2.1 An Introduction to computer vision

Computer vision is the field of having a computer understand and label what is present in an image.
So one way to solve that is to use lots of pictures of clothing and tell the computer what that's a picture of and then have the computer figure out the patterns that give you the difference between a shoe, and a shirt, and a handbag, and a coat.
Fortunately, there's a data set called **Fashion MNIST** which gives a 70 thousand images spread across 10 different items of clothing. These images have been scaled down to 28 by 28 pixels.
https://github.com/zalandoresearch/fashion-mnist

2.2 Writing code to load training data

```
fashion_mnist = keras.datasets.fashion_mnist
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

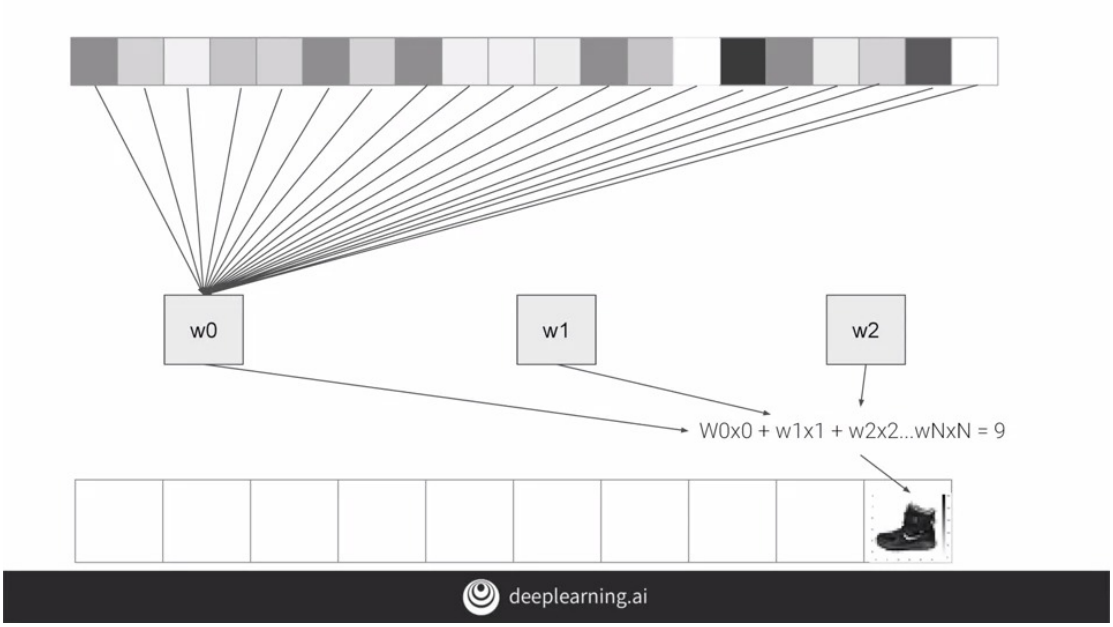
when building a neural network like this, it's a nice strategy to use some of your data to train
the neural network and similar data that the model hasn't yet seen to test how good it is at recognizing the images. So in the Fashion-MNIST data set, 60,000 of the 70,000 images are used to train the network, and then 10,000 images, one that it hasn't previously seen, can be used to test just how good or how bad it is performing.

Machine Learning Fairness
https://developers.google.com/machine-learning/fairness-overview/

2.3 Coding a Computer Vision Neural Network

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

Now we have three layers.
The last layer has 10 neurons in it because we have ten classes of clothing in the dataset.
The first layer is a flatten layer with the input shaping 28 by 28. Now, if you remember our images are 28 by 28, so we're specifying that this is the shape that we should expect the data to be in. Flatten takes this 28 by 28 square and turns it into a simple linear array.



2.4 Walk through a Notebook for computer vision

neural networks work better with normalized data
to make a guess as to what the relationship is between the input data and the output data, measure how well or how badly it did using the loss function, use the optimizer to generate a new gas and repeat
Course 1 - Part 4 - Lesson 2 – Notebook

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 just takes that square and turns it into a 1 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, but just use these for now.
Relu effectively means "If X>0 return X, else return 0" -- so what it does it it only passes values 0 or greater to the next layer in the network.
Softmax takes a set of values, and effectively picks the biggest one, so, for example, if the output of the last layer looks like [0.1, 0.1, 0.05, 0.1, 9.5, 0.1, 0.05, 0.05, 0.05], it saves you from fishing through it looking for the biggest value, and turns it into [0,0,0,0,1,0,0,0,0] -- The goal is to save a lot of coding!

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 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 wriitng 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.

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
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.

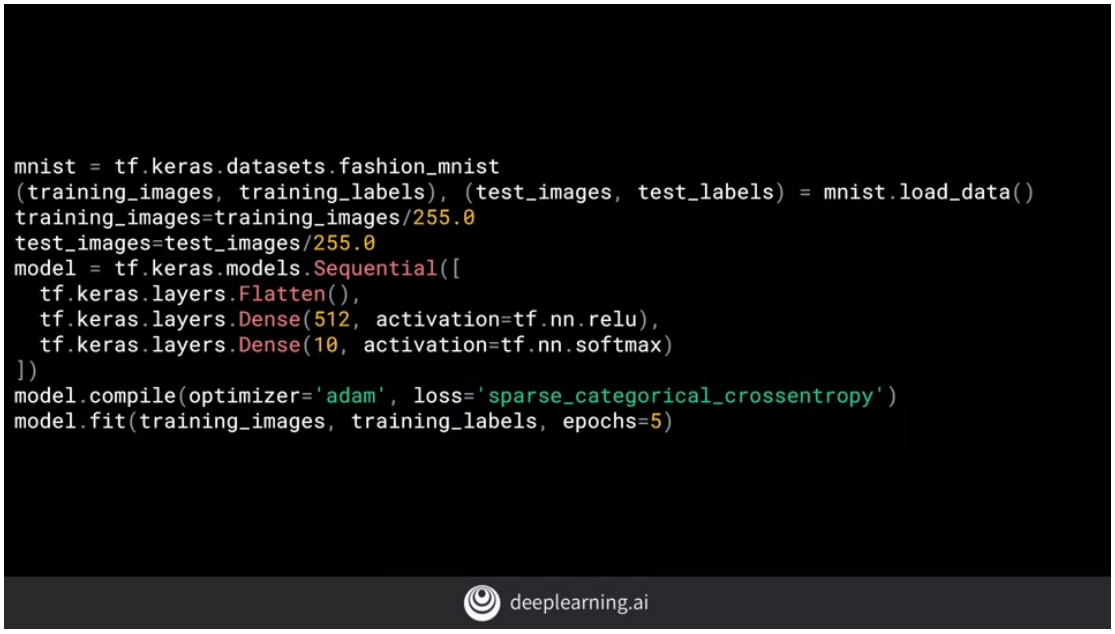
Consider the effects of additional layers in the network. What will happen if you add another layer between the one with 512 and the final layer with 10.
Ans: There isn't a significant impact -- because this is relatively simple data. For far more complex data (including color images to be classified as flowers that you'll see in the next lesson), extra layers are often necessary.

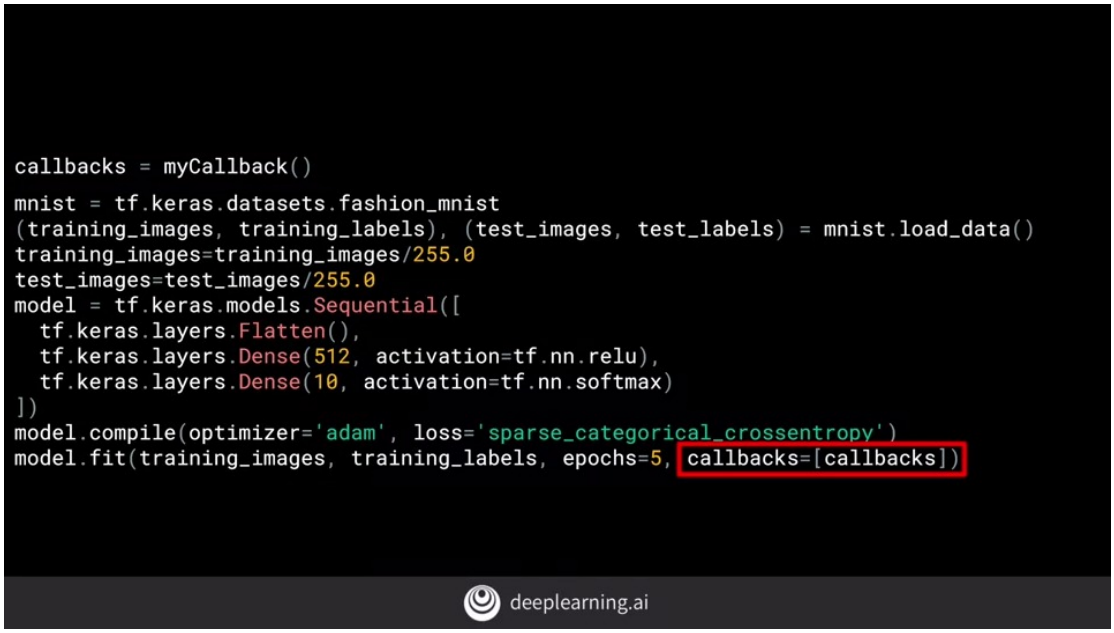
Consider the impact of training for more or less epochs. Why do you think that would be the case?
Try 15 epochs -- you'll probably get a model with a much better loss than the one with 5 Try 30 epochs -- you might see the loss value stops decreasing, and sometimes increases. This is a side effect of something called '**overfittin**g' which you can learn about 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! :)

2.5 Using Callbacks to control training

the training loop does support callbacks. So in every epoch, you can callback to a code function, having checked the metrics. If they're what you want to say, then you can cancel the training at that point.

Course 1 - Part 4 - Lesson 4 - Notebook.ipynb





2.6 Walk through a notebook with Callbacks

As the second epoch begins, it has already dropped below 0.4, but the callback hasn't been hit yet. That's because we set it up for on epoch end. It's good practice to do this, because with some data and some algorithms, the loss may vary up and down during the epoch, because all of the data hasn't yet been processed. So, I like to wait for the end to be sure.