# C1W3_Assignment

March 8, 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!"

[1]:
```
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.

[2]:
```
# 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:

[3]:
```
# GRADED FUNCTION: reshape_and_normalize

def reshape_and_normalize(images):

    ### START CODE HERE

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

    # Normalize pixel values
    images = np.divide(images,255)

    ### END CODE HERE

    return images
```

Test your function with the next cell:

[4]:
```
# 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).

[5]:
```
# GRADED CLASS: myCallback
### START CODE HERE

# Remember to inherit from the correct class
class myCallback(tf.keras.callbacks.Callback):
    # Define the method that checks the accuracy at the end of each epoch
    def on_epoch_end(self, epoch, logs={}):
            if logs.get('accuracy') is not None and logs.get('accuracy') > 0.
 ↪995:
                print("\nReached 99% accuracy so cancelling training!")

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

### 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.

[6]:
```
# GRADED FUNCTION: convolutional_model
def convolutional_model():
    ### START CODE HERE

    # Define the model
    model = tf.keras.models.Sequential([
      tf.keras.layers.Conv2D(64, (3, 3), activation='relu', input_shape=(28,␣
 ↪28, 1)),
      tf.keras.layers.MaxPooling2D(2, 2),
      tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
      tf.keras.layers.MaxPooling2D(2, 2),
      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
```

[7]:
```
# 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 [==============================] - 83s 44ms/step - loss: 0.1235 -
accuracy: 0.9614
Epoch 2/10
1875/1875 [==============================] - 83s 44ms/step - loss: 0.0415 -
accuracy: 0.9873
Epoch 3/10
1875/1875 [==============================] - 81s 43ms/step - loss: 0.0267 -
accuracy: 0.9918
Epoch 4/10
1875/1875 [==============================] - 81s 43ms/step - loss: 0.0198 -
accuracy: 0.9933
Epoch 5/10
1875/1875 [==============================] - ETA: 0s - loss: 0.0146 - accuracy:
0.9952
Reached 99% accuracy so cancelling training!
1875/1875 [==============================] - 81s 43ms/step - loss: 0.0146 -
accuracy: 0.9952
```

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:

[8]:
```
print(f"Your model was trained for {len(history.epoch)} epochs")
```

```
Your model was trained for 5 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:

[9]:
```
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:

[10]:
```
# 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 =␣
↪"CiAgIC0gQSBDb252MkQgbGF5ZXIgd2l0aCAzMiBmaWx0ZXJzLCBhIGtlcm5lbF9zaXplIG9mIDN4MywgUmVMVSBhY3
encoded_answer = encoded_answer.encode('ascii')
answer = base64.b64decode(encoded_answer)
answer = answer.decode('ascii')

print(answer)
```

```
   - A Conv2D layer with 32 filters, a kernel_size of 3x3, ReLU activation
function and an input shape that matches that of every image in the training set
   - A MaxPooling2D layer with a pool_size of 2x2
   - A Flatten layer with no arguments
   - A Dense layer with 128 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 CNN to assist you in the image classification task. Nice job!

**Keep it up!**

[ ]: