Open in Colab

Ungraded Lab: Beyond Hello World, A Computer Vision Example

In the previous exercise, you saw how to create a neural network that figured out the problem you were trying to solve. This gave an explicit example of learned behavior. Of course, in that instance, it was a bit of overkill because it would have been easier to write the function `y=2x-1` directly instead of bothering with using machine learning to learn the relationship between `x` and `y`.

But what about a scenario where writing rules like that is much more difficult -- for example a computer vision problem? Let's take a look at a scenario where you will build a neural network to recognize different items of clothing, trained from a dataset containing 10 different types.

Start Coding

Let's start with our import of TensorFlow.

```
import tensorflow as tf

print(tf.__version__)
```

```
2.8.0
```

The [Fashion MNIST dataset](https://github.com/zalandoresearch/fashion-mnist) is a collection of grayscale 28x28 pixel clothing images. Each image is associated with a label as shown in this table!?

Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress

Label	Description
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

This dataset is available directly in the [tf.keras.datasets](https://www.tensorflow.org/api_docs/python/tf/keras/datasets) API and you load it like this:

```
# Load the Fashion MNIST dataset
fmnist = tf.keras.datasets.fashion_mnist
```

Calling `load_data()` on this object will give you two tuples with two lists each. These will be the training and testing values for the graphics that contain the clothing items and their labels.

```
# Load the training and test split of the Fashion MNIST dataset
(training_images, training_labels), (test_images, test_labels) = fmnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-labels-idx1-ubyte.gz
32768/29515 [=================================] - 0s 0us/step
40960/29515 [=========================================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/train-images-idx3-ubyte.gz
26427392/26421880 [==============================] - 0s 0us/step
26435584/26421880 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-labels-idx1-ubyte.gz
16384/5148 [=============================================================================================] - 0s 0us
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/t10k-images-idx3-ubyte.gz
4423680/4422102 [==============================] - 0s 0us/step
4431872/4422102 [==============================] - 0s 0us/step
```

What does these values look like? Let's print a training image (both as an image and a numpy array), and a training label to see. Experiment with

```
import numpy as np
import matplotlib.pyplot as plt

# You can put between 0 to 59999 here
index = 0

# Set number of characters per row when printing
np.set_printoptions(linewidth=320)

# Print the label and image
print(f'LABEL: {training_labels[index]}')
print(f'\nIMAGE PIXEL ARRAY:\n {training_images[index]}')

# Visualize the image
plt.imshow(training_images[index])
```

```
LABEL: 9

IMAGE PIXEL ARRAY:
 [[  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   1   0   0  13  73   0   0   1   4   0   0   0   0   1   1   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   3   0  36 136 127  62  54   0   0   0   1   3   4   0   0   3]
 [  0   0   0   0   0   0   0   0   0   0   0   0   6   0 102 204 176 134 144 123  23   0   0   0   0  12  10   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0   0 155 236 207 178 107 156 161 109  64  23  77 130  72  15]
 [  0   0   0   0   0   0   0   0   0   0   0   1   0  69 207 223 218 216 216 163 127 121 122 146 141  88 172  66]
 [  0   0   0   0   0   0   0   0   0   1   1   1   0 200 232 232 233 229 223 223 215 213 164 127 123 196 229   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 183 225 216 223 228 235 227 224 222 224 221 223 245 173   0]
 [  0   0   0   0   0   0   0   0   0   0   0   0   0 193 228 218 213 198 180 212 210 211 213 223 220 243 202   0]
 [  0   0   0   0   0   0   0   0   0   1   3   0  12 219 220 212 218 192 169 227 208 218 224 212 226 197 209  52]
 [  0   0   0   0   0   0   0   0   0   0   6   0  99 244 222 220 218 203 198 221 215 213 222 220 245 119 167  56]
 [  0   0   0   0   0   0   0   0   0   4   0   0  55 236 228 230 228 240 232 213 218 223 234 217 217 209  92   0]
 [  0   0   1   4   6   7   2   0   0   0   0   0 237 226 217 223 222 219 222 221 216 223 229 215 218 255  77   0]
 [  0   3   0   0   0   0   0   0   0  62 145 204 228 207 213 221 218 208 211 218 224 223 219 215 224 244 159   0]
 [  0   0   0   0  18  44  82 107 189 228 220 222 217 226 200 205 211 230 224 234 176 188 250 248 233 238 215   0]
 [  0  57 187 208 224 221 224 208 204 214 208 209 200 159 245 193 206 223 255 255 221 234 221 211 220 232 246   0]
 [  3 202 228 224 221 211 211 214 205 205 205 220 240  80 150 255 229 221 188 154 191 210 204 209 222 228 225   0]
 [ 98 233 198 210 222 229 229 234 249 220 194 215 217 241  65  73 106 117 168 219 221 215 217 223 223 224 229  29]
 [ 75 204 212 204 193 205 211 225 216 185 197 206 198 213 240 195 227 245 239 223 218 212 209 222 220 221 230  67]
 [ 48 203 183 194 213 197 185 190 194 192 202 214 219 221 220 236 225 216 199 206 186 181 177 172 181 205 206 115]
 [  0 122 219 193 179 171 183 196 204 210 213 207 211 210 200 196 194 191 195 191 198 192 176 156 167 177 210  92]
```

You'll notice that all of the values in the number are between 0 and 255. If you are training a neural network especially in image processing, for various reasons it will usually learn better if you scale all values to between 0 and 1. It's a process called *normalization* and fortunately in Python, it's easy to normalize an array without looping. You do it like this:


```
# Normalize the pixel values of the train and test images
training_images  = training_images / 255.0
test_images = test_images / 255.0
```

Now you might be wondering why the dataset is split into two: training and testing? Remember we spoke about this in the intro? The idea is to have 1 set of data for training, and then another set of data that the model hasn't yet seen. This will be used to evaluate how good it would be at classifyi[redacted]

Let's now design the model. There's quite a few new concepts here. But don't worry, you'll get the hang of them.

```
# Build the classification model
model = tf.keras.models.Sequential([tf.keras.layers.Flatten(),
                                    tf.keras.layers.Dense(128, activation=tf.nn.relu),
                                    tf.keras.layers.Dense(10, activation=tf.nn.softmax)])
```

[Sequential](): That defines a sequence of layers in the neural network.

[Flatten](): Remember earlier where our images were a 28x28 pixel matrix when you printed them out? Flatten just takes that square and turns it into a 1-dimensional array.

[Dense](): Adds a layer of neurons

Each layer of neurons need an [activation function]() to tell them what to do. There are a lot of options, but just use these for now:

[ReLU]() effectively means:

```
if x > 0:
  return x

else:
  return 0
```

In other words, it it only passes values 0 or greater to the next layer in the network.

[Softmax]() takes a list of values and scales these so the sum of all elements will be equal to 1. When applied to model outputs, you can think of the scaled values as the probability for that class. For example, in your classification model which has 10 units in the output dense layer, having the highest value at `index = 4` means that the model is most confident that the input clothing image is a coat. If it is at index = 5, then it is a sandal, and so forth. See the short code block below which demonstrates these concepts. You can also watch this [lecture]() if you want to know more about the Softmax function and how the values are computed.

```
# Declare sample inputs and convert to a tensor
inputs = np.array([[1.0, 3.0, 4.0, 2.0]])
inputs = tf.convert_to_tensor(inputs)
print(f'input to softmax function: {inputs.numpy()}')

# Feed the inputs to a softmax activation function
outputs = tf.keras.activations.softmax(inputs)
print(f'output of softmax function: {outputs.numpy()}')

# Get the sum of all values after the softmax
sum = tf.reduce_sum(outputs)
print(f'sum of outputs: {sum}')

# Get the index with highest value
prediction = np.argmax(outputs)
print(f'class with highest probability: {prediction}')
```

```
input to softmax function: [[1. 3. 4. 2.]]
output of softmax function: [[0.0320586  0.23688282 0.64391426 0.08714432]]
sum of outputs: 1.0
class with highest probability: 2
```

```
total=np.exp(1)+np.exp(3)+np.exp(4)+np.exp(2)
```

```
np.exp(1)/total
```

```
0.03205860328008499
```

The next thing to do, now that the model is defined, is to actually build it. You do this by compiling it with an optimizer and loss function as before -- and then you train it by calling `model.fit()` asking it to fit your training data to your training labels. It will figure out the relationship between the training data and its actual labels so in the future if you have inputs that looks like the training data, then it can predict what the label for that input is.

```
model.compile(optimizer = tf.optimizers.Adam(),
              loss = 'sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(training_images, training_labels, epochs=5)
```

```
Epoch 1/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.4971 - accuracy: 0.8252
Epoch 2/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3756 - accuracy: 0.8644
Epoch 3/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3382 - accuracy: 0.8769
Epoch 4/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.3133 - accuracy: 0.8858
Epoch 5/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2938 - accuracy: 0.8922
<keras.callbacks.History at 0x7fbea17ffb10>
```

Once it's done training -- you should see an accuracy value at the end of the final epoch. It might look something like `0.9098`. This tells you that your neural network is about 91% accurate in classifying the training data. That is, it figured out a pattern match between the image and the labels that worked 91% of the time. Not great, but not bad considering it was only trained for 5 epochs and done quite quickly.

But how would it work with unseen data? That's why we have the test images and labels. We can call [`model.evaluate()`](#) with this test dataset as inputs and it will report back the loss and accuracy of the model. Let's give it a try:

```
# Evaluate the model on unseen data
model.evaluate(test_images, test_labels)
```

```
313/313 [==============================] - 1s 1ms/step - loss: 0.3642 - accuracy: 0.8733
[0.3641885817050934, 0.8733000159263611]
```

You can expect the accuracy here to be about `0.88` which means it was 88% accurate on the entire test set. As expected, it probably would not do as well with *unseen* data as it did with data it was trained on! As you go through this course, you'll look at ways to improve this.

Exploration Exercises

To explore further and deepen your understanding, try the below exercises:

Exercise 1:

For this first exercise run the below code: It creates a set of classifications for each of the test images, and then prints the first entry in the classifications. The output, after you run it is a list of numbers. Why do you think this is, and what do those numbers represent?

```
classifications = model.predict(test_images)

print(classifications[0])
```

```
[3.45551343e-05 5.09785103e-09 2.36808523e-06 9.00163286e-08 2.36252731e-06 3.50199221e-03 5.36902699e-05 2.41864786e
```

Hint: try running `print(test_labels[0])` -- and you'll get a `9`. Does that help you understand why this list looks the way it does?

```
print(test_labels[0])
```

```
9
```

E1Q1: What does this list represent?

1. It's 10 random meaningless values
2. It's the first 10 classifications that the computer made

3. It's the probability that this item is each of the 10 classes

▼ Click for Answer

Answer:

The correct answer is (3)

The output of the model is a list of 10 numbers. These numbers are a probability that the value being classified is the corresponding value (https://github.com/zalandoresearch/fashion-mnist#labels), i.e. the first value in the list is the probability that the image is of a '0' (T-shirt/top), the next is a '1' (Trouser) etc. Notice that they are all VERY LOW probabilities.

For index 9 (Ankle boot), the probability was in the 90's, i.e. the neural network is telling us that the image is most likely an ankle boot.

E1Q2: How do you know that this list tells you that the item is an ankle boot?

1. There's not enough information to answer that question
2. The 10th element on the list is the biggest, and the ankle boot is labelled 9
3. The ankle boot is label 9, and there are 0->9 elements in the list

▼ Click for Answer

Answer

The correct answer is (2). Both the list and the labels are 0 based, so the ankle boot having label 9 means that it is the 10th of the 10 classes. The list having the 10th element being the highest value means that the Neural Network has predicted that the item it is classifying is most likely an ankle boot

Exercise 2:

Let's now look at the layers in your model. Experiment with different values for the dense layer with 512 neurons. What different results do you get for loss, training time etc? Why do you think that's the case?

```
mnist = tf.keras.datasets.mnist

(training_images, training_labels) ,  (test_images, test_labels) = mnist.load_data()

training_images = training_images/255.0
test_images = test_images/255.0

model = tf.keras.models.Sequential([tf.keras.layers.Flatten(),
                                    tf.keras.layers.Dense(1024, activation=tf.nn.relu), # Try experimenting with this laye
                                    tf.keras.layers.Dense(10, activation=tf.nn.softmax)])

model.compile(optimizer = 'adam',
              loss = 'sparse_categorical_crossentropy')

model.fit(training_images, training_labels, epochs=5)

model.evaluate(test_images, test_labels)

classifications = model.predict(test_images)

print(classifications[0])
print(test_labels[0])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
11501568/11490434 [==============================] - 0s 0us/step
Epoch 1/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.1866
Epoch 2/5
1875/1875 [==============================] - 12s 7ms/step - loss: 0.0767
Epoch 3/5
1875/1875 [==============================] - 11s 6ms/step - loss: 0.0483
Epoch 4/5
1875/1875 [==============================] - 12s 7ms/step - loss: 0.0360
Epoch 5/5
1875/1875 [==============================] - 12s 6ms/step - loss: 0.0259
313/313 [==============================] - 1s 3ms/step - loss: 0.0958
```

```
[1.03522568e-09 1.30515367e-08 1.15607275e-08 7.94161460e-04 4.52195703e-12 5.25995993e-08 3.26235160e-13 9.99201000e
7
```

```
mnist = tf.keras.datasets.mnist

(training_images, training_labels) ,  (test_images, test_labels) = mnist.load_data()

training_images = training_images/255.0
test_images = test_images/255.0

model = tf.keras.models.Sequential([tf.keras.layers.Flatten(),
                                    tf.keras.layers.Dense(512, activation=tf.nn.relu), # Try experimenting with this layer
                                    tf.keras.layers.Dense(10, activation=tf.nn.softmax)])

model.compile(optimizer = 'adam',
              loss = 'sparse_categorical_crossentropy')

model.fit(training_images, training_labels, epochs=5)

model.evaluate(test_images, test_labels)

classifications = model.predict(test_images)

print(classifications[0])
print(test_labels[0])
```

```
Epoch 1/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2001
Epoch 2/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0818
Epoch 3/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0533
Epoch 4/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0381
Epoch 5/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0280
313/313 [==============================] - 1s 2ms/step - loss: 0.0789
```

```
[6.2099019e-09 1.0282420e-10 5.0267285e-08 1.9889719e-06 5.1036942e-13 1.1057601e-08 2.2859202e-13 9.9999690e-01 1.56
7
```

E2Q1: Increase to 1024 Neurons -- What's the impact?

1. Training takes longer, but is more accurate
2. Training takes longer, but no impact on accuracy
3. Training takes the same time, but is more accurate

▼ Click for Answer

Answer

The correct answer is (1) by adding more Neurons we have to do more calculations, slowing down the process, but in this case they have a good impact -- we do get more accurate. That doesn't mean it's always a case of 'more is better', you can hit the law of diminishing returns very quickly!

Exercise 3:

E3Q1: What would happen if you remove the Flatten() layer. Why do you think that's the case?

▼ Click for Answer

Answer

You get an error about the shape of the data. It may seem vague right now, but it reinforces the rule of thumb that the first layer in your network should be the same shape as your data. Right now our data is 28x28 images, and 28 layers of 28 neurons would be infeasible, so it makes more sense to 'flatten' that 28,28 into a 784x1. Instead of writng all the code to handle that ourselves, we add the Flatten() layer at the begining, and when the arrays are loaded into the model later, they'll automatically be flattened for us.

```
mnist = tf.keras.datasets.mnist
```

```
(training_images, training_labels) ,  (test_images, test_labels) = mnist.load_data()

training_images = training_images/255.0
test_images = test_images/255.0

model = tf.keras.models.Sequential([tf.keras.layers.Flatten(), #Try removing this layer
                                    tf.keras.layers.Dense(64, activation=tf.nn.relu),
                                    tf.keras.layers.Dense(10, activation=tf.nn.softmax)])

model.compile(optimizer = 'adam',
              loss = 'sparse_categorical_crossentropy')

model.fit(training_images, training_labels, epochs=5)

model.evaluate(test_images, test_labels)

classifications = model.predict(test_images)

print(classifications[0])
print(test_labels[0])
```

```
Epoch 1/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.3083
Epoch 2/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1483
Epoch 3/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.1093
Epoch 4/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0860
Epoch 5/5
1875/1875 [==============================] - 3s 2ms/step - loss: 0.0709
313/313 [==============================] - 1s 1ms/step - loss: 0.0888
[3.1270240e-05 2.7959496e-07 8.6200525e-05 2.3762627e-02 2.0591280e-09 5.7553311e-06 6.9258974e-11 9.7570002e-01 2.83
7
```

Exercise 4:

Consider the final (output) layers. Why are there 10 of them? What would happen if you had a different amount than 10? For example, try training the network with 5.

▼ Click for Answer

Answer

You get an error as soon as it finds an unexpected value. Another rule of thumb -- the number of neurons in the last layer should match the number of classes you are classifying for. In this case it's the digits 0-9, so there are 10 of them, hence you should have 10 neurons in your final layer.

```
mnist = tf.keras.datasets.mnist

(training_images, training_labels) ,  (test_images, test_labels) = mnist.load_data()

training_images = training_images/255.0
test_images = test_images/255.0

model = tf.keras.models.Sequential([tf.keras.layers.Flatten(),
                                    tf.keras.layers.Dense(64, activation=tf.nn.relu),
                                    tf.keras.layers.Dense(5, activation=tf.nn.softmax) # Try experimenting with this layer
                                   ])

model.compile(optimizer = 'adam',
              loss = 'sparse_categorical_crossentropy')

model.fit(training_images, training_labels, epochs=5)

model.evaluate(test_images, test_labels)

classifications = model.predict(test_images)

print(classifications[0])
print(test_labels[0])
```

```
Epoch 1/5
---------------------------------------------------------------------------
InvalidArgumentError                      Traceback (most recent call last)
<ipython-input-18-6f7206ecf9a2> in <module>()
     14               loss = 'sparse_categorical_crossentropy')
     15 
---> 16 model.fit(training_images, training_labels, epochs=5)
     17 
     18 model.evaluate(test_images, test_labels)

1 frames

/usr/local/lib/python3.7/dist-packages/tensorflow/python/eager/execute.py in quick_execute(op_name, num_outputs, inpu
     53     ctx.ensure_initialized()
     54     tensors = pywrap_tfe.TFE_Py_Execute(ctx._handle, device_name, op_name,
---> 55                                         inputs, attrs, num_outputs)
     56   except core._NotOkStatusException as e:
     57     if name is not None:

InvalidArgumentError: Graph execution error:

Detected at node 'sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/SparseSoftmaxCrossEntropyWithLc
call last):
    File "/usr/lib/python3.7/runpy.py", line 193, in _run_module_as_main
      "__main__", mod_spec)
    File "/usr/lib/python3.7/runpy.py", line 85, in _run_code
      exec(code, run_globals)
    File "/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py", line 16, in <module>
      app.launch_new_instance()
    File "/usr/local/lib/python3.7/dist-packages/traitlets/config/application.py", line 846, in launch_instance
      app.start()
    File "/usr/local/lib/python3.7/dist-packages/ipykernel/kernelapp.py", line 499, in start
      self.io_loop.start()
    File "/usr/local/lib/python3.7/dist-packages/tornado/platform/asyncio.py", line 132, in start
      self.asyncio_loop.run_forever()
    File "/usr/lib/python3.7/asyncio/base_events.py", line 541, in run_forever
      self._run_once()
    File "/usr/lib/python3.7/asyncio/base_events.py", line 1786, in _run_once
      handle._run()
    File "/usr/lib/python3.7/asyncio/events.py", line 88, in _run
      self._context.run(self._callback, *self._args)
    File "/usr/local/lib/python3.7/dist-packages/tornado/ioloop.py", line 758, in _run_callback
```

```
    ret = callback()
  File "/usr/local/lib/python3.7/dist-packages/tornado/stack_context.py", line 300, in null_wrapper
    return fn(*args, **kwargs)
  File "/usr/local/lib/python3.7/dist-packages/zmq/eventloop/zmqstream.py", line 536, in <lambda>
    self.io_loop.add_callback(lambda: self._handle_events(self.socket, 0))
  File "/usr/local/lib/python3.7/dist-packages/zmq/eventloop/zmqstream.py", line 452, in _handle_events
    self._handle_recv()
  File "/usr/local/lib/python3.7/dist-packages/zmq/eventloop/zmqstream.py", line 481, in _handle_recv
    self._run_callback(callback, msg)
  File "/usr/local/lib/python3.7/dist-packages/zmq/eventloop/zmqstream.py", line 431, in _run_callback
    callback(*args, **kwargs)
  File "/usr/local/lib/python3.7/dist-packages/tornado/stack_context.py", line 300, in null_wrapper
    return fn(*args, **kwargs)
  File "/usr/local/lib/python3.7/dist-packages/ipykernel/kernelbase.py", line 283, in dispatcher
    return self.dispatch_shell(stream, msg)
  File "/usr/local/lib/python3.7/dist-packages/ipykernel/kernelbase.py", line 233, in dispatch_shell
    handler(stream, idents, msg)
  File "/usr/local/lib/python3.7/dist-packages/ipykernel/kernelbase.py", line 399, in execute_request
    user_expressions, allow_stdin)
  File "/usr/local/lib/python3.7/dist-packages/ipykernel/ipkernel.py", line 208, in do_execute
    res = shell.run_cell(code, store_history=store_history, silent=silent)
  File "/usr/local/lib/python3.7/dist-packages/ipykernel/zmqshell.py", line 537, in run_cell
    return super(ZMQInteractiveShell, self).run_cell(*args, **kwargs)
  File "/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py", line 2718, in run_cell
    interactivity=interactivity, compiler=compiler, result=result)
  File "/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py", line 2822, in run_ast_nodes
    if self.run_code(code, result):
  File "/usr/local/lib/python3.7/dist-packages/IPython/core/interactiveshell.py", line 2882, in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)
  File "<ipython-input-18-6f7206ecf9a2>", line 16, in <module>
    model.fit(training_images, training_labels, epochs=5)
  File "/usr/local/lib/python3.7/dist-packages/keras/utils/traceback_utils.py", line 64, in error_handler
    return fn(*args, **kwargs)
  File "/usr/local/lib/python3.7/dist-packages/keras/engine/training.py", line 1384, in fit
    tmp_logs = self.train_function(iterator)
  File "/usr/local/lib/python3.7/dist-packages/keras/engine/training.py", line 1021, in train_function
    return step_function(self, iterator)
  File "/usr/local/lib/python3.7/dist-packages/keras/engine/training.py", line 1010, in step_function
    outputs = model.distribute_strategy.run(run_step, args=(data,))
  File "/usr/local/lib/python3.7/dist-packages/keras/engine/training.py", line 1000, in run_step
    outputs = model.train_step(data)
```

```
    File "/usr/local/lib/python3.7/dist-packages/keras/engine/training.py", line 860, in train_step
      loss = self.compute_loss(x, y, y_pred, sample_weight)
    File "/usr/local/lib/python3.7/dist-packages/keras/engine/training.py", line 919, in compute_loss
      y, y_pred, sample_weight, regularization_losses=self.losses)
    File "/usr/local/lib/python3.7/dist-packages/keras/engine/compile_utils.py", line 201, in __call__
      loss_value = loss_obj(y_t, y_p, sample_weight=sw)
    File "/usr/local/lib/python3.7/dist-packages/keras/losses.py", line 141, in __call__
      losses = call_fn(y_true, y_pred)
    File "/usr/local/lib/python3.7/dist-packages/keras/losses.py", line 245, in call
      return ag_fn(y_true, y_pred, **self._fn_kwargs)
    File "/usr/local/lib/python3.7/dist-packages/keras/losses.py", line 1863, in sparse_categorical_crossentropy
      y_true, y_pred, from_logits=from_logits, axis=axis)
    File "/usr/local/lib/python3.7/dist-packages/keras/backend.py", line 5203, in sparse_categorical_crossentropy
      labels=target, logits=output)
Node: 'sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/SparseSoftmaxCrossEntropyWithLogits'
Received a label value of 9 which is outside the valid range of [0, 5).  Label values: 5 6 0 3 0 5 3 6 9 0 5 7 4 3 4
         [[{{node sparse_categorical_crossentropy/SparseSoftmaxCrossEntropyWithLogits/SparseSoftmaxCrossEntropyWithLc
[Op:__inference_train_function_133449]
```

SEARCH STACK OVERFLOW

Exercise 5:

Consider the effects of additional layers in the network. What will happen if you add another layer between the one with 512 and the final layer with 10.

▶ Click for Answer

```
mnist = tf.keras.datasets.mnist

(training_images, training_labels) ,  (test_images, test_labels) = mnist.load_data()

training_images = training_images/255.0
test_images = test_images/255.0

model = tf.keras.models.Sequential([tf.keras.layers.Flatten(),
```

```
                                    tf.keras.layers.Dense(512, activation=tf.nn.relu),
                                    # Add a layer here,
                                    tf.keras.layers.Dense(256, activation=tf.nn.relu),
                                    # Add a layer here
                                    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
                                  ])

model.compile(optimizer = 'adam',
              loss = 'sparse_categorical_crossentropy')

model.fit(training_images, training_labels, epochs=5)

model.evaluate(test_images, test_labels)

classifications = model.predict(test_images)

print(classifications[0])
print(test_labels[0])
```

```
Epoch 1/5
1875/1875 [==============================] - 11s 5ms/step - loss: 0.1860
Epoch 2/5
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0786
Epoch 3/5
1875/1875 [==============================] - 12s 7ms/step - loss: 0.0552
Epoch 4/5
1875/1875 [==============================] - 13s 7ms/step - loss: 0.0410
Epoch 5/5
1875/1875 [==============================] - 9s 5ms/step - loss: 0.0334
313/313 [==============================] - 1s 2ms/step - loss: 0.0752
[8.7240895e-15 1.3932053e-09 1.7279129e-09 1.5269232e-10 7.4466762e-13 2.3543857e-14 3.4480312e-17 1.0000000e+00 3.39
7
```

Exercise 6:

E6Q1: Consider the impact of training for more or less epochs. Why do you think that would be the case?

- Try 15 epochs -- you'll probably get a model with a much better loss than the one with 5
- Try 30 epochs -- you might see the loss value stops decreasing, and sometimes increases.

This is a side effect of something called 'overfitting' which you can learn about later and it's something you need to keep an eye out for when training neural networks. There's no point in wasting your time training if you aren't improving your loss, right! :)

```
mnist = tf.keras.datasets.mnist

(training_images, training_labels) ,  (test_images, test_labels) = mnist.load_data()

training_images = training_images/255.0
test_images = test_images/255.0

model = tf.keras.models.Sequential([tf.keras.layers.Flatten(),
                                    tf.keras.layers.Dense(128, activation=tf.nn.relu),
                                    tf.keras.layers.Dense(10, activation=tf.nn.softmax)])

model.compile(optimizer = 'adam',
              loss = 'sparse_categorical_crossentropy')

model.fit(training_images, training_labels, epochs=5) # Experiment with the number of epochs

model.evaluate(test_images, test_labels)

classifications = model.predict(test_images)

print(classifications[34])
print(test_labels[34])
```

```
Epoch 1/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.2561
Epoch 2/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.1142
Epoch 3/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0784
Epoch 4/5
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0584
Epoch 5/5
```

```
1875/1875 [==============================] - 4s 2ms/step - loss: 0.0472
313/313 [==============================] - 1s 1ms/step - loss: 0.0804
[4.5672674e-10 3.7722353e-07 4.2801323e-05 2.7070224e-05 1.1856756e-10 1.1732532e-10 1.9565301e-11 9.9992299e-01 6.63
7
```

Exercise 7:

Before you trained, you normalized the data, going from values that were 0-255 to values that were 0-1. What would be the impact of removing that? Here's the complete code to give it a try. Why do you think you get different results?

```
mnist = tf.keras.datasets.mnist
(training_images, training_labels), (test_images, test_labels) = mnist.load_data()
training_images=training_images/255.0 # Experiment with removing this line
test_images=test_images/255.0 # Experiment with removing this line
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy')
model.fit(training_images, training_labels, epochs=5)
model.evaluate(test_images, test_labels)
classifications = model.predict(test_images)
print(classifications[0])
print(test_labels[0])
```

```
Epoch 1/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1982
Epoch 2/5
1875/1875 [==============================] - 10s 5ms/step - loss: 0.0796
Epoch 3/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0529
Epoch 4/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0368
Epoch 5/5
1875/1875 [==============================] - 7s 4ms/step - loss: 0.0269
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0833
[1.6795008e-11 3.4542820e-09 8.5565155e-08 1.3560115e-05 1.9320485e-13 1.1306874e-09 6.0706592e-16 9.9998569e-01 3.49
7
```

Exercise 8:

Earlier when you trained for extra epochs you had an issue where your loss might change. It might have taken a bit of time for you to wait for the training to do that, and you might have thought 'wouldn't it be nice if I could stop the training when I reach a desired value?' -- i.e. 95% accuracy might be enough for you, and if you reach that after 3 epochs, why sit around waiting for it to finish a lot more epochs....So how would you fix that? Like any other program...you have callbacks! Let's see them in action...

```
class myCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if(logs.get('accuracy') >= 0.6): # Experiment with changing this value
      print("\nReached 60% accuracy so cancelling training!")
      self.model.stop_training = True

callbacks = myCallback()
mnist = tf.keras.datasets.fashion_mnist
(training_images, training_labels), (test_images, test_labels) = mnist.load_data()
training_images=training_images/255.0
test_images=test_images/255.0
model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
model.fit(training_images, training_labels, epochs=5, callbacks=[callbacks])
```

```
Epoch 1/5
1869/1875 [============================>.] - ETA: 0s - loss: 0.4786 - accuracy: 0.8295
Reached 60% accuracy so cancelling training!
```

```
1875/1875 [==============================] - 7s 4ms/step - loss: 0.4783 - accuracy: 0.8295
<keras.callbacks.History at 0x7fbea1693d90>
```

✓ 7s completed at 2:28 PM