DEEP LEARNING

BY M.J.PASSLAR

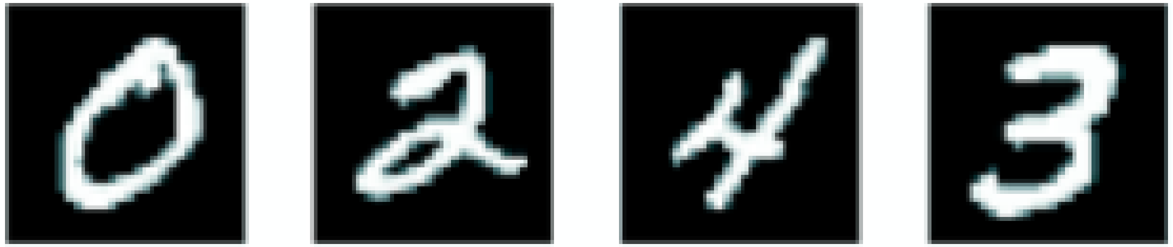
COMPUTERONIC - TEHRAN - IRAN

2023

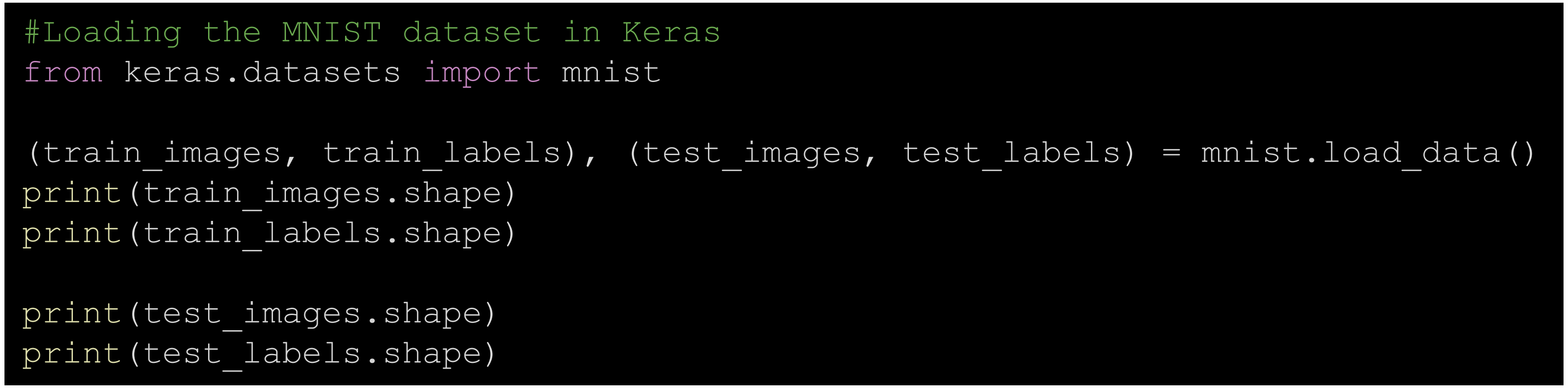
CHAPTER 3 : MNIST

What's MNIST problem ?

- It's a set of 60,000 training images, plus 10,000 test images
- assembled by the National Institute of Standards and Technology (the NIST in MNIST) in the 1980s
- It's a multi, multilable Classification problem type
- So last layer activation -> softmax & loss function -> categorical_crossentropy



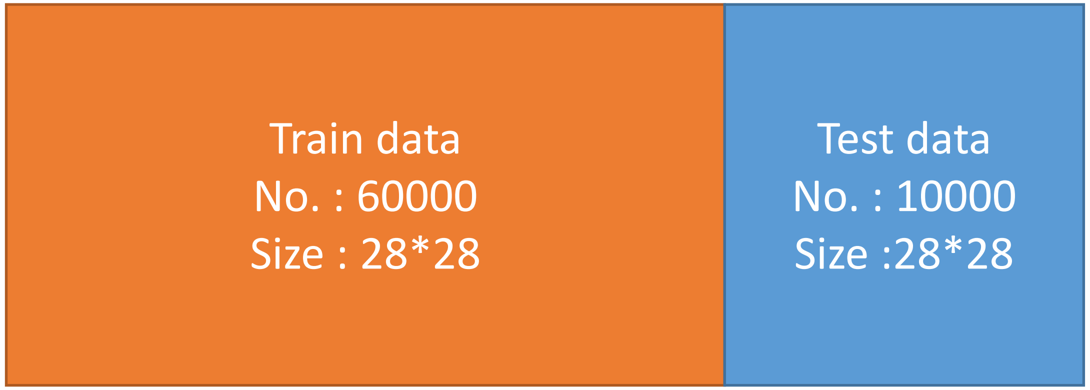
Loading the MNIST dataset in Keras



```
#Loading the MNIST dataset in Keras
from keras.datasets import mnist

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()
print(train_images.shape)
print(train_labels.shape)

print(test_images.shape)
print(test_labels.shape)
```



Train data
No. : 60000
Size : 28*28

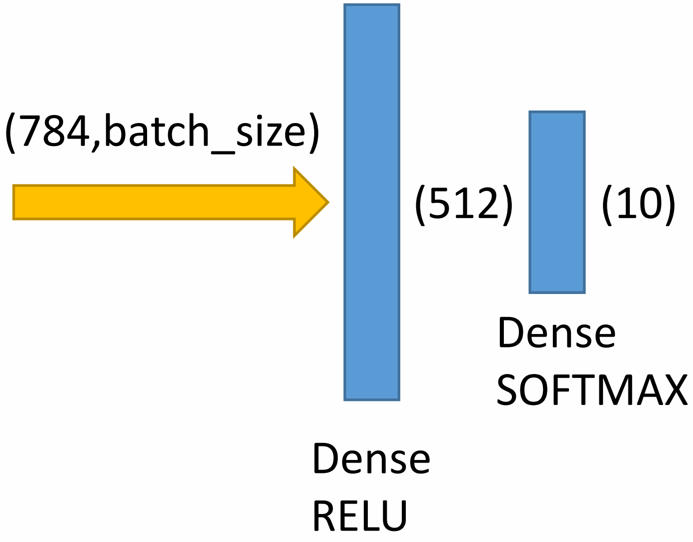
Test data
No. : 10000
Size :28*28

Create and compile your model

```
#The network architecture
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'))

network.summary()

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



(784,batch_size)

(512)

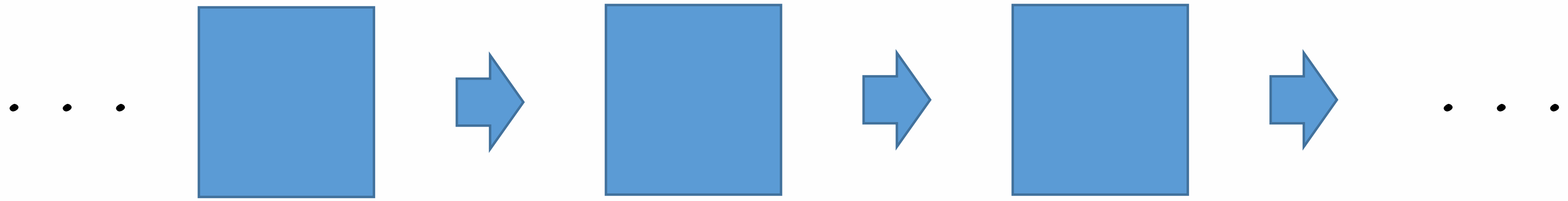
(10)

Dense
SOFTMAX

Dense
RELU

Sequence models

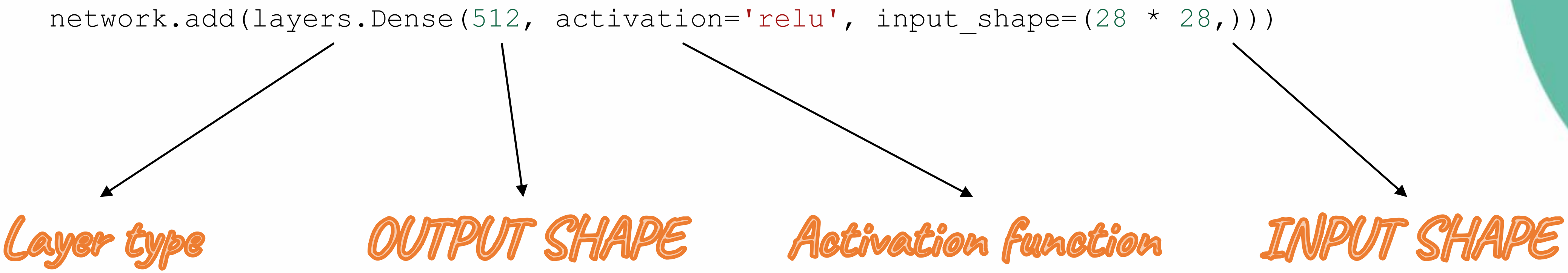
A Sequential model is appropriate for a plain stack of layers where each layer has exactly one input tensor and one output tensor.



A Sequential model is **not appropriate** when:

- Your model has multiple inputs or multiple outputs
- Any of your layers has multiple inputs or multiple outputs
- You need to do layer sharing
- You want non-linear topology (e.g. a residual connection, a multi-branch model)

Adding layers



```
network.add(layers.Dense(512, activation='relu', input_shape=(28 * 28,)))
```

Layer type

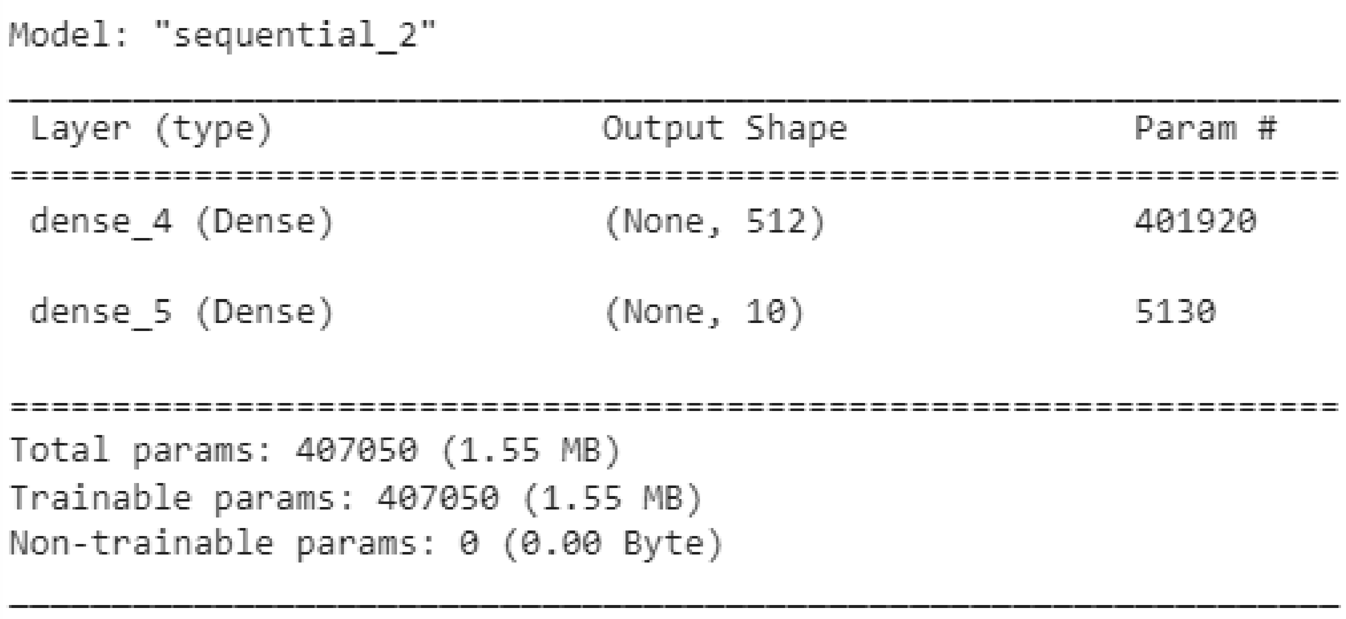
OUTPUT SHAPE

Activation function

INPUT SHAPE

Network summary

```
network.summary()
```



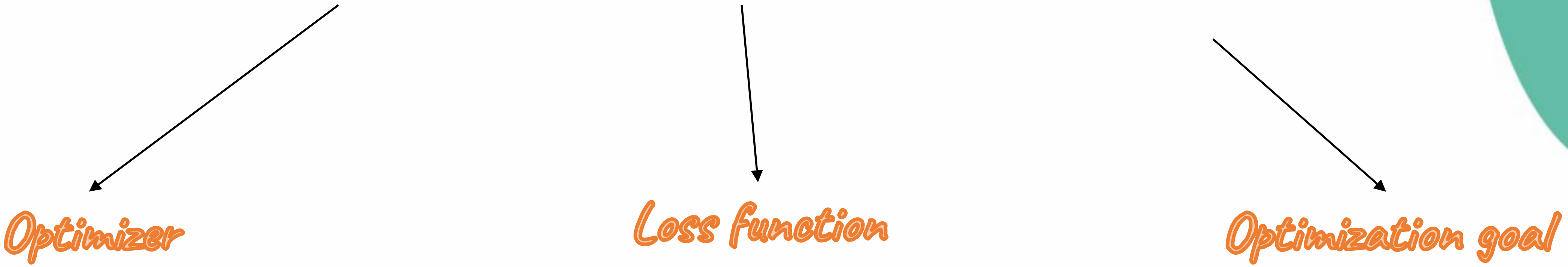
```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense_4 (Dense)             (None, 512)               401920

 dense_5 (Dense)             (None, 10)                5130

=================================================================
Total params: 407050 (1.55 MB)
Trainable params: 407050 (1.55 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Compiling model

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

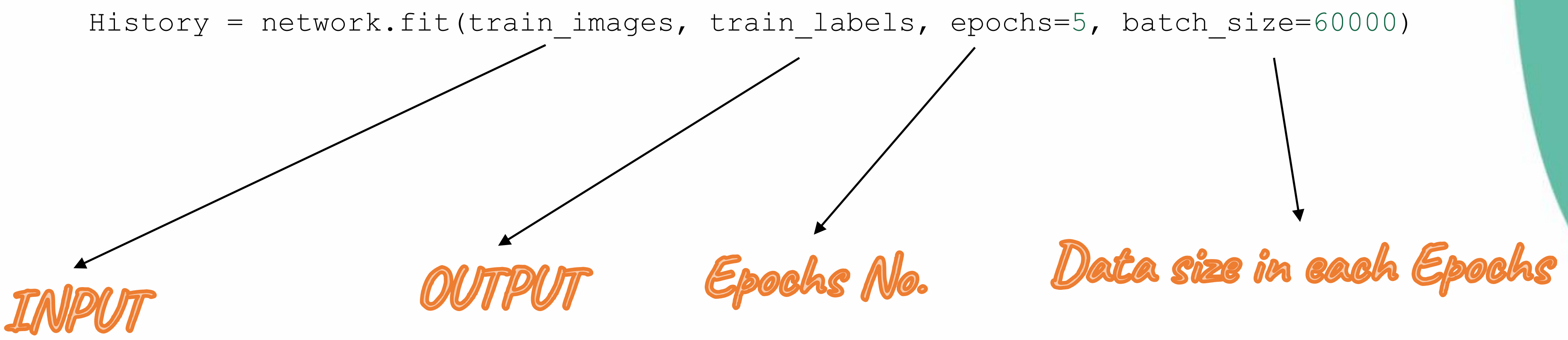


Optimizer

Loss function

Optimization goal

Train your network



```
History = network.fit(train_images, train_labels, epochs=5, batch_size=60000)
```

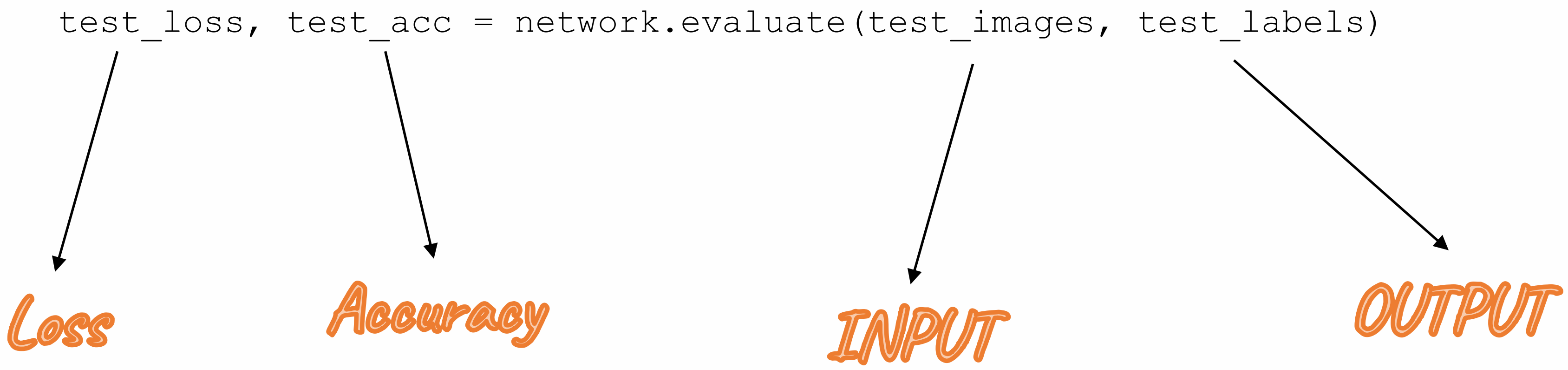
INPUT

OUTPUT

Epochs No.

Data size in each Epochs

Evaluate your model



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

Loss

Accuracy

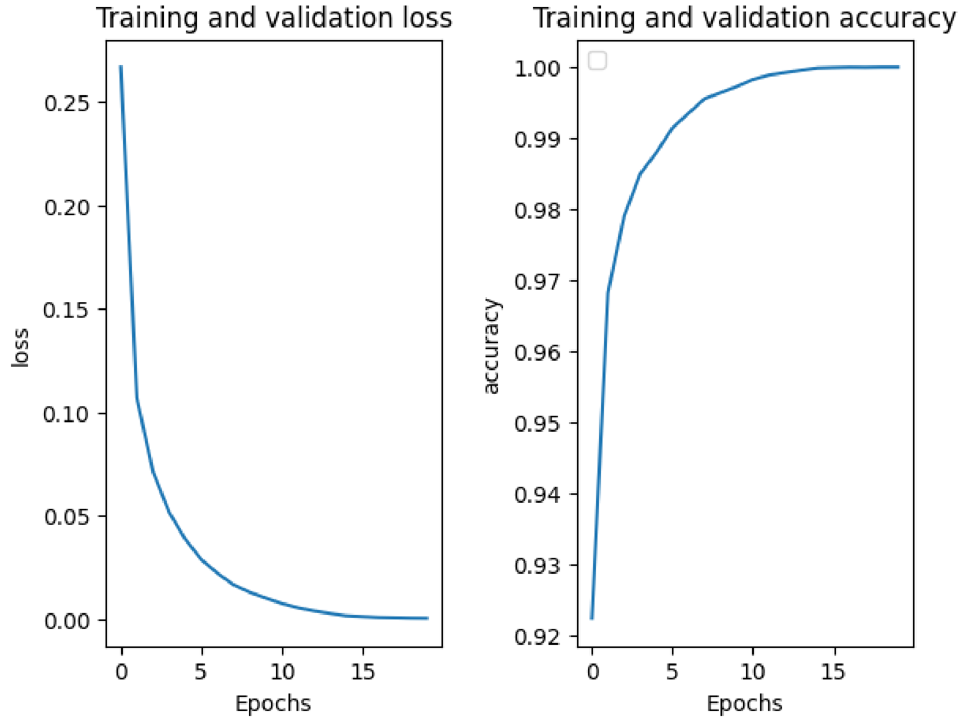
INPUT

OUTPUT

Lets visualize data ...

```
#visiualize data
history_dict = history.history
loss_values = history_dict['loss']
accuracy_values = history_dict['accuracy']

import matplotlib.pyplot as plt
plt.subplots_adjust(left=0.1, bottom=0.1, right=0.9,
top=0.9, wspace=0.4,hspace=0.4)
plt.subplot(1,2,1)
plt.title('Training and validation loss')
plt.plot(loss_values)
plt.xlabel('Epochs')
plt.ylabel('loss')
plt.subplot(1,2,2)
plt.title('Training and validation accuracy')
plt.plot(accuracy_values)
plt.xlabel('Epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()
```



Now it's your turn ...

Parameter	Smaller	Bigger	Result
Meddle layer size			
Epochs number			
Batch size			
Meddle layer activation function			
Add layer before output layer			
New created layer size			