About Keras
Getting started
Developer guides
Keras API reference
Code examples
Computer Vision
Natural Language Processing
Structured Data
Timeseries
Audio Data
Generative Deep Learning
Reinforcement Learning
Graph Data
Quick Keras Recipes
Why choose Keras?
Community & governance
Contributing to Keras
KerasTuner

Simple MNIST convnet

Author: fchollet
Date created: 2015/06/19
Last modified: 2020/04/21
Description: A simple convnet that achieves ~99% test accuracy on MNIST.

View in Colab • GitHub source

Setup

```
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
```

Prepare the data

```
# Model / data parameters
num_classes = 10
input_shape = (28, 28, 1)

# the data, split between train and test sets
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Scale images to the [0, 1] range
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
# Make sure images have shape (28, 28, 1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
print("x_train shape:", x_train.shape)
print(x_train.shape[0], "train samples")
print(x_test.shape[0], "test samples")


# convert class vectors to binary class matrices
y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

```
x_train shape: (60000, 28, 28, 1)
60000 train samples
10000 test samples
```

Build the model

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
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 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1600)              0
_________________________________________________________________
dropout (Dropout)            (None, 1600)              0
_________________________________________________________________
dense (Dense)                (None, 10)                16010
=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
_________________________________________________________________
```

Train the model

```
batch_size = 128
epochs = 15

model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=["accuracy"])

model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, validation_split=0.1)
```

```
Epoch 1/15
422/422 [==============================] - 13s 29ms/step - loss: 0.7840 - accuracy: 0.7643 -
val_loss: 0.0780 - val_accuracy: 0.9780
Epoch 2/15
422/422 [==============================] - 13s 31ms/step - loss: 0.1199 - accuracy: 0.9639 -
val_loss: 0.0559 - val_accuracy: 0.9843
Epoch 3/15
422/422 [==============================] - 14s 33ms/step - loss: 0.0845 - accuracy: 0.9737 -
val_loss: 0.0469 - val_accuracy: 0.9877
Epoch 4/15
422/422 [==============================] - 14s 33ms/step - loss: 0.0762 - accuracy: 0.9756 -
val_loss: 0.0398 - val_accuracy: 0.9895
Epoch 5/15
422/422 [==============================] - 15s 35ms/step - loss: 0.0621 - accuracy: 0.9812 -
val_loss: 0.0378 - val_accuracy: 0.9890
Epoch 6/15
422/422 [==============================] - 17s 40ms/step - loss: 0.0547 - accuracy: 0.9825 -
val_loss: 0.0360 - val_accuracy: 0.9910
Epoch 7/15
422/422 [==============================] - 17s 41ms/step - loss: 0.0497 - accuracy: 0.9840 -
val_loss: 0.0311 - val_accuracy: 0.9920
Epoch 8/15
422/422 [==============================] - 16s 39ms/step - loss: 0.0443 - accuracy: 0.9862 -
val_loss: 0.0346 - val_accuracy: 0.9910
Epoch 9/15
422/422 [==============================] - 17s 39ms/step - loss: 0.0436 - accuracy: 0.9860 -
val_loss: 0.0325 - val_accuracy: 0.9915
Epoch 10/15
422/422 [==============================] - 16s 38ms/step - loss: 0.0407 - accuracy: 0.9865 -
val_loss: 0.0301 - val_accuracy: 0.9920
Epoch 11/15
422/422 [==============================] - 16s 37ms/step - loss: 0.0406 - accuracy: 0.9874 -
val_loss: 0.0303 - val_accuracy: 0.9920
Epoch 12/15
237/422 [===============>..............] - ETA: 7s - loss: 0.0398 - accuracy: 0.9877
```

Evaluate the trained model

```python
score = model.evaluate(x_test, y_test, verbose=0)
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

```
Test loss: 0.023950600996613503
Test accuracy: 0.9922000169754028
```