

TensorFlow 2 项目实战进阶



扫码试看/订阅

《TensorFlow 2 项目进阶实战》视频课程

快速上手篇：动手训练模型和部署服务

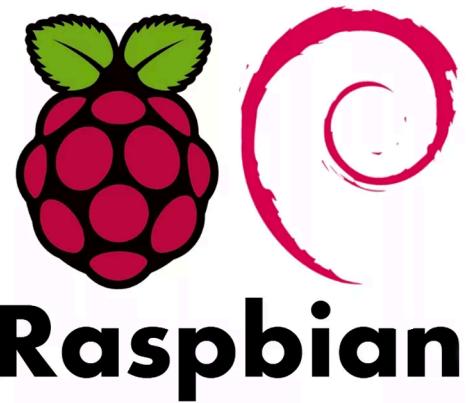
目录

- TensorFlow 2 开发环境搭建
- 使用 `tf.keras.datasets` 加载数据
- 使用 `tf.data.Dataset` 加载数据
- 使用 `tf.keras.Model` 管理模型
- Fashion MNIST 数据集介绍
- 使用 TensorFlow 2 训练分类网络

TensorFlow 2 开发环境搭建

TensorFlow 2 支持的操作系统

- Python 3.5–3.7
- Ubuntu 16.04 or later
- Windows 7 or later
- macOS 10.12.6 (Sierra) or later (no GPU support)
- Raspbian 9.0 or later



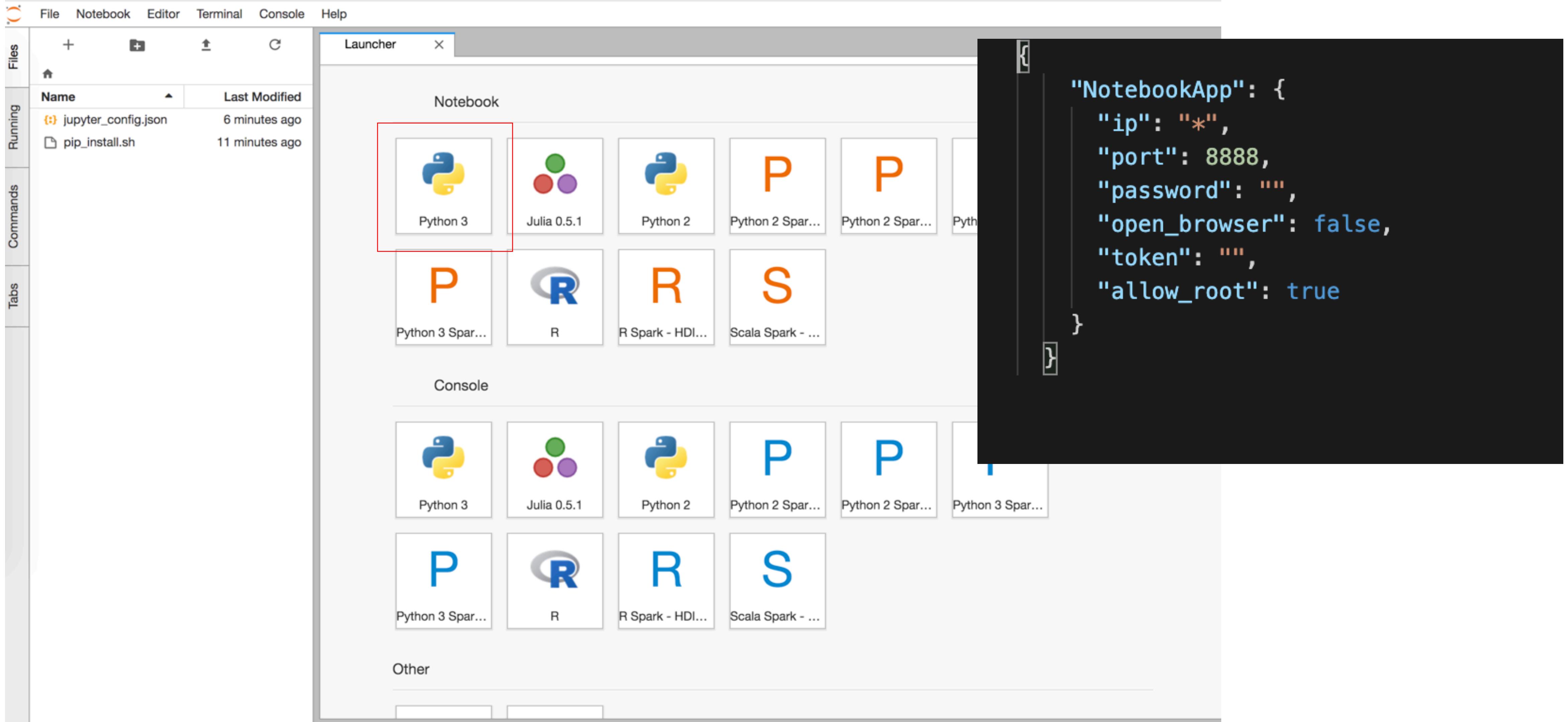
使用 pip3 安装 TensorFlow 2

```
# 更新 Pip  
pip install -i https://mirrors.aliyun.com/pypi/simple --upgrade pip  
  
# 安装 TensorFlow 2.2.0 RC4  
pip install -i https://mirrors.aliyun.com/pypi/simple tensorflow==2.2.0rc4  
  
# 安装 Jupyter Lab 作为交互式开发环境  
pip install -i https://mirrors.aliyun.com/pypi/simple jupyter lab
```

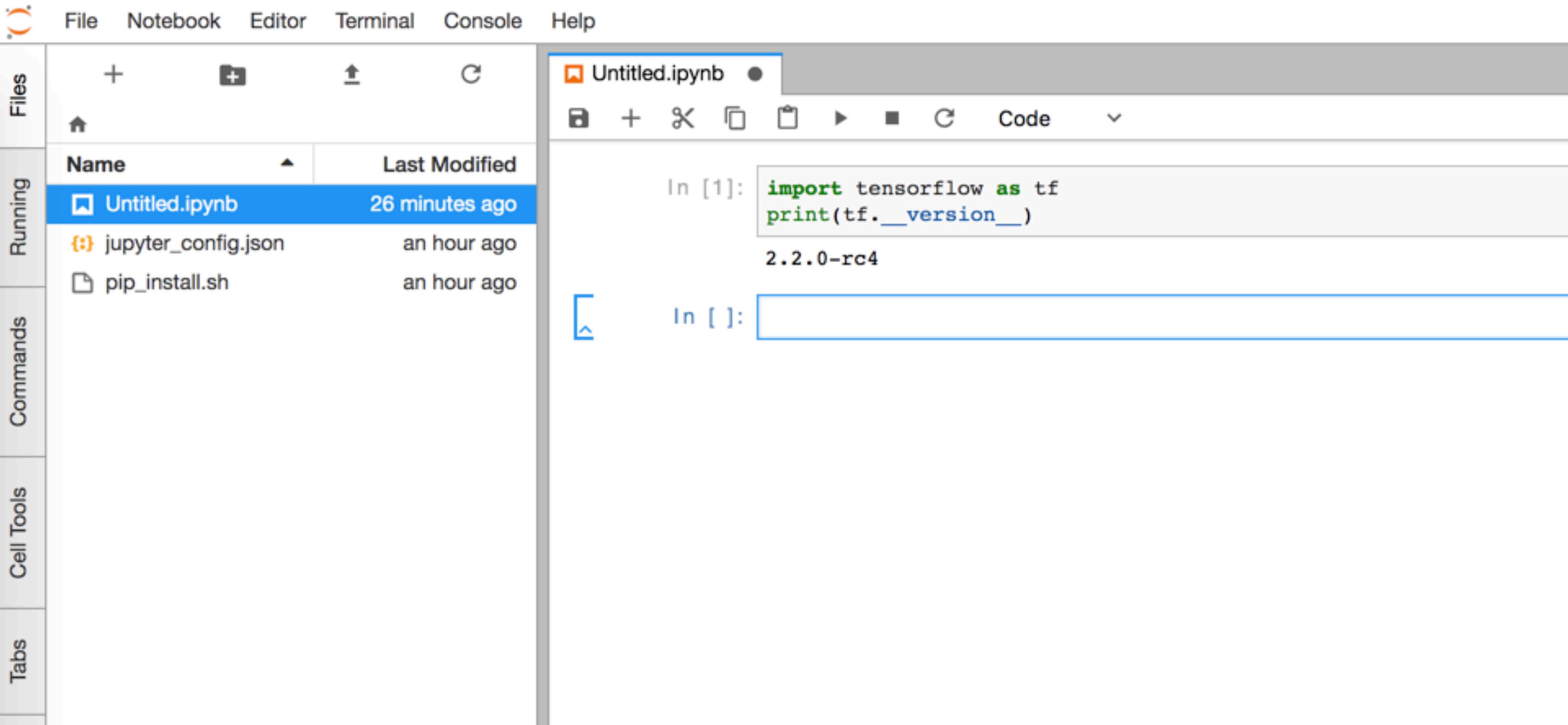
在 Jupyter Lab 中使用 TensorFlow 2

```
django@dev-tf2:~/advanced_practice/install$ jupyter lab --config jupyter_config.json
[W 05:08:55.656 LabApp] Config option `allow_root` not recognized by `LabApp` . Did you mean one of: `allow_origin, allow_origin_pat`?
[W 05:08:55.662 LabApp] Config option `allow_root` not recognized by `LabApp` . Did you mean one of: `allow_origin, allow_origin_pat`?
[W 05:08:55.788 LabApp] WARNING: The notebook server is listening on all IP addresses and not using encryption. This is not recommended.
[W 05:08:55.788 LabApp] WARNING: The notebook server is listening on all IP addresses and not using authentication. This is highly insecure and not recommended.
[W 05:08:55.793 LabApp] JupyterLab server extension not enabled, manually loading...
[I 05:08:55.793 LabApp] JupyterLab alpha preview extension loaded from /anaconda/envs/py35/lib/python3.5/site-packages/jupyterlab
JupyterLab v0.26.4
Known labextensions:
[I 05:08:55.794 LabApp] Running the core application with no additional extensions or settings
[E 05:08:55.795 LabApp] schemas_dir: /anaconda/envs/py35/lib/python3.5/site-packages/jupyterlab/schemas
[I 05:08:55.797 LabApp] Serving notebooks from local directory: /home/django/advanced_practice/install
[I 05:08:55.798 LabApp] 0 active kernels
[I 05:08:55.798 LabApp] The Jupyter Notebook is running at: http://[all ip addresses on your system]:8888/
[I 05:08:55.798 LabApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
```

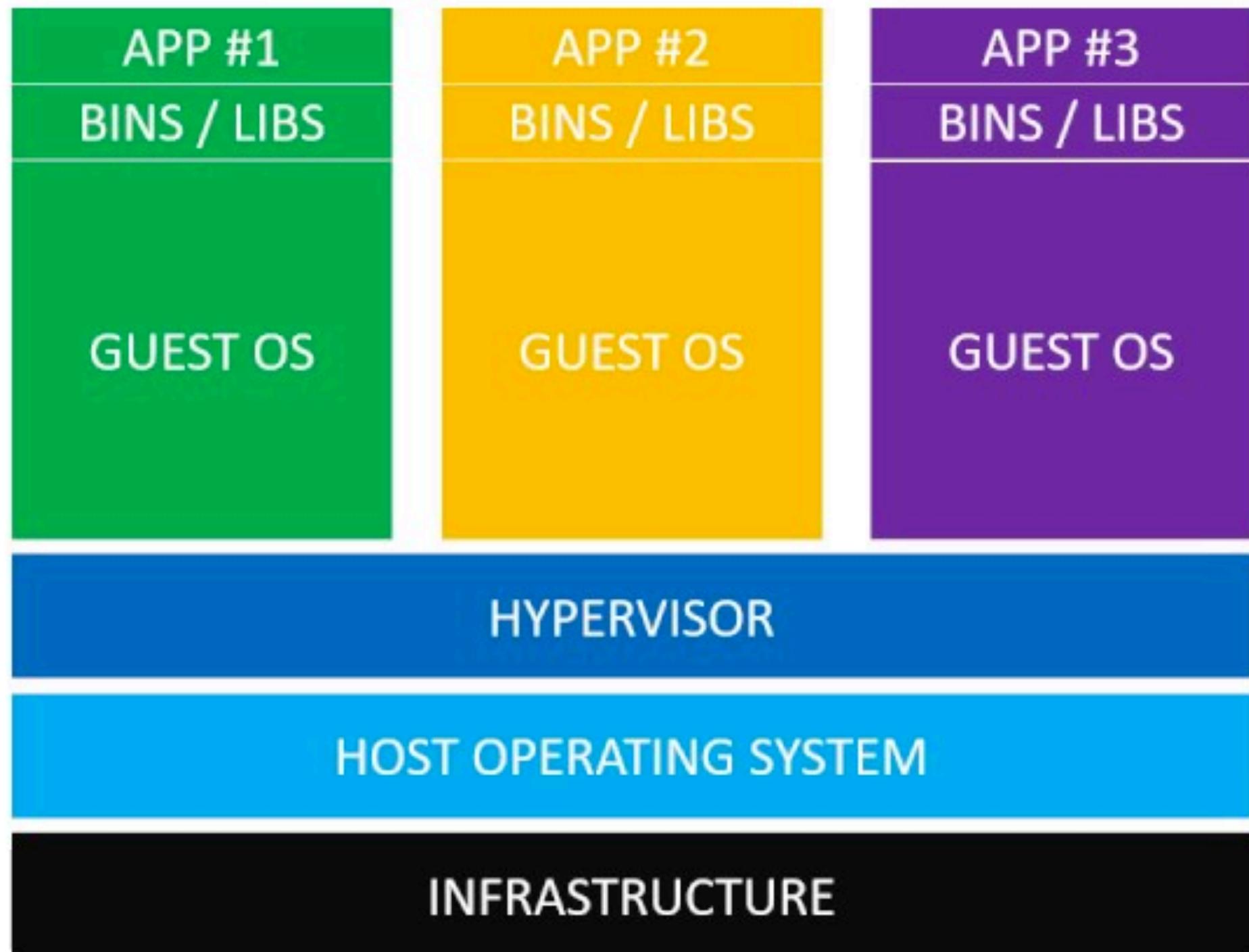
在Jupyter Lab 中使用 TensorFlow 2



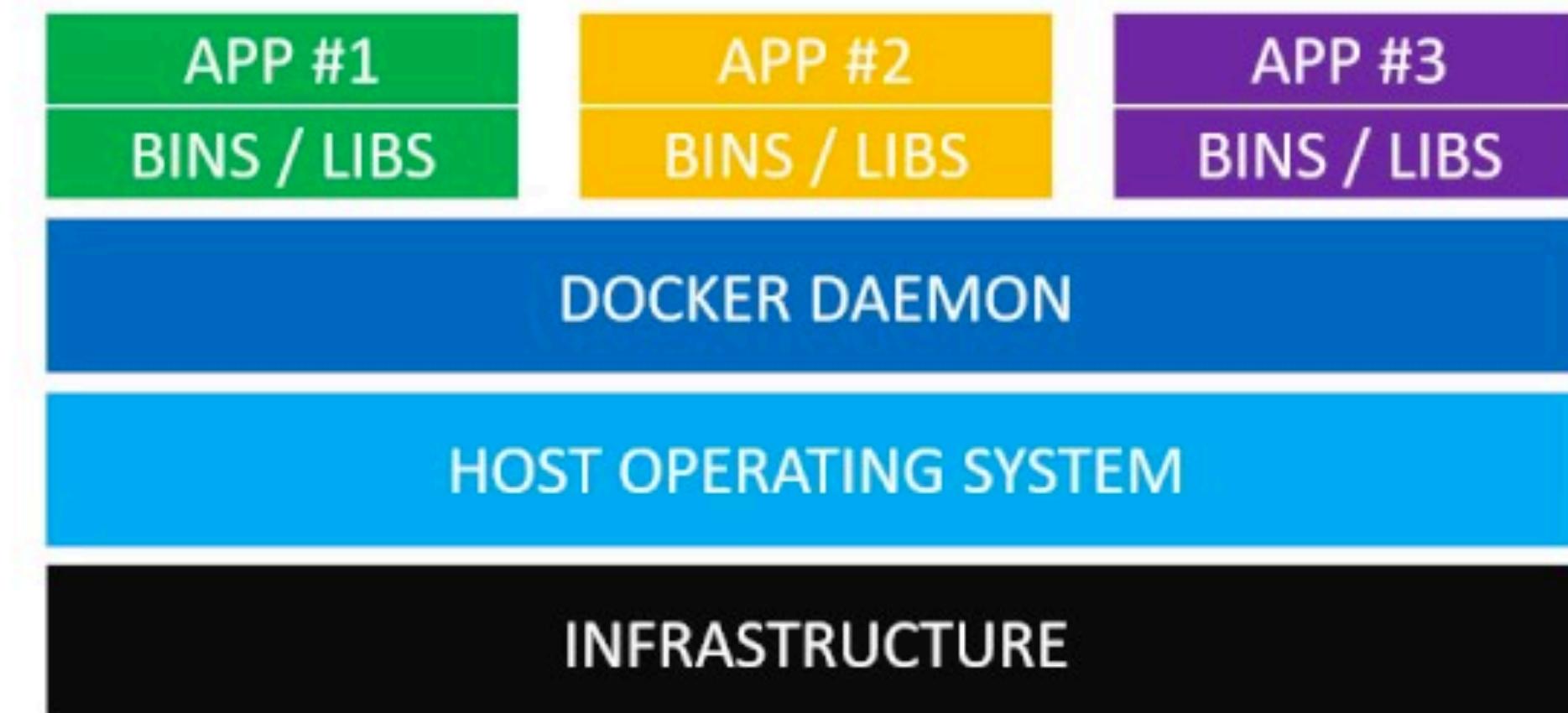
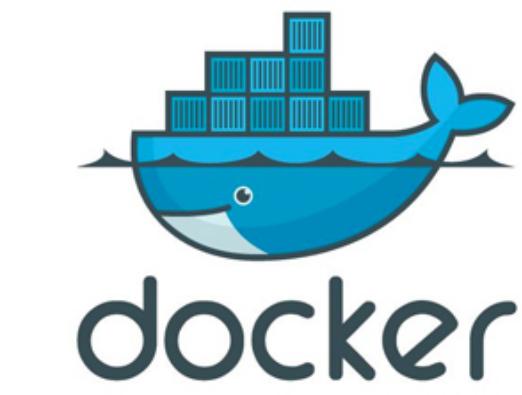
在Jupyter Lab 中使用 TensorFlow 2



Docker 容器与虚拟机



虚拟机



Docker 容器

在 Docker 中使用 TensorFlow 2

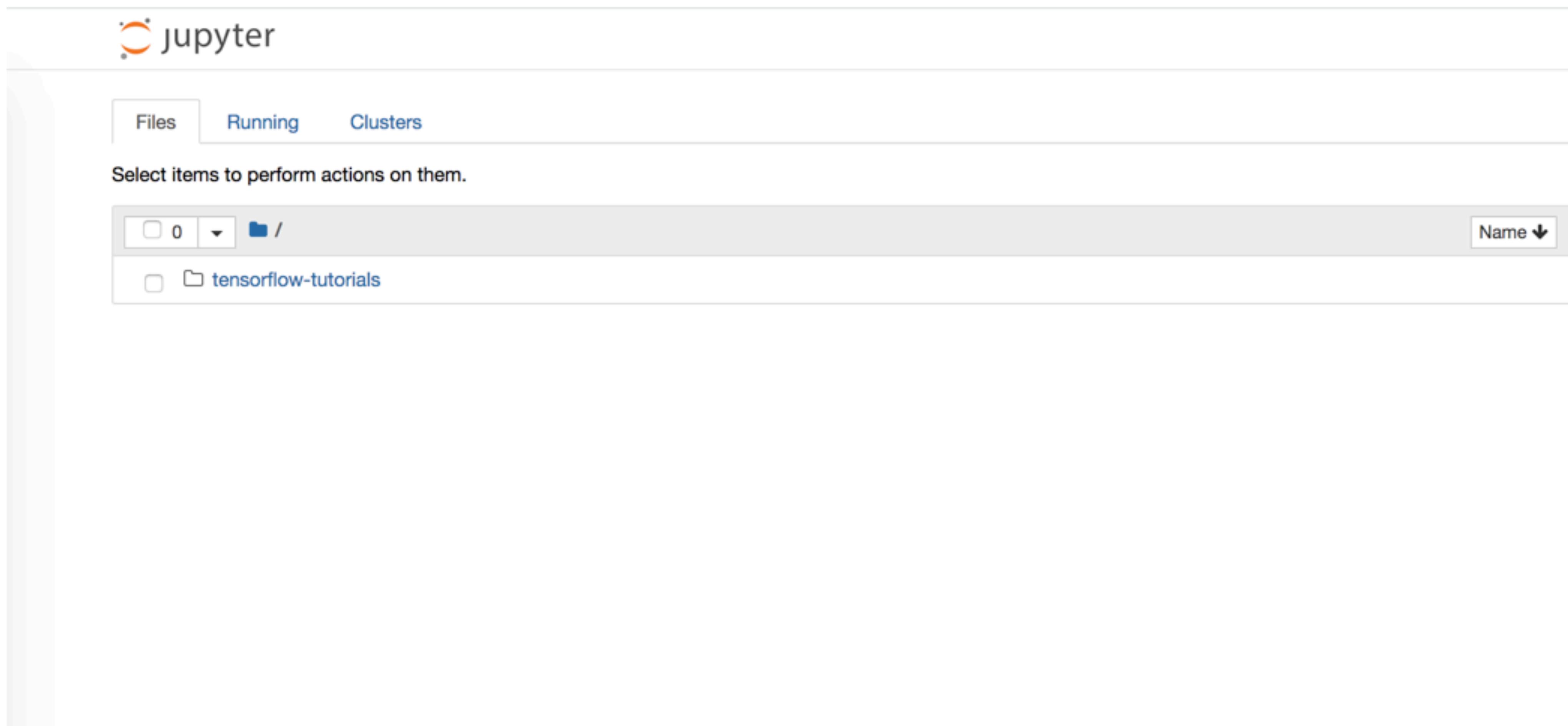
```
django@dev-tf2:~/advanced_practice/install $ sudo docker run -it -p 8888:8888 tensorflow/tensorflow:2.2.0rc4-jupyter
Unable to find image 'tensorflow/tensorflow:2.2.0rc4-jupyter' locally
2.2.0rc4-jupyter: Pulling from tensorflow/tensorflow
23884877105a: Pulling fs layer
bc38caa0f5b9: Pulling fs layer
2910811b6c42: Pulling fs layer
36505266dcc6: Waiting
49b62c208964: Waiting
0eed2e253368: Waiting
6e96dc8601eb: Waiting
2e7d0b5b1540: Waiting
f65c844bdd2c: Waiting
3c7423a0cd91: Waiting
282930c7512c: Waiting
3534c13b42db: Waiting
438a44c74bf6: Waiting
928f5c4a2608: Waiting
c541daaabd98: Waiting
441ce73ef16d: Waiting
845b44a04597: Waiting
df3ec2d4217e: Waiting
e6cad6aee343: Waiting
60d86d9f0b2d: Waiting
db44bb841228: Waiting
55f54a4d310e: Waiting
92d8c786052e: Waiting
bf1e2d268bc4: Waiting
0b5f6d50dead: Waiting
e6ca097c0aa9: Waiting
8d98b4a61ef7: Waiting
```

在 Docker 中使用 TensorFlow 2

```
Digest: sha256:b08518e183be034c99de228310f5f506af2e2ce369323ee5994447d29a537019
Status: Downloaded newer image for tensorflow/tensorflow:2.2.0rc4-jupyter
[I 05:20:07.286 NotebookApp] Writing notebook server cookie secret to /root/.local/share/jupyter/runtime/notebook_cookie_secret
jupyter_http_over_ws extension initialized. Listening on /http_over_websocket
[I 05:20:07.549 NotebookApp] Serving notebooks from local directory: /tf
[I 05:20:07.550 NotebookApp] The Jupyter Notebook is running at:
[I 05:20:07.550 NotebookApp] http://0005478527d6:8888/?token=51a44dbd0904acb48e8a65dd678c79f4b6885a066bc978c2
[I 05:20:07.550 NotebookApp] or http://127.0.0.1:8888/?token=51a44dbd0904acb48e8a65dd678c79f4b6885a066bc978c2
[I 05:20:07.550 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 05:20:07.554 NotebookApp]

To access the notebook, open this file in a browser:
  file:///root/.local/share/jupyter/runtime/nbserver-1-open.html
Or copy and paste one of these URLs:
  http://0005478527d6:8888/?token=51a44dbd0904acb48e8a65dd678c79f4b6885a066bc978c2
  or http://127.0.0.1:8888/?token=51a44dbd0904acb48e8a65dd678c79f4b6885a066bc978c2
[DW 05:20:09.483 NotebookApp] Forbidden
[DW 05:20:09.483 NotebookApp] 403 GET /api/contents/?content=1&1588569609412 (58.247.11.58) 1.36ms referer=http://139.219.130.157:8888/lab?
[DW 05:20:13.398 NotebookApp] Forbidden
[DW 05:20:13.398 NotebookApp] 403 GET /api/kernelspecs?1588569613362 (58.247.11.58) 1.39ms referer=http://139.219.130.157:8888/lab?
[DW 05:20:15.375 NotebookApp] Forbidden
[DW 05:20:15.376 NotebookApp] 403 GET /api/sessions?1588569615271 (58.247.11.58) 0.99ms referer=http://139.219.130.157:8888/lab?
[DW 05:20:15.383 NotebookApp] Forbidden
[DW 05:20:15.383 NotebookApp] 403 GET /api/terminals?1588569615272 (58.247.11.58) 0.75ms referer=http://139.219.130.157:8888/lab?
[DW 05:20:19.566 NotebookApp] Forbidden
[DW 05:20:19.567 NotebookApp] 403 GET /api/contents/?content=1&1588569619447 (58.247.11.58) 1.02ms referer=http://139.219.130.157:8888/lab?
[DW 05:20:25.341 NotebookApp] Forbidden
```

在 Docker 中使用 TensorFlow 2



Try it!

使用 `tf.keras.datasets` 加载数据

使用 tf.keras.datasets 预置数据集

```
tf.graph_util
tf.image
tf.io
tf.keras
  Overview
  Input
  Model
  Sequential
  activations
  applications
  backend
  callbacks
  constraints
  datasets
    Overview
    boston_housing
    cifar10
    cifar100
    fashion_mnist
    imdb
    mnist
    reuters
    estimator
    experimental
    initializers
```

Module: tf.keras.datasets



TensorFlow 1 version

Public API for tf.keras.datasets namespace.

Modules

[boston_housing](#) module: Boston housing price regression dataset.

[cifar10](#) module: CIFAR10 small images classification dataset.

[cifar100](#) module: CIFAR100 small images classification dataset.

[fashion_mnist](#) module: Fashion-MNIST dataset.

[imdb](#) module: IMDB sentiment classification dataset.

[mnist](#) module: MNIST handwritten digits dataset.

[reuters](#) module: Reuters topic classification dataset.

使用 tf.keras.datasets.mnist 数据集

The screenshot shows a Jupyter Notebook interface with the following details:

- File Menu:** File, Notebook, Editor, Terminal, Console, Help.
- Files Sidebar:** chapter-2, files, mnist_model, data.ipynb (selected), model.ipynb, mnist_model.h5.
- Code Cell 2:** Contains the code:

```
import tensorflow as tf  
mnist = tf.keras.datasets.mnist  
(x_train, y_train), (x_test, y_test) = mnist.load_data()  
x_train, x_test = x_train / 255.0, x_test / 255.0
```

 The output of this cell is: Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>. A blue bracket highlights this output.
- Code Cell 5:** Contains the code:

```
x_train.shape
```

 The output of this cell is: Out[5]: (60000, 28, 28).

使用 tf.keras.datasets.mnist 数据集

The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Notebook, Editor, Terminal, Console, Help.
- Left Sidebar:** Files, Running, Commands, Cell Tools, Tabs.
- Current File:** data.ipynb (selected in the sidebar).
- Code Cells:**
 - In [6]:

```
y_train = y_train.astype('float32')
y_test = y_test.astype('float32')

# Reserve 10,000 samples for validation
x_val = x_train[-10000:]
y_val = y_train[-10000:]
x_train = x_train[:-10000]
y_train = y_train[:-10000]
```
 - In [7]:

```
print(x_train.shape)
print(x_val.shape)

(50000, 28, 28)
(10000, 28, 28)
```
 - In [12]:

```
%matplotlib inline

import matplotlib.pyplot as plt

fig = plt.figure()
for i in range(15):
    plt.subplot(3,5,i+1) # 绘制前15个手写体数字, 以3行5列子图形式展示
    plt.tight_layout() # 自动适配子图尺寸
    plt.imshow(x_train[i], cmap='Greys') # 使用灰色显示像素灰度值
    plt.title("Label: {}".format(y_train[i])) # 设置标签为子图标题
    plt.xticks([]) # 删除x轴标记
    plt.yticks([]) # 删除y轴标记
```
- Output:** Displays 15 handwritten digit images in a 3x5 grid. Each image is labeled with its corresponding digit value:
 - Row 1: Label: 5.0 (digit 5), Label: 0.0 (digit 0), Label: 4.0 (digit 4), Label: 1.0 (digit 1), Label: 9.0 (digit 9)
 - Row 2: Label: 2.0 (digit 2), Label: 1.0 (digit 1), Label: 3.0 (digit 3), Label: 1.0 (digit 1), Label: 4.0 (digit 4)
 - Row 3: (empty)

Try it!

使用 tf.data.Dataset 加载数据

使用 `tf.data.Dataset.from_tensor_slices` 加载 List

The screenshot shows a Jupyter Notebook interface with a single code cell. The cell title is "List 列表数据". The code in the cell is:

```
In [13]: dataset = tf.data.Dataset.from_tensor_slices([1, 2, 3])
for element in dataset:
    print(element)
```

The output of the code is displayed below the cell, showing three tensors:

```
tf.Tensor(1, shape=(), dtype=int32)
tf.Tensor(2, shape=(), dtype=int32)
tf.Tensor(3, shape=(), dtype=int32)
```

使用 tf.data.Dataset.from_generator 加载 Generator

The screenshot shows a Jupyter Notebook interface with the title bar 'data.ipynb'. The notebook has three cells:

- Cell 43:** Contains Python code defining a generator function 'gen' that yields pairs of integers and their squares.
- Cell 44:** Contains Python code to create a TensorFlow dataset from the generator. The line 'dataset = tf.data.Dataset.from_generator(' is highlighted with a red box.
- Cell 45:** Contains Python code to list the first three elements of the dataset as NumPy iterators.

The output of Cell 45 is:

```
Out[45]: [(1, array([1])), (2, array([1, 1])), (3, array([1, 1, 1]))]
```

使用 tf.data.TextLineDataset 加载文本

The screenshot shows a Jupyter Notebook interface with a sidebar on the left and a main workspace on the right.

Sidebar:

- File: File, Notebook, Editor, Terminal, Console, Help
- Files: chapter-2
- Running: data.ipynb (selected), files, mnist_model, model.ipynb, mnist_model.h5
- Commands
- Cell Tools
- Tabs

Main Workspace (data.ipynb):

```
In [39]: parent_dir = "chapter-2/files"
FILE_NAMES = ['cowper.txt', 'derby.txt', 'butler.txt']

In [40]: def labeler(example, index):
    return example, tf.cast(index, tf.int64)

labeled_data_sets = []

for i, file_name in enumerate(FILE NAMES):
    lines_dataset = tf.data.TextLineDataset(os.path.join(parent_dir, file_name))
    labeled_dataset = lines_dataset.map(lambda ex: labeler(ex, i))
    labeled_data_sets.append(labeled_dataset)

In [41]: BUFFER_SIZE = 50000
BATCH_SIZE = 64
TAKE_SIZE = 5000

all_labeled_data = labeled_data_sets[0]
for labeled_dataset in labeled_data_sets[1:]:
    all_labeled_data = all_labeled_data.concatenate(labeled_dataset)

In [42]: for ex in all_labeled_data.take(5):
    print(ex)

(<tf.Tensor: shape=(), dtype=string, numpy=b'\xef\xbb\xbfAchilles sing, O Goddess! Peleus' son;'), <tf.Tensor: shape=(), dtype=int64, numpy=0>)
(<tf.Tensor: shape=(), dtype=string, numpy=b'His wrath pernicious, who ten thousand woes';, <tf.Tensor: shape=(), dtype=int64, numpy=0>)
(<tf.Tensor: shape=(), dtype=string, numpy=b"Caused to Achaia's host, sent many a soul", <tf.Tensor: shape=(), dtype=int64, numpy=0>)
(<tf.Tensor: shape=(), dtype=string, numpy=b'Illustrious into Ades premature,'), <tf.Tensor: shape=(), dtype=int64, numpy=0>)
(<tf.Tensor: shape=(), dtype=string, numpy=b'And Heroes gave (so stood the will of Jove)', <tf.Tensor: shape=(), dtype=int64, numpy=0>)

In [ ]:
```

Try it!

使用 tf.keras.Model 管理模型

历史上的 tf.keras.Model

- Class [tf.compat.v1.keras.Model](#)
- Class [tf.compat.v1.keras.models.Model](#)
- Class [tf.compat.v2.keras.Model](#)
- Class [tf.compat.v2.keras.models.Model](#)
- Class [tf.keras.models.Model](#)

使用 tf.keras.Model 构建模型

构建CNN模型网络（Sequential）

```
In [3]: model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10)
])
```

使用 tf.keras.Model 构建模型

构建CNN模型网络 (Functional API)

```
In [1]: import tensorflow as tf

inputs = tf.keras.Input(shape=(3,))
x = tf.keras.layers.Dense(4, activation=tf.nn.relu)(inputs)
outputs = tf.keras.layers.Dense(5, activation=tf.nn.softmax)(x)
model = tf.keras.Model(inputs=inputs, outputs=outputs)
```

使用 tf.keras.Model 训练模型

```
In [4]: model.compile(optimizer='adam',
                     loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                     metrics=['accuracy'])

In [5]: model.fit(x_train, y_train, epochs=5)

Epoch 1/5

1/1875 [........................] - ETA: 0s - accuracy: 0.0000e+00 - loss: 2.4046
32/1875 [........................] - ETA: 2s - accuracy: 0.5342 - loss: 1.5983
63/1875 [>.....] - ETA: 2s - accuracy: 0.6607 - loss: 1.1938
95/1875 [>.....] - ETA: 2s - accuracy: 0.7135 - loss: 0.9955
126/1875 [=>.....] - ETA: 2s - accuracy: 0.7490 - loss: 0.8747
157/1875 [=>.....] - ETA: 2s - accuracy: 0.7721 - loss: 0.7990
187/1875 [=>.....] - ETA: 2s - accuracy: 0.7884 - loss: 0.7420
219/1875 [==>.....] - ETA: 2s - accuracy: 0.8024 - loss: 0.6890
251/1875 [==>.....] - ETA: 2s - accuracy: 0.8141 - loss: 0.6492
284/1875 [==>.....] - ETA: 2s - accuracy: 0.8220 - loss: 0.6184
316/1875 [====>.....] - ETA: 2s - accuracy: 0.8288 - loss: 0.5947
348/1875 [====>.....] - ETA: 2s - accuracy: 0.8351 - loss: 0.5712
380/1875 [=====>.....] - ETA: 2s - accuracy: 0.8406 - loss: 0.5505
411/1875 [=====>.....] - ETA: 2s - accuracy: 0.8451 - loss: 0.5323
444/1875 [=====>.....] - ETA: 2s - accuracy: 0.8483 - loss: 0.5185
475/1875 [=====>.....] - ETA: 2s - accuracy: 0.8517 - loss: 0.5061
506/1875 [=====>.....] - ETA: 2s - accuracy: 0.8557 - loss: 0.4933
537/1875 [=====>.....] - ETA: 2s - accuracy: 0.8593 - loss: 0.4819
569/1875 [=====>.....] - ETA: 2s - accuracy: 0.8628 - loss: 0.4706
600/1875 [=====>.....] - ETA: 2s - accuracy: 0.8659 - loss: 0.4595
632/1875 [=====>.....] - ETA: 1s - accuracy: 0.8687 - loss: 0.4487
663/1875 [=====>.....] - ETA: 1s - accuracy: 0.8711 - loss: 0.4407
695/1875 [=====>.....] - ETA: 1s - accuracy: 0.8733 - loss: 0.4330
727/1875 [=====>.....] - ETA: 1s - accuracy: 0.8753 - loss: 0.4251
758/1875 [=====>.....] - ETA: 1s - accuracy: 0.8776 - loss: 0.4182
790/1875 [=====>.....] - ETA: 1s - accuracy: 0.8799 - loss: 0.4109
817/1875 [=====>.....] - ETA: 1s - accuracy: 0.8814 - loss: 0.4059
851/1875 [=====>.....] - ETA: 1s - accuracy: 0.8832 - loss: 0.3986
885/1875 [=====>.....] - ETA: 1s - accuracy: 0.8843 - loss: 0.3941
```

保存和加载 h5 模型

保存为 h5 模型

```
In [11]: model.save("mnist_model.h5")
```

加载 h5 模型

```
In [15]: h5_model = tf.keras.models.load_model("mnist_model.h5")
```

```
In [16]: h5_model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
<hr/>		
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 128)	100480
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1290
<hr/>		
Total params: 101,770		
Trainable params: 101,770		
Non-trainable params: 0		

保存和加载 SavedModel 模型

保存为 SavedModel 模型

```
In [17]: model.save("mnist_model")
```

```
INFO:tensorflow:Assets written to: mnist_model/assets
```

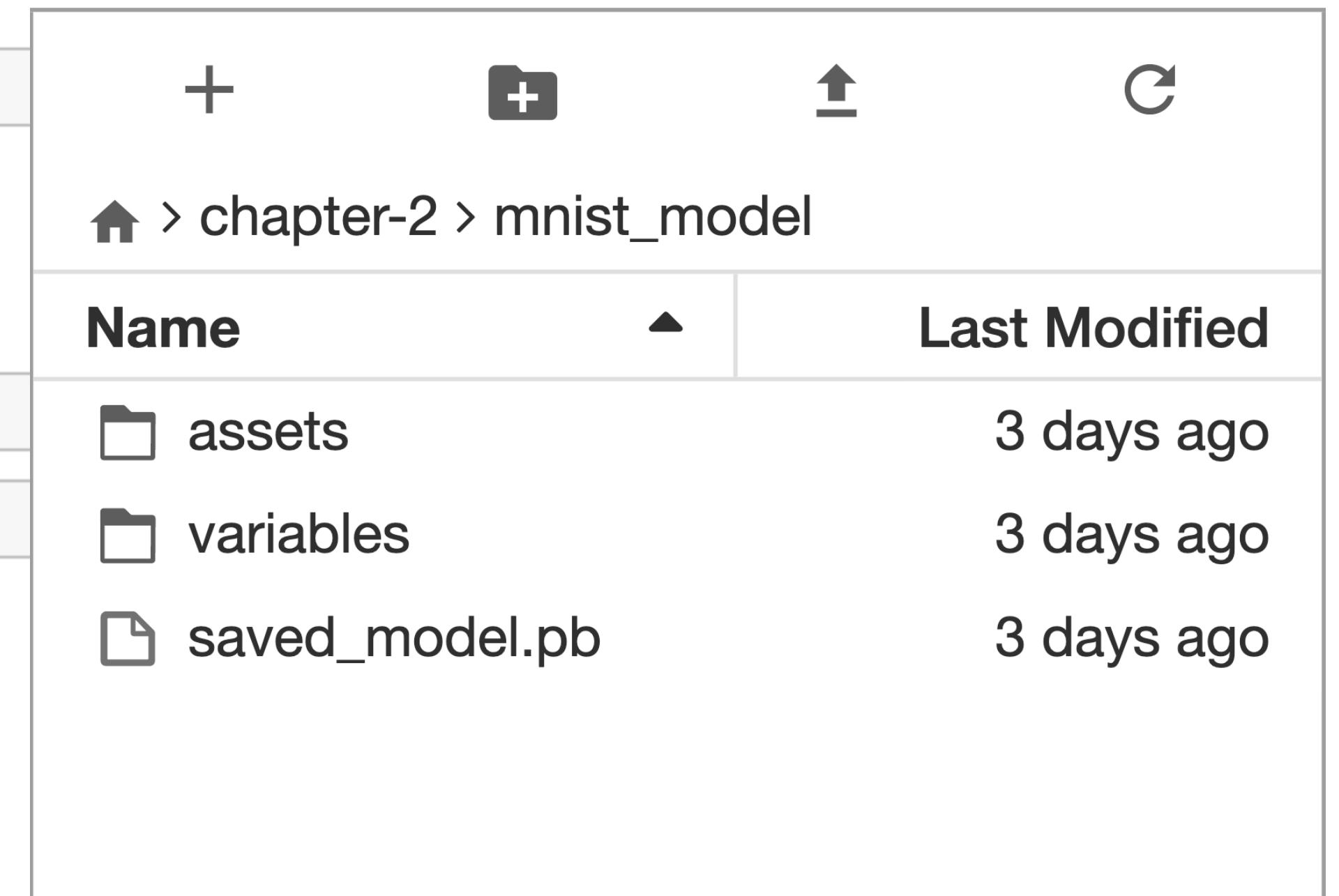
加载 SavedModel 模型

```
In [18]: sm_model = tf.keras.models.load_model("mnist_model.h5")
```

```
In [19]: sm_model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #
<hr/>		
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 128)	100480
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 10)	1290
<hr/>		
Total params: 101,770		
Trainable params: 101,770		
Non-trainable params: 0		



Fashion MNIST 数据集介绍

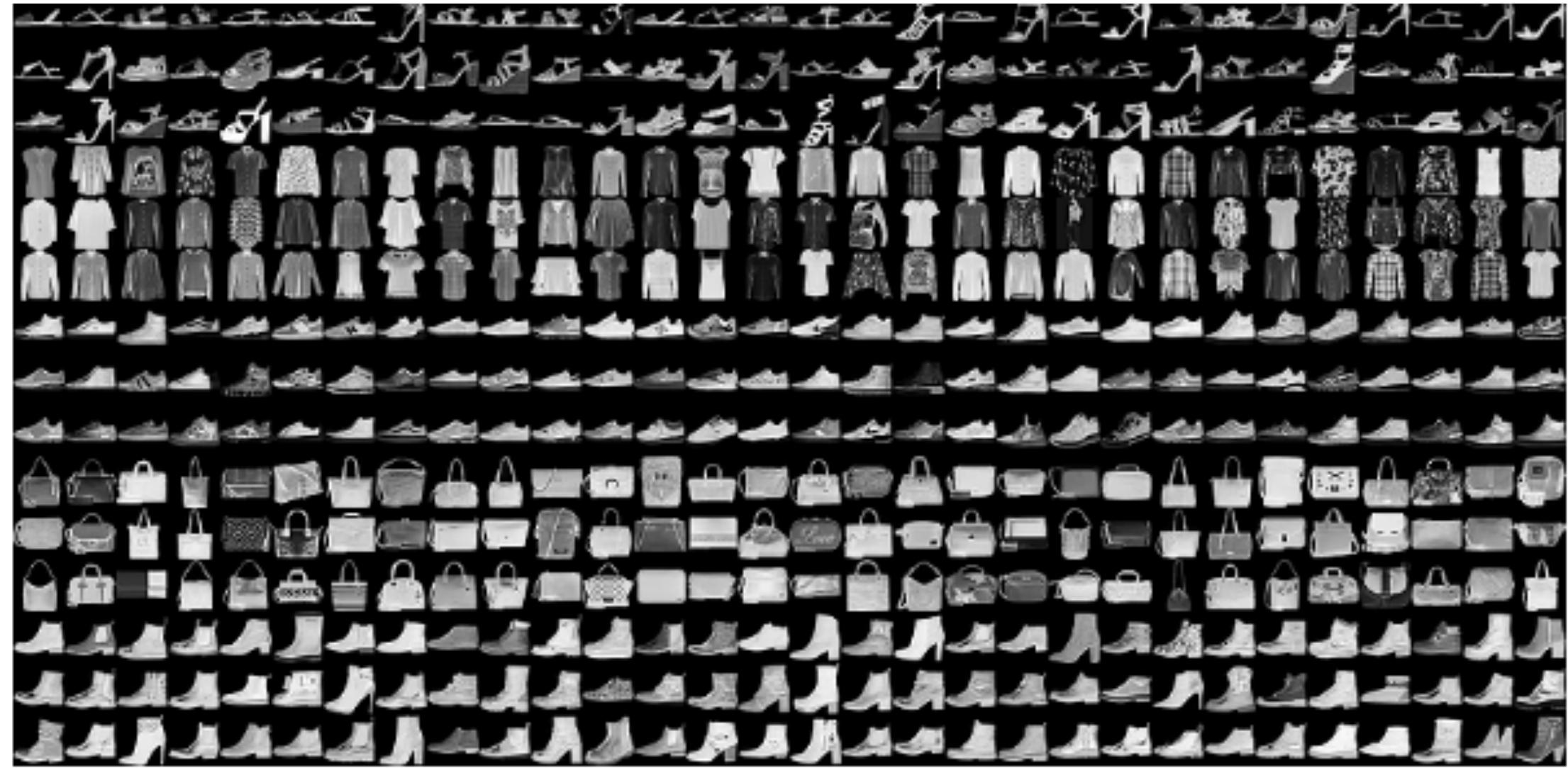
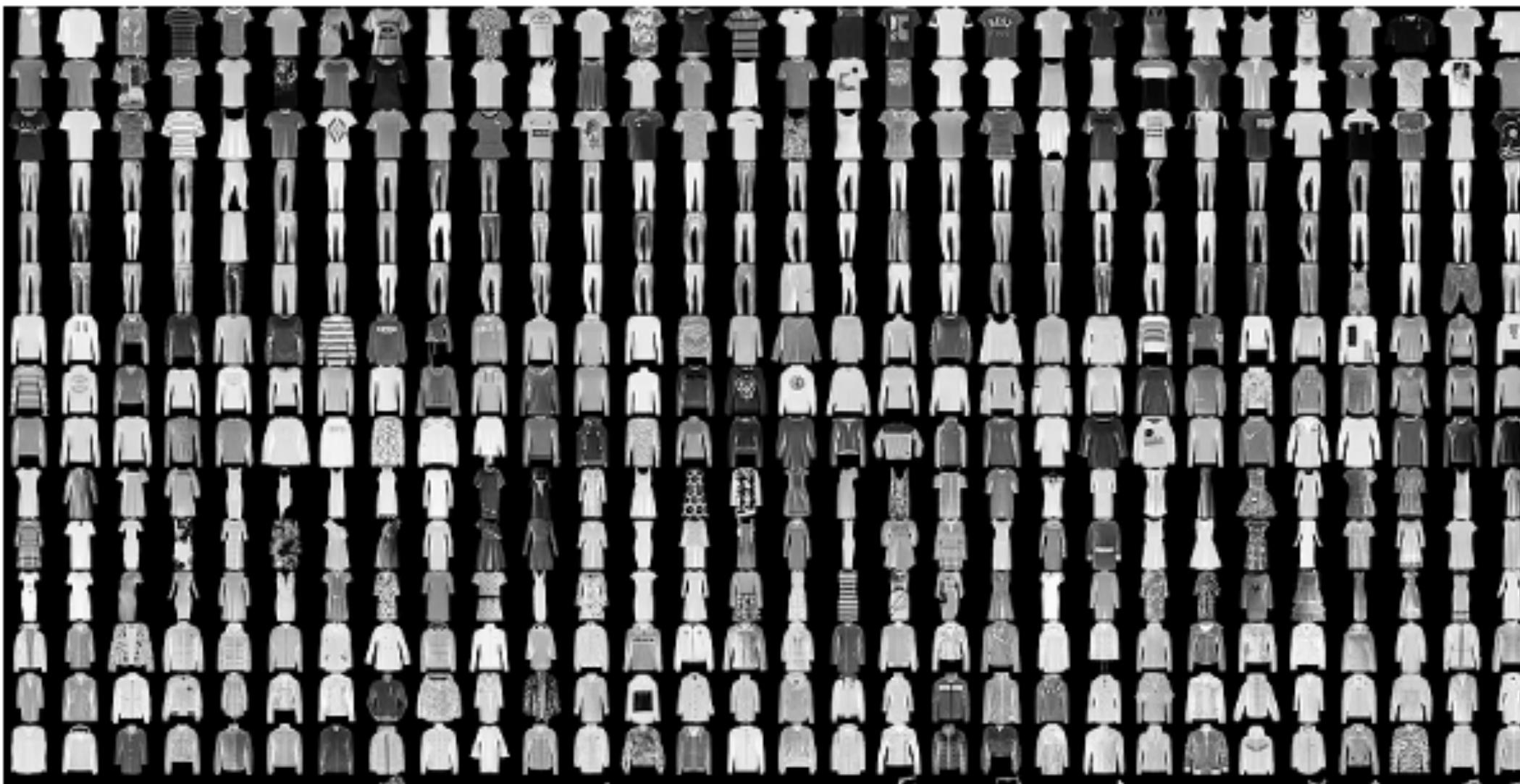
Original MNIST dataset

The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

721041495906901597349665407
605499219487397444925476790
361113952945939036557227128
471240274330031965259793042
283824503171579719214292049
395213136574226326548971303
686857860240223197510846247
107707944855408210845040615
834408833173596326136072171
144602914747398847121223232
902519781041796426813754418
785979696374453547878076887
818033723621611379080540287
441291469939844313188794887
67205520220249809946549183

Fashion MNIST dataset

This guide uses the **Fashion MNIST** dataset which contains 70,000 grayscale images in 10 categories. The images show individual articles of clothing at low resolution (28 by 28 pixels). <https://github.com/zalandoresearch/fashion-mnist>



Why we need Fashion MNIST

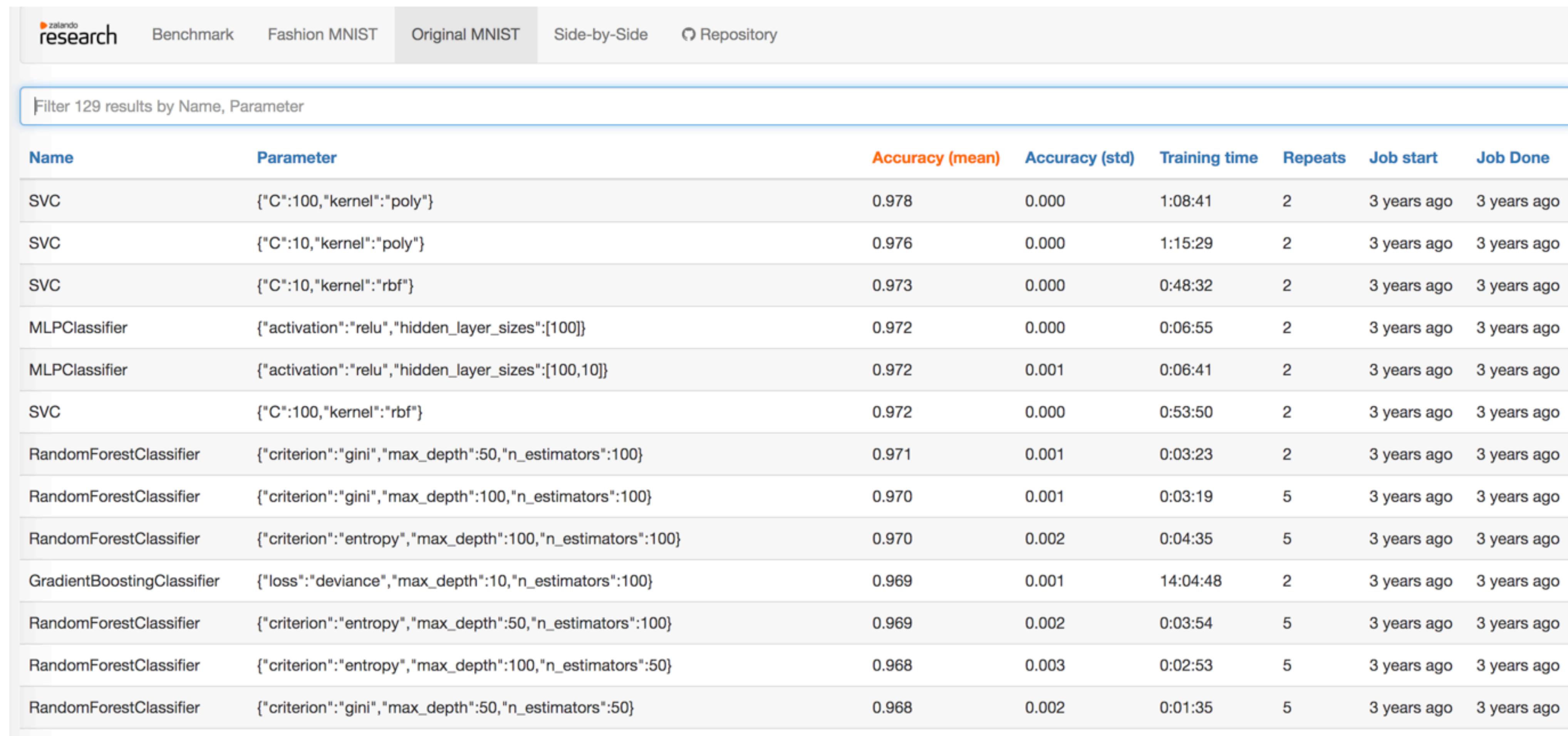
The original [MNIST dataset](#) contains a lot of handwritten digits. Members of the AI/ML/Data Science community love this dataset and use it as a benchmark to validate their algorithms. In fact, MNIST is often the first dataset researchers try. "*If it doesn't work on MNIST, it won't work at all*", they said. "*Well, if it does work on MNIST, it may still fail on others.*"

To Serious Machine Learning Researchers

Seriously, we are talking about replacing MNIST. Here are some good reasons:

- **MNIST is too easy.** Convolutional nets can achieve 99.7% on MNIST. Classic machine learning algorithms can also achieve 97% easily. Check out [our side-by-side benchmark for Fashion-MNIST vs. MNIST](#), and read "[Most pairs of MNIST digits can be distinguished pretty well by just one pixel.](#)"
- **MNIST is overused.** In [this April 2017 Twitter thread](#), Google Brain research scientist and deep learning expert Ian Goodfellow calls for people to move away from MNIST.
- **MNIST can not represent modern CV tasks,** as noted in [this April 2017 Twitter thread](#), deep learning expert/Keras author François Chollet.

Benchmark on original MNIST



The screenshot shows a web-based benchmarking interface for the 'Original MNIST' dataset. At the top, there's a navigation bar with links for 'Benchmark', 'Fashion MNIST', 'Original MNIST' (which is the active tab), 'Side-by-Side', and 'Repository'. Below the navigation is a search/filter bar with the placeholder 'Filter 129 results by Name, Parameter'. The main content area is a table with the following columns: Name, Parameter, Accuracy (mean), Accuracy (std), Training time, Repeats, Job start, and Job Done. The table lists 129 entries, mostly for SVC and MLPClassifier models with various parameter configurations.

Name	Parameter	Accuracy (mean)	Accuracy (std)	Training time	Repeats	Job start	Job Done
SVC	{"C":100,"kernel":"poly"}	0.978	0.000	1:08:41	2	3 years ago	3 years ago
SVC	{"C":10,"kernel":"poly"}	0.976	0.000	1:15:29	2	3 years ago	3 years ago
SVC	{"C":10,"kernel":"rbf"}	0.973	0.000	0:48:32	2	3 years ago	3 years ago
MLPClassifier	{"activation":"relu","hidden_layer_sizes":[100]}	0.972	0.000	0:06:55	2	3 years ago	3 years ago
MLPClassifier	{"activation":"relu","hidden_layer_sizes":[100,10]}	0.972	0.001	0:06:41	2	3 years ago	3 years ago
SVC	{"C":100,"kernel":"rbf"}	0.972	0.000	0:53:50	2	3 years ago	3 years ago
RandomForestClassifier	{"criterion":"gini","max_depth":50,"n_estimators":100}	0.971	0.001	0:03:23	2	3 years ago	3 years ago
RandomForestClassifier	{"criterion":"gini","max_depth":100,"n_estimators":100}	0.970	0.001	0:03:19	5	3 years ago	3 years ago
RandomForestClassifier	{"criterion":"entropy","max_depth":100,"n_estimators":100}	0.970	0.002	0:04:35	5	3 years ago	3 years ago
GradientBoostingClassifier	{"loss":"deviance","max_depth":10,"n_estimators":100}	0.969	0.001	14:04:48	2	3 years ago	3 years ago
RandomForestClassifier	{"criterion":"entropy","max_depth":50,"n_estimators":100}	0.969	0.002	0:03:54	5	3 years ago	3 years ago
RandomForestClassifier	{"criterion":"entropy","max_depth":100,"n_estimators":50}	0.968	0.003	0:02:53	5	3 years ago	3 years ago
RandomForestClassifier	{"criterion":"gini","max_depth":50,"n_estimators":50}	0.968	0.002	0:01:35	5	3 years ago	3 years ago

Benchmark on Fashion MNIST

Name	Parameter	Accuracy (mean)	Accuracy (std)	Training time	Repeats	Job start	Job Done
SVC	{"C":10,"kernel":"poly"}	0.897	0.000	1:12:39	2	3 years ago	3 years ago
SVC	{"C":100,"kernel":"poly"}	0.896	0.000	1:12:30	2	3 years ago	3 years ago
SVC	{"C":10,"kernel":"rbf"}	0.896	0.000	1:15:25	2	3 years ago	3 years ago
SVC	{"C":100,"kernel":"rbf"}	0.893	0.000	1:18:01	2	3 years ago	3 years ago
GradientBoostingClassifier	{"loss":"deviance","max_depth":10,"n_estimators":100}	0.888	0.001	17:38:22	5	3 years ago	3 years ago
SVC	{"C":1,"kernel":"rbf"}	0.884	0.000	1:15:54	2	3 years ago	3 years ago
GradientBoostingClassifier	{"loss":"deviance","max_depth":10,"n_estimators":50}	0.880	0.001	16:42:47	2	3 years ago	3 years ago
RandomForestClassifier	{"criterion":"entropy","max_depth":50,"n_estimators":100}	0.879	0.000	0:08:39	2	3 years ago	3 years ago
RandomForestClassifier	{"criterion":"entropy","max_depth":100,"n_estimators":100}	0.877	0.001	0:08:29	2	3 years ago	3 years ago
MLPClassifier	{"activation":"relu","hidden_layer_sizes":[100]}	0.877	0.005	0:16:03	5	3 years ago	3 years ago
RandomForestClassifier	{"criterion":"gini","max_depth":100,"n_estimators":100}	0.876	0.002	0:05:48	5	3 years ago	3 years ago
RandomForestClassifier	{"criterion":"gini","max_depth":50,"n_estimators":100}	0.876	0.003	0:09:55	5	3 years ago	3 years ago
RandomForestClassifier	{"criterion":"entropy","max_depth":100,"n_estimators":50}	0.875	0.001	0:06:25	5	3 years ago	3 years ago

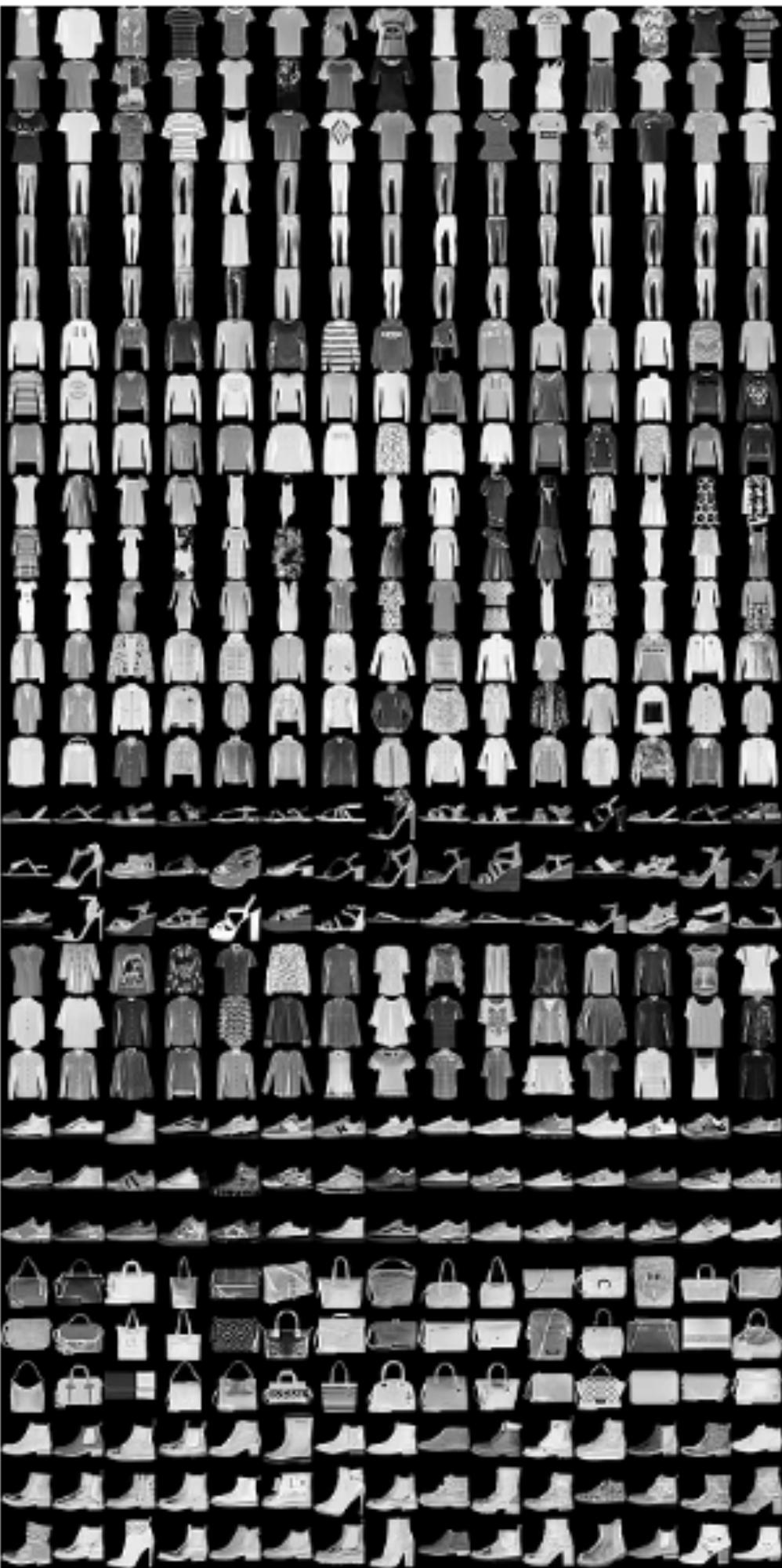
Benchmark Side-by-side

zalando research Benchmark Fashion MNIST Original MNIST Side-by-Side Repository

Filter 129 results by Name, Parameter

Name	Parameter	Fashion Accuracy (mean)	MNIST Accuracy (mean)	Fashion Accuracy (std)	MNIST Accuracy (std)
SVC	{"C":10,"kernel":"poly"}	0.897	0.976	0.000	0.000
SVC	{"C":100,"kernel":"poly"}	0.896	0.978	0.000	0.000
SVC	{"C":10,"kernel":"rbf"}	0.896	0.973	0.000	0.000
SVC	{"C":100,"kernel":"rbf"}	0.893	0.972	0.000	0.000
GradientBoostingClassifier	{"loss":"deviance","max_depth":10,"n_estimators":100}	0.888	0.969	0.001	0.001
SVC	{"C":1,"kernel":"rbf"}	0.884	0.966	0.000	0.000
GradientBoostingClassifier	{"loss":"deviance","max_depth":10,"n_estimators":50}	0.880	0.964	0.001	0.003
RandomForestClassifier	{"criterion":"entropy","max_depth":50,"n_estimators":100}	0.879	0.969	0.000	0.002
MLPClassifier	{"activation":"relu","hidden_layer_sizes":[100]}	0.877	0.972	0.005	0.000
RandomForestClassifier	{"criterion":"entropy","max_depth":100,"n_estimators":100}	0.877	0.970	0.001	0.002
RandomForestClassifier	{"criterion":"gini","max_depth":50,"n_estimators":100}	0.876	0.971	0.003	0.001
RandomForestClassifier	{"criterion":"gini","max_depth":100,"n_estimators":100}	0.876	0.970	0.002	0.001
RandomForestClassifier	{"criterion":"entropy","max_depth":100,"n_estimators":50}	0.875	0.968	0.001	0.003

Fashion MNIST dataset



Label	Description
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot



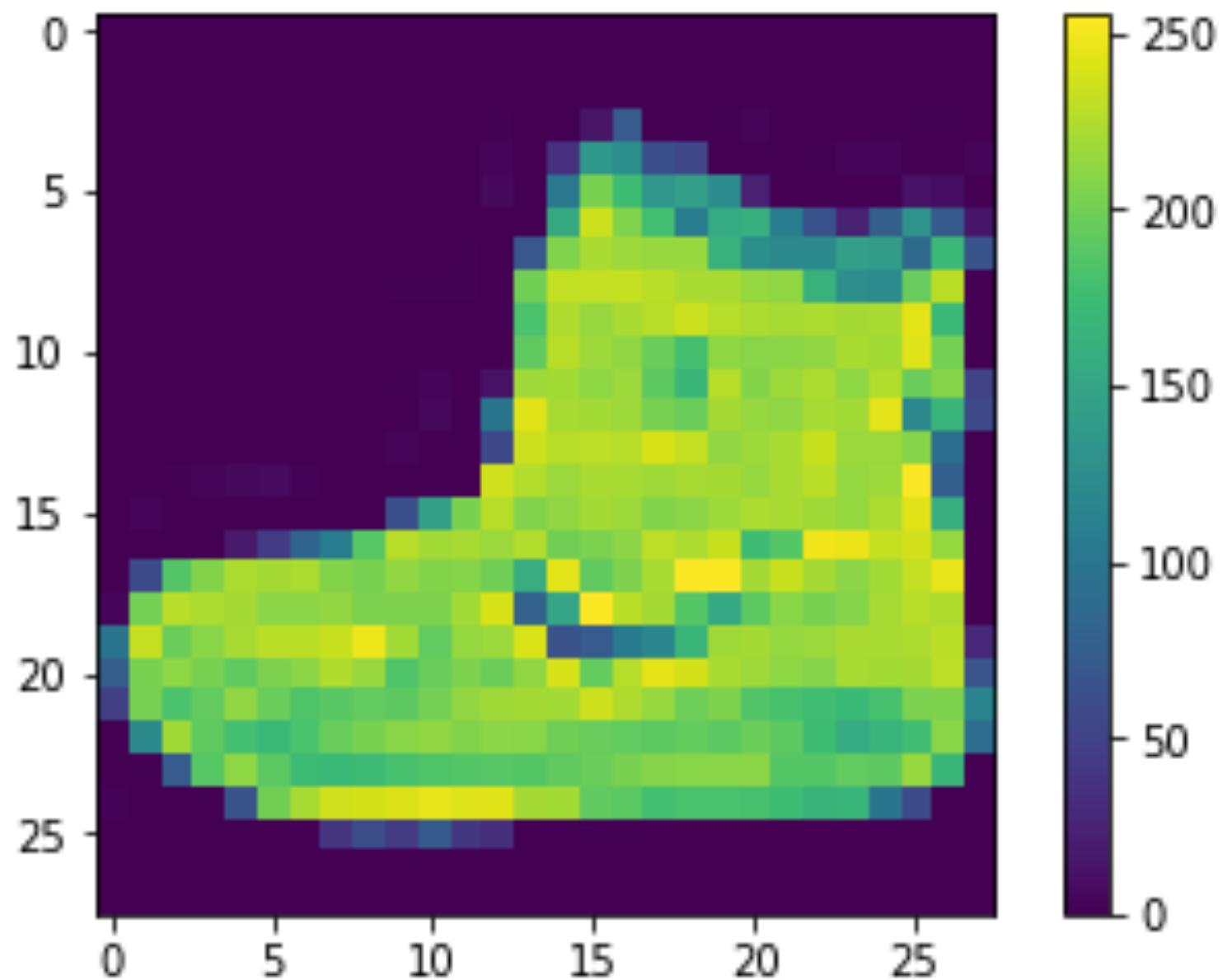
使用 TensorFlow 2 训练分类网络

Get Fashion MNIST dataset

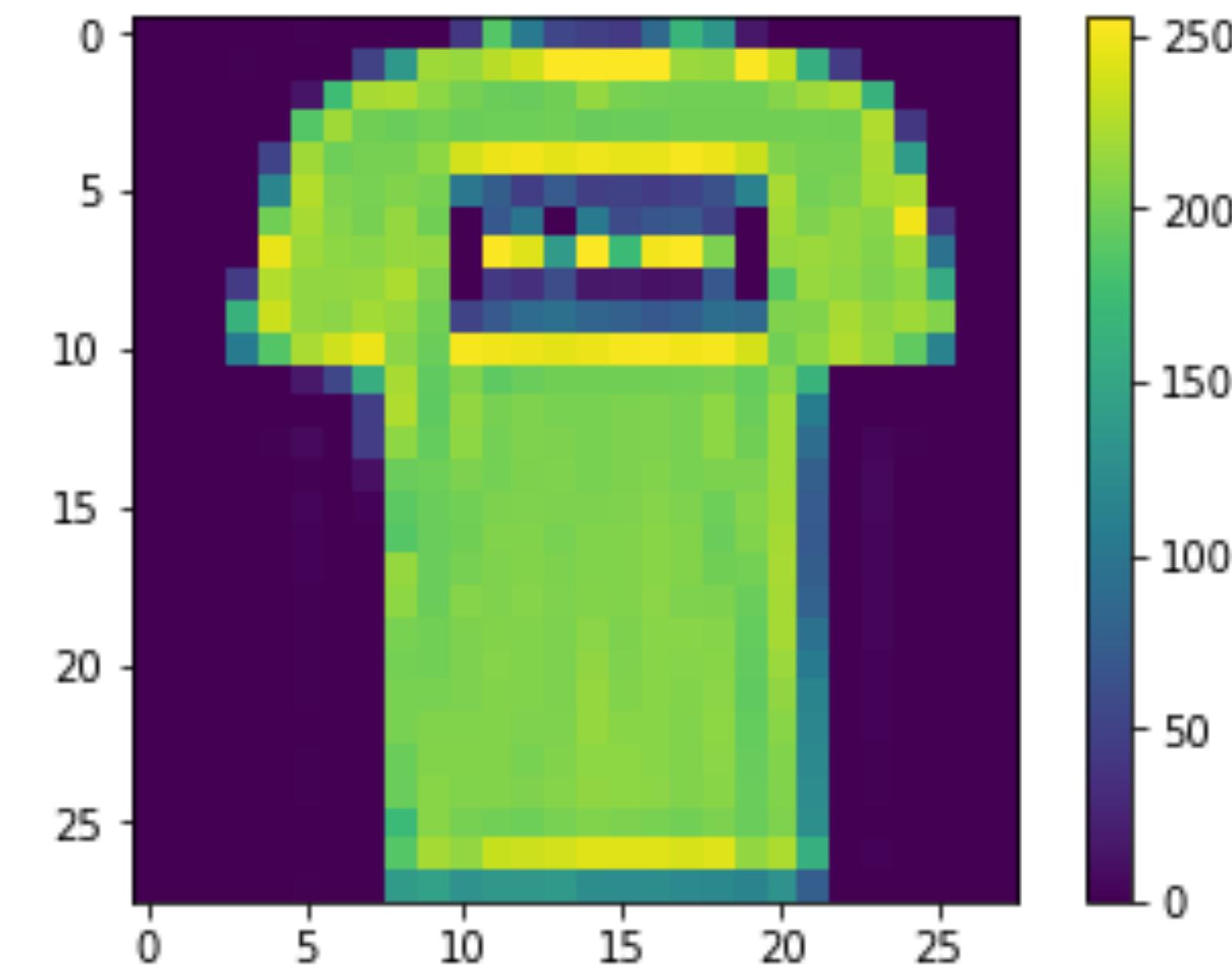
```
from tensorflow import keras  
  
import tensorflow as tf  
  
fashion_mnist = keras.datasets.fashion_mnist  
  
(train_images, train_labels), (test_images, test_labels) = fashion_mnist.load_data()
```

Preprocess data

```
import matplotlib.pyplot as plt  
plt.figure()  
plt.imshow(train_images[0])  
plt.colorbar()  
plt.grid(False)  
plt.show()
```



```
import matplotlib.pyplot as plt  
plt.figure()  
plt.imshow(train_images[1])  
plt.colorbar()  
plt.grid(False)  
plt.show()
```



Preprocess data

```
class_names = [ 'T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
                 'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot' ]  
  
train_images = train_images / 255.0  
test_images = test_images / 255.0  
  
plt.figure(figsize=(8, 8))  
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.binary)  
    plt.xlabel(class_names[train_labels[i]])  
    plt.show( )
```



Build the model

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)),
    keras.layers.Dense(128, activation=tf.nn.relu),
    keras.layers.Dense(10, activation=tf.nn.softmax)
])

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

model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
<hr/>		
flatten_3 (Flatten)	(None, 784)	0
dense_4 (Dense)	(None, 128)	100480
dense_5 (Dense)	(None, 10)	1290
<hr/>		
Total params:	101,770	
Trainable params:	101,770	
Non-trainable params:	0	

Train and evaluate

```
model.fit(train_images, train_labels, epochs=5)

Epoch 1/5
60000/60000 [=====] - 6s 100us/sample - loss: 3.0744 - accuracy: 0.6726
Epoch 2/5
60000/60000 [=====] - 6s 96us/sample - loss: 0.7389 - accuracy: 0.7142
Epoch 3/5
60000/60000 [=====] - 5s 83us/sample - loss: 0.6300 - accuracy: 0.7473
Epoch 4/5
60000/60000 [=====] - 6s 98us/sample - loss: 0.5631 - accuracy: 0.7987
Epoch 5/5
60000/60000 [=====] - 5s 86us/sample - loss: 0.5366 - accuracy: 0.8128
```

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

print('Test accuracy:', test_acc)

10000/10000 [=====] - 1s 63us/sample - loss: 0.6564 - accuracy: 0.7953
Test accuracy: 0.7953
```

Make prediction

```
predictions = model.predict(test_images)
```

```
import numpy as np  
np.argmax(predictions[0])
```

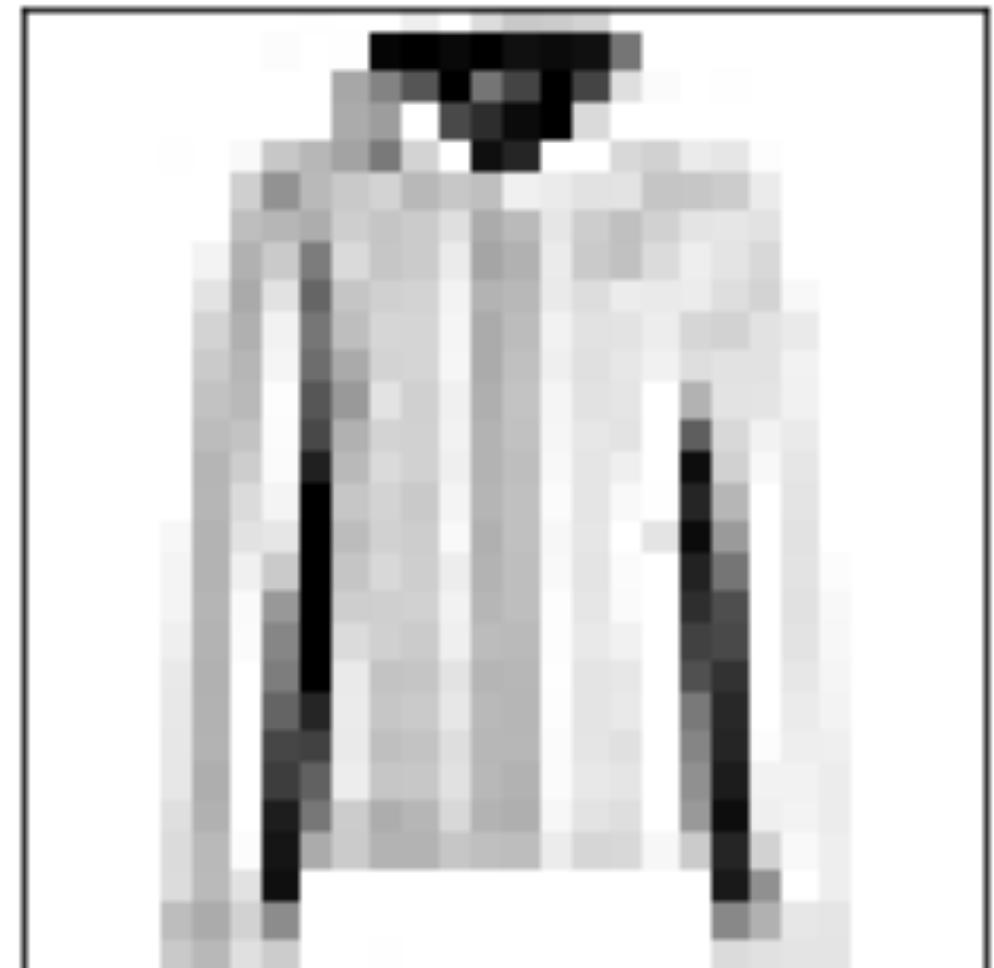
9

```
test_labels[0]
```

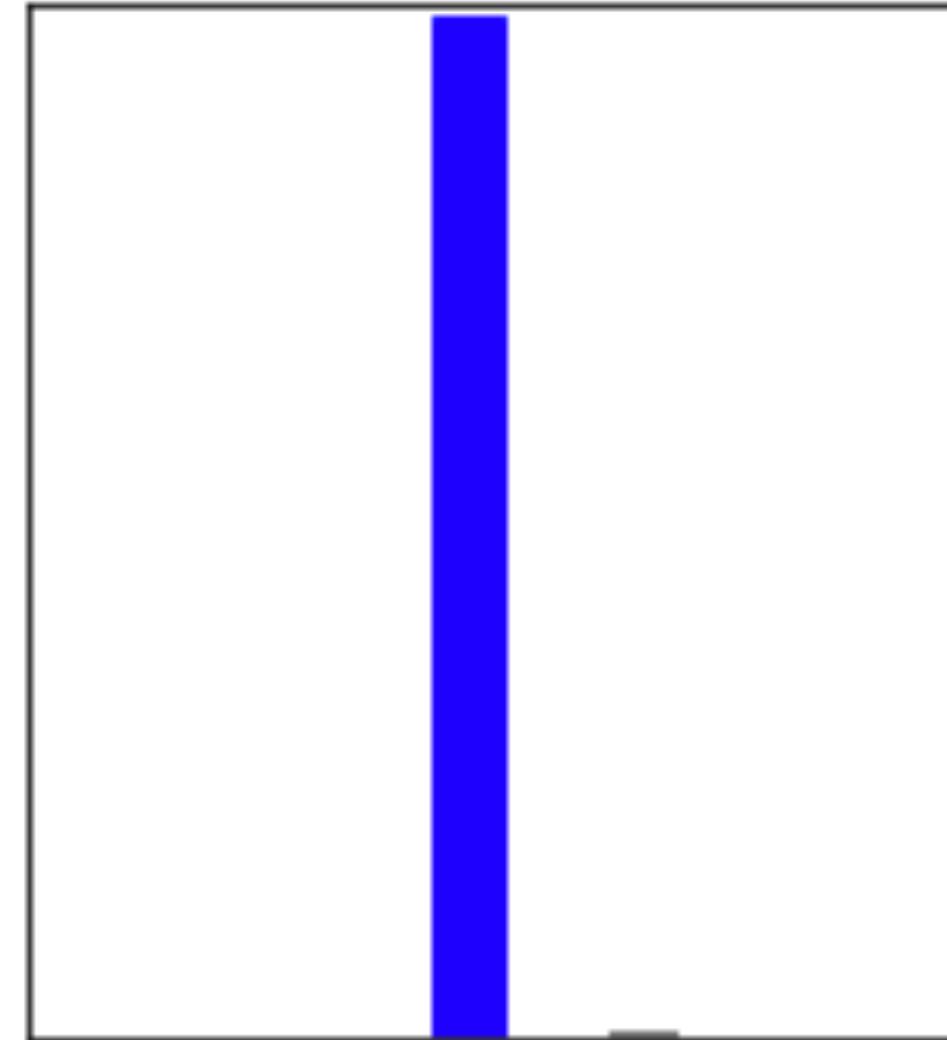
9

Visualize prediction

```
i = 6  
plt.figure(figsize=(6,3))  
plt.subplot(1,2,1)  
plot_image(i, predictions, test_labels, test_images)  
plt.subplot(1,2,2)  
plot_value_array(i, predictions, test_labels)  
plt.show()
```



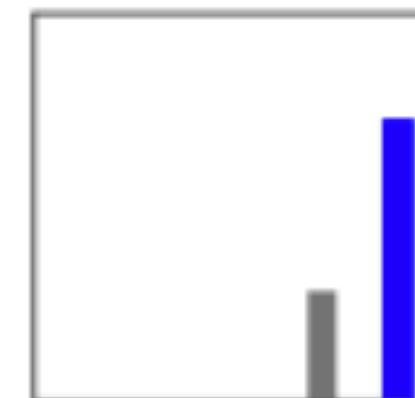
Coat 99% (Coat)



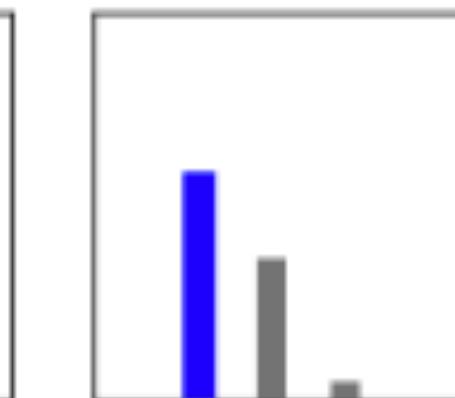
Visualize prediction



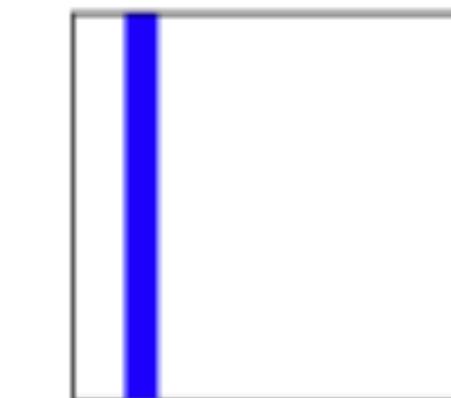
Ankle boot 72% (Ankle boot)



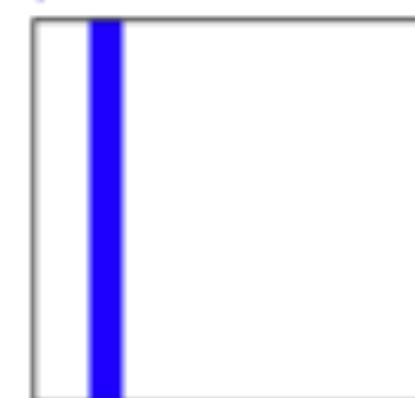
Pullover 59% (Pullover)



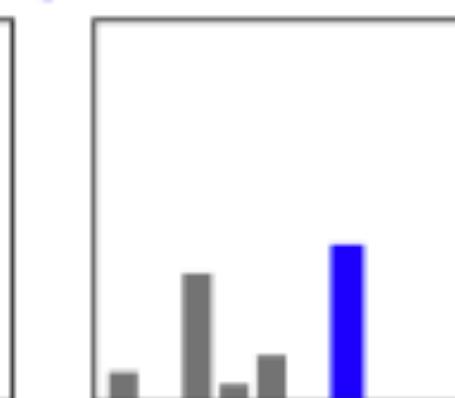
Trouser 100% (Trouser)



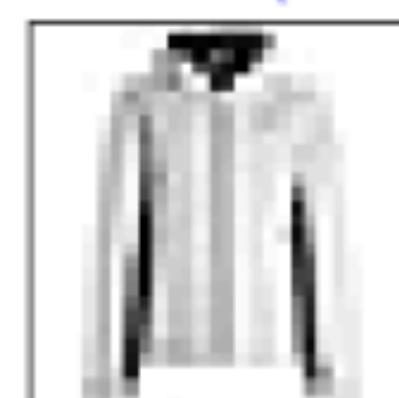
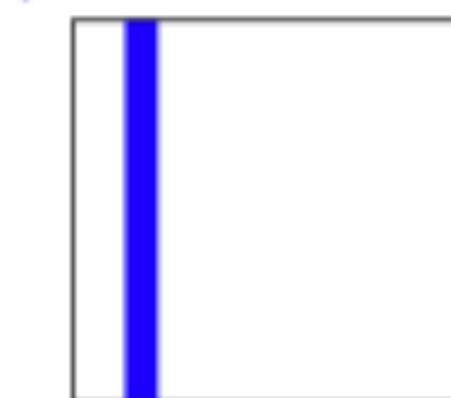
Trouser 100% (Trouser)



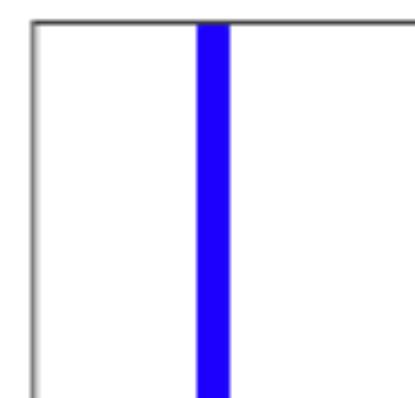
Shirt 41% (Shirt)



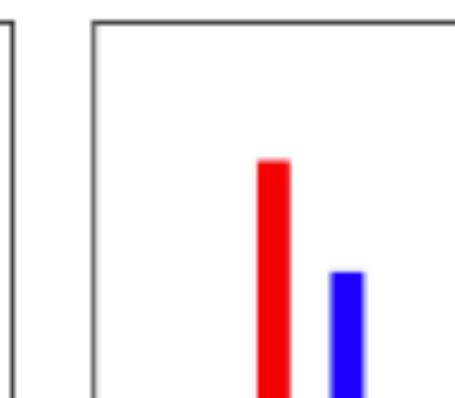
Trouser 100% (Trouser)



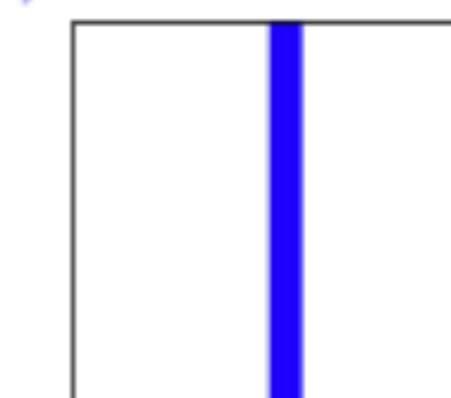
Coat 99% (Coat)



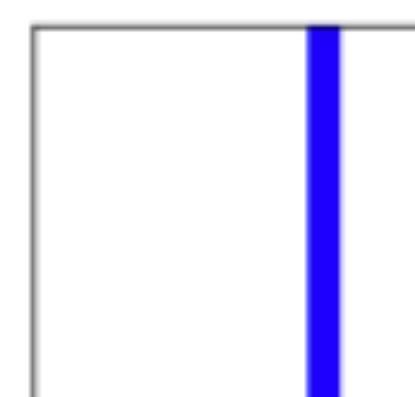
Coat 64% (Shirt)



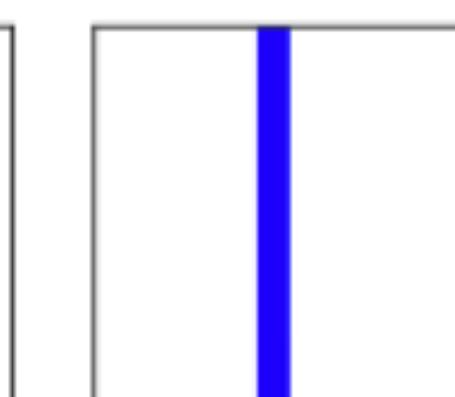
Sandal 100% (Sandal)



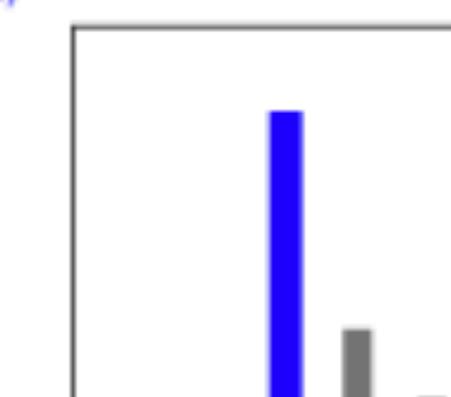
Sneaker 100% (Sneaker)



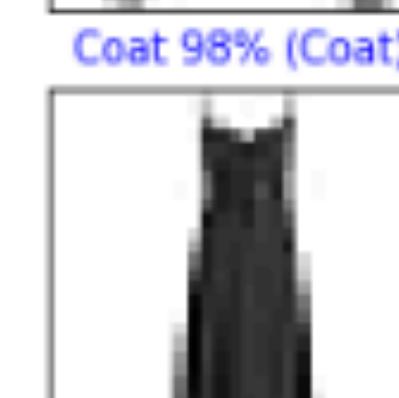
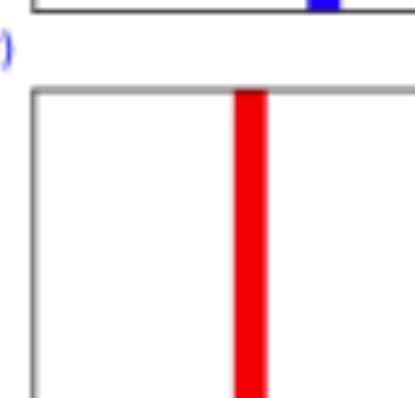
Coat 98% (Coat)



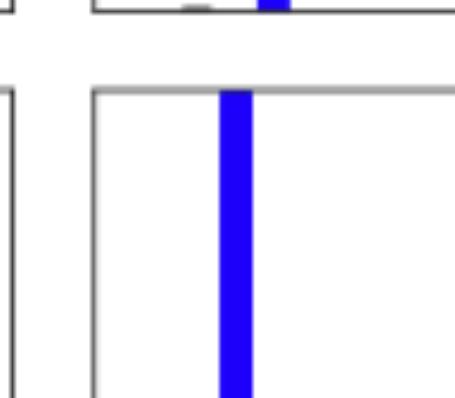
Sandal 77% (Sandal)



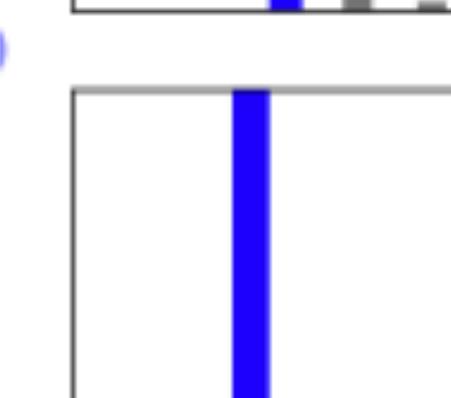
Sandal 100% (Sandal)



Dress 99% (Dress)



Coat 100% (Coat)



Try it!



扫码试看/订阅

《TensorFlow 2 项目进阶实战》视频课程