C1W3_Assignment

June 28, 2023

1 Week 3: Improve MNIST with Convolutions

In the videos you looked at how you would improve Fashion MNIST using Convolutions. For this exercise see if you can improve MNIST to 99.5% accuracy or more by adding only a single convolutional layer and a single MaxPooling 2D layer to the model from the assignment of the previous week.

You should stop training once the accuracy goes above this amount. It should happen in less than 10 epochs, so it's ok to hard code the number of epochs for training, but your training must end once it hits the above metric. If it doesn't, then you'll need to redesign your callback.

When 99.5% accuracy has been hit, you should print out the string "Reached 99.5% accuracy so cancelling training!"

```
[3]: # 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 C1W3_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.*

```
[4]: # grader-required-cell

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

1.1 Load 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.

[5]:
```python
# 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")

# Get only training set
(training_images, training_labels), _ = tf.keras.datasets.mnist.
 ↪load_data(path=data_path)
```

1.2 Pre-processing the data

One important step when dealing with image data is to preprocess the data. During the preprocess step you can apply transformations to the dataset that will be fed into your convolutional neural network.

Here you will apply two transformations to the data: - Reshape the data so that it has an extra dimension. The reason for this is that commonly you will use 3-dimensional arrays (without counting the batch dimension) to represent image data. The third dimension represents the color using RGB values. This data might be in black and white format so the third dimension doesn't really add any additional information for the classification process but it is a good practice regardless.

- Normalize the pixel values so that these are values between 0 and 1. You can achieve this by dividing every value in the array by the maximum.

Remember that these tensors are of type `numpy.ndarray` so you can use functions like reshape or divide to complete the `reshape_and_normalize` function below:

[6]:
```python
# grader-required-cell

# GRADED FUNCTION: reshape_and_normalize

def reshape_and_normalize(images):

    ### START CODE HERE

    # Reshape the images to add an extra dimension
    images = images.reshape(60000,28,28,1)

    # Normalize pixel values
```

```
        images = images/255.0
        ### END CODE HERE

        return images
```

Test your function with the next cell:

```
[7]:  # grader-required-cell

      # 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(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}")
```

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)

Expected Output:

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)

1.3   Defining your callback

Now complete the callback that will ensure that training will stop after an accuracy of 99.5% is reached.

Define your callback in such a way that it checks for the metric `accuracy` (`acc` can normally be used as well but the grader expects this metric to be called `accuracy` so to avoid getting grading errors define it using the full word).

```
[14]:  # grader-required-cell

       # GRADED CLASS: myCallback
       ### START CODE HERE

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

```python
    def on_epoch_end(self,epoch,logs={}):
# Define the method that checks the accuracy at the end of each epoch
        if(logs.get('accuracy')>=0.995):
            print('\nReached 99.5% accuracy so cancelling training!')
            self.model.stop_training = True


    pass



### END CODE HERE
```

1.4 Convolutional Model

Finally, complete the `convolutional_model` function below. This function should return your convolutional neural network.

Your model should achieve an accuracy of 99.5% or more before 10 epochs to pass this assignment.

Hints: - You can try any architecture for the network but try to keep in mind you don't need a complex one. For instance, only one convolutional layer is needed. - In case you need extra help you can check out an architecture that works pretty well at the end of this notebook. - To avoid timeout issues with the autograder, please limit the number of units in your convolutional and dense layers. An exception will be raised if your model is too large.

```python
[9]: # grader-required-cell

# GRADED FUNCTION: convolutional_model
def convolutional_model():
    ### START CODE HERE

    # Define the model
    model = tf.keras.models.Sequential([
        tf.keras.layers.Conv2D(32 , (3,3), activation = 'relu' , input_shape =⌴
    ↪(28,28,1)),
        tf.keras.layers.MaxPooling2D(2,2),
        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(64 , activation = tf.nn.relu),
        tf.keras.layers.Dense(10 , activation = tf.nn.softmax)
    ])
    ### END CODE HERE

    # Compile the model
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
```

```
                    metrics=['accuracy'])

    return model
```

[15]:
```python
# grader-required-cell

# Save your untrained model
model = convolutional_model()

# Get number of weights
model_params = model.count_params()

# Unit test to limit the size of the model
assert model_params < 1000000, (
    f'Your model has {model_params:,} params. For successful grading, please
 ↪keep it '
    f'under 1,000,000 by reducing the number of units in your Conv2D and/or
 ↪Dense layers.'
)

# Instantiate the callback class
callbacks = myCallback()

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

```
Epoch 1/10
1875/1875 [==============================] - 31s 16ms/step - loss: 0.1621 -
accuracy: 0.9529
Epoch 2/10
1875/1875 [==============================] - 30s 16ms/step - loss: 0.0562 -
accuracy: 0.9834
Epoch 3/10
1875/1875 [==============================] - 30s 16ms/step - loss: 0.0386 -
accuracy: 0.9879
Epoch 4/10
1875/1875 [==============================] - 30s 16ms/step - loss: 0.0273 -
accuracy: 0.9914
Epoch 5/10
1875/1875 [==============================] - 30s 16ms/step - loss: 0.0193 -
accuracy: 0.9938
Epoch 6/10
1875/1875 [==============================] - ETA: 0s - loss: 0.0143 - accuracy:
0.9951
Reached 99.5% accuracy so cancelling training!
1875/1875 [==============================] - 30s 16ms/step - loss: 0.0143 -
```

```
accuracy: 0.9951
```

If you see the message that you defined in your callback printed out after less than 10 epochs it means your callback worked as expected. You can also double check by running the following cell:

```
[16]:  # grader-required-cell

       print(f"Your model was trained for {len(history.epoch)} epochs")
```

```
Your model was trained for 6 epochs
```

If your callback didn't stop training, one cause might be that you compiled your model using a metric other than `accuracy` (such as `acc`). Make sure you set the metric to `accuracy`. You can check by running the following cell:

```
[17]:  if not "accuracy" in history.model.metrics_names:
           print("Use 'accuracy' as metric when compiling your model.")
       else:
           print("The metric was correctly defined.")
```

```
The metric was correctly defined.
```

1.5 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 =␣
       ↪"CiAgICOgQSBDb252MkQgbGF5ZXIgd2l0aCAzMiBmaWx0ZXJzLCBhIGtlcm5lbF9zaXplIG9mIDN4MywgUmVMVSBhY3
      encoded_answer = encoded_answer.encode('ascii')
      answer = base64.b64decode(encoded_answer)
      answer = answer.decode('ascii')

      print(answer)
```

Congratulations on finishing this week's assignment!

You have successfully implemented a CNN to assist you in the image classification task. Nice job!

Keep it up!

Please click here if you want to experiment with any of the non-graded code.

Important Note: Please only do this when you've already passed the assignment to avoid problems with the autograder.

On the notebook's menu, click "View" > "Cell Toolbar" > "Edit Metadata"

Hit the "Edit Metadata" button next to the code cell which you want to lock/unlock

Set the attribute value for "editable" to:

"true" if you want to unlock it

"false" if you want to lock it

```
    </li>
    <li> On the notebook's menu, click "View" > "Cell Toolbar" > "None" </li>
</ol>
<p> Here's a short demo of how to do the steps above:
    <br>
    <img src="https://drive.google.com/uc?export=view&id=14Xy_Mb17CZVgzVAgq7NCjMVBvSae3xO1" al
```