Week 2: Implementing Callbacks in TensorFlow using the MNIST Dataset

In the course you learned how to do classification using Fashion MNIST, a data set containing items of clothing. There's another, similar dataset called MNIST which has items of handwriting -- the digits 0 through 9.

Write an MNIST classifier that trains to 99% accuracy and stops once this threshold is achieved. In the lecture you saw how this was done for the loss but here you will be using accuracy instead.

Some notes:

1. Your network should succeed in less than 9 epochs.
2. When it reaches 99% or greater it should print out the string "Reached 99% accuracy so cancelling training!" and stop training.
3. If you add any additional variables, make sure you use the same names as the ones used in the class. This is important for the function signatures (the parameters and names) of the callbacks.

```
# IMPORTANT: This will check your notebook's metadata for grading.
# Please do not continue the lab unless the output of this cell tells you to proceed.
!python add_metadata.py --filename C1W2_Assignment.ipynb
```

```
Grader metadata detected! You can proceed with the lab!
```

NOTE: *To prevent errors from the autograder, you are not allowed to edit or delete non-graded cells in this notebook . Please only put your solutions in between the* `### START CODE HERE` *and* `### END CODE HERE` *code comments, and also refrain from adding any new cells.* ***Once you have passed this assignment*** *and want to experiment with any of the non-graded code, you may follow the instructions at the bottom of this notebook.*

```
# grader-required-cell

import os
import tensorflow as tf
from tensorflow import keras
```

Load and inspect the data

Begin by loading the data. A couple of things to notice:

- The file `mnist.npz` is already included in the current workspace under the `data` directory. By default the `load_data` from Keras accepts a path relative to `~/.keras/datasets` but in this case it is stored somewhere else, as a result of this, you need to specify the full path.

- `load_data` returns the train and test sets in the form of the tuples `(x_train, y_train), (x_test, y_test)` but in this exercise you will be needing only the train set so you can ignore the second tuple.

```
# grader-required-cell

# Load the data

# Get current working directory
current_dir = os.getcwd()

# Append data/mnist.npz to the previous path to get the full path
data_path = os.path.join(current_dir, "data/mnist.npz")

# Discard test set
(x_train, y_train), _ =
tf.keras.datasets.mnist.load_data(path=data_path)

# Normalize pixel values
x_train = x_train / 255.0
```

Now take a look at the shape of the training data:

```
# grader-required-cell

data_shape = x_train.shape

print(f"There are {data_shape[0]} examples with shape
({data_shape[1]}, {data_shape[2]})")
```

```
There are 60000 examples with shape (28, 28)
```

Defining your callback

Now it is time to create your own custom callback. For this complete the `myCallback` class and the `on_epoch_end` method in the cell below. If you need some guidance on how to proceed, check out this link.

```
# grader-required-cell

# GRADED CLASS: myCallback
### START CODE HERE

# Remember to inherit from the correct class
class myCallback(tf.keras.callbacks.Callback):
        # Define the correct function signature for on_epoch_end
        def on_epoch_end(self, epoch, logs={}):
            if logs.get('accuracy') is not None and
logs.get('accuracy') > 0.99: # @KEEP
```

```
                print("\nReached 99% accuracy so cancelling
training!")

                # Stop training once the above condition is met
                self.model.stop_training = True

### END CODE HERE
```

Create and train your model

Now that you have defined your callback it is time to complete the `train_mnist` function below.

You must set your model to train for 10 epochs and the callback should fire before the 9th epoch for you to pass this assignment.

Hint:

- Feel free to try the architecture for the neural network that you see fit but in case you need extra help you can check out an architecture that works pretty well at the end of this notebook.

```
# grader-required-cell

# GRADED FUNCTION: train_mnist
def train_mnist(x_train, y_train):

    ### START CODE HERE

    # Instantiate the callback class
    callbacks = myCallback()

    # Define the model
    model = tf.keras.models.Sequential([
        keras.layers.Flatten(input_shape=(28, 28)),
        keras.layers.Dense(512, activation=tf.nn.relu),
        keras.layers.Dense(10, activation=tf.nn.softmax)
    ])

    # Compile the model
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    # Fit the model for 10 epochs adding the callbacks
    # and save the training history
    history = model.fit(x_train, y_train, epochs=10,
callbacks=[callbacks])
```

```
    ### END CODE HERE

    return history
```

Call the `train_mnist` passing in the appropiate parameters to get the training history:

```
# grader-required-cell

hist = train_mnist(x_train, y_train)
```

```
Epoch 1/10
1875/1875 [==============================] - 8s 4ms/step - loss:
0.1995 - accuracy: 0.9402: 2s - ETA: 0s - loss: 0.210
Epoch 2/10
1875/1875 [==============================] - 8s 4ms/step - loss:
0.0784 - accuracy: 0.9754
Epoch 3/10
1875/1875 [==============================] - 8s 4ms/step - loss:
0.0507 - accuracy: 0.9843: 5s - loss: 0
Epoch 4/10
1875/1875 [==============================] - 8s 4ms/step - loss:
0.0356 - accuracy: 0.9879: 0s - loss: 0.0359 - accuracy: 0.
Epoch 5/10
1852/1875 [============================>.] - ETA: 0s - loss: 0.0279 -
accuracy: 0.9907 ETA: 6s - loss: 0 - ETA: 5s - loss: 0.0202 -  - ETA:
4s - loss: 0.0 - ETA: 3s - loss:
Reached 99% accuracy so cancelling training!
1875/1875 [==============================] - 8s 4ms/step - loss:
0.0280 - accuracy: 0.9906
```

If you see the message `Reached 99% accuracy so cancelling training!` printed out after less than 9 epochs it means your callback worked as expected.

Need more help?

Run the following cell to see an architecture that works well for the problem at hand:

```
# WE STRONGLY RECOMMEND YOU TO TRY YOUR OWN ARCHITECTURES FIRST
# AND ONLY RUN THIS CELL IF YOU WISH TO SEE AN ANSWER

import base64

encoded_answer =
"CiAgIC0gQSBGbGF0dGVuIGxheWVyIHRoYXQgcmVjZWl2ZXMgaW5wdXRzIHdpdGggdGhlI
HNhbWUgc2hhcGUgYXMgdGhlIGltYWdlcwogICAtIEEgRGVuc2UgbGF5ZXIgd2l0aCA1MTI
gdW5pdHMgYW5kIFJlTFUgYWN0aXZhdGlvbiBmdW5jdGlvbgogICAtIEEgRGVuc2UgbGF5Z
XIgd2l0aCAxMCB1bml0cyBhbmQgc29mdG1heCBhY3RpdmF0aW9uIGZ1bmN0aW9uCg=="
encoded_answer = encoded_answer.encode('ascii')
answer = base64.b64decode(encoded_answer)
```

```
answer = answer.decode('ascii')

print(answer)


   - A Flatten layer that receives inputs with the same shape as the
images
   - A Dense layer with 512 units and ReLU activation function
   - A Dense layer with 10 units and softmax activation function
```

Congratulations on finishing this week's assignment!

You have successfully implemented a callback that gives you more control over the training loop for your model. Nice job!

Keep it up!