20162191_result

June 9, 2021

1 Assignment_5

1.0.1 Open Source SW project(Deep Learning)

- Run the provided code in Jupyter Notebook('./TensorFlow_mnist_example.ipynb') for three different convolutional neural networks (CNNs) for the popular classification dataset, MNIST, on your machine.

1.0.2 Import library

[1]:
```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers, models
import numpy as np
import matplotlib.pyplot as plt
```

1.0.3 Load the MNIST dataset

- Split data into (train_images, train_labels) and (test_images, test_labels).
- train_images: (60000, 28, 28)
- train_labels: (60000,)
- test_images : (10000, 28, 28)
- test_labels : (10000,)
- Above matrix shows that, there are total **60000 train images and labels** for training model. (**6000 images for single class**)
- Also, **10000 test images** and labels for testing.
- Shape of each **data(feature)** is **28x28** (pixel)
- The **range of label: 0 ... 9** (10 labels)

```
print(max(train_labels))    # 9
print(min(train_labels))    # 0
print(max(test_labels))     # 9
print(min(test_labels))     # 0
```

[2]:
```
mnist = keras.datasets.mnist
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

1.0.4 Plot first train image

- value is close to 0: dark
- value is close to 255: white

[3]:
```
def plot_train_images(title, iter_num):
    for i in range(iter_num):
        plt.subplot(1, iter_num, i+1)
        plt.xticks([])
        plt.yticks([])
        plt.imshow(train_images[i])

    plt.title(title)
    plt.tight_layout()
    plt.show()
    print('\nTrain labels match with Train label sequentially\n', train_labels[:
 ↪iter_num])

title = 'First 10 Train images in MNIST dataset'
plot_train_images(title, 10)
```


```
Train labels match with Train label sequentially
 [5 0 4 1 9 2 1 3 1 4]
```

1.0.5 Change data shape (60000 x 28 x 28) -> (60000 x 28 x 28 x 1)

- **Tip**: reshape 1- , .

[4]:
```
train_images = tf.reshape(train_images, [-1, 28, 28, 1])    # (60000, 28, 28, 1)
test_images = tf.reshape(test_images, [-1, 28, 28, 1])      # (10000, 28, 28, 1)
```

1.0.6 Select one Convolution model below (3 example models.)

:: Models ::

	Layers	Convolution layer
Model1	3	1
Model2	5	2
Model3	7	4

- Easiest way to build a model is to use function “sequential”

[5]:
```
def select_model(model_number):
    if model_number == 1:
        model = keras.models.Sequential([
                    keras.layers.Conv2D(32, (3,3), activation = 'relu',␣
↪input_shape = (28, 28,1)),  # layer 1
                    keras.layers.MaxPool2D((2,2)),                      ␣
↪                   # layer 2
                    keras.layers.Flatten(),
                    keras.layers.Dense(10, activation = 'softmax')])    ␣
↪                   # layer 3

    if model_number == 2:
        model = keras.models.Sequential([
                    keras.layers.Conv2D(32, (3,3), activation = 'relu',␣
↪input_shape=(28,28,1)),     # layer 1
                    keras.layers.MaxPool2D((2,2)),                      ␣
↪                   # layer 2
                    keras.layers.Conv2D(64, (3,3), activation = 'relu'),␣
↪                   # layer 3
                    keras.layers.MaxPool2D((2,2)),                      ␣
↪                   # layer 4
                    keras.layers.Flatten(),
                    keras.layers.Dense(10, activation = 'softmax')])    ␣
↪                   # layer 5

    if model_number == 3:
        model = keras.models.Sequential([
                    keras.layers.Conv2D(32, (3,3), activation = 'relu',␣
↪input_shape = (28, 28,1)),  # layer 1
                    keras.layers.MaxPool2D((2,2)),                      ␣
↪                   # layer 2
                    keras.layers.Conv2D(64, (3,3), activation = 'relu'),␣
↪                   # layer 3
                    keras.layers.Conv2D(64, (3,3), activation = 'relu'),␣
↪                   # layer 4
                    keras.layers.MaxPool2D((2,2)),                      ␣
↪                   # layer 5
```

```
                    keras.layers.Conv2D(128, (3,3), activation = 'relu'),
↪                     # layer 6
                    keras.layers.Flatten(),
                    keras.layers.Dense(10, activation = 'softmax')])
↪                     # layer 7

    return model
```

[29]:
```
model = select_model(3)
```

[30]:
```
#model.summary()
```

sample output of `model.summary()`

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
flatten (Flatten)            (None, 5408)              0
_________________________________________________________________
dense (Dense)                (None, 10)                54090
=================================================================
Total params: 54,410
Trainable params: 54,410
Non-trainable params: 0
_________________________________________________________________



Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1600)              0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                16010
```

```
=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
_________________________________________________________________


Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 9, 9, 64)          36928
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 2, 2, 128)         73856
_________________________________________________________________
flatten_2 (Flatten)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130
=================================================================
Total params: 134,730
Trainable params: 134,730
Non-trainable params: 0
_________________________________________________________________
```

1.0.7 Components in training step

Optimizer Loss function accuracy metrics

[31]:
```
model.compile(
    optimizer = 'adam',
    loss = 'sparse_categorical_crossentropy',
    metrics = ['accuracy']
)
```

1.0.8 Training Step

- **Training for 5 epochs**

[32]:
```
model.fit(train_images, train_labels,  epochs = 5)
```

```
Epoch 1/5
1875/1875 [==============================] - 17s 9ms/step - loss: 0.1946 -
accuracy: 0.9552
Epoch 2/5
1875/1875 [==============================] - 17s 9ms/step - loss: 0.0557 -
accuracy: 0.9832
Epoch 3/5
1875/1875 [==============================] - 18s 9ms/step - loss: 0.0488 -
accuracy: 0.9854
Epoch 4/5
1875/1875 [==============================] - 18s 9ms/step - loss: 0.0404 -
accuracy: 0.9879
Epoch 5/5
1875/1875 [==============================] - 18s 9ms/step - loss: 0.0351 -
accuracy: 0.9894
```

[32]: `<tensorflow.python.keras.callbacks.History at 0x7f498e33a880>`

1.0.9 Test Step

- **Perform test with test data**

[33]:
```
test_loss, accuracy = model.evaluate(test_images, test_labels, verbose = 2)
print('\nTest loss : ', test_loss)
print('Test accuracy :', accuracy)
```

```
313/313 - 1s - loss: 0.0647 - accuracy: 0.9824

Test loss :  0.0646621361374855
Test accuracy : 0.9824000000953674
```

sample output of `test with each model`

```
313/313 - 0s - loss: 0.1372 - accuracy: 0.9677

Test loss :  0.13722403347492218
Test accuracy : 0.9677000045776367


313/313 - 1s - loss: 0.0604 - accuracy: 0.9848

Test loss :  0.060430023819208145
Test accuracy : 0.9847999811172485


313/313 - 1s - loss: 0.0647 - accuracy: 0.9824

Test loss :  0.0646621361374855
Test accuracy : 0.9824000000953674
```

1.0.10 Change test image's type to float32, before prediction

[34]:
```
test_images = tf.cast(test_images, tf.float32)
pred = model.predict(test_images)
Number = [0,1,2,3,4,5,6,7,8,9]
```

[35]:
```
print('Prediction : ', pred.shape)
print('Test labels : ', test_labels.shape)
```

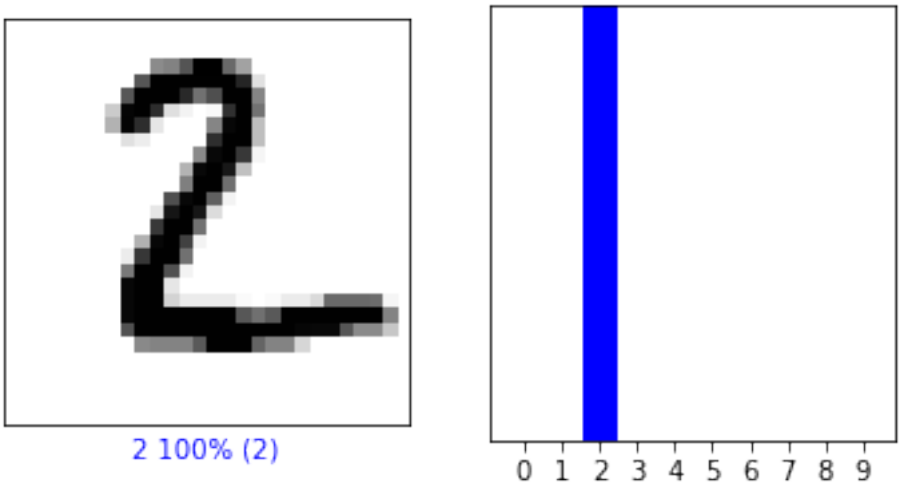
```
Prediction :  (10000, 10)
Test labels :  (10000,)
```

1.0.11 Functions for plot images, and probability

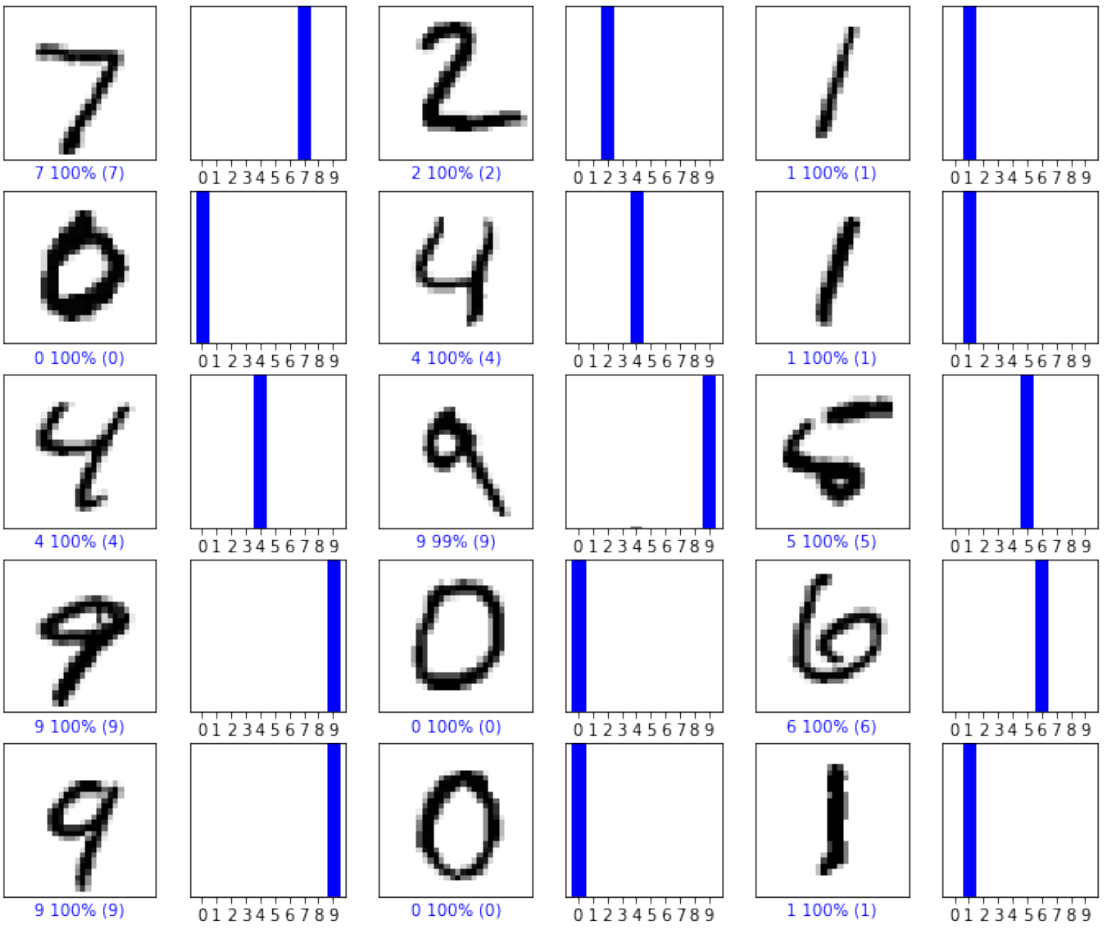
[36]:
```
def plot_image(i, predictions_array, true_label, img):
  predictions_array, true_label, img = predictions_array[i], true_label[i],␣
 ↪img[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])

  plt.imshow(img, cmap=plt.cm.binary)

  predicted_label = np.argmax(predictions_array)
  if predicted_label == true_label:
    color = 'blue'
  else:
    color = 'red'

  plt.xlabel("{} {:2.0f}% ({})".format(Number[predicted_label],
                                100*np.max(predictions_array),
                                Number[true_label]),
                                color=color)

def plot_value_array(i, predictions_array, true_label):
  predictions_array, true_label = predictions_array[i], true_label[i]
  plt.grid(False)
  plt.xticks([])
  plt.yticks([])
  thisplot = plt.bar(range(10), predictions_array, color="#777777")
  plt.ylim([0, 1])
  predicted_label = np.argmax(predictions_array)
  plt.xticks(Number)

  thisplot[predicted_label].set_color('red')
  thisplot[true_label].set_color('blue')
```

[37]:
```
# Reload data
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

[38]:
```
i = 1
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, pred, test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, pred,  test_labels)
plt.show()
```



[39]:
```
num_rows = 5
num_cols = 3
num_images = num_rows*num_cols
plt.figure(figsize=(2*2*num_cols, 2*num_rows))
for i in range(num_images):
  plt.subplot(num_rows, 2*num_cols, 2*i+1)
  plot_image(i, pred, test_labels, test_images)
  plt.subplot(num_rows, 2*num_cols, 2*i+2)
  plot_value_array(i, pred, test_labels)
plt.show()
```



1.0.12 Plot images and probability that model predicted wrong

[40]:
```
def error_mnist(prediction_array, true_label):
    error_index = []

    for i in range(true_label.shape[0]):
        if np.argmax(prediction_array[i]) != true_label[i]:
            error_index.append(i)
    return error_index

# change num_cols, num_rows if you want to see more result.
def plot_error(index, prediction_array, true_label):
    num_cols = 5
    num_rows = 5
    plt.figure(figsize=(2*2*num_cols, 2*num_rows))

    assert len(index) < num_cols * num_rows
    for i in range(len(index)):
```

```
        plt.subplot(num_rows, 2*num_cols, 2*i+1)
        idx = index[i]
        plt.imshow(test_images[idx])
        plt.subplot(num_rows, 2*num_cols, 2*i+2)
        plt.bar(range(10), prediction_array[idx])
        plt.xticks(Number)
```

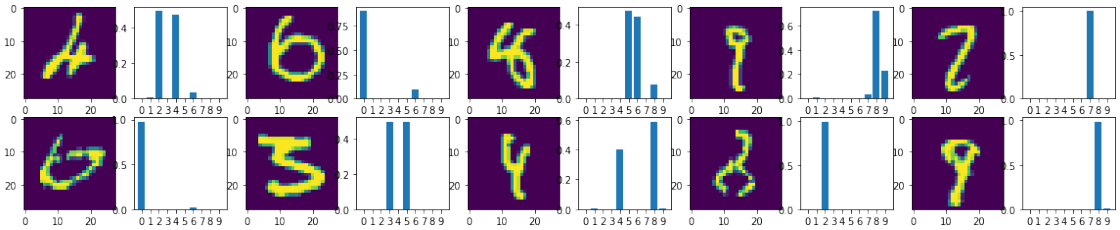
1.0.13 Find index of wrong prediction

- Plot first 10 wrong predicted images and probability

[41]:
```
index = error_mnist(pred, test_labels)
index_slice = index[:10]
print(index[:10])
```

```
[247, 259, 290, 320, 321, 445, 449, 497, 582, 593]
```

[42]:
```
plot_error(index_slice, pred, test_labels)
```



1.0.14 End of assignment_05