIZE).prefetch(1)
validation_batches = validation_examples.map(format_image).batch(BATCH_SIZE).
    ↳prefetch(1)
test_batches = test_examples.map(format_image).batch(1)
```

```
[ ]: # check if the batches have the correct size and the images have the correct_
    ↳shape
for image_batch, label_batch in train_batches.take(1):
    pass

print(image_batch.shape)
```

## 1.5 Define and Configure the Model

As with other strategies, setting up the model requires minimal code changes. Let's first define a utility function to build and compile the model.

```
[ ]: # tells if we want to freeze the layer weights of our feature extractor during
      ↪ training
do_fine_tuning = False

[ ]: def build_and_compile_model():
      print("Building model with", MODULE_HANDLE)

      # configures the feature extractor fetched from TF Hub
      feature_extractor = hub.KerasLayer(MODULE_HANDLE,
                                          input_shape=IMAGE_SIZE + (3,),
                                          trainable=do_fine_tuning)

      # define the model
      model = tf.keras.Sequential([
          feature_extractor,
          # append a dense with softmax for the number of classes
          tf.keras.layers.Dense(num_classes, activation='softmax')
      ])

      # display summary
      model.summary()

      # configure the optimizer, loss and metrics
      optimizer = tf.keras.optimizers.SGD(lr=0.002, momentum=0.9) if
      ↪ do_fine_tuning else 'adam'
      model.compile(optimizer=optimizer,
                    loss='sparse_categorical_crossentropy',
                    metrics=['accuracy'])

      return model
```

You can now call the function under the strategy scope. This places variables and computations on the device you specified earlier.

```
[ ]: # build and compile under the strategy scope
with one_strategy.scope():
    model = build_and_compile_model()
```

`model.fit()` can be run as usual.

```
[ ]: EPOCHS = 5
hist = model.fit(train_batches,
                 epochs=EPOCHS,
```

```
validation_data=validation_batches)
```

Once everything is working correctly, you can switch to a different device or a different strategy that distributes to multiple devices.