

---

---

# Machine Learning Google Colab Tutorial

TA : 許湛然 (Chan-Jan Hsu)  
2021.03.05

---

---

# Introduction

Colaboratory, or "Colab" for short, allows you to write and execute Python in your browser, with

- Zero configuration required
- Free access to GPUs
- Easy sharing

# Introduction

Colab Demo : <https://reurl.cc/ra63jE>

In this demo, you will learn the following :

- Download files using colab
- Connect google colab with your **google drive**
- Pytorch examples and common errors

# Introduction

You can type python code in the code block, or use a leading exclamation mark ! to change the code block to treating the input as a shell script

```
import numpy  
import math  
...
```

python

```
!ls -l
```

shell script

# Introduction

Exclamation mark (!) starts a new shell, does the operations, and then kills that shell, while percentage (%) affects the process associated with the notebook, and it is called a magic function.

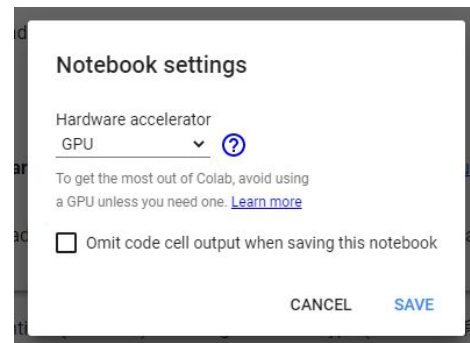
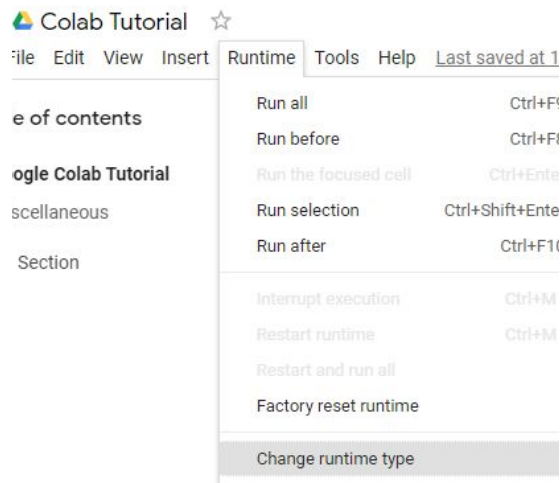
**Use % instead of ! for `cd` (change directory) command**

! shell script  
% magic function

# Changing Runtime

To utilize the free GPU provided by google, click on "Runtime"(執行階段) -> "Change Runtime Type"(變更執行階段類型). There are three options under "Hardware Accelerator"(硬體加速器), select "GPU".

\* Doing this will restart the session, so make sure you change to the desired runtime before executing any code.



# Executing Code Block

Click on the play button to execute the code block. This code downloads a file from google drive

## 1. Download Files via google drive

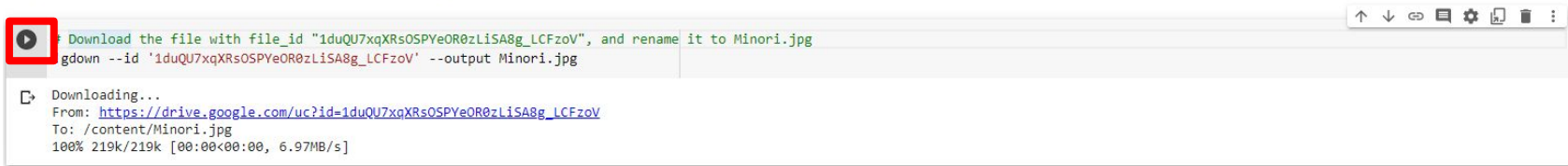
A file stored in Google Drive has the following sharing link :

[https://drive.google.com/open?id=1duQU7xqXRsoSPYeOR0zLiSA8g\\_LCFzoV](https://drive.google.com/open?id=1duQU7xqXRsoSPYeOR0zLiSA8g_LCFzoV)

The random string after "open?id=" is the **file\_id**

[https://drive.google.com/open?id=1duQU7xqXRsoSPYeOR0zLiSA8g\\_LCFzoV](https://drive.google.com/open?id=1duQU7xqXRsoSPYeOR0zLiSA8g_LCFzoV)

It is possible to download the file via Colab knowing the **file\_id**, using the following command.



The image shows a Google Colab code block. On the left, a red square contains a black play button icon. The code block has a light gray header with a toolbar containing icons for undo, redo, link, comment, settings, copy, and delete. The code area contains the following text:

```
Download the file with file_id "1duQU7xqXRsoSPYeOR0zLiSA8g_LCFzoV", and rename it to Minori.jpg  
gdown --id '1duQU7xqXRsoSPYeOR0zLiSA8g_LCFzoV' --output Minori.jpg
```

Below the code, the execution progress is shown:

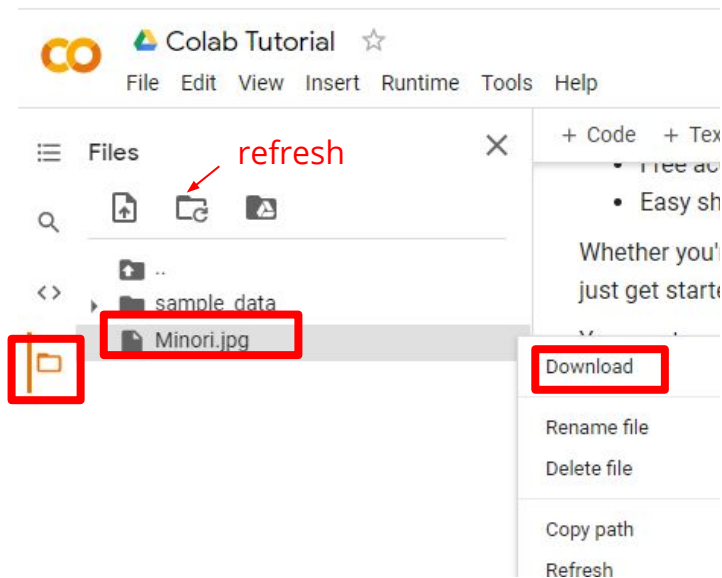
```
↳ Downloading...  
From: https://drive.google.com/uc?id=1duQU7xqXRsoSPYeOR0zLiSA8g\_LCFzoV  
To: /content/Minori.jpg  
100% 219k/219k [00:00<00:00, 6.97MB/s]
```

# File Structure

Clicking on the folder icon will give you the visualization of the file structure

There should be a jpg file, if you do not see it, click the refresh button

The file is temporarily stored, and will be removed once you end your session. You can download the file to your local directory.





# Mounting Google Drive

Execute the code block with `drive.mount('/content/drive')`

or click on the Google Drive icon, a code block will appear



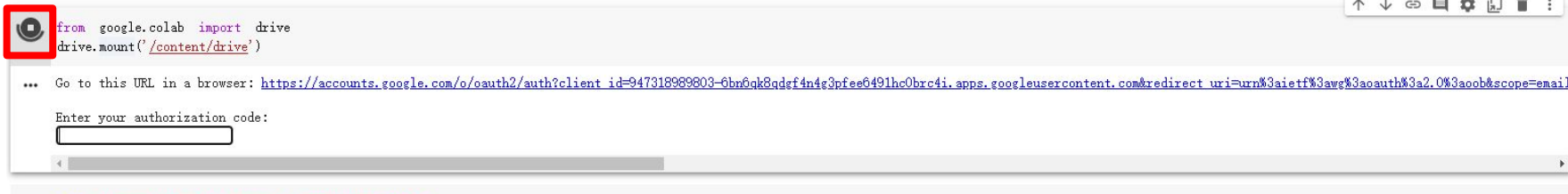
The screenshot shows the Google Colab interface. In the top left, the Google Colab logo and 'Google Colab Tutorial' are visible. Below them is a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. On the left side, there is a 'Files' panel with a search icon and a file explorer. A red box highlights the Google Drive icon in the file explorer. A red arrow points from this icon to a code block in the main editor area. The code block contains the following Python code:

```
[1] # Import a library named google.colab
from google.colab import drive
# mount the content to the directory '/content/drive'
drive.mount('/content/drive', force_remount=True)
```

Below the code block, the output shows 'Mounted at /content/drive'. At the bottom of the code block, there is a red box highlighting the code again.

# Mounting Google Drive

Sign in to your google account to get the authorization code. Enter the authorization code in the box below.



# Mounting Google Drive

Execute the following three code blocks in order

This will download the image to your google drive, so you can access it later

```
%cd /content/drive/MyDrive
#change directory to google drive
!mkdir ML2021 #make a directory named ML2021
%cd ./ML2021
#change directory to ML2021

/content/drive/MyDrive
mkdir: cannot create directory 'ML2021': File exists
/content/drive/MyDrive/ML2021
```

Use bash command pwd to output the current directory

```
!pwd #output the current directory

/content/drive/MyDrive/ML2021
```

Repeat the downloading process, this time, the file will be stored permanently in your goog

```
[ ] # Download the file with file_id "1duQU7xqXRsoSPYeOR0zLiSA8g_LCFzoV", and rena
!gdown --id "1duQU7xqXRsoSPYeOR0zLiSA8g_LCFzoV" --output Minor1.jpg
```



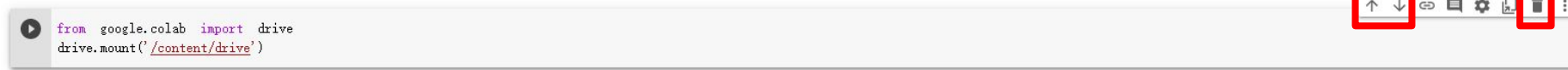
# Moving and Creating a New Code Block

You can create a new code block by clicking on +Code(程式碼) on the top

Move cell up

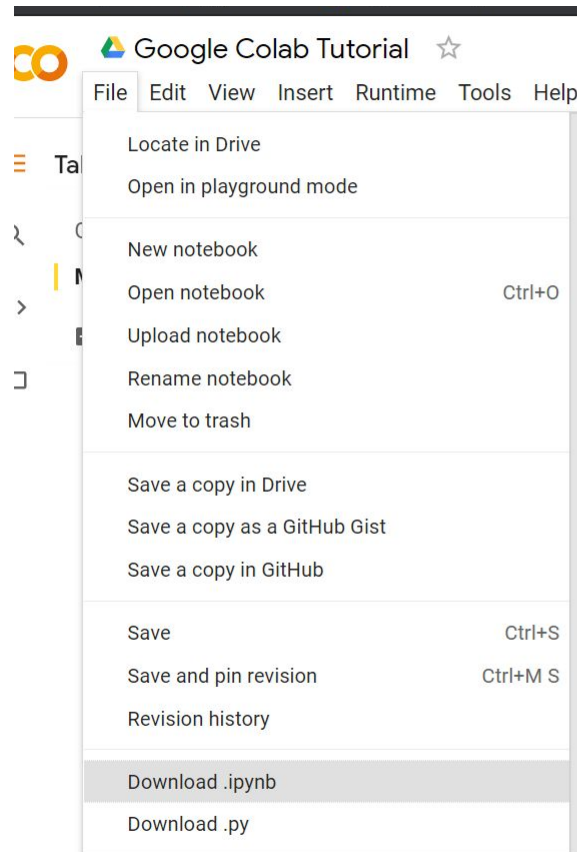
Move cell down

Delete cell



# Saving Colab

You can download the ipynb file to your local device ( File > Download .ipynb), or save the colab notebook to your google drive (File > Save a copy in Drive).



# Recovering Files in Google Drive

Right Click on File > **Manage Versions** (版本管理) to recover old files that have been accidentally overwritten.

## Manage versions

Older versions of 'main.py' may be deleted after 30 days or after 100 versions are stored. To avoid deletion, select **Keep forever** in the file's context menu. [Learn more](#)

UPLOAD NEW VERSION



Current version main.py

Dec 18, 2020, 1:40 AM 許湛然



Version 1 main.py

Dec 17, 2020, 9:21 AM 許湛然



CLOSE

# Useful Linux Commands (in Colab)

**ls** : List all files in the current directory

**ls -l** : List all files in the current directory with more detail

**pwd** : Output the working directory

**mkdir <dirname>** : Create a directory named <dirname>

**cd <dirname>** : Move to directory named <dirname>

**gdown** : Download files from google drive

**wget** : Download files from the internet

**python <python\_file>** : Executes a python file

---

# Machine Learning Pytorch Tutorial 2

## Documentation and Common Errors

---

TA : 許湛然 (Chan-Jan Hsu)  
2021.03.05

---



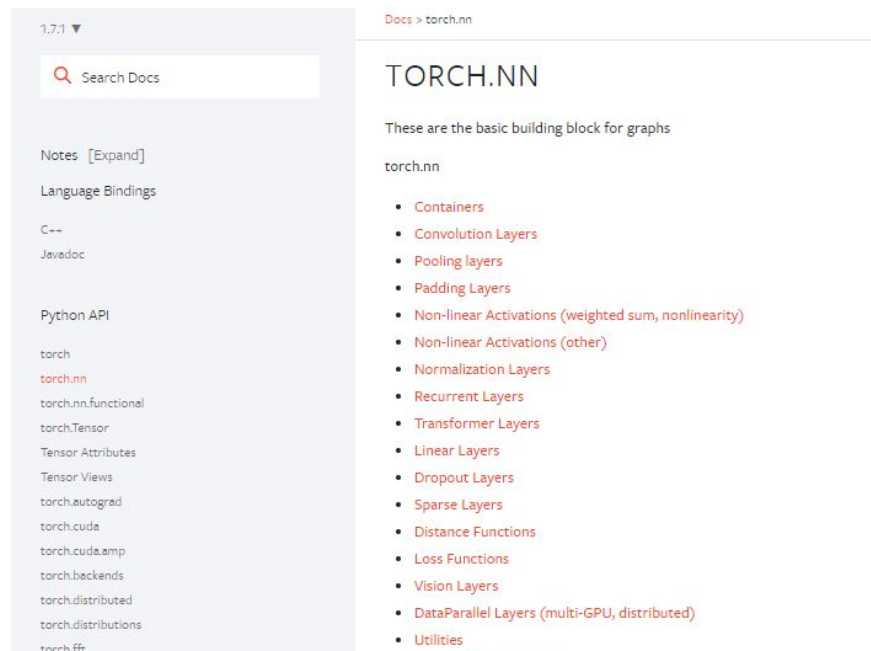
# PyTorch Documentation

<https://pytorch.org/docs/stable/>

torch.nn -> neural network

torch.optim -> optimization algorithms

torch.utils.data -> dataset, dataloader



The screenshot displays the PyTorch documentation interface. On the left, a sidebar lists navigation options: 1.7.1, Search Docs, Notes [Expand], Language Bindings (C++, Javadoc), Python API (torch, torch.nn, torch.nn.functional, torch.Tensor, Tensor Attributes, Tensor Views, torch.autograd, torch.cuda, torch.cuda.amp, torch.backends, torch.distributed, torch.distributions, torch.fft), and torch.nn. The main content area is titled 'TORCH.NN' and includes the text 'These are the basic building block for graphs'. Below this, a list of torch.nn components is shown: Containers, Convolution Layers, Pooling layers, Padding Layers, Non-linear Activations (weighted sum, nonlinearity), Non-linear Activations (other), Normalization Layers, Recurrent Layers, Transformer Layers, Linear Layers, Dropout Layers, Sparse Layers, Distance Functions, Loss Functions, Vision Layers, DataParallel Layers (multi-GPU, distributed), and Utilities.

# PyTorch Documentation Example

function inputs and outputs

data type and explanation  
of each input

## TORCH.MAX

```
torch.max(input) → Tensor
```

Returns the maximum value of all elements in the `input` tensor.

### • WARNING

This function produces deterministic (sub)gradients unlike `max(dim=0)`

### Parameters

**input** (*Tensor*) – the input tensor.

# PyTorch Documentation Example

Some functions behave differently with different inputs

Parameters : You don't need to specify the name of the argument (Positional Arguments)

Keyword Arguments : You have to specify the name of the argument

*They are separated by \**

```
torch.max(input, dim, keepdim=False, *, out=None) -> (Tensor, LongTensor)
```

Returns a namedtuple (values, indices) where values is the maximum value of each row of the input tensor in the given dimension dim. And indices is the index location of each maximum value found (argmax).

If keepdim is True, the output tensors are of the same size as input except in the dimension dim where they are of size 1. Otherwise, dim is squeezed (see torch.squeeze()), resulting in the output tensors having 1 fewer dimension than input.

## • NOTE

If there are multiple maximal values in a reduced row then the indices of the first maximal value are returned.

## Parameters

- **input** (Tensor) – the input tensor.
- **dim** (int) – the dimension to reduce.
- **keepdim** (bool) – whether the output tensor has dim retained or not. Default: False.

## Keyword Arguments

**out** (tuple, optional) – the result tuple of two output tensors (max, max\_indices)

# PyTorch Documentation Example

Some functions behave differently with different inputs

Arguments with default value :  
Some arguments have a default value (keepdim=False), so passing a value of this argument is optional

```
torch.max(input, dim, keepdim=False, *, out=None) -> (Tensor, LongTensor)
```

Returns a namedtuple (values, indices) where values is the maximum value of each row of the input tensor in the given dimension dim. And indices is the index location of each maximum value found (argmax).

If keepdim is True, the output tensors are of the same size as input except in the dimension dim where they are of size 1. Otherwise, dim is squeezed (see `torch.squeeze()`), resulting in the output tensors having 1 fewer dimension than input.

## • NOTE

If there are multiple maximal values in a reduced row then the indices of the first maximal value are returned.

## Parameters

- **input** (*Tensor*) – the input tensor.
- **dim** (*int*) – the dimension to reduce.
- **keepdim** (*bool*) – whether the output tensor has dim retained or not. Default: False.

## Keyword Arguments

**out** (*tuple, optional*) – the result tuple of two output tensors (max, max\_indices)

# PyTorch Documentation Example

## Three Kinds of torch.max

1. `torch.max(input) → Tensor`
2. `torch.max(input, dim, keepdim=False, *, out=None) → (Tensor, LongTensor)`
3. `torch.max(input, other, *, out=None) → Tensor`

`input : Tensor, dim : int, keepdim : bool`  
`other : Tensor`

# PyTorch Documentation Example

**1. `torch.max(input)` → `Tensor`**

Find the maximum value of a tensor, and return that value.

input

[[1 2 3]

[5 6 4]]

# PyTorch Documentation Example

2. `torch.max(input, dim, keepdim=False, *, out=None) → (Tensor, LongTensor)`

Find the maximum value of a tensor along a dimension, and return that value, along with the index corresponding to that value.

input

```
[[1  2  7]
 [5  6  4]]
```

# PyTorch Documentation Example

`3. torch.max(input) → Tensor`

Perform element-wise comparison between two tensors of the same size, and select the maximum of the two to construct a tensor with the same size.

input

[[1	2	3]	[[2	4	6]
[5	6	4]]	[1	3	5]]



# PyTorch Documentation Example (Colab)

## Three Kinds of torch.max

1. `torch.max(input) → Tensor`
2. `torch.max(input, dim, keepdim=False, *, out=None) → (Tensor, LongTensor)`
3. `torch.max(input, other, *, out=None) → Tensor`

`input : Tensor`

`dim : int`

`keepdim : bool`

`other : Tensor`

## Colab code

```
x = torch.randn(4,5)
```

```
y = torch.randn(4,5)
```

```
1. m = torch.max(x)
```

```
2. m, idx = torch.max(x,0) → O
```

```
m, idx = torch.max(input = x,dim=0) → O
```

```
m, idx = torch.max(x,0,False) → O
```

```
m, idx = torch.max(x,0,keepdim=True) → O
```

```
m, idx = torch.max(x,0,False,out=p) → O
```

```
m, idx = torch.max(x,0,False,p) → x
```

*\*out is a keyword argument*

```
m, idx = torch.max(x,True) → x
```

*\*did not specify dim*

```
3. t = torch.max(x,y)
```

# Common Errors -- Tensor on Different Device to Model

```
model = torch.nn.Linear(5,1).to("cuda:0")
```

```
x = torch.Tensor([1,2,3,4,5]).to("cpu")
```

```
y = model(x)
```

Tensor for \* is on CPU, but expected them to be on GPU

=> send the tensor to GPU

```
x = torch.Tensor([1,2,3,4,5]).to("cuda:0")
```

```
y = model(x)
```

```
print(y.shape)
```

# Common Errors -- Mismatched Dimensions

```
x = torch.randn(4,5)
y = torch.randn(5,4)
z = x + y
```

The size of tensor a (5) must match the size of tensor b (4) at non-singleton dimension 1

=> the shape of a tensor is incorrect, use **transpose**, **squeeze**, **unsqueeze** to align the dimensions

```
y = y.transpose(0,1)
z = x + y
print(z.shape)
```

# Common Errors -- Cuda Out of Memory

```
import torch
import torchvision.models as models
resnet18 = models.resnet18().to("cuda:0") # Neural Networks for Image Recognition
data = torch.randn(512,3,244,244) # Create fake data (512 images)
out = resnet18(data.to("cuda:0")) # Use Data as Input and Feed to Model
print(out.shape)
```

CUDA out of memory. Tried to allocate 350.00 MiB (GPU 0; 14.76 GiB total capacity; 11.94 GiB already allocated; 123.75 MiB free; 13.71 GiB reserved in total by PyTorch)

=> The batch size of data is too large to fit in the GPU. Reduce the batch size.

# Common Errors -- Cuda Out of Memory

If the data is iterated (batch size = 1), the problem will be solved. You can also use DataLoader

```
for d in data:  
    out = resnet18(d.to("cuda:0").unsqueeze(0))  
    print(out.shape)
```

# Common Errors -- Mismatched Tensor Type

```
import torch.nn as nn
L = nn.CrossEntropyLoss()
outs = torch.randn(5,5)
labels = torch.Tensor([1,2,3,4,0])
lossval = L(outs,labels) # Calculate CrossEntropyLoss between outs and labels
```

expected scalar type Long but found Float

=> labels must be long tensors, cast it to type "Long" to fix this issue

```
labels = labels.long()
lossval = L(outs,labels)
print(lossval)
```