Intro to Tensorflow (TF)

Deep Learning for Computer Vision
ELEC4240/COMP4471

About me

- Name: Chenyang QI
- Email: cqiaa@connect.ust.hk
- 3nd year PhD student in CSE,
  supervised by Prof. Qifeng Chen
- Mainly doing research in computational photography

Today’s Tutorial

1. Quick overview of Tensorflow
2. Installation
3. Example {from loading data to evaluating model}
4. Save & Load model
5. Try it yourself

Source of tutorial : tf_basic

Read the documentation to understand more. This tutorial only provide **the basic**

What is tensorflow?

Open Source software library (by google)

Well known usage : machine learning; but has other usage as well

Why tensorflow?

1. You’re already familiar with tensorflow
2. You work with google
3. You want faster deployment and production

In general :

More flexible than pytorch → more complicated

MXNET provides alternative but community is not as big

Installation

Latest version : Tensorflow 2.0

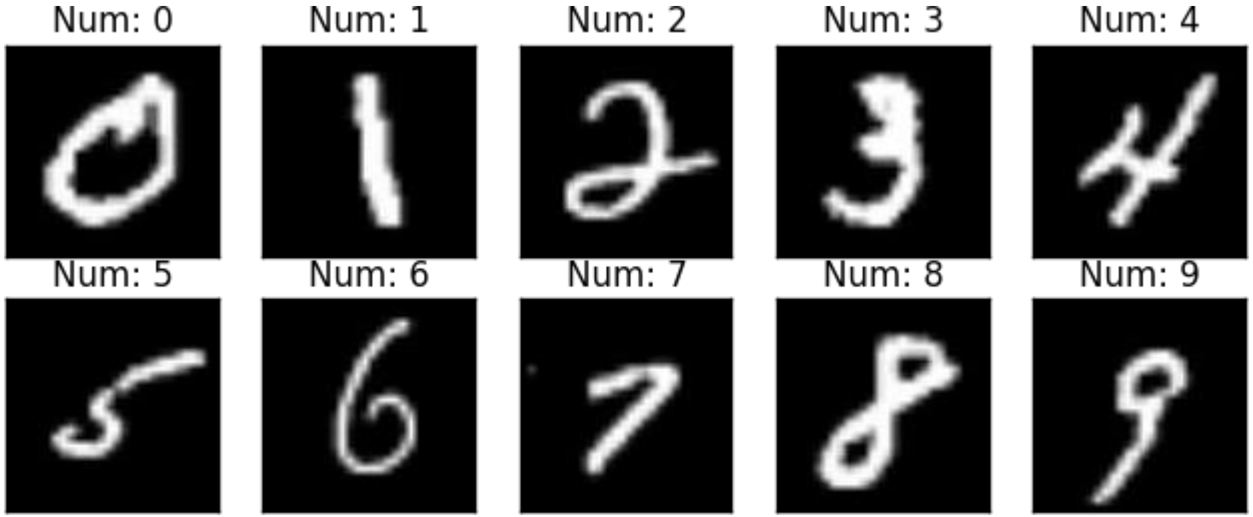
Command :

```
Pip install tensorflow

Conda install tensorflow
```

General guideline

1. Design your application {supervised/unsupervised}
2. Make sure you have necessary components (i.e. opensource, data&label)
3. Decide on the input & output → training & deployment have to be **consistent**
4. Test, fine tune, & debug (complexity, overfitting, etc)

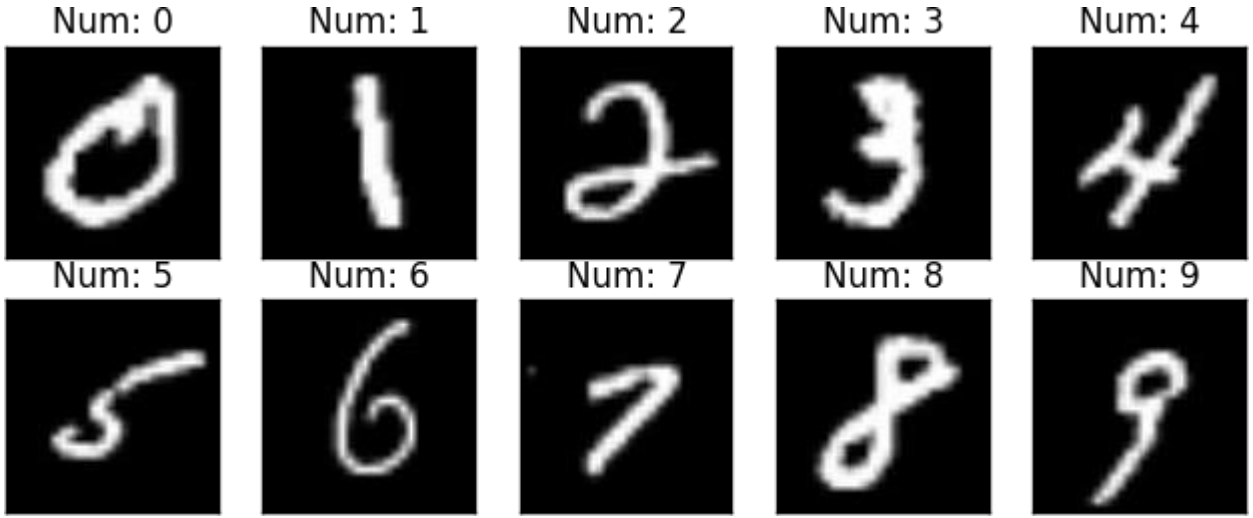
Example - MNIST classification



Task :

Build a **model** to classify input image as digits

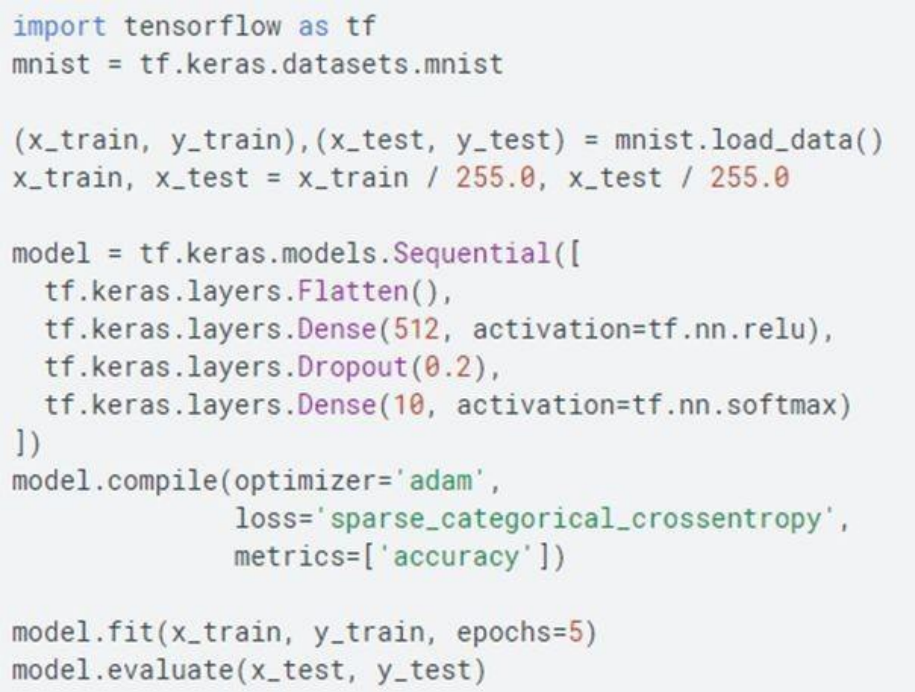
Example - MNIST classification



1. Get the data & label for training → preprocessing helps {observe}
2. Set the input & output for your model → based on how you want to deploy
3. Debug → until satisfactory

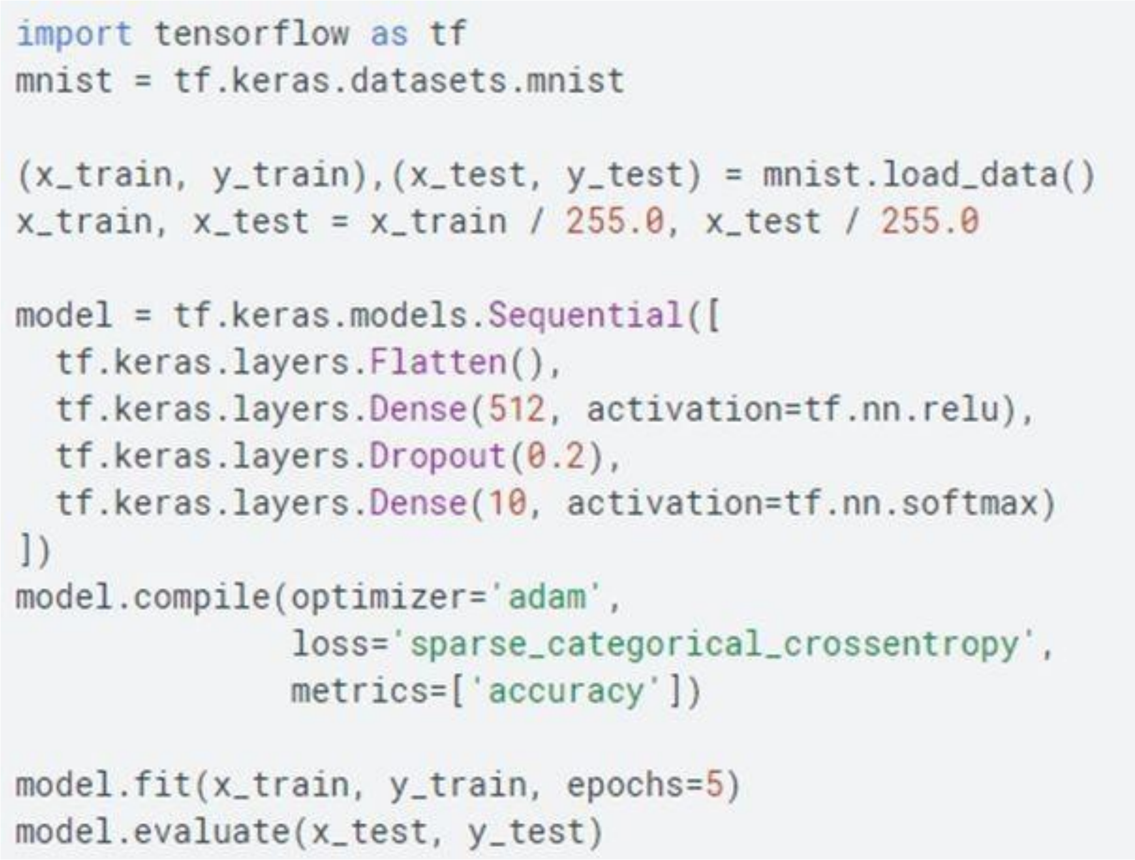
Quick overview

Implementation using Tensorflow - just **few lines**



```
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

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Deeper dive

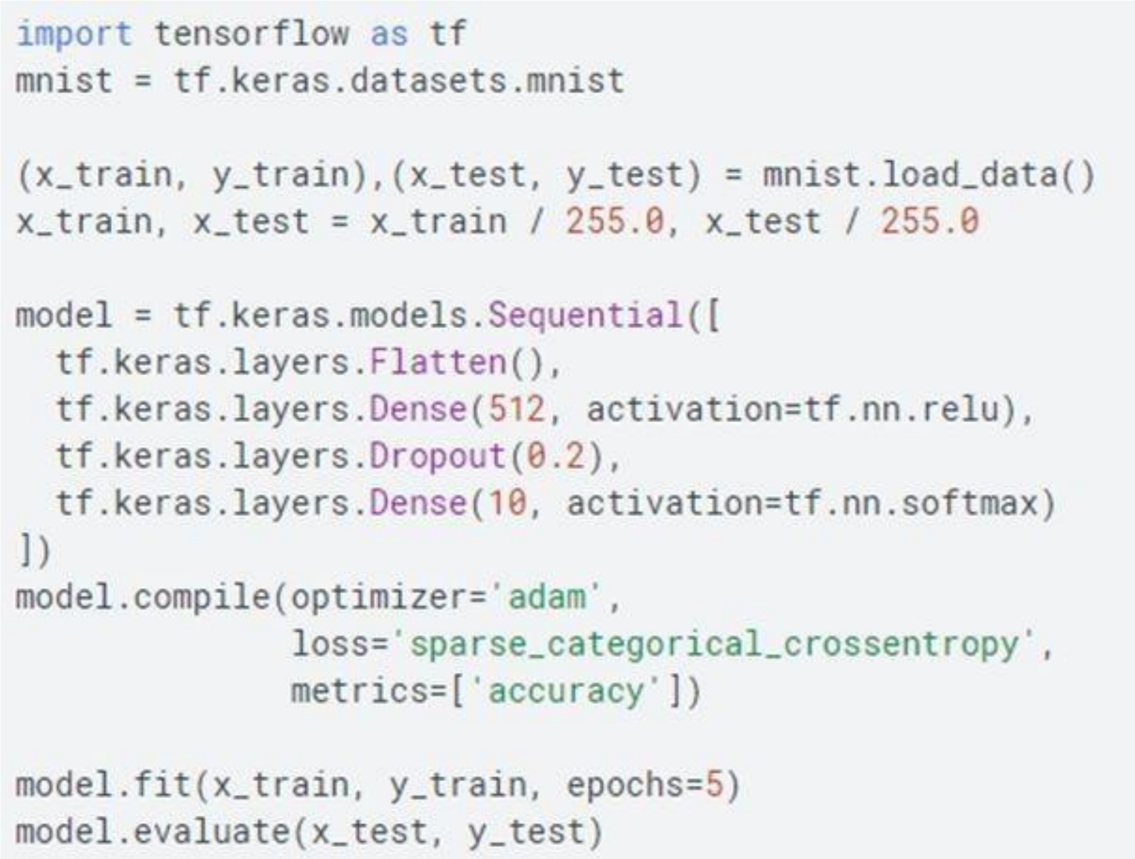


```
import tensorflow as tf
mnist = tf.keras.datasets.mnist

(x_train, y_train),(x_test, y_test) = mnist.load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Step1

Call the **necessarypackage (import)**

Deeper dive



```
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

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Step1

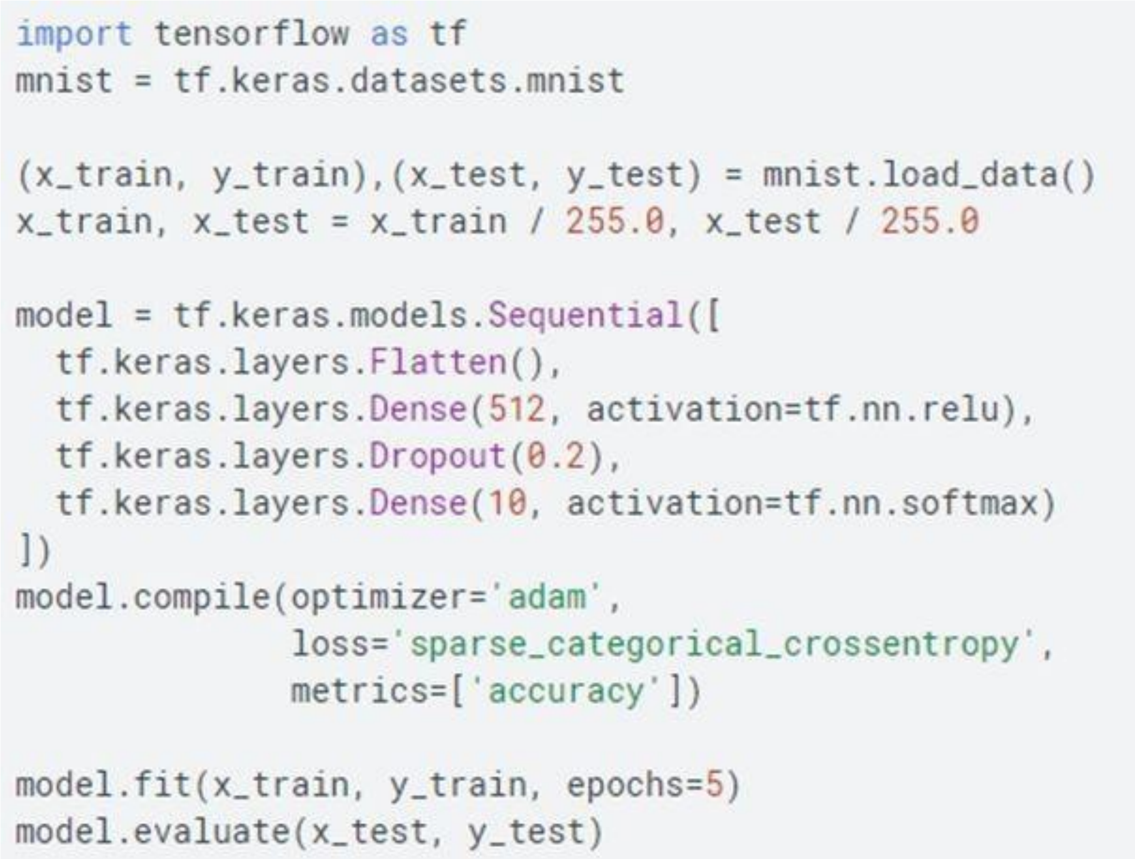
Call the **necessarypackage (import)**

Step2

Load the dataset (MNIST is **common** hence simple)

Preprocess data {normalize} → why? Does it matter?

Deeper dive



```
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

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Step1

Call the **necessarypackage (import)**

Step2

Load the dataset (MNIST is **common** hence simple)

Preprocess data {normalize} → why? Does it matter?

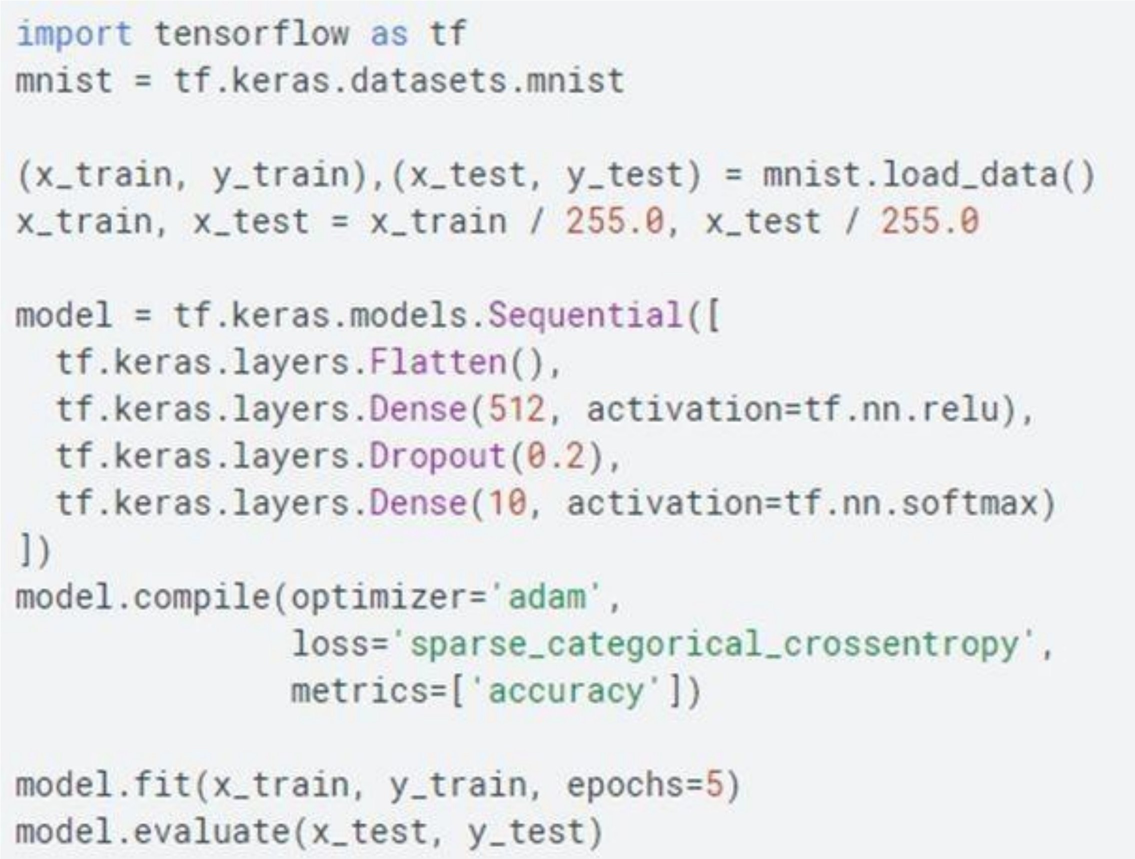
Step 3

Design your model.

Is this model effective? Why this model?

What is the **input** and **output?**

Deeper dive



```
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

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Step1

Call the **necessarypackage (import)**

Step2

Load the dataset (MNIST is **common** hence simple)

Preprocess data {normalize} → why? Does it matter?

Step 3

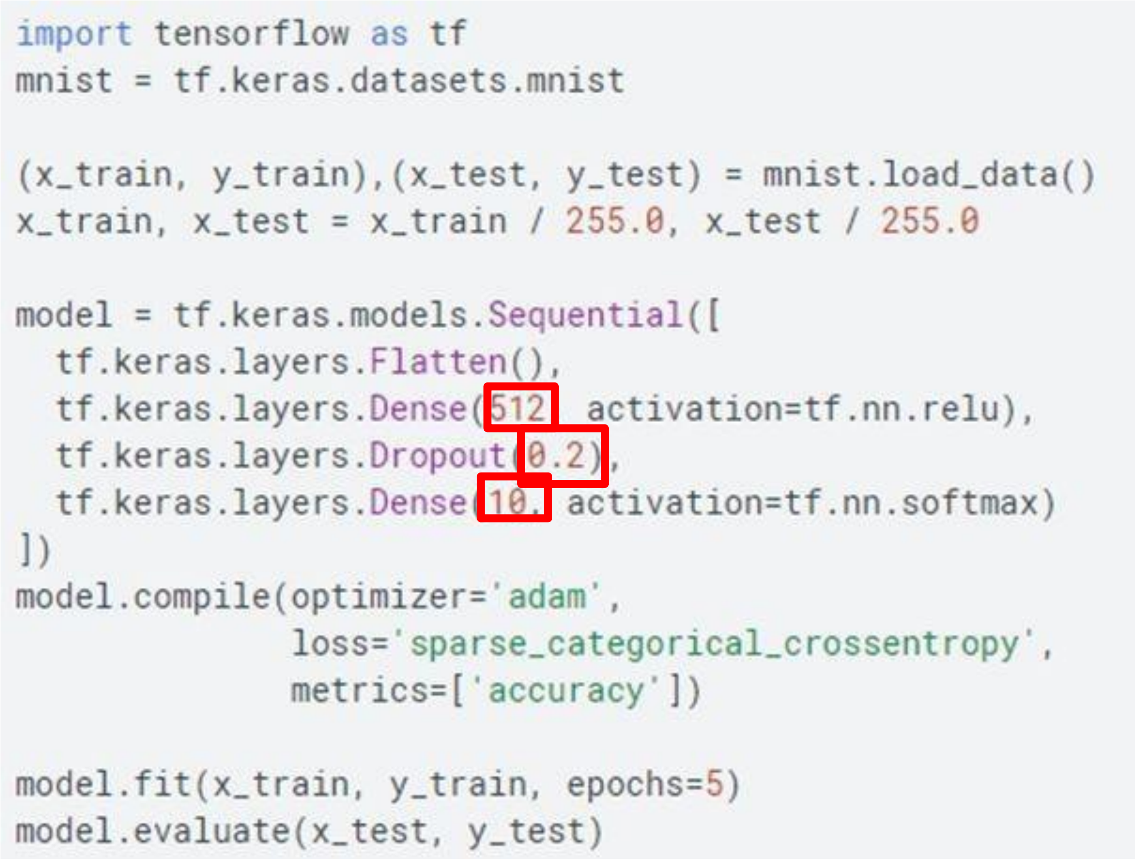
Design your model.

Is this model effective? Why this model?

What is the **input** and **output?**

Given images, what will be the output of model?

Deeper dive



```
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

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Step1

Call the **necessarypackage (import)**

Step2

Load the dataset (MNIST is **common** hence simple)

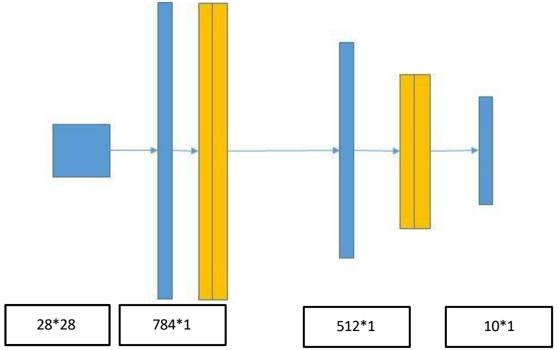
Preprocess data {normalize} → why? Does it matter?

Step 3

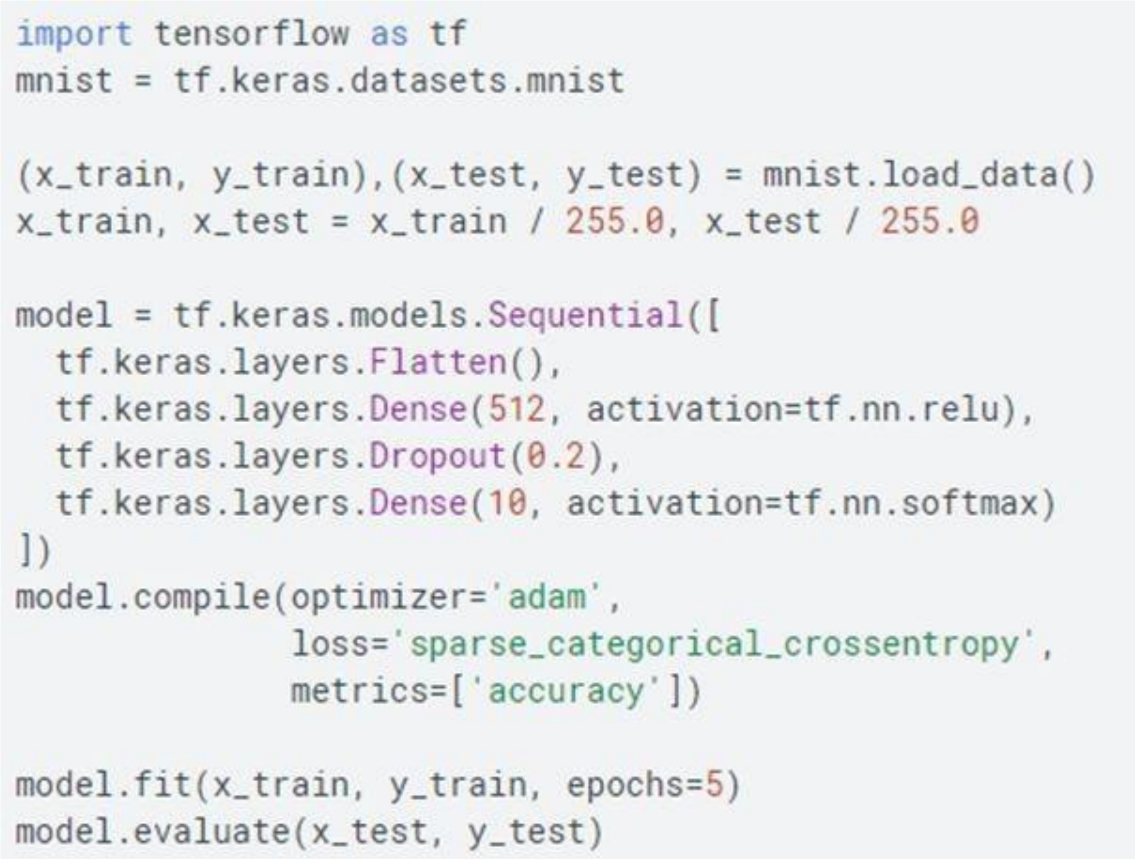
Design your model.

Is this model effective? Why this model?

What is the **input** and **output?**



Deeper dive



```
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

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Step1

Call the **necessarypackage (import)**

Step2

Load the dataset (MNIST is **common** hence simple)

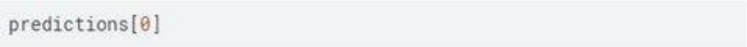
Preprocess data {normalize} → why? Does it matter?

Step 3

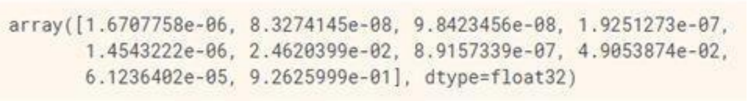
Design your model.

Is this model effective? Why this model?

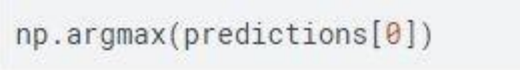
What is the **input** and **output?**



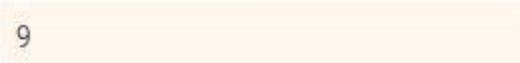
```
predictions[0]
```



```
array([1.6707758e-06, 8.3274145e-08, 9.8423456e-08, 1.9251273e-07,
       1.4543222e-06, 2.4620399e-02, 8.9157339e-07, 4.9053874e-02,
       6.1236402e-05, 9.2625999e-01], dtype=float32)
```

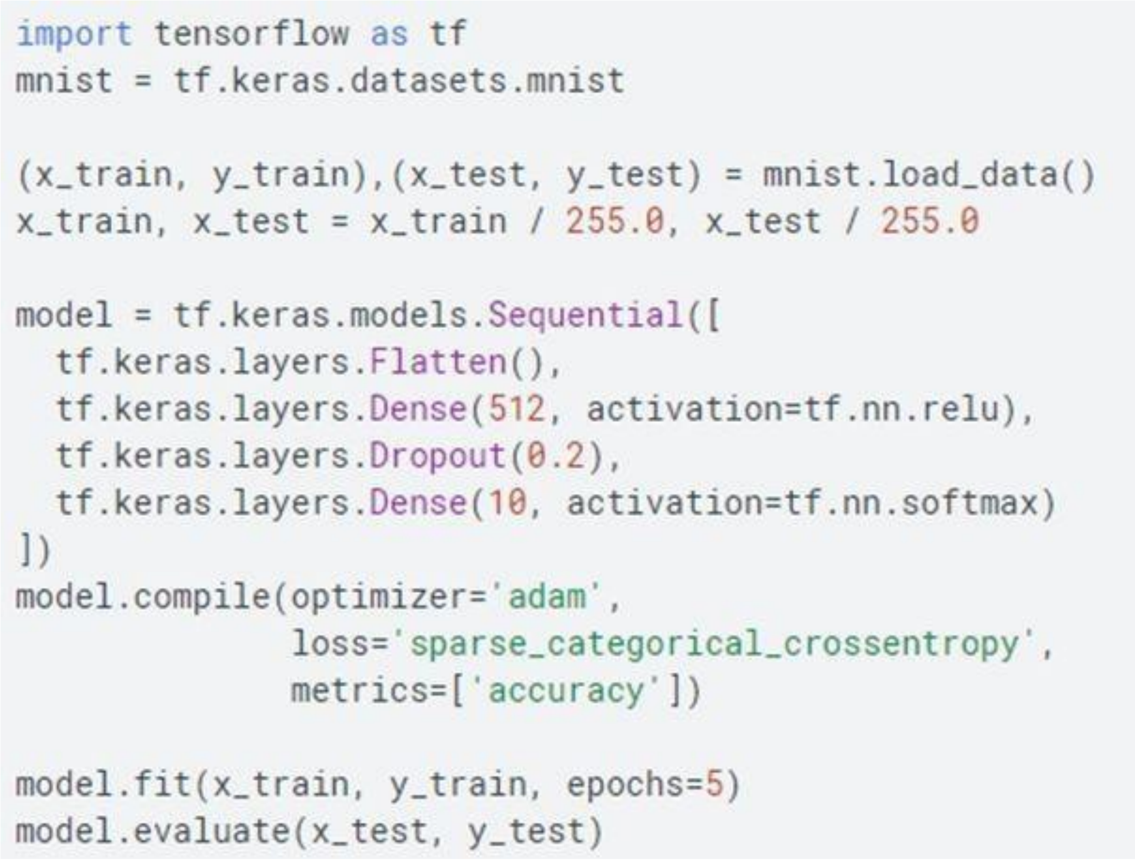


```
np.argmax(predictions[0])
```



```
9
```

Deeper dive



```
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

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Step 1

Call the **necessarypackage (import)**

Step 2

Load the dataset (MNIST is **common** hence simple)

Preprocess data {normalize} → why? Does it matter?

Step 3

Design your model.

Is this model effective? Why this model?

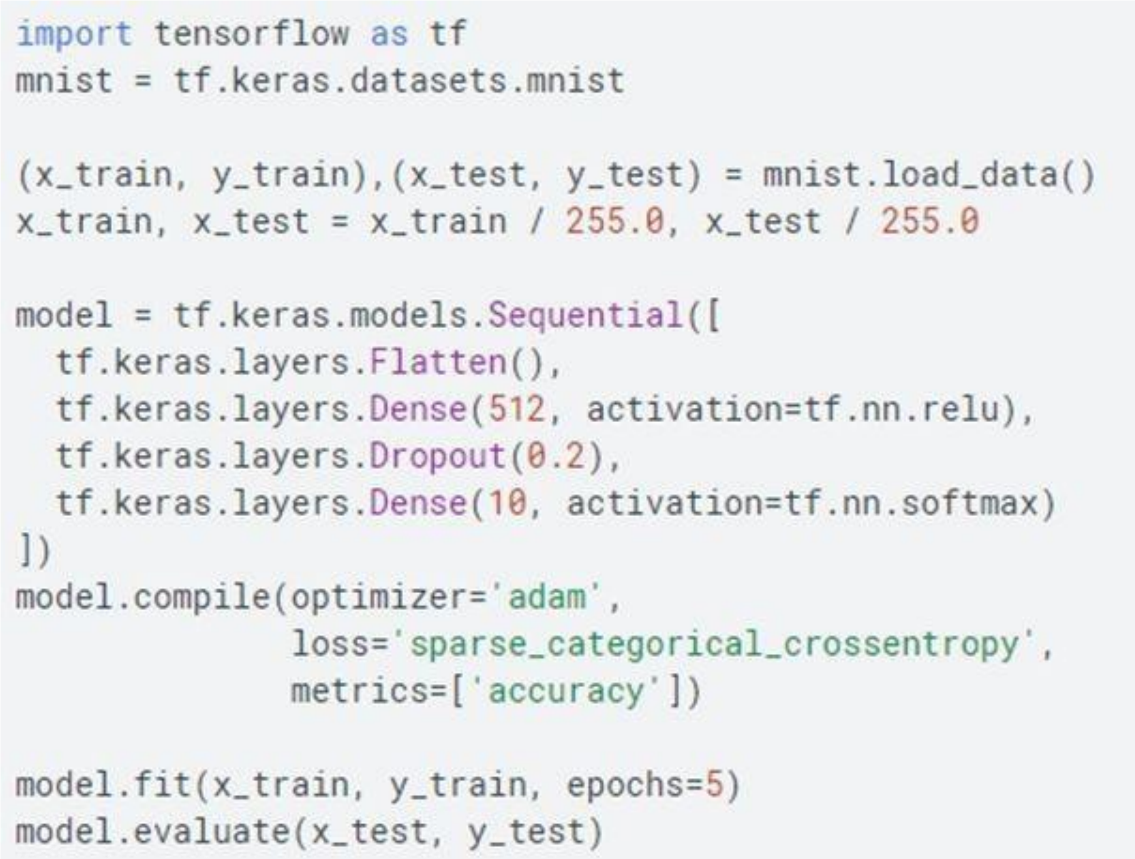
What is the **input** and **output?**

Step 4

Decide on how you want to train the model

Sparse_categorical vs categorical?

Deeper dive



```
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

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Step1

Call the **necessarypackage (import)**

Step2

Load the dataset (MNIST is **common** hence simple)

Preprocess data {normalize} → why? Does it matter?

Step 3

Design your model.

Is this model effective? Why this model?

What is the **input** and **output?**

Step4

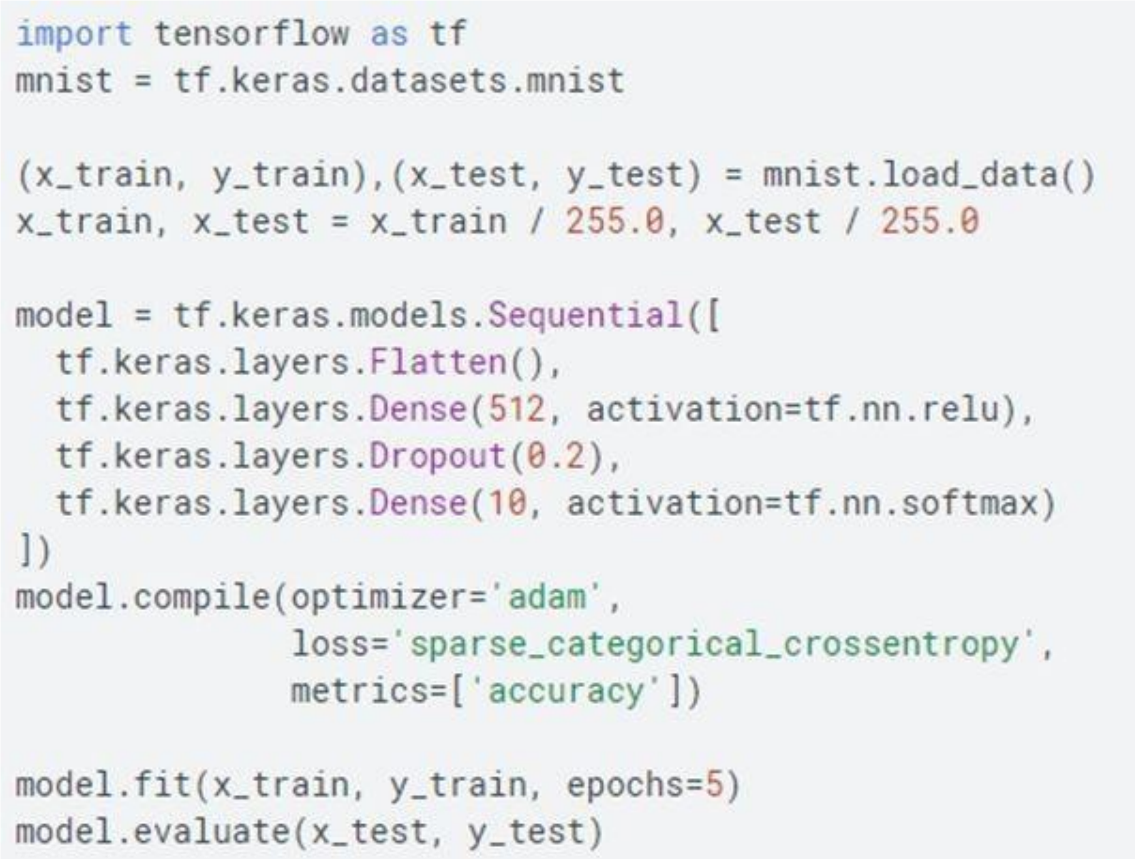
Decide on how you want to train the model

Sparse_categorical vs categorical?

Sparse_categorical , label = [0, 2, ….]

categorical , label = [[1 0 0 0], [0 0 1 0], ….]

Deeper dive



```
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

model = tf.keras.models.Sequential([
  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(512, activation=tf.nn.relu),
  tf.keras.layers.Dropout(0.2),
  tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=5)
model.evaluate(x_test, y_test)
```

Step1

Call the **necessarypackage (import)**

Step2

Load the dataset (MNIST is **common** hence simple)

Preprocess data {normalize} → why? Does it matter?

Step 3

Design your model.

Is this model effective? Why this model?

What is the **input** and **output?**

Step4

Decide on how you want to train the model

Step5

Train & Evaluate on non-training set. (**why?**)

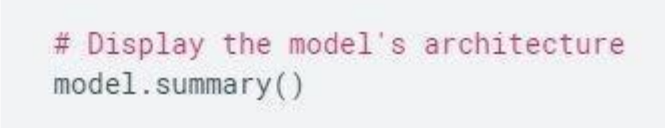
Model

Training is often expensive. After training, what to do?

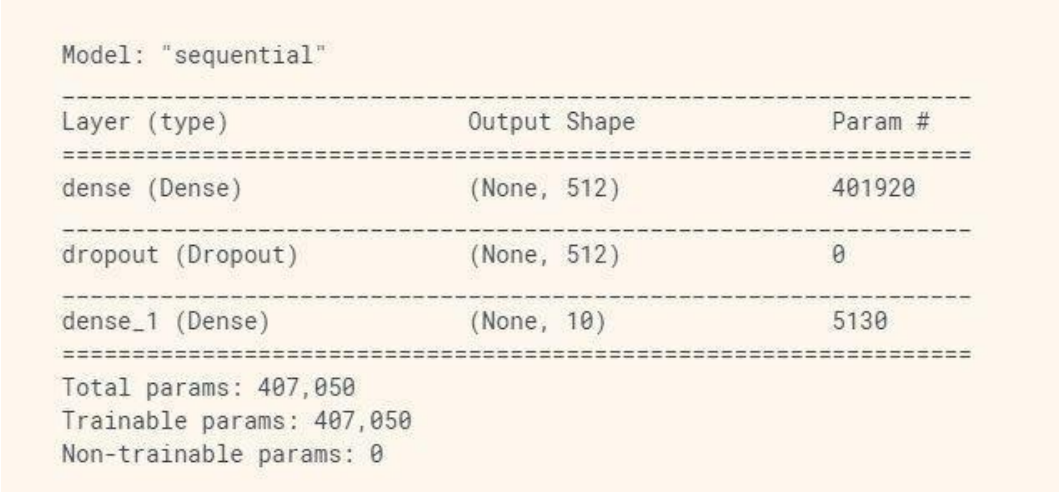
A. Close program & do nothing
B. Save the training weight & model architecture

Model : Check

Debug & Check if model is correctly created:



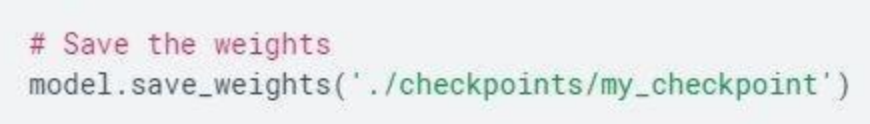
```
# Display the model's architecture
model.summary()
```



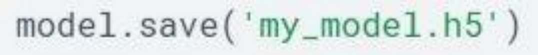
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 512)               401920
_________________________________________________________________
dropout (Dropout)            (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130
=================================================================
Total params: 407,050
Trainable params: 407,050
Non-trainable params: 0
```

Model : Save

What is the difference?



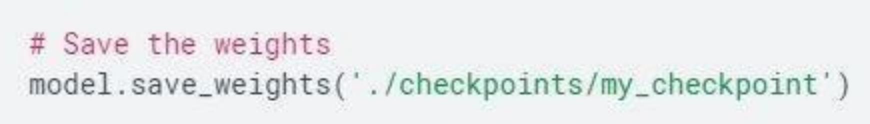
```
# Save the weights
model.save_weights('./checkpoints/my_checkpoint')
```



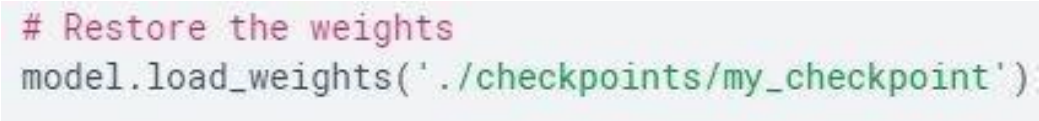
```
model.save('my_model.h5')
```

Model : Save & Load

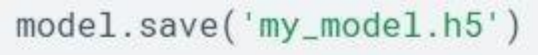
What is the difference?



```
# Save the weights
model.save_weights('./checkpoints/my_checkpoint')
```



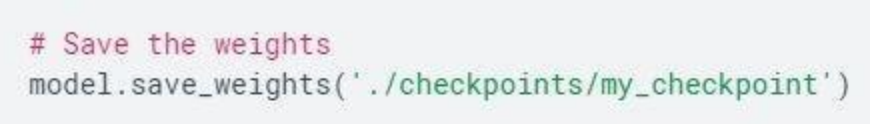
```
# Restore the weights
model.load_weights('./checkpoints/my_checkpoint')
```



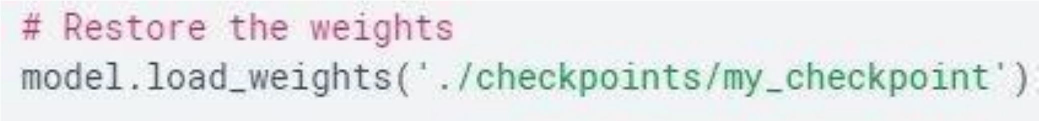
```
model.save('my_model.h5')
```

Model : Save & Load

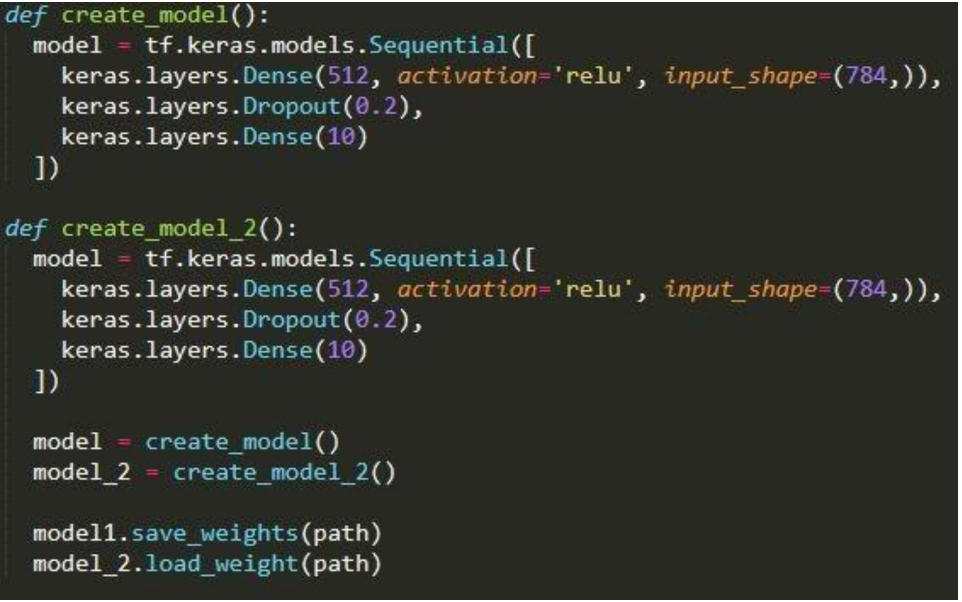
What is the output?



```
# Save the weights
model.save_weights('./checkpoints/my_checkpoint')
```



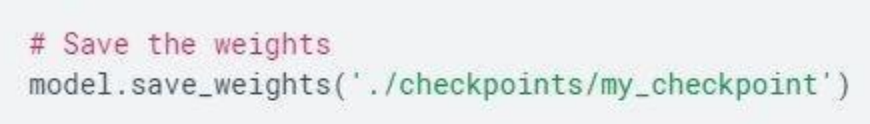
```
# Restore the weights
model.load_weights('./checkpoints/my_checkpoint')
```



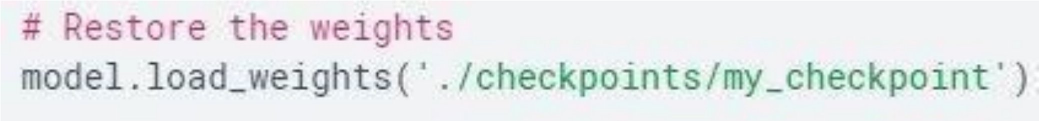
```
def create_model():
  model = tf.keras.models.Sequential([
    keras.layers.Dense(512, activation='relu', input_shape=(784,)),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(10)
  ])

def create_model_2():
  model = tf.keras.models.Sequential([
    keras.layers.Dense(512, activation='relu', input_shape=(784,)),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(10)
  ])

  model = create_model()
  model_2 = create_model_2()

  model1.save_weights(path)
  model_2.load_weight(path)
```

Model : Save & Load

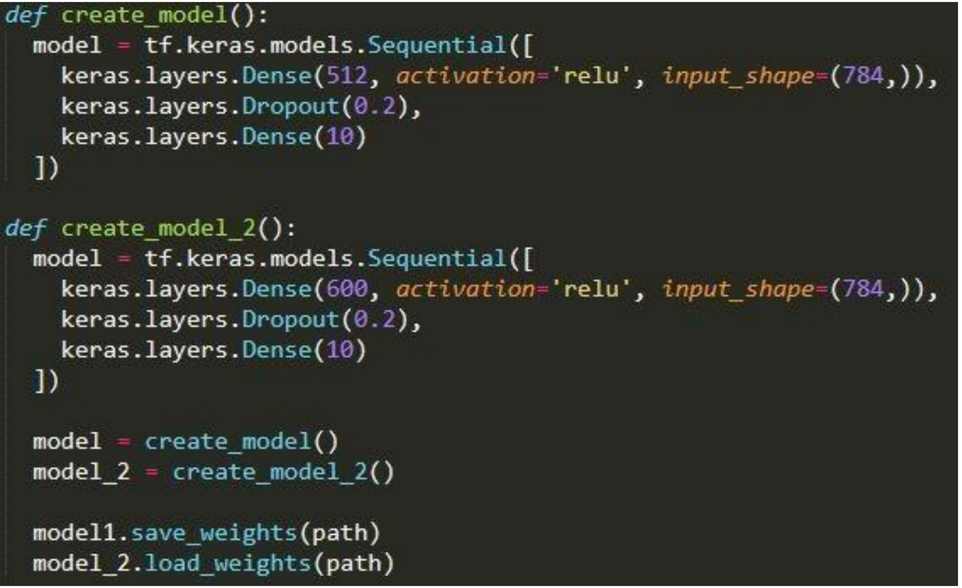
What is the output?



```
# Save the weights
model.save_weights('./checkpoints/my_checkpoint')
```



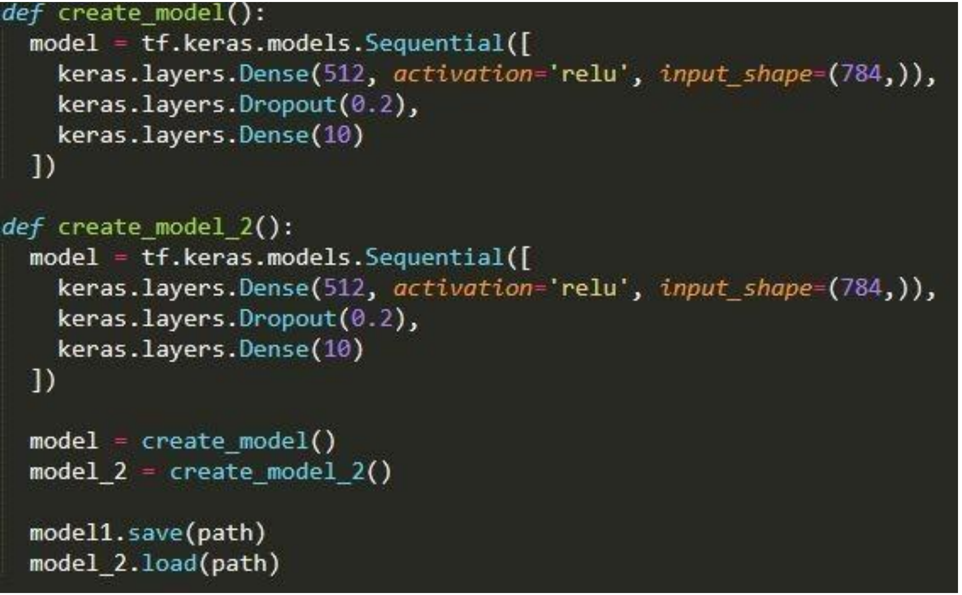
```
# Restore the weights
model.load_weights('./checkpoints/my_checkpoint')
```



```
def create_model():
  model = tf.keras.models.Sequential([
    keras.layers.Dense(512, activation='relu', input_shape=(784,)),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(10)
  ])

def create_model_2():
  model = tf.keras.models.Sequential([
    keras.layers.Dense(600, activation='relu', input_shape=(784,)),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(10)
  ])

  model = create_model()
  model_2 = create_model_2()

  model1.save_weights(path)
  model_2.load_weights(path)
```

Model : Save & Load

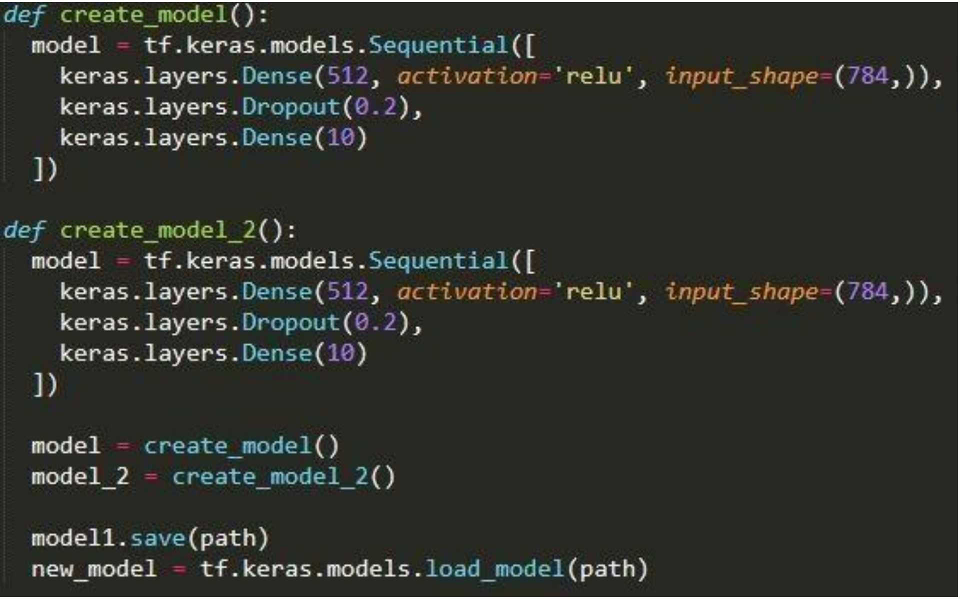
How about this one?



```
def create_model():
  model = tf.keras.models.Sequential([
    keras.layers.Dense(512, activation='relu', input_shape=(784,)),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(10)
  ])

def create_model_2():
  model = tf.keras.models.Sequential([
    keras.layers.Dense(512, activation='relu', input_shape=(784,)),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(10)
  ])

  model = create_model()
  model_2 = create_model_2()

  model1.save(path)
  model_2.load(path)
```

Model : Save & Load

Last one



```
def create_model():
  model = tf.keras.models.Sequential([
    keras.layers.Dense(512, activation='relu', input_shape=(784,)),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(10)
  ])

def create_model_2():
  model = tf.keras.models.Sequential([
    keras.layers.Dense(512, activation='relu', input_shape=(784,)),
    keras.layers.Dropout(0.2),
    keras.layers.Dense(10)
  ])

  model = create_model()
  model_2 = create_model_2()

  model1.save(path)
  new_model = tf.keras.models.load_model(path)
```

Recap

Utilize model saving & loading → checkpoint at every few iters

Try it yourself : Fashion MNIST

Check the jupyter notebook file and try to build your own model → self explanatory

Do not trust the code as is. I modified the code [hint: related to tutorial]

Take knowledge from today’s tutorial and try to build a classifier

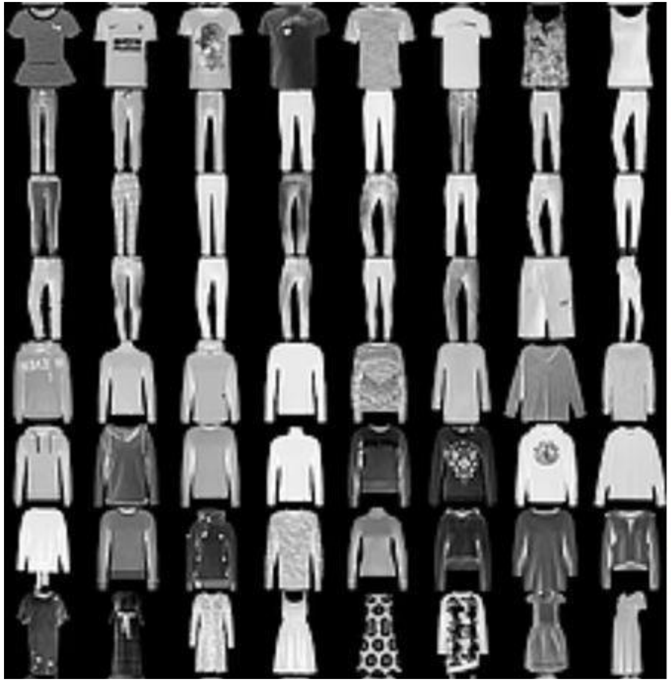
But before that, ……..

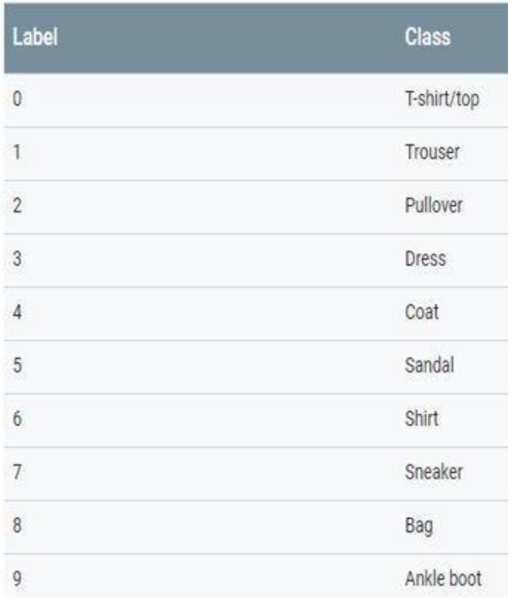
p.s. → might **be easier** to write your own code from scratch

→ solution here

Another simple dataset : Fashion MNIST

It’s just like MNIST but with clothing

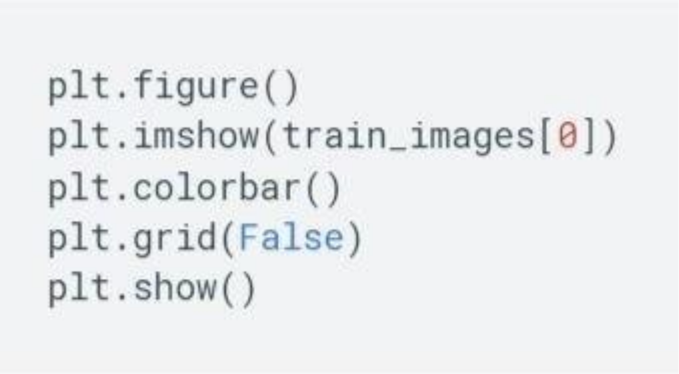




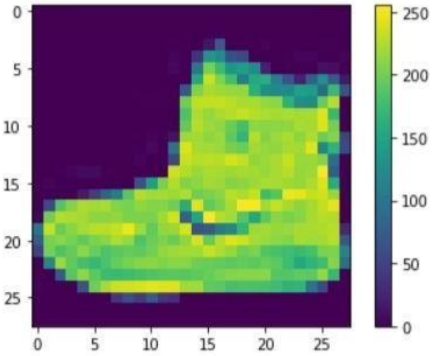
Label	Class
0	T-shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal
6	Shirt
7	Sneaker
8	Bag
9	Ankle boot

Might be helpful for debugging

1. Check input → is it correct?

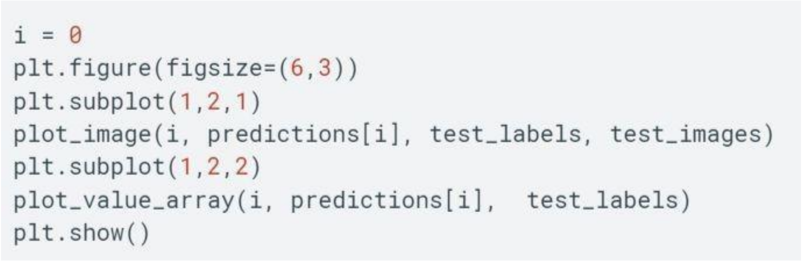


```
plt.figure()
plt.imshow(train_images[0])
plt.colorbar()
plt.grid(False)
plt.show()
```

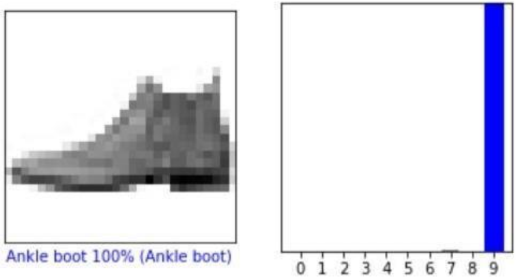


Might be helpful for debugging

1. Check input → is it correct?
2. Verify predictions



```
i = 0
plt.figure(figsize=(6,3))
plt.subplot(1,2,1)
plot_image(i, predictions[i], test_labels, test_images)
plt.subplot(1,2,2)
plot_value_array(i, predictions[i],  test_labels)
plt.show()
```



Have fun!