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 v example of learned behavior. Of course, in that instance, it was a bit of overkill because it would ha directly, instead of bothering with using Machine Learning to learn the relationship between X and for all values.

But what about a scenario where writing rules like that is much more difficult -- for example a com scenario where we can recognize different items of clothing, trained from a dataset containing 10

Start Coding

Let's start with our import of TensorFlow

```
import tensorflow as tf
print(tf.__version__)
```

```
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/dtypes.py:5
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/dtypes.py:5
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/dtypes.py:5
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/dtypes.py:5
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/dtypes.py:5
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/dtypes.py:5
  np_resource = np.dtype([("resource", np.ubyte, 1)])
2.0.0-alpha0
/usr/local/lib/python3.6/dist-packages/tensorboard/compat/tensorflow_stub/dtyp
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/usr/local/lib/python3.6/dist-packages/tensorboard/compat/tensorflow_stub/dtyp
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/usr/local/lib/python3.6/dist-packages/tensorboard/compat/tensorflow_stub/dtyp
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/usr/local/lib/python3.6/dist-packages/tensorboard/compat/tensorflow_stub/dtyp
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/usr/local/lib/python3.6/dist-packages/tensorboard/compat/tensorflow_stub/dtyp
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/usr/local/lib/python3.6/dist-packages/tensorboard/compat/tensorflow_stub/dtyp
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

```
!pip install tensorflow==2.0.0-alpha0
```

```
Requirement already satisfied: tensorflow==2.0.0-alpha0 in /usr/local/lib/pyth
Requirement already satisfied: astor>=0.6.0 in /usr/local/lib/python3.6/dist-p
Requirement already satisfied: wheel>=0.26 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: keras-applications>=1.0.6 in /usr/local/lib/pyt
Requirement already satisfied: keras-preprocessing>=1.0.5 in /usr/local/lib/py
Requirement already satisfied: tb-nightly<1.14.0a20190302,>=1.14.0a20190301 in
Requirement already satisfied: grpcio>=1.8.6 in /usr/local/lib/python3.6/dist-
Requirement already satisfied: protobuf>=3.6.1 in /usr/local/lib/python3.6/dis
Requirement already satisfied: numpy<2.0,>=1.14.5 in /usr/local/lib/python3.6/
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.6/di
Requirement already satisfied: tf-estimator-nightly<1.14.0.dev2019030116,>=1.1
Requirement already satisfied: gast>=0.2.0 in /usr/local/lib/python3.6/dist-pa
Requirement already satisfied: google-pasta>=0.1.2 in /usr/local/lib/python3.6
Requirement already satisfied: absl-py>=0.7.0 in /usr/local/lib/python3.6/dist
Requirement already satisfied: h5py in /usr/local/lib/python3.6/dist-packages
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.6/dis
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.6/d
Requirement already satisfied: setuptools in /usr/local/lib/python3.6/dist-pac
```

The Fashion MNIST data is available directly in the tf.keras datasets API. You load it like this:

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

Calling load_data on this object will give you two sets of two lists, these will be the training and tes
clothing items and their labels.

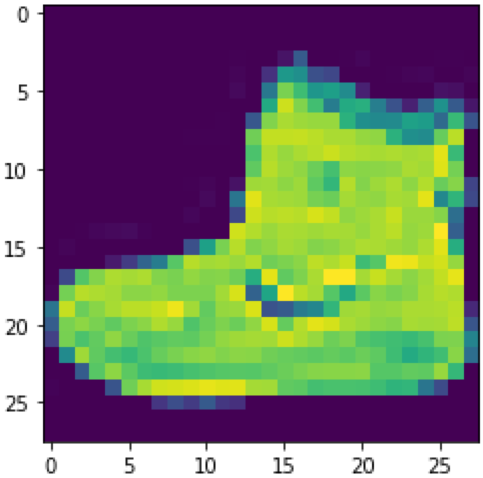
```
(training_images, training_labels), (test_images, test_labels) = mnist.load_da
```

What does these values look like? Let's print a training image, and a training label to see...Experim
example, also take a look at index 42...that's a a different boot than the one at index 0

```
import matplotlib.pyplot as plt
plt.imshow(training_images[0])
print(training_labels[0])
print(training_images[0])
```

--NORMAL--

```
[  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]
 [  0   0  74 189 212 191 175 172 175 181 185 188 189 188 193 198 204 209
  210 210 211 188 188 194 192 216 170   0]
 [  2   0   0   0  66 200 222 237 239 242 246 243 244 221 220 193 191 179
  182 182 181 176 166 168  99  58   0   0]
 [  0   0   0   0   0   0   0  40  61  44  72  41  35   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]]
```



You'll notice that all of the values in the number are between 0 and 255. If we are training a neural treat all values as between 0 and 1, a process called '**normalizing**'...and fortunately in Python it's ea looping. You do it like this:

```
training_images  = training_images / 255.0
test_images = test_images / 255.0
```

Now you might be wondering why there are 2 sets...training and testing -- remember we spoke abo of data for training, and then another set of data...that the model hasn't yet seen...to see how good when you're done, you're going to want to try it out with data that it hadn't previously seen!

Let's now design the model. There's quite a few new concepts here, but don't worry, you'll get the h

```
model = tf.keras.models.Sequential([tf.keras.layers.Flatten(),
                                    tf.keras.layers.Dense(128, activation=tf.r
                                    tf.keras.layers.Dense(10, activation=tf.nn
```

Sequential: That defines a SEQUENCE of layers in the neural network

Flatten: Remember earlier where our images were a square, when you printed them out? Flatten ju dimensional set.

Dense: Adds a layer of neurons

Each layer of neurons need an **activation function** to tell them what to do. There's lots of options, b

Relu effectively means "If X>0 return X, else return 0" -- so what it does it it only passes values 0 or

Softmax takes a set of values, and effectively picks the biggest one, so, for example, if the output c 9.5, 0.1, 0.05, 0.05, 0.05], it saves you from fishing through it looking for the biggest value, and turn a lot of coding!

The next thing to do, now the model is defined, is to actually build it. You do this by compiling it wit and then you train it by calling **model.fit ** asking it to fit your training data to your training labels -- the training data and its actual labels, so in future if you have data that looks like the training data, data would look like.

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

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

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-8-9eacb0a4baaf> in <module>()
----> 1 model.compile(optimizer = tf.train.AdamOptimizer(),
      2               loss = 'sparse_categorical_crossentropy',
      3               metrics=['accuracy'])
      4
      5 model.fit(training_images, training_labels, epochs=5)

AttributeError: module 'tensorflow._api.v2.train' has no attribute 'AdamOptimi
```

SEARCH STACK OVERFLOW

Once it's done training -- you should see an accuracy value at the end of the final epoch. It might lo your neural network is about 91% accurate in classifying the training data. I.E., it figured out a patte that worked 91% of the time. Not great, but not bad considering it was only trained for 5 epochs ar

But how would it work with unseen data? That's why we have the test images. We can call model.e report back the loss for each. Let's give it a try:

```
model.evaluate(test_images, test_labels)
```

For me, that returned a accuracy of about .8838, which means it was about 88% accurate. As expe *unseen* data as it did with data it was trained on! As you go through this course, you'll look at ways

To explore further, try the below exercises:

Exploration Exercises

Exercise 1:

For this first exercise run the below code: It creates a set of classifications for each of the test ima classifications. The output, after you run it is a list of numbers. Why do you think this is, and what

```
classifications = model.predict(test_images)

print(classifications[0])
```

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

Hint: try running print(test_labels[0]) -- and you'll get a 9. Does that help you understand why this lis

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

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

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 bei
the first value in the list is the probability that the handwriting is of a '0', the next is a '1' etc. Notice

For the 7, the probability was .999+, i.e. the neural network is telling us that it's almost certainly a 7

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

Answer

The correct answer is (2). Both the list and the labels are 0 based, so the ankle boot having label 9
The list having the 10th element being the highest value means that the Neural Network has predic
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
get for loss, training time etc? Why do you think that's the case?

```
import tensorflow as tf
print(tf.__version__)

mnist = tf.keras.datasets.mnist

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

training_images = training_images/255.0
test_images = test_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
                                    tf.keras.layers.Dense(10, activation=tf.nn

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])
```

Question 1. 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

Answer

The correct answer is (1) by adding more Neurons we have to do more calculations, slowing down good impact -- we do get more accurate. That doesn't mean it's always a case of 'more is better', y quickly!

Exercise 3:

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

You get an error about the shape of the data. It may seem vague right now, but it reinforces the rul should be the same shape as your data. Right now our data is 28x28 images, and 28 layers of 28 r sense to 'flatten' that 28,28 into a 784x1. Instead of wriitng all the code to handle that ourselves, w when the arrays are loaded into the model later, they'll automatically be flattened for us.

```
import tensorflow as tf
print(tf.__version__)

mnist = tf.keras.datasets.mnist

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

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
                                    tf.keras.layers.Dense(10, activation=tf.nn

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])
```

Exercise 4:

Consider the final (output) layers. Why are there 10 of them? What would happen if you had a diffe training the network with 5

You get an error as soon as it finds an unexpected value. Another rule of thumb -- the number of ne number of classes you are classifying for. In this case it's the digits 0-9, so there are 10 of them, he layer.

```
import tensorflow as tf
print(tf.__version__)

mnist = tf.keras.datasets.mnist

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

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
                                    tf.keras.layers.Dense(5, activation=tf.nn

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])
```

Exercise 5:

Consider the effects of additional layers in the network. What will happen if you add another layer with 10.

Ans: There isn't a significant impact -- because this is relatively simple data. For far more complex

```
import tensorflow as tf
print(tf.__version__)

mnist = tf.keras.datasets.mnist

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

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.r
                                    tf.keras.layers.Dense(256, activation=tf.r
                                    tf.keras.layers.Dense(10, activation=tf.nn

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])
```

Exercise 6:

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 epoc decreasing, and sometimes increases. This is a side effect of something called 'overfitting' which y something you need to keep an eye out for when training neural networks. There's no point in wast your loss, right! :)

```
import tensorflow as tf
print(tf.__version__)

mnist = tf.keras.datasets.mnist

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

```

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.n
                                    tf.keras.layers.Dense(10, activation=tf.nn

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

model.fit(training_images, training_labels, epochs=30)

model.evaluate(test_images, test_labels)

classifications = model.predict(test_images)

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

Exercise 7:

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

```
import tensorflow as tf
print(tf.__version__)
mnist = tf.keras.datasets.mnist
(training_images, training_labels), (test_images, test_labels) = mnist.load_da
training_images=training_images
test_images=test_images
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])
```

```
2.0.0-alpha0
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datas
11493376/11490434 [==============================] - 0s 0us/step
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-10-79832ad8cc4f> in <module>()
     11 ])
     12 model.compile(optimizer='adam', loss='sparse_categorical_crossentropy'
---> 13 model.fit(training_images, training_labels, epochs=5)
     14 model.evaluate(test_images, test_labels)
     15 classifications = model.predict(test_images)

7 frames
/usr/local/lib/python3.6/dist-packages/tensorflow/python/keras/layers/core.py
    961     if not (dtype.is_floating or dtype.is_complex):
    962       raise TypeError('Unable to build `Dense` layer with non-floating
--> 963                       'dtype %s' % (dtype,))
    964     input_shape = tensor_shape.TensorShape(input_shape)
    965     if tensor_shape.dimension_value(input_shape[-1]) is None:

TypeError: Unable to build `Dense` layer with non-floating point dtype <dtype:
```

SEARCH STACK OVERFLOW

Exercise 8:

Earlier when you trained for extra epochs you had an issue where your loss might change. It might the training to do that, and you might have thought 'wouldn't it be nice if I could stop the training w accuracy might be enough for you, and if you reach that after 3 epochs, why sit around waiting for you fix that? Like any other program...you have callbacks! Let's see them in action...

```
import tensorflow as tf
print(tf.__version__)

class myCallback(tf.keras.callbacks.Callback):
  def on_epoch_end(self, epoch, logs={}):
    if(logs.get('loss')<0.4):
      print("\nReached 60% accuracy so cancelling training!")
      self.model.stop_training = True

callbacks = myCallback()
```

```
callbacks = myCallback()
mnist = tf.keras.datasets.fashion_mnist
(training_images, training_labels), (test_images, test_labels) = mnist.load_da
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')
model.fit(training_images, training_labels, epochs=5, callbacks=[callbacks])
```

```
2.0.0-alpha0
Epoch 1/5
60000/60000 [==============================] - 8s 127us/sample - loss: 0.4725
Epoch 2/5
59872/60000 [============================>.] - ETA: 0s - loss: 0.3587
Reached 60% accuracy so cancelling training!
60000/60000 [==============================] - 7s 119us/sample - loss: 0.3587
<tensorflow.python.keras.callbacks.History at 0x7fd19831b6d8>
```