Introduction to Keras

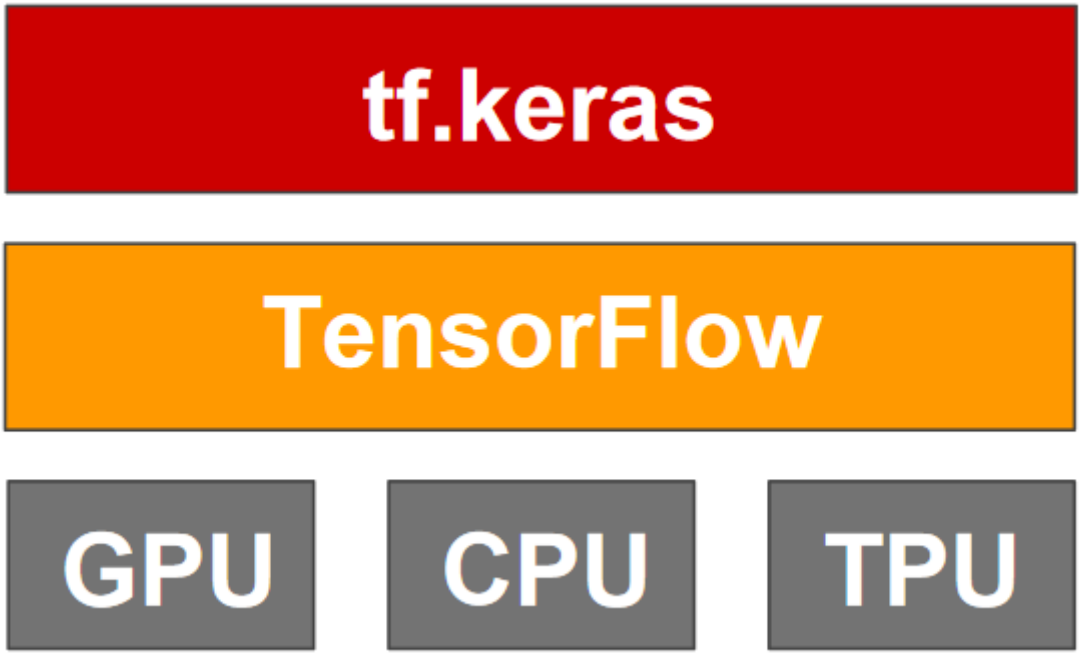
Keras (https://keras.io/) is a Python deep-learning framework authored by Francois Chollet (https://ai.google/research/people/105096) that provides a **convenient way**

- to define
- and train

deep-learning model.

or simply
Keras: an API for specifying & training differentiable programs[1] (https://web.stanford.edu/class/cs20si/lectures/march9guestlecture.pdf)

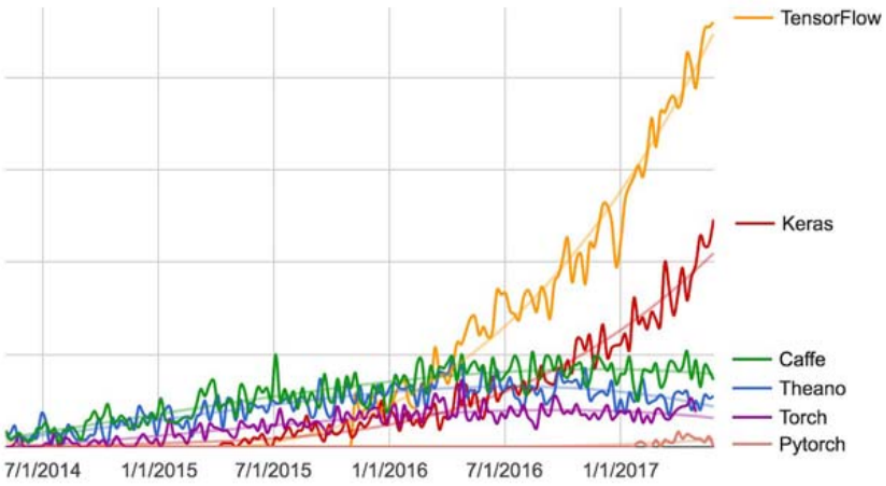
It is the official high-level API of TensorFlow



Keras has the following key features:

1. Inherits all pros of the backend ML library (No need to worry about Eager)
2. It has a user-friendly API that makes it easy to quickly prototype deep-learning models.
3. It has built-in support for various neural network architecturs
    - convolutional networks (for computer vision)
    - recurrent networks (for sequence processing)
    - and any combination of both

Interests in Keras [2] (https://twitter.com/fchollet/status/871089784898310144)

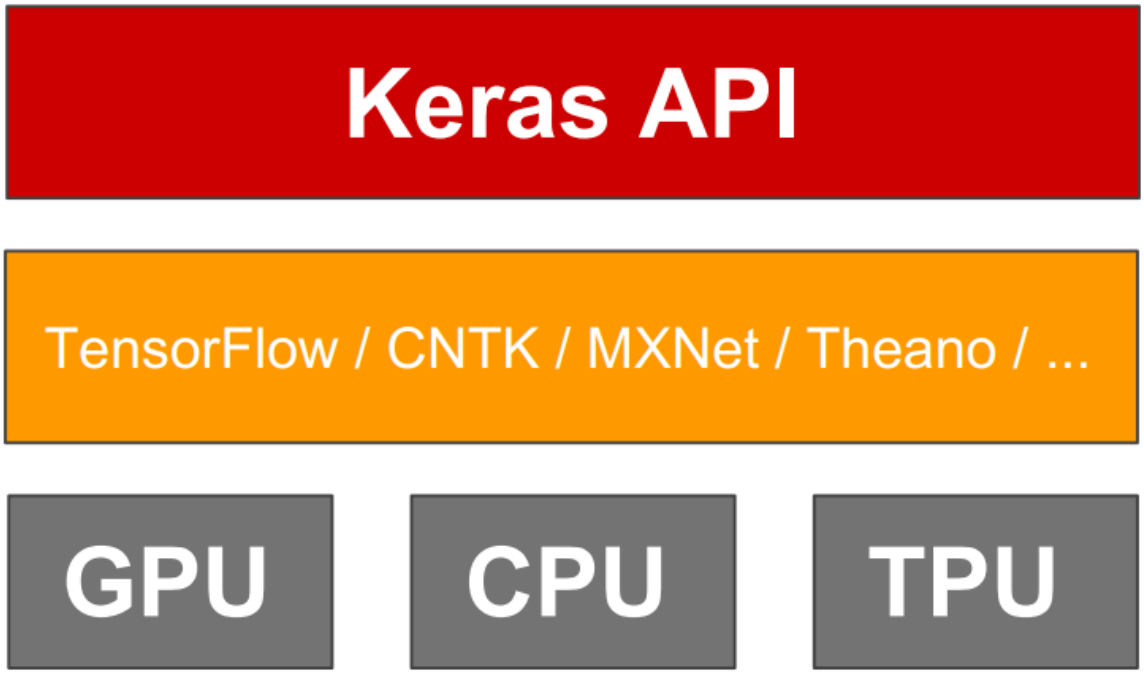


Google web search interest for different deep-learning frameworks over time

Who or what makes up Keras?

Keras does not execute low-level operations.
Instead it relies on a specialized, well-optimized tensor library to handle low level operations e.g TensorFlow
For handling operations such as tensor manipulation and differentiation, serving as the backend engine of Keras.

Keras handles the problem in a modular way.



In [1]:

```
"""
importing modules
"""
from keras import backend as BackEnd # the module that allows us to manipulate our Kera
s backend
import os # library that will give us system access to the keras backend file
from importlib import reload # the library that will we will use reload a function
```

```
Using TensorFlow backend.
```

Switching to CNTK[3] (https://github.com/Microsoft/CNTK) as our backend.

In [2]:

```
"""
# setting our backend to CNTK
i.e Microsoft Cognitive toolkit
"""
os.environ['KERAS_BACKEND'] = "cntk"
reload(BackEnd)
```

```
Using CNTK backend
```

Out[2]:

```
<module 'keras.backend' from 'c:\\intelpython3\\lib\\site-packages\\keras
\\backend\\__init__.py'>
```

Switching to Theano[4] (http://deeplearning.net/software/theano) as our backend.

In [3]:

```
"""
# setting our backend to Theano
developed by the MILA lab at Université de Montréal,
"""
os.environ['KERAS_BACKEND'] = "theano"
reload(BackEnd)
```

```
Using Theano backend.
```

Out[3]:

```
<module 'keras.backend' from 'c:\\intelpython3\\lib\\site-packages\\keras
\\backend\\__init__.py'>
```

In [4]:

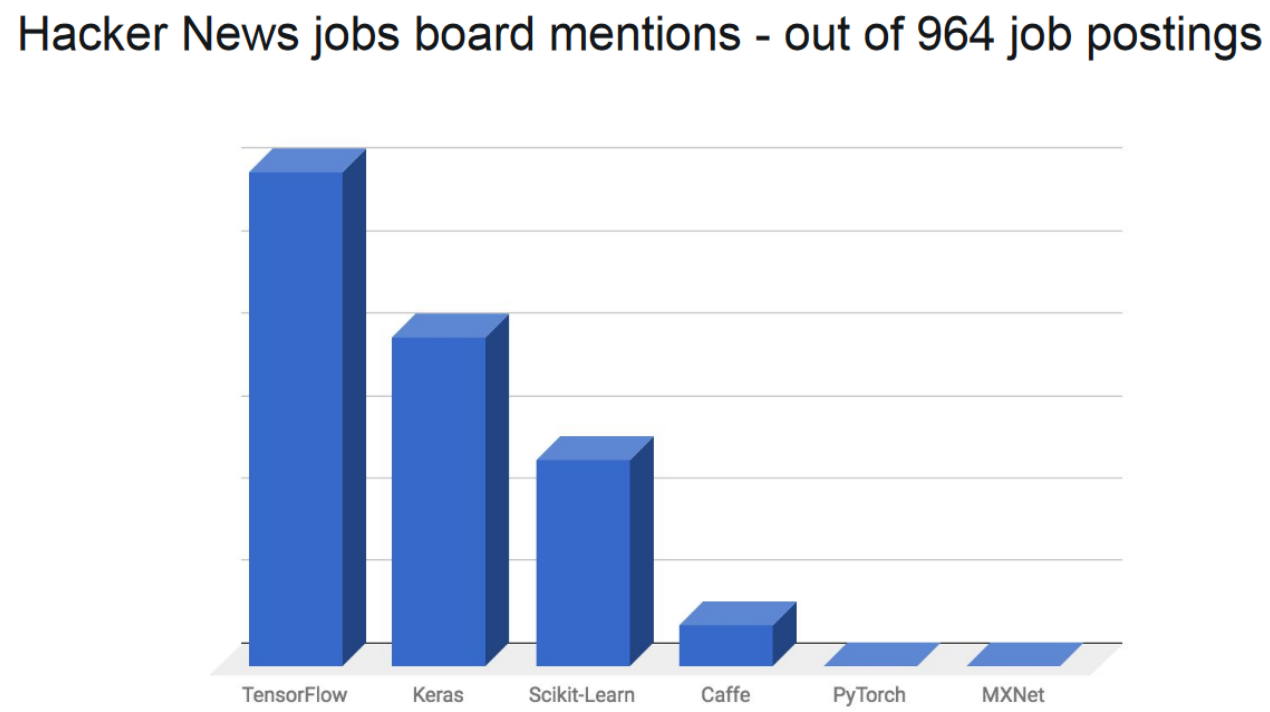
```
"""
Let's make life easier by making a function;
this function will let us just use one line to change out backend
"""
def set_keras_backend(backend):# set_keras_backend will be the name of our function
                               # the argument we will take in will be our backend frame
work
    if BackEnd.backend() != backend:
        os.environ['KERAS_BACKEND'] = backend
        reload(BackEnd)
        assert BackEnd.backend() == backend
```

In [5]:

```
set_keras_backend("tensorflow")
```

```
Using TensorFlow backend.
```

Scaling on the job market

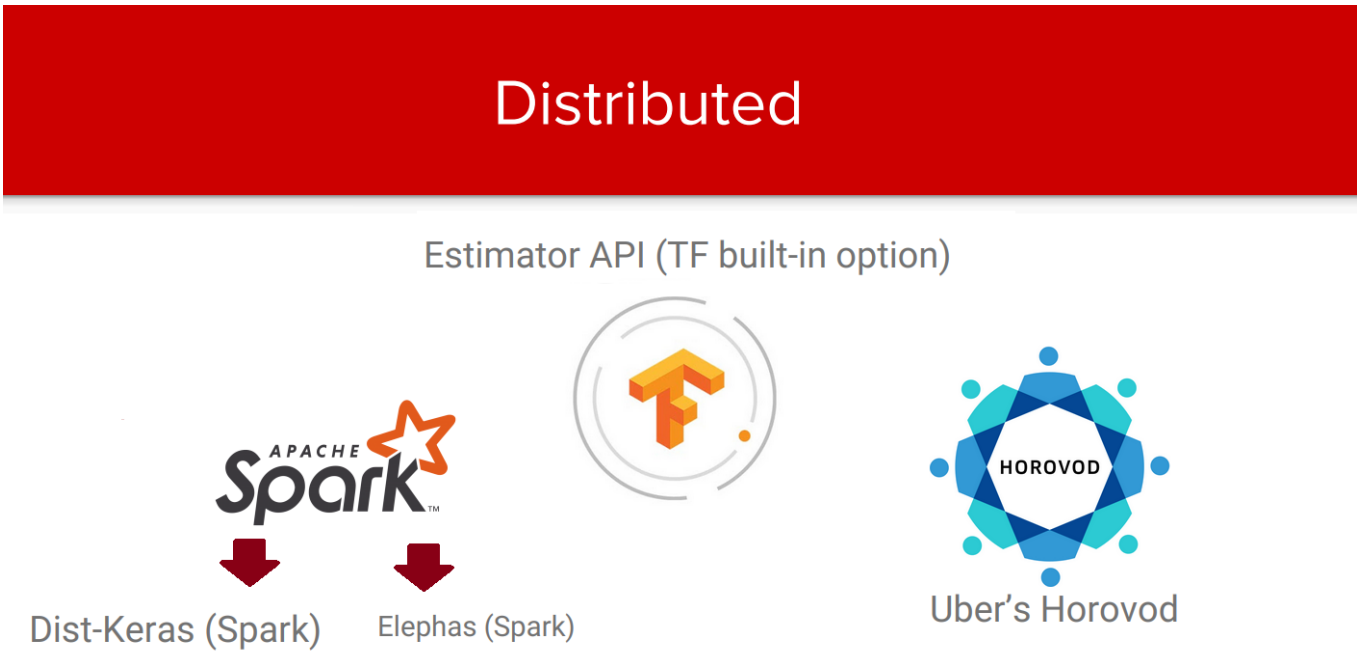


Who is using Keras?

Industry traction

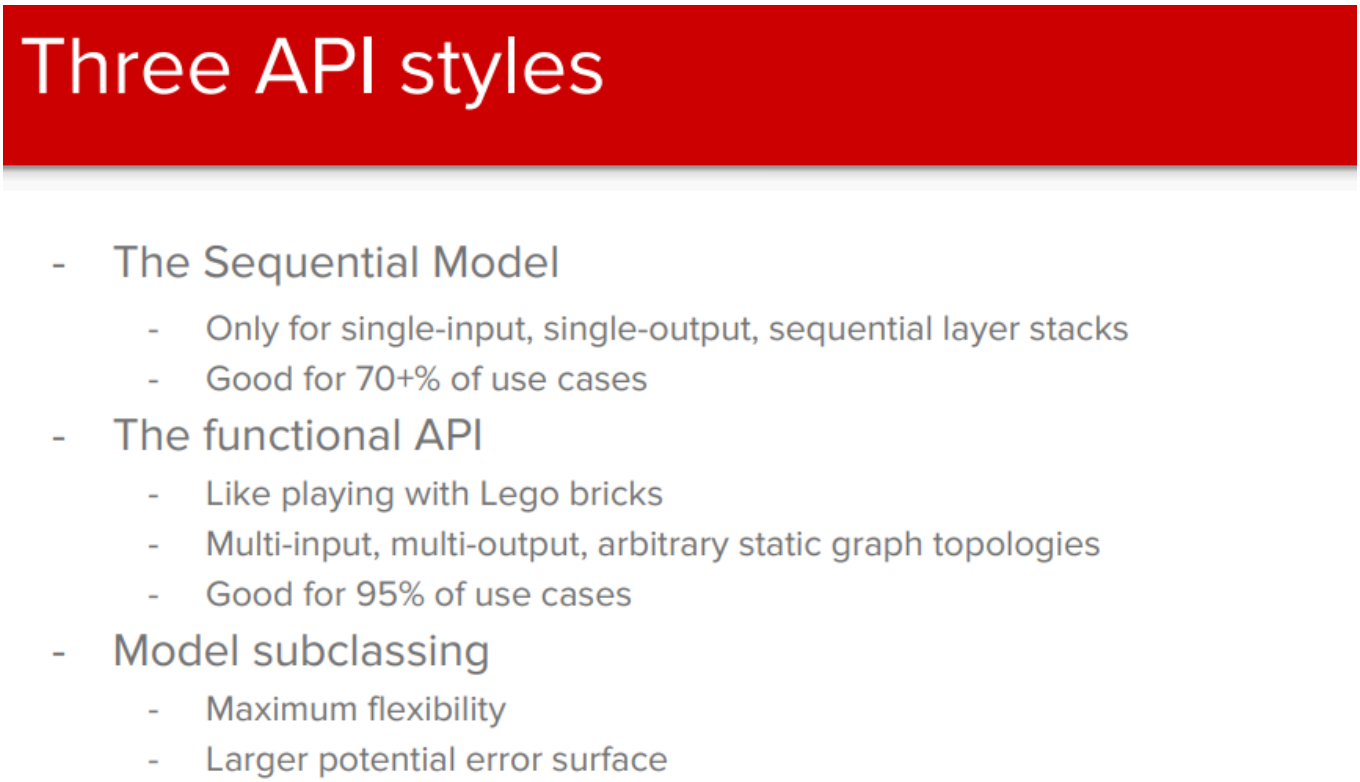
etc...

Distributed Computing



How to use Keras

Keras is a simple but powerful tool or users of every experience level



The Sequential API

In [ ]:

```
import keras
from keras import layers
```

In [ ]:

```
model = keras.Sequential()
```

In [ ]:

```
model.add(layers.Dense(20, activation="relu", input_shape=(10,)))
model.add(layers.Dense(20, activation="relu",))
model.add(layers.Dense(10, activation="softmax"))
```

In [ ]:

```
model.fit(x, y, epochs=10, batch_size=32)
```

The Functional API

In [ ]:

```
import keras
from keras import layers
```

In [ ]:

```
inputs = keras.Input(shape=(10,))
```

In [ ]:

```
x = layers.Dense(20, activation="relu")(x)
x = layers.Dense(20, activation="relu")(x)
```

In [ ]:

```
outputs = layers.Dense(10, activation="softmax")(x)
```

In [ ]:

```
model = keras.Model(inputs, outputs)
```

In [ ]:

```
model.fit(x, y, epochs=10, batch_size=32)
```

Model subclassing

In [ ]:

```
import keras
from keras import layers
```

In [ ]:

```
class MyModel(keras.Model):
    def__init__(self):
        super(MyModel, self).__init__()
        self.dense1 = layers.Dense(20, activation="relu")
        self.dense2 = layers.Dense(20, activation="relu")
        self.dense3 = layers.Dense(10, activation="softmax")

    def call(self, inputs):
        x = self.dense1(x)
        x = self.dense2(x)
        return self.dense3(x)

model = MyModel()
model.fit(x, y, epochs, batch_size=32)
```

MNIST

In [6]:

```
# mnist comes with keras as numpy arrays
from keras.datasets import mnist
```

In [7]:

```
# loading the training and test set images along with the labels
(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
```

In [8]:

```
# to see the number of training images and dimensions
train_images.shape
```

Out[8]:

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

In [9]:

```
# viewing the number of labels "training"
len(train_labels)
```

Out[9]:

```
60000
```

In [10]:

```
# to view the stored form and stored format of the training labels
train_labels
```

Out[10]:

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

We move onto the test data now

In [11]:

```
# to see the number of testing images and dimensions
test_images.shape
```

Out[11]:

```
(10000, 28, 28)
```

In [12]:

```
# viewing the number of labels "testing"
len(test_labels)
```

Out[12]:

```
10000
```

In [13]:

```
# to view the stored form and stored format of the testing labels
test_labels
```

Out[13]:

```
array([7, 2, 1, ..., 4, 5, 6], dtype=uint8)
```

About the network architecture

In [14]:

```
from keras import models
from keras import layers
```

In [15]:

```
network = models.Sequential()

# the sequential model is connected to 512 neurons in the network
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))

# Each score will be the probability that the current digit image belongs to
# one of our 10 digit classes.
network.add(layers.Dense(10, activation='softmax'))
# it will return an array of 10 probability scores
# (summing to 1)
```

-> A loss function—How the network will be able to measure its performance on the training data, and thus how it will be able to steer itself in the right direction.

-> An optimizer—The mechanism through which the network will update itself based on the data it sees and its loss function.

-> Metrics to monitor during training and testing—Here, we'll only care about accuracy (the fraction of the images that were correctly classified).

The compilation step

In [16]:

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

Before training, we'll preprocess the data by reshaping it into the shape the network expects and scaling it so that all values are in the [0, 1] interval.

Previously, our training images, for instance, were stored in an array of shape (60000, 28, 28) of type uint8 with values in the [0, 255] interval.

In [17]:

```
# We transform it into a float32 array of
# shape (60000, 28 * 28) with values between 0 and 1.

train_images = train_images.reshape((60000, 28 * 28))
train_images = train_images.astype('float32') / 255

# dividing by 255 is a form of data normalisation
test_images = test_images.reshape((10000, 28 * 28))
test_images = test_images.astype('float32') / 255
```

In [18]:

```
# Linking the labels and images

from keras.utils import to_categorical
# We need to categorically encode the labels

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

In [19]:

```
# We're now ready to train the network, which in Keras is done via a call to the net-
# work's fit method—we fit the model to its training data

#network.fit(train_images, train_labels, epochs=10, batch_size=128)

network.fit(train_images, train_labels,batch_size=128, epochs=10, verbose=1, validation
_split=0.1)
```

```
Train on 54000 samples, validate on 6000 samples
Epoch 1/10
54000/54000 [==============================] - 4s 73us/step - loss: 0.2777
- acc: 0.9193 - val_loss: 0.1233 - val_acc: 0.9630
Epoch 2/10
54000/54000 [==============================] - 4s 73us/step - loss: 0.1136
- acc: 0.9666 - val_loss: 0.0958 - val_acc: 0.9733
Epoch 3/10
54000/54000 [==============================] - 4s 75us/step - loss: 0.0748
- acc: 0.9777 - val_loss: 0.0780 - val_acc: 0.9788
Epoch 4/10
54000/54000 [==============================] - 4s 76us/step - loss: 0.0534
- acc: 0.9838 - val_loss: 0.0851 - val_acc: 0.9732
Epoch 5/10
54000/54000 [==============================] - 4s 81us/step - loss: 0.0397
- acc: 0.9882 - val_loss: 0.0647 - val_acc: 0.9815
Epoch 6/10
54000/54000 [==============================] - 4s 76us/step - loss: 0.0305
- acc: 0.9905 - val_loss: 0.0585 - val_acc: 0.9832
Epoch 7/10
54000/54000 [==============================] - 4s 79us/step - loss: 0.0234
- acc: 0.9930 - val_loss: 0.0611 - val_acc: 0.9837
Epoch 8/10
54000/54000 [==============================] - 5s 84us/step - loss: 0.0175
- acc: 0.9949 - val_loss: 0.0704 - val_acc: 0.9825
Epoch 9/10
54000/54000 [==============================] - 5s 84us/step - loss: 0.0130
- acc: 0.9965 - val_loss: 0.0714 - val_acc: 0.9823
Epoch 10/10
54000/54000 [==============================] - 5s 89us/step - loss: 0.0106
- acc: 0.9967 - val_loss: 0.0712 - val_acc: 0.9827
```

Out[19]:

```
<keras.callbacks.History at 0x175106e63c8>
```

In [20]:

```
# Now let's check that the model performs well on the test set
test_loss, test_acc = network.evaluate(test_images, test_labels)
print('test_acc:', test_acc)
```

```
10000/10000 [==============================] - 1s 50us/step
test_acc: 0.9821
```

Thank you

Shout out to our sponsor

visit their website (http://deepanalytics.ai/)

Like their Facebook page (https://www.facebook.com/DeepAnalyticsAI/)

Join our Facebook Group (https://www.facebook.com/groups/harareschoolofai/)