hy-23 / TensorFlow_Tutorials Private

Code Issues Pull requests Actions Projects Security Insights Settings

main

TensorFlow_Tutorials / TensorFlow_2.x / Chapter4_LeNet.ipynb

hy23 Folder restructuring ... History

0 contributors

10.2 MB

Open in Colab

In [1]:
```
import tensorflow as tf
from tensorflow.keras import datasets, layers, models, optimizers
```

In [2]:
```
# Load the TensorBoard notebook extension
# https://www.tensorflow.org/tensorboard/get_started

%load_ext tensorboard

import datetime
```

In [3]:
```
# Clear any logs from previous runs
!rm -rf /content/logs
```

In [4]:
```
# network and training
EPOCHS                  = 10
BATCH_SIZE              = 128
VERBOSE                 = 0
OPTIMIZER               = tf.keras.optimizers.Adam()
VALIDATION_SPLIT        = 0.95

# input image dimensions
IMG_ROWS, IMG_COLS      = 28, 28
INPUT_SHAPE             = (IMG_ROWS, IMG_COLS, 1)

# number of outputs = number of digits
NB_CLASSES              = 10
```

In [5]:
```
def build(input_shape, classes):
  model = models.Sequential()
  model.add(layers.Conv2D(filters=20, kernel_size=(5,5), padding='same',
                          activation='relu', input_shape=input_shape))
  model.add(layers.MaxPool2D(pool_size=(2,2), strides=(2,2)))
  model.add(layers.Conv2D(filters=50, kernel_size=(5,5), padding='same',
                          activation='relu'))
  model.add(layers.MaxPool2D(pool_size=(2,2), strides=(2,2)))
  model.add(layers.Flatten())
  model.add(layers.Dense(500, activation='relu'))
  # a softmax classifier
  model.add(layers.Dense(classes, activation='softmax'))
  return model
```

In [6]:
```
# Load data
(x_train, y_train), (x_test, y_test) = datasets.mnist.load_data()

print("Original shape of x_train is {}".format(x_train.shape))
```

```
Original shape of x_train is (60000, 28, 28)
```

In [7]:
```
# reshape
x_train = x_train.reshape((60000, 28, 28, 1))
x_test  = x_test.reshape((10000, 28, 28, 1))

# normalize
x_train, x_test = x_train / 255.0, x_test / 255.0

# cast
x_train = x_train.astype('float32')
x_test  = x_test.astype('float32')
```

In [8]:
```
print("Original shape of y_train is {}".format(y_train.shape))
```

```
Original shape of y_train is (60000,)
```

In [9]:
```
y_train = tf.keras.utils.to_categorical(y_train, NB_CLASSES)
y_test  = tf.keras.utils.to_categorical(y_test, NB_CLASSES)

print("Shape of y_train is {}".format(y_train.shape))
```

```
Shape of y_train is (60000, 10)
```

In [10]:

```python
in_shape = x_train.shape[1:]

# get the model
model = build(in_shape, NB_CLASSES)
```

In [11]:

```python
model.compile(optimizer=OPTIMIZER, loss='categorical_crossentropy',
              metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 20)        520

 max_pooling2d (MaxPooling2D  (None, 14, 14, 20)       0
 )

 conv2d_1 (Conv2D)           (None, 14, 14, 50)        25050

 max_pooling2d_1 (MaxPooling  (None, 7, 7, 50)         0
 2D)

 flatten (Flatten)           (None, 2450)              0

 dense (Dense)               (None, 500)               1225500

 dense_1 (Dense)             (None, 10)                5010

=================================================================
Total params: 1,256,080
Trainable params: 1,256,080
Non-trainable params: 0
_________________________________________________________________
```

In [12]:

```python
# use TensorBoard
callbacks = [
             # write TensorBoard logs to './logs' directory
             tf.keras.callbacks.TensorBoard(log_dir='./logs')
]
```

In [13]:

```python
# train the model
history = model.fit(x=x_train, y=y_train, batch_size=BATCH_SIZE, epochs=EPOCHS,
                    verbose=VERBOSE, callbacks=callbacks,
                    validation_split=VALIDATION_SPLIT)
```

In [14]:

```python
score = model.evaluate(x_test, y_test, verbose=VERBOSE)
```

In [15]:

```python
print("test loss is {}".format(score[0]))
print("test accu is {}".format(score[1]))
```

```
test loss is 0.09741149842739105
test accu is 0.9700000286102295
```

TensorBoard

In [16]:

```python
# How to use TensorBoard?
# https://www.tensorflow.org/tensorboard/get_started
%tensorboard --logdir logs/train
```

```
Reusing TensorBoard on port 6007 (pid 713), started 4:00:51 ago. (Use '!kill 713' to kill it.)
```

In [17]:

```python
# How to visualize feature maps.

for layer in model.layers:
  # from model.summary(), we know the names of the 'conv' layers
  if 'conv2d' not in layer.name:
    continue
  # get filter weights
  filters, biases = layer.get_weights()
  print(layer.name, filters.shape)
```

```
conv2d (5, 5, 1, 20)
conv2d_1 (5, 5, 20, 50)
```

conv2d_1 (5, 5, 20, 50)

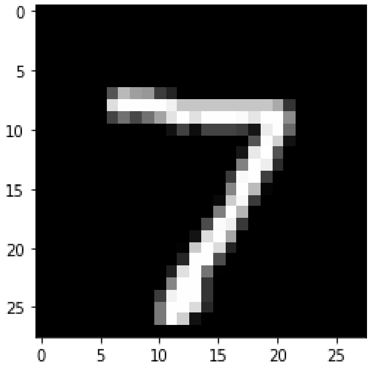
Visualize Feature Maps

In [18]:
```
# https://machinelearningmastery.com/how-to-visualize-filters-and-feature-maps-in-convolutional-neural-n

# The idea of visualizing a feature map for a specific input image would be to
# understand what features of the input are detected or preserved in the feature
# maps. The expectation would be that the feature maps close to the input detect
# small or fine-grained detail, whereas feature maps close to the output of the
# model capture more general features.
```

In [19]:
```
from matplotlib import pyplot as plt
```

In [20]:
```
plt.imshow(x_test[0].reshape(28,28), cmap='gray')
plt.show()
```



In [21]:
```
layer_outputs = [layer.output for layer in model.layers]
layer_outputs
```

Out[21]:
```
[<KerasTensor: shape=(None, 28, 28, 20) dtype=float32 (created by layer 'conv2d')>,
 <KerasTensor: shape=(None, 14, 14, 20) dtype=float32 (created by layer 'max_pooling2d')>,
 <KerasTensor: shape=(None, 14, 14, 50) dtype=float32 (created by layer 'conv2d_1')>,
 <KerasTensor: shape=(None, 7, 7, 50) dtype=float32 (created by layer 'max_pooling2d_1')>,
 <KerasTensor: shape=(None, 2450) dtype=float32 (created by layer 'flatten')>,
 <KerasTensor: shape=(None, 500) dtype=float32 (created by layer 'dense')>,
 <KerasTensor: shape=(None, 10) dtype=float32 (created by layer 'dense_1')>]
```

In [22]:
```
feature_map_model = tf.keras.models.Model(inputs=model.input, outputs=layer_outputs)
```

In [23]:
```
sample = x_test[0]
print("shape of the sample is {}".format(sample.shape))
sample = sample.reshape(1, 28, 28, 1)
print("shape of the sample is {}".format(sample.shape))
```

```
shape of the sample is (28, 28, 1)
shape of the sample is (1, 28, 28, 1)
```

In [24]:
```
fmap = feature_map_model.predict(sample)
```

In [25]:
```
print("type of the fmap is {}".format(type(fmap)))
print("shape of the fmap is {}".format(len(fmap)))
```

```
type of the fmap is <class 'list'>
shape of the fmap is 7
```

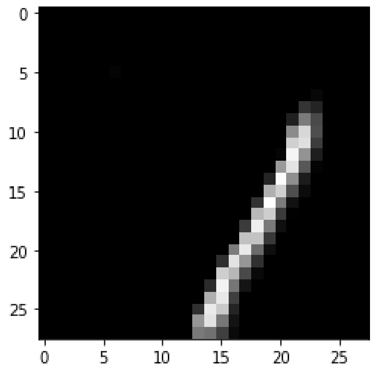
In [26]:
```
# each fmap corresponds to each layer.
# we know that layer0 is a convolution layer.

fmap_0 = fmap[0]
print("shape of the feature map 0 is {}".format(fmap_0.shape))
```

```
shape of the feature map 0 is (1, 28, 28, 20)
```

In [27]:
```
# example plot
plt.imshow(fmap_0[0, :, :, 1], cmap='gray')
```

Out[27]: <matplotlib.image.AxesImage at 0x7fda77646450>



In [28]:

```python
# let us plot all the 20 feature maps of covolution layer 0
no_of_plots_in_x = 5
no_of_plots_in_y = 4

index = 1
for x in range(no_of_plots_in_x):
  for y in range(no_of_plots_in_y):
    subplot = plt.subplot(no_of_plots_in_x, no_of_plots_in_y, index)
    subplot.set_xticks([])
    subplot.set_yticks([])
    # buf = sprintf("feature map # %d", index)
    # buf = f"depth #{index-1:d}"
    # subplot.set_title(buf)
    plt.imshow(fmap_0[0, :, :, index-1], cmap='gray')
    index += 1

print("Feature maps of convolution layer 0")
plt.show()
```

Feature maps of convolution layer 0

In [29]:

```python
fmap_2 = fmap[2]
print("shape of the feature map 0 is {}".format(fmap_2.shape))
```

shape of the feature map 0 is (1, 14, 14, 50)

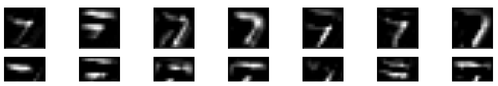
In [30]:

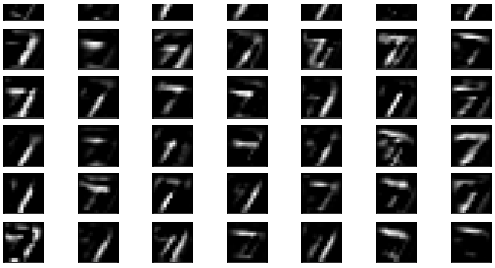
```python
# let us plot all the 50 feature maps of covolution layer 0
no_of_plots_in_x = 7
no_of_plots_in_y = 7

index = 1
for x in range(no_of_plots_in_x):
  for y in range(no_of_plots_in_y):
    subplot = plt.subplot(no_of_plots_in_x, no_of_plots_in_y, index)
    subplot.set_xticks([])
    subplot.set_yticks([])
    # buf = sprintf("feature map # %d", index)
    # buf = f"depth #{index-1:d}"
    # subplot.set_title(buf)
    plt.imshow(fmap_2[0, :, :, index-1], cmap='gray')
    index += 1

print("Feature maps of convolution layer 2")
plt.show()
```

Feature maps of convolution layer 2





In [30]: