```
import os
import base64
import numpy as np
import tensorflow as tf



# 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, "/content/DL2.html")

# Load data (discard test set)
(training_images, training_labels), _ = tf.keras.datasets.mnist.load_data(path=dat

print(f"training_images is of type {type(training_images)}.\ntraining_labels is of

# Inspect shape of the data
data_shape = training_images.shape

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

```
training_images is of type <class 'numpy.ndarray'>.
training_labels is of type <class 'numpy.ndarray'>

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

```
def reshape_and_normalize(images):
    """Reshapes the array of images and normalizes pixel values.

    Args:
        images (numpy.ndarray): The images encoded as numpy arrays

    Returns:
        numpy.ndarray: The reshaped and normalized images.
    """

    ### START CODE HERE ###

    # Reshape the images to add an extra dimension (at the right-most side of the
    images = np.expand_dims(images, axis=-1)

    # Normalize pixel values
    images = images/255.0

    ### END CODE HERE ###

    return images


# Reload the images in case you run this cell multiple times
(training_images, _), _ = tf.keras.datasets.mnist.load_data(path=data_path)
```

```
# Apply your function
training_images = reshape_and_normalize(training_images)

print('Name: A.ARUVI.          RegisterNumber: 212222230014.         \n')
print(f"Maximum pixel value after normalization: {np.max(training_images)}\n")
print(f"Shape of training set after reshaping: {training_images.shape}\n")
print(f"Shape of one image after reshaping: {training_images[0].shape}")
```

```
Name: A.ARUVI.          RegisterNumber: 212222230014.

Maximum pixel value after normalization: 1.0

Shape of training set after reshaping: (60000, 28, 28, 1)

Shape of one image after reshaping: (28, 28, 1)
```

```
# GRADED CLASS: EarlyStoppingCallback

### START CODE HERE ###

# Remember to inherit from the correct class
class EarlyStoppingCallback(tf.keras.callbacks.Callback):

    # Define the correct function signature for on_epoch_end method
    def on_epoch_end(self,epoch, logs={}):

        # Check if the accuracy is greater or equal to 0.98
        if (logs.get('accuracy')>=0.995):

            # Stop training once the above condition is met
            self.model.stop_training=True
            print("\nReached 98% accuracy so cancelling training!")
print('Name: A.ARUVI.            Register Number: 212222230014.        \n')
### END CODE HERE ###
```

```
Name: A.ARUVI.            Register Number: 212222230014.
```

```
def convolutional_model():
    """Returns the compiled (but untrained) convolutional model.

    Returns:
        tf.keras.Model: The model which should implement convolutions.
    """

    ## START CODE HERE ###

    # Define the model
    model = tf.keras.models.Sequential([

    # Add convolutions and max pooling
    tf.keras.Input(shape=(28,28,1)),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
```

```
    tf.keras.layers.MaxPooling2D(2, 2),
    tf.keras.layers.Conv2D(64, (3,3), activation='relu'),
    tf.keras.layers.MaxPooling2D(2,2),

    # Add the same layers as before
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')
])

    ### END CODE HERE ###

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

```
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	
conv2d_2 (Conv2D)	(None, 26, 26, 64)	
max_pooling2d_2 (MaxPooling2D)	(None, 13, 13, 64)	
conv2d_3 (Conv2D)	(None, 11, 11, 64)	
max_pooling2d_3 (MaxPooling2D)	(None, 5, 5, 64)	
flatten_1 (Flatten)	(None, 1600)	
dense_2 (Dense)	(None, 128)	
dense_3 (Dense)	(None, 10)	

Total params: 731,360 (2.79 MB)
Trainable params: 243,786 (952.29 KB)
Non-trainable params: 0 (0.00 B)
Optimizer params: 487,574 (1.86 MB)

```
### Model compiling and Training

# Define your compiled (but untrained) model
model = convolutional_model()

# Train your model (this can take up to 5 minutes)
training_history = model.fit(training_images, training_labels, epochs=10, callbacks=[Earl
```

```
Epoch 1/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 99s 52ms/step - accuracy: 0.9109 - loss: 0.2842
Epoch 2/10
```

```
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 90s 48ms/step - accuracy: 0.9860 - loss: 0.0433
Epoch 3/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 91s 48ms/step - accuracy: 0.9919 - loss: 0.0262
Epoch 4/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 146s 51ms/step - accuracy: 0.9943 - loss: 0.0178
Epoch 5/10
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 0s 48ms/step - accuracy: 0.9957 - loss: 0.0134
Reached 98% accuracy so cancelling training!
1875/1875 ━━━━━━━━━━━━━━━━━━━━ 91s 48ms/step - accuracy: 0.9957 - loss: 0.0134
```

Start coding or generate with AI.