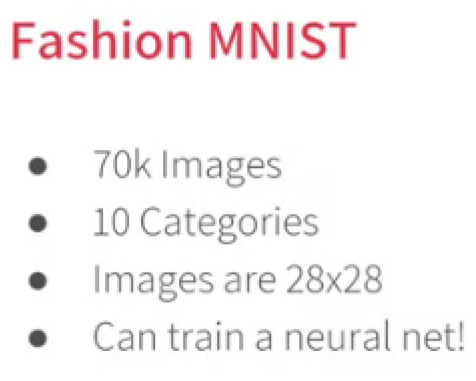
Introduction to TF for AI — Week 02

aakashgoel12.medium.com/introduction-to-tf-for-ai-week-02-ed5d6e190361

January 28, 2021


Fashion MNIST

- 70k Images
- 10 Categories
- Images are 28x28
- Can train a neural net!

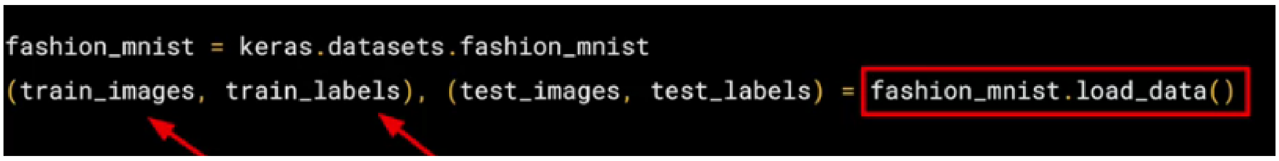
Aakash Goel

·6 days ago

Responsible AI practices — https://ai.google/responsibilities/responsible-ai-practices/

Fashion MNIST

- 70k Images
- 10 Categories
- Images are 28x28
- Can train a neural net!



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

Using a number is a first step in avoiding bias — instead of labelling it with words in a specific language and excluding people who don't speak that language! You can learn more about bias and techniques to avoid it here **(Impt.)**

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

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 testing
contain the clothing items and their labels.

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

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

```
import numpy as np
np.set_printoptions(linewidth=200)
import matplotlib.pyplot as plt
plt.imshow(training_images[0])
print(training_labels[0])
print(training_images[0])
```

You'll notice that all of the values in the number are between 0 and 255. If we are training a neural network, for various reasons it's easier if we treat all values as between 0 and 1, a process called '**normalizing**'...and fortunately in Python it's easy to normalize a list like this without 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 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...to see how good it would be at classifying values. After all, 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 hang of them.

```
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 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!

The next thing to do, now 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 – i.e. have it figure out the relationship between the training data and its actual labels, so in future if you have data that looks like the training data, then it can make a prediction for what that data would look like.

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

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

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. I.E., 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. We can call model.evaluate, and pass in the two sets, and it will report back the loss for each. Let's give it a try:

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

Experiment with Number of neurons in Dense Layer i.e. make it to 1024

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

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),
                                    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])
```

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 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!

Experiment with Flatten() Layer

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.

Experiment with output Layer

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

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.

Experiment with more Dense Layers

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.

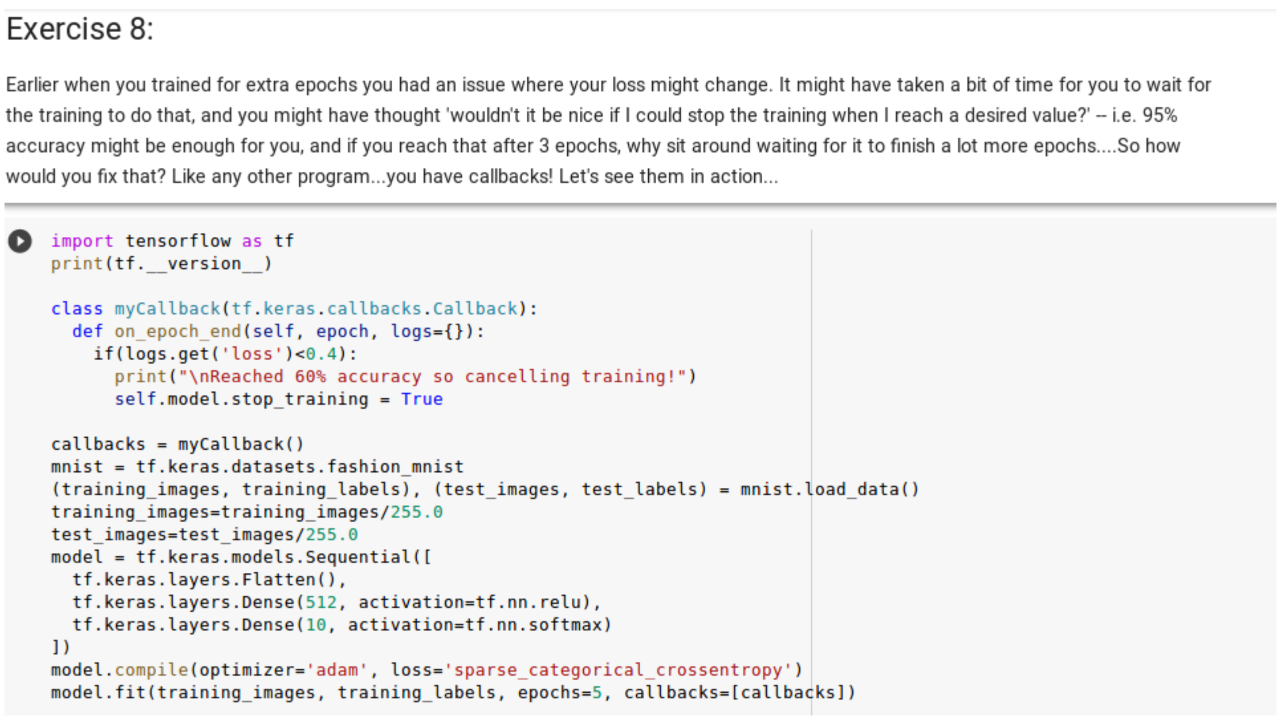
```
import tensorflow as tf
print(tf.__version__)

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),
                                    tf.keras.layers.Dense(256, activation=tf.nn.relu),
                                    tf.keras.layers.Dense(10, activation=tf.nn.softmax)])
```

Experiment with more epochs

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 [somewhere] 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! :)

Stop Training if Loss reaches value X — Use Callbacks



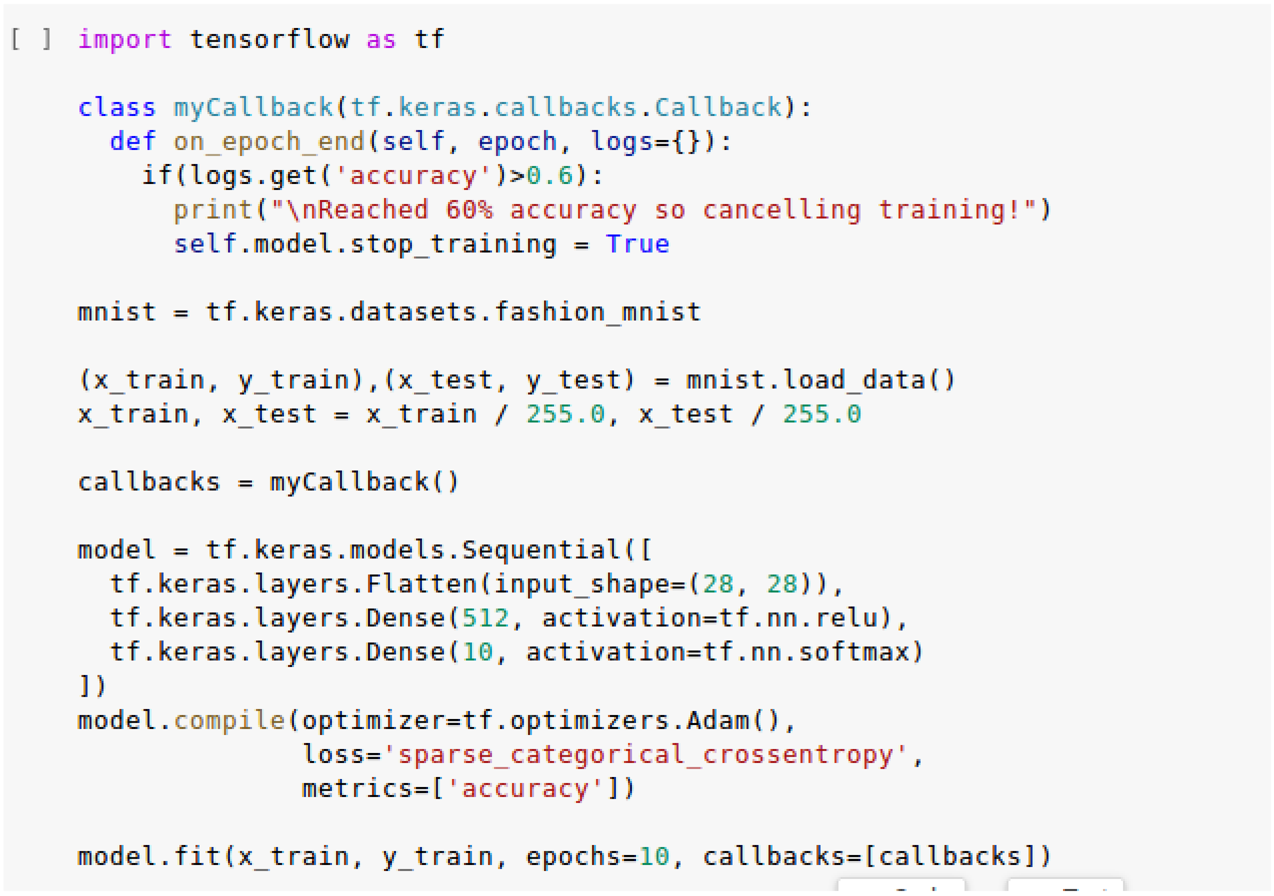
Exercise 8:

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...

```
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()
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')
model.fit(training_images, training_labels, epochs=5, callbacks=[callbacks])
```

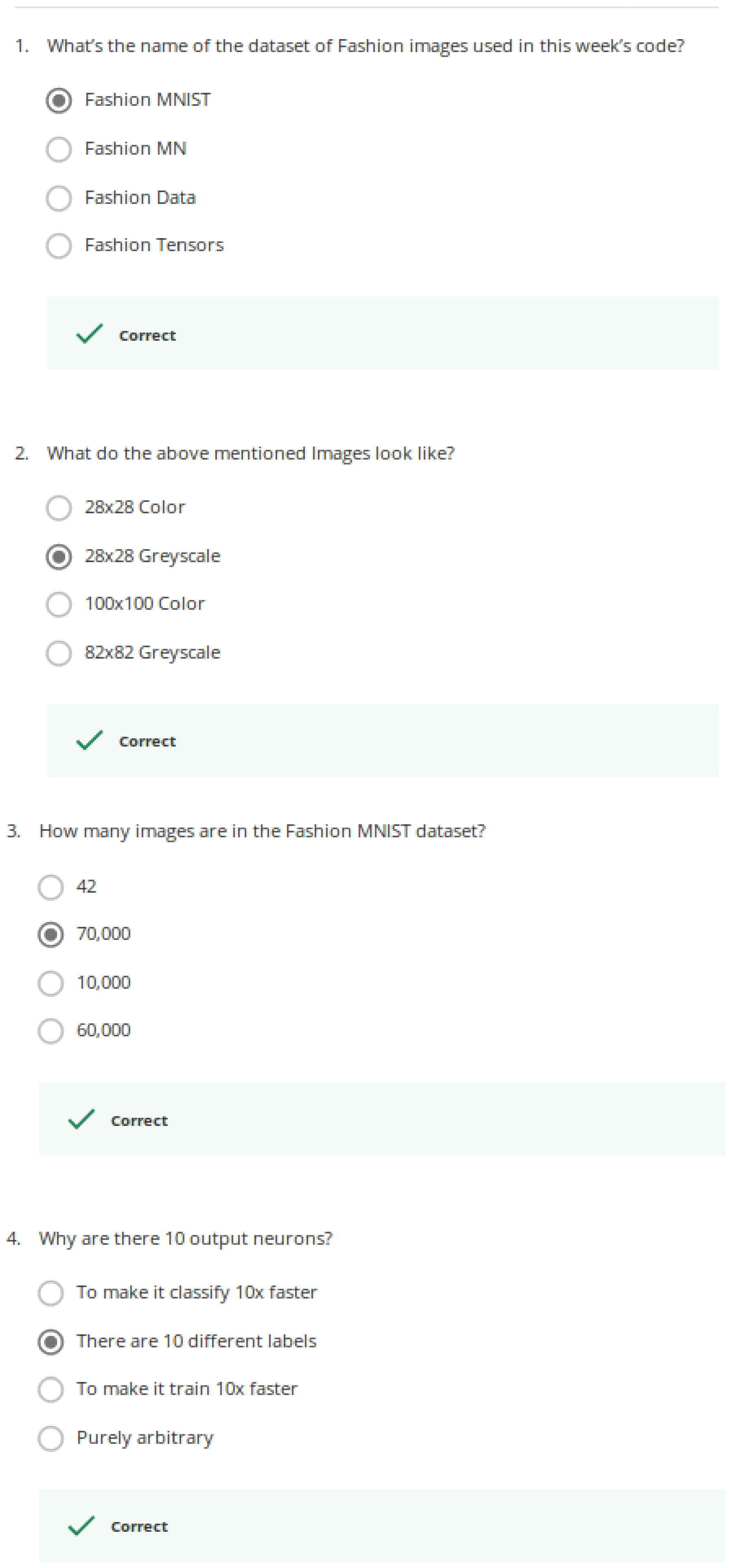
All Experiments in Notebook — https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/Course%201%20-%20Part%204%20-%20Lesson%202%20-%20Notebook.ipynb#scrollTo=E7W2PT66ZBHQ

Callbacks Example → https://colab.research.google.com/github/lmoroney/dlaicourse/blob/master/Course%201%20-%20Part%204%20-%20Lesson%204%20-%20Notebook.ipynb



```
[ ] import tensorflow as tf

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

    mnist = tf.keras.datasets.fashion_mnist

    (x_train, y_train),(x_test, y_test) = mnist.load_data()
    x_train, x_test = x_train / 255.0, x_test / 255.0

    callbacks = myCallback()

    model = tf.keras.models.Sequential([
      tf.keras.layers.Flatten(input_shape=(28, 28)),
      tf.keras.layers.Dense(512, activation=tf.nn.relu),
      tf.keras.layers.Dense(10, activation=tf.nn.softmax)
    ])
    model.compile(optimizer=tf.optimizers.Adam(),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    model.fit(x_train, y_train, epochs=10, callbacks=[callbacks])
```

QUIZ



1. What's the name of the dataset of Fashion images used in this week's code?

   [✓ Fashion MNIST]
   Fashion MN
   Fashion Data
   Fashion Tensors

   ✓ Correct

2. What do the above mentioned Images look like?

   28x28 Color
   [✓ 28x28 Greyscale]
   100x100 Color
   82x82 Greyscale

   ✓ Correct

3. How many images are in the Fashion MNIST dataset?

   42
   [✓ 70,000]
   10,000
   60,000

   ✓ Correct

4. Why are there 10 output neurons?

   To make it classify 10x faster
   [✓ There are 10 different labels]
   To make it train 10x faster
   Purely arbitrary

   ✓ Correct

5. What does Relu do?

- ○ For a value x, it returns 1/x
- ○ It only returns x if x is less than zero
- ○ It returns the negative of x
- [✓ It only returns x if x is greater than zero]

✓ Correct

6. Why do you split data into training and test sets?

- ○ To make testing quicker
- ○ To make training quicker
- [✓ To test a network with previously unseen data]
- ○ To train a network with previously unseen data

✓ Correct

7. What method gets called when an epoch finishes?

- [✓ on_epoch_end]
- ○ on_end
- ○ On_training_complete
- ○ on_epoch_finished

✓ Correct

8. What parameter to you set in your fit function to tell it to use callbacks?

- ○ callback=
- ○ oncallback=
- [✓ callbacks=]
- ○ oncallbacks=

✓ Correct

lmoroney/dlaicourse

Notebooks for learning deep learning. Contribute to lmoroney/dlaicourse development by creating an account on GitHub.

github.com