Week1_MNIST_CNN-Implementation

January 18, 2020

1 MNIST - Convolutional Neural Networks

Here a state of the art Convolutional Neural Networks is shown to perform decently on the MNIST dataset. Keras module makes this implementation very simple. See how the model can achieve high performance.

```
In [4]: # TensorFlow and tf.keras
        import tensorflow as tf
        from tensorflow import keras

        # Helper libraries
        import numpy as np
        import matplotlib.pyplot as plt
        import random

        print(tf.__version__)
```

```
2.1.0
```

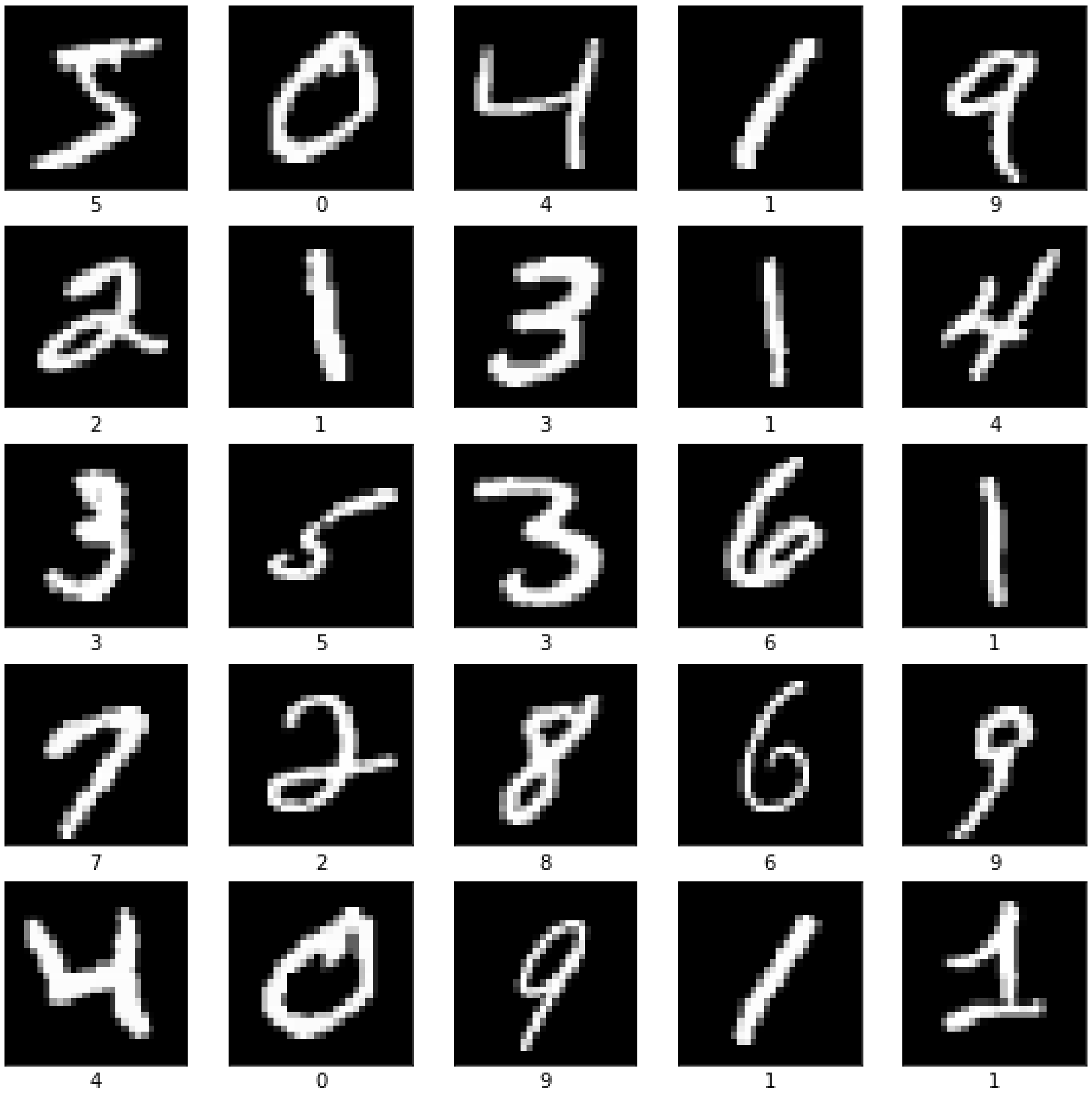
```
In [5]: # Keras provides a handy API to download the MNIST dataset, and split them into
        # "train" dataset and "test" dataset.
        mnist = keras.datasets.mnist
        (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
In [6]: # Normalize the input image so that each pixel value is between 0 to 1.
        train_images = train_images / 255.0
        test_images = test_images / 255.0
        print('Pixels are normalized')
```

```
Pixels are normalized
```

```
In [7]: # Show the first 25 images in the training dataset.
        plt.figure(figsize=(10,10))
        for i in range(25):
          plt.subplot(5,5,i+1)
          plt.xticks([])
```

```
  plt.yticks([])
  plt.grid(False)
  plt.imshow(train_images[i], cmap=plt.cm.gray)
  plt.xlabel(train_labels[i])
plt.show()
```



```
In [8]: # Define the model architecture
        model = keras.Sequential([
          keras.layers.InputLayer(input_shape=(28, 28)),
          keras.layers.Reshape(target_shape=(28, 28, 1)),
          keras.layers.Conv2D(filters=12, kernel_size=(3, 3), activation=tf.nn.relu),
          keras.layers.MaxPooling2D(pool_size=(2, 2)),
          keras.layers.Flatten(),
          keras.layers.Dense(10, activation=tf.nn.softmax)
        ])

        # Define how to train the model
        model.compile(optimizer='adam',
                      loss='sparse_categorical_crossentropy',
                      metrics=['accuracy'])

        # Train the digit classification model
        model.fit(train_images, train_labels, epochs=5)
```

```
Train on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 8s 139us/sample - loss: 0.3124 - accuracy: 0.910
Epoch 2/5
60000/60000 [==============================] - 8s 135us/sample - loss: 0.1317 - accuracy: 0.963
Epoch 3/5
60000/60000 [==============================] - 8s 134us/sample - loss: 0.0887 - accuracy: 0.975
Epoch 4/5
60000/60000 [==============================] - 8s 137us/sample - loss: 0.0715 - accuracy: 0.979
Epoch 5/5
60000/60000 [==============================] - 8s 133us/sample - loss: 0.0603 - accuracy: 0.982
```

```
Out[8]: <tensorflow.python.keras.callbacks.History at 0x16821f400>
```

```
In [9]: model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
reshape (Reshape)            (None, 28, 28, 1)         0
_________________________________________________________________
conv2d (Conv2D)              (None, 26, 26, 12)        120
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 12)        0
_________________________________________________________________
flatten (Flatten)            (None, 2028)              0
_________________________________________________________________
dense (Dense)                (None, 10)                20290
=================================================================
Total params: 20,410
Trainable params: 20,410
Non-trainable params: 0
_________________________________________________________________
```

```
In [10]: # Evaluate the model using all images in the test dataset.
         test_loss, test_acc = model.evaluate(test_images, test_labels)
         print('Test accuracy:', test_acc)
```

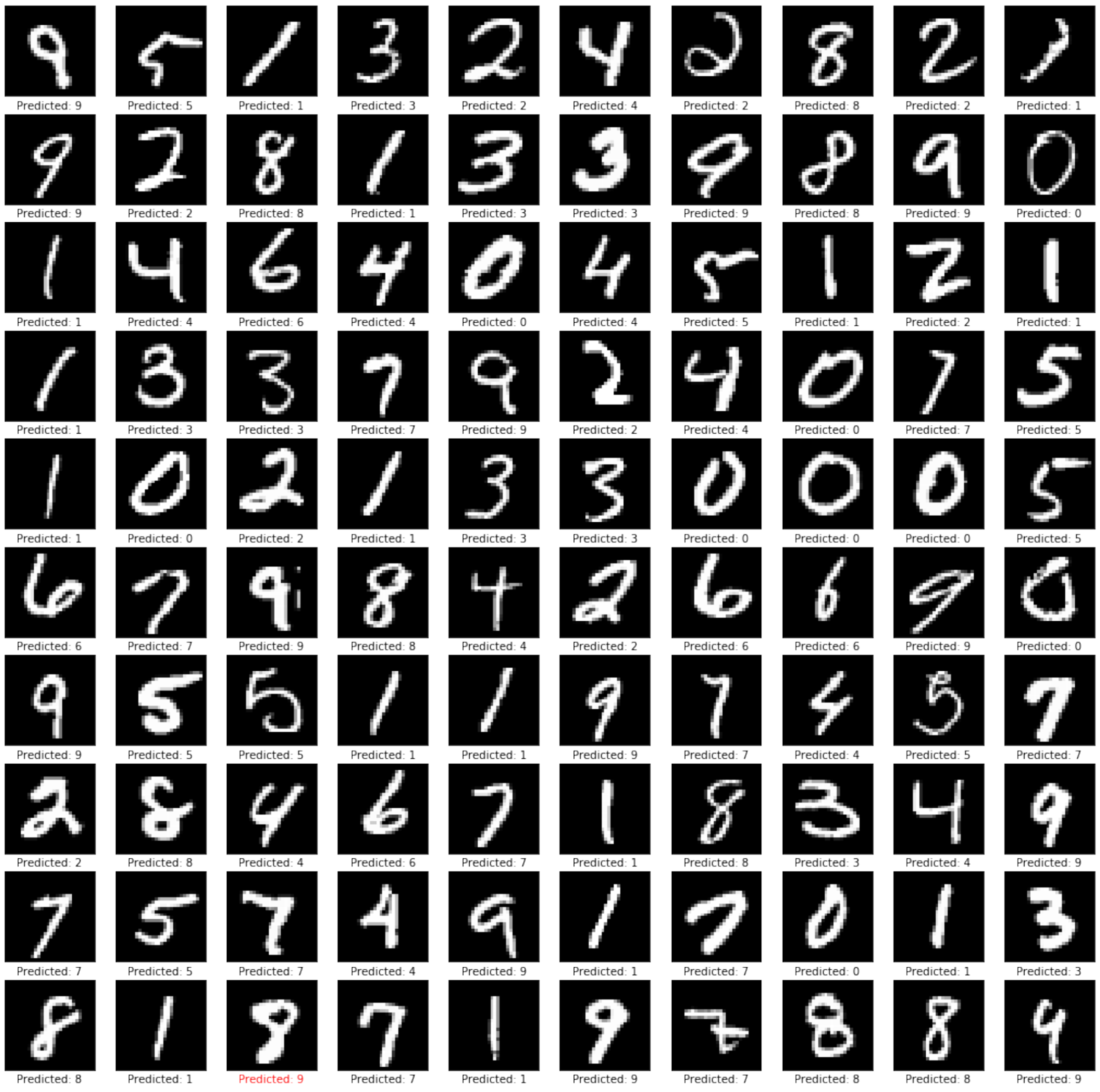
```
10000/10000 [==============================] - 1s 57us/sample - loss: 0.0614 - accuracy: 0.980
Test accuracy: 0.9807
```

```
In [11]: # A helper function that returns 'red'/'black' depending on if its two input
         # parameter matches or not.
         def get_label_color(val1, val2):
           if val1 == val2:
             return 'black'
           else:
             return 'red'

         # Predict the labels of digit images in our test dataset.
         predictions = model.predict(test_images)

         # As the model output 10 float representing the probability of the input image
         # being a digit from 0 to 9, we need to find the largest probability value
         # to find out which digit the model predicts to be most likely in the image.
         prediction_digits = np.argmax(predictions, axis=1)

         # Then plot 100 random test images and their predicted labels.
         # If a prediction result is different from the label provided label in "test"
         # dataset, we will highlight it in red color.
         plt.figure(figsize=(18, 18))
         for i in range(100):
           ax = plt.subplot(10, 10, i+1)
           plt.xticks([])
           plt.yticks([])
           plt.grid(False)
           image_index = random.randint(0, len(prediction_digits))
           plt.imshow(test_images[image_index], cmap=plt.cm.gray)
           ax.xaxis.label.set_color(get_label_color(prediction_digits[image_index],\
                                                    test_labels[image_index]))
           plt.xlabel('Predicted: %d' % prediction_digits[image_index])
         plt.show()
```


Predicted: 9
Predicted: 5
Predicted: 1
Predicted: 3
Predicted: 2
Predicted: 4
Predicted: 2
Predicted: 8
Predicted: 2
Predicted: 1
Predicted: 9
Predicted: 2
Predicted: 8
Predicted: 1
Predicted: 3
Predicted: 3
Predicted: 9
Predicted: 8
Predicted: 9
Predicted: 0
Predicted: 1
Predicted: 4
Predicted: 6
Predicted: 4
Predicted: 0
Predicted: 4
Predicted: 5
Predicted: 1
Predicted: 2
Predicted: 1
Predicted: 1
Predicted: 3
Predicted: 3
Predicted: 7
Predicted: 9
Predicted: 2
Predicted: 4
Predicted: 0
Predicted: 7
Predicted: 5
Predicted: 1
Predicted: 0
Predicted: 2
Predicted: 1
Predicted: 3
Predicted: 3
Predicted: 0
Predicted: 0
Predicted: 0
Predicted: 5
Predicted: 6
Predicted: 7
Predicted: 9
Predicted: 8
Predicted: 4
Predicted: 2
Predicted: 6
Predicted: 6
Predicted: 9
Predicted: 0
Predicted: 9
Predicted: 5
Predicted: 5
Predicted: 1
Predicted: 1
Predicted: 9
Predicted: 7
Predicted: 4
Predicted: 5
Predicted: 7
Predicted: 2
Predicted: 8
Predicted: 4
Predicted: 6
Predicted: 7
Predicted: 1
Predicted: 8
Predicted: 3
Predicted: 4
Predicted: 9
Predicted: 7
Predicted: 5
Predicted: 7
Predicted: 4
Predicted: 9
Predicted: 1
Predicted: 7
Predicted: 0
Predicted: 1
Predicted: 3
Predicted: 8
Predicted: 1
Predicted: 9
Predicted: 7
Predicted: 1
Predicted: 9
Predicted: 7
Predicted: 8
Predicted: 8
Predicted: 9