mnist with cnn.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Comment Share

Files

..
sample_data
epoch.png

Disk 85.14 GB available

+ Code + Text

RAM Disk Editing

```
[1] from keras.datasets import mnist
    (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11490434/11490434 [==============================] - 0s 0us/step
```

```
train_images.shape
```

```
(60000, 28, 28)
```

```
[3] len(train_labels)
```

```
60000
```

```
[4] train_labels
```

```
array([5, 0, 4, ..., 5, 6, 8], dtype=uint8)
```

```
[5] from keras import models
    from keras import layers
    network = models.Sequential()
    network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
    network.add(layers.Dense(10, activation='softmax'))
```

```
[6] network.compile(optimizer='rmsprop',
      loss='categorical_crossentropy',
      metrics=['accuracy'])
```

```
[7] train_images = train_images.reshape((60000, 28 * 28))
    train_images = train_images.astype('float32') / 255
    test_images = test_images.reshape((10000, 28 * 28))
    test_images = test_images.astype('float32') / 255
```

```
[8] from keras.utils import to_categorical
    train_labels = to_categorical(train_labels)
    test_labels = to_categorical(test_labels)
```

```
[9] network.fit(train_images, train_labels, epochs=6, batch_size=128)
```

```
Epoch 1/6
469/469 [==============================] - 7s 14ms/step - loss: 0.2614 - accuracy: 0.9241
Epoch 2/6
469/469 [==============================] - 5s 11ms/step - loss: 0.1049 - accuracy: 0.9690
Epoch 3/6
469/469 [==============================] - 6s 13ms/step - loss: 0.0684 - accuracy: 0.9798
Epoch 4/6
469/469 [==============================] - 5s 11ms/step - loss: 0.0501 - accuracy: 0.9851
Epoch 5/6
469/469 [==============================] - 5s 10ms/step - loss: 0.0370 - accuracy: 0.9886
Epoch 6/6
469/469 [==============================] - 5s 10ms/step - loss: 0.0287 - accuracy: 0.9916
<keras.callbacks.History at 0x7f0e75fd5510>
```

```
[10] test_loss, test_acc = network.evaluate(test_images, test_labels)
     print('test_acc:', test_acc)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.0680 - accuracy: 0.9801
test_acc: 0.9800999760627747
```

98% accuracy is not the best, let's try with convnets (conv2D)

```
[11] model = models.Sequential()
     model.add(layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)))
     model.add(layers.MaxPooling2D((2, 2)))
     model.add(layers.Conv2D(64, (3, 3), activation='relu'))
     model.add(layers.MaxPooling2D((2, 2)))
     model.add(layers.Conv2D(64, (3, 3), activation='relu'))
```

```
[12] model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 3, 3, 64)          36928

=================================================================
Total params: 55,744
Trainable params: 55,744
Non-trainable params: 0
_________________________________________________________________
```

Now we add flatten 3D outputs to 1D, and then stack on top some dense layers

```
[13] model.add(layers.Flatten())
     model.add(layers.Dense(64, activation='relu'))
     model.add(layers.Dense(10, activation='softmax'))
```

```
[14] model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 26, 26, 32)        320

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 conv2d_1 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 conv2d_2 (Conv2D)           (None, 3, 3, 64)          36928

 flatten (Flatten)           (None, 576)               0

 dense_2 (Dense)             (None, 64)                36928

 dense_3 (Dense)             (None, 10)                650

=================================================================
Total params: 93,322
Trainable params: 93,322
Non-trainable params: 0
_________________________________________________________________
```

As it can be seen, the (3, 3, 64) outputs are flattened into vectors of shape (576,) before going through two Dense layers

```
[54] (train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

```
[55] train_images = train_images.reshape((60000, 28, 28, 1))
     train_images = train_images.astype('float32') / 255
     test_images = test_images.reshape((10000, 28, 28, 1))
     test_images = test_images.astype('float32') / 255
     train_labels = to_categorical(train_labels)
     test_labels = to_categorical(test_labels)
     model.compile(optimizer='rmsprop',
       loss='categorical_crossentropy',
       metrics=['accuracy'])
     model.fit(train_images, train_labels, epochs=6, batch_size=64, validation_data=(test_images, test_labels))
```

```
Epoch 1/6
938/938 [==============================] - 59s 63ms/step - loss: 0.0055 - accuracy: 0.9987 - val_loss: 0.0691 - val_accuracy: 0.9908
Epoch 2/6
938/938 [==============================] - 58s 61ms/step - loss: 0.0024 - accuracy: 0.9994 - val_loss: 0.0769 - val_accuracy: 0.9917
Epoch 3/6
938/938 [==============================] - 57s 61ms/step - loss: 0.0033 - accuracy: 0.9990 - val_loss: 0.0672 - val_accuracy: 0.9918
Epoch 4/6
938/938 [==============================] - 58s 62ms/step - loss: 0.0031 - accuracy: 0.9992 - val_loss: 0.0758 - val_accuracy: 0.9919
Epoch 5/6
938/938 [==============================] - 58s 62ms/step - loss: 0.0034 - accuracy: 0.9992 - val_loss: 0.0799 - val_accuracy: 0.9926
Epoch 6/6
938/938 [==============================] - 57s 61ms/step - loss: 0.0023 - accuracy: 0.9995 - val_loss: 0.0878 - val_accuracy: 0.9903
<keras.callbacks.History at 0x7f0e63891f10>
```

```
test_loss, test_acc = model.evaluate(test_images, test_labels)
test_acc
```

```
313/313 [==============================] - 4s 12ms/step - loss: 0.0878 - accuracy: 0.9903
0.9902999997138977
```