

# How To Run CUDA C/C++ on Jupyter notebook in Google Colaboratory

Difficulty Level : Easy • Last Updated : 03 Feb, 2022

## What is CUDA?

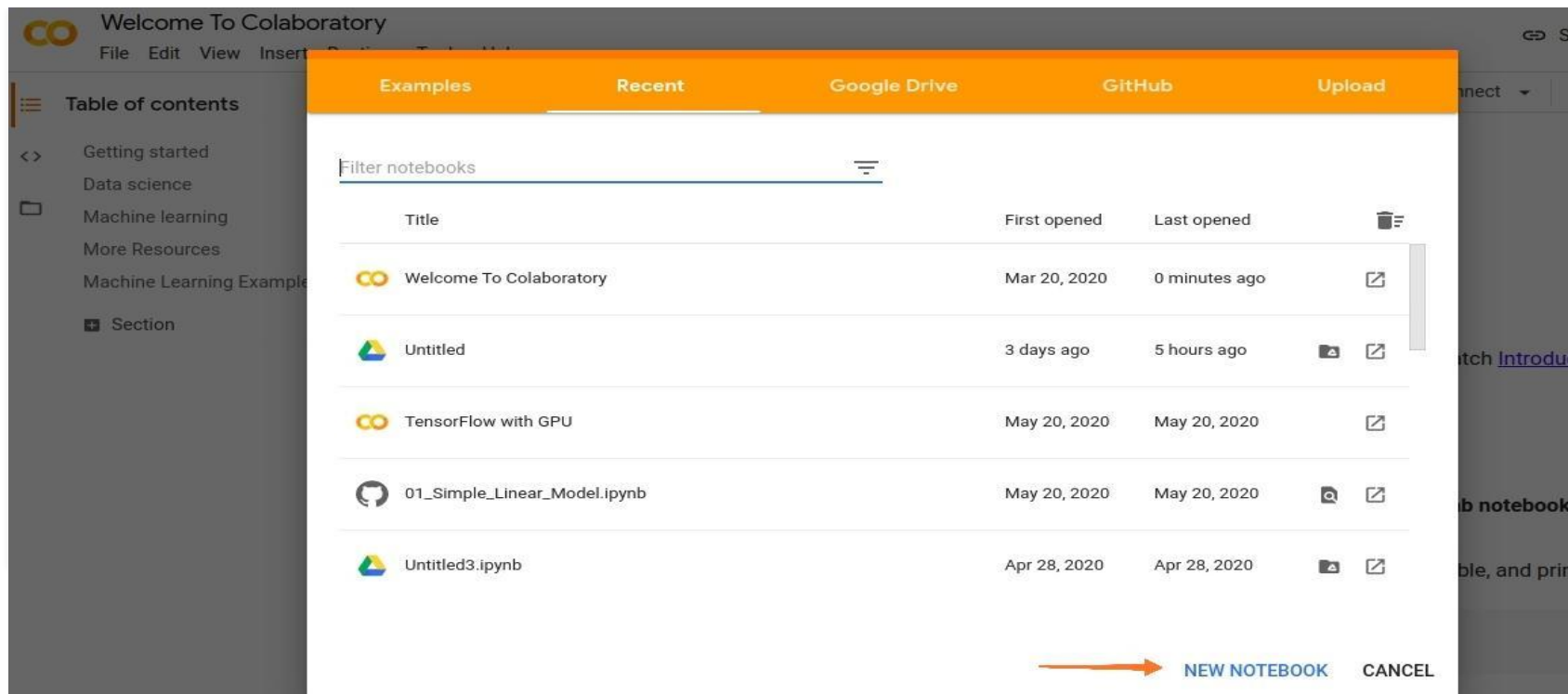
CUDA is a model created by Nvidia for parallel computing platform and application programming interface. CUDA is the parallel computing architecture of NVIDIA which allows for dramatic increases in computing performance by harnessing the power of the GPU.

## What is Google Colab?

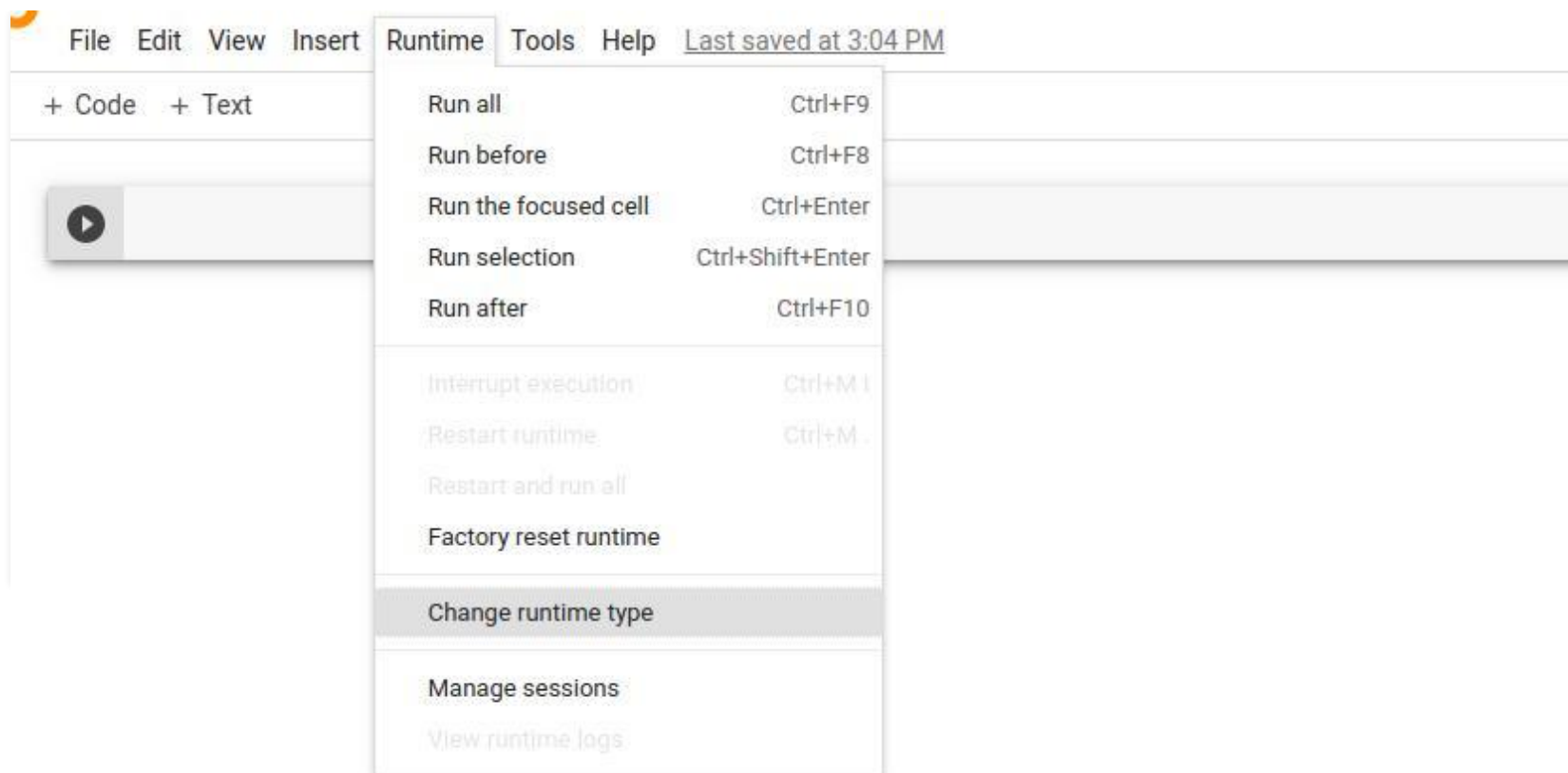
Google Colab is a free cloud service and the most important feature able to distinguish Colab from other free cloud services is; Colab offers GPU and is completely free! With Colab you can work on the GPU with CUDA C/C++ for free! CUDA code will not run on AMD CPU or Intel HD graphics unless you have NVIDIA hardware inside your machine. On Colab you can take advantage of Nvidia GPU as well as being a fully functional Jupyter Notebook with pre-installed Tensorflow and some other ML/DL tools.

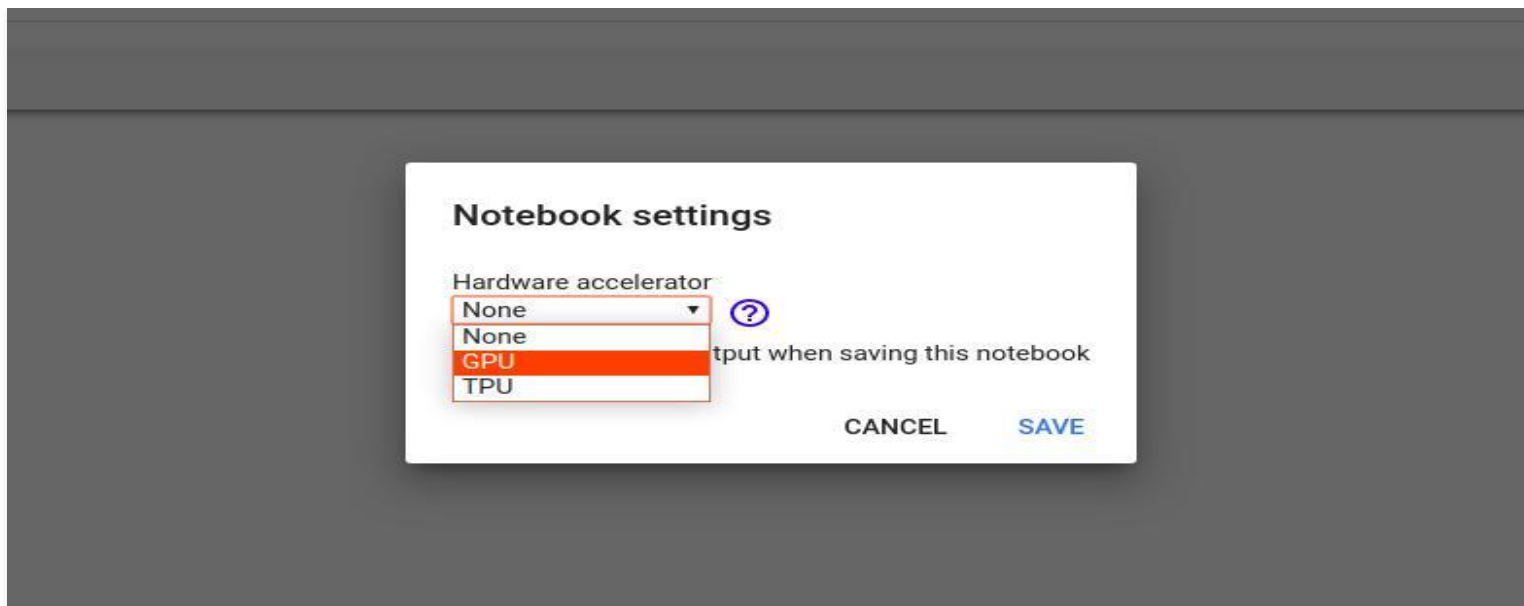
I guess, I'm done with the introduction. Let's configure our learning environment.

**Step 1: Go to <https://colab.research.google.com> in Browser and Click on New Notebook.**



**Step 2: We need to switch our runtime from CPU to GPU. Click on Runtime > Change runtime type > Hardware Accelerator > GPU > Save.**





**Step 3: Completely uninstall any previous CUDA versions. We need to refresh the Cloud Instance of CUDA.**

```
!apt-get --purge remove cuda nvidia* libnvidia-*  
!dpkg -l | grep cuda- | awk '{print $2}' | xargs -n1 dpkg --purge  
!apt-get remove cuda-*  
!apt autoremove  
!apt-get update
```

Write code in a separate code Block and Run that code. Every line that starts with '!', it will be executed as a command line command.

#### **Step 4: Install CUDA Version 9 (You can just copy it in separate code block).**

```
!wget https://developer.nvidia.com/compute/cuda/9.2/Prod/local_installers/cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64.deb
!dpkg -i cuda-repo-ubuntu1604-9-2-local_9.2.88-1_amd64.deb
!apt-key add /var/cuda-repo-9-2-local/7fa2af80.pub
!apt-get update
!apt-get install cuda-9.2
```

#### **Step 5: Now you can check your CUDA installation by running the command given below :**

```
!nvcc --version
```

Output will be something like this:

```
vcc: NVIDIA (R) Cuda compiler driver  
Copyright (c) 2005-2018 NVIDIA Corporation  
Built on Wed_Apr_11_23:16:29_CDT_2018  
Cuda compilation tools, release 9.2, V9.2.88
```

**Step 6: Run the given command to install a small extension to run nvcc from the Notebook cells.**

```
!pip install git+git://github.com/andreinechaev/nvcc4jupyter.git
```

**Step 7: Load the extension using the code given below:**

```
%load_ext nvcc_plugin
```





**Step 8: Execute the code given below to check if CUDA is working or not.**

Now we are ready to run CUDA C/C++ code right in your Notebook.

*Important Note: To check the following code is working or not, write that code in a separate code block and Run that only again when you update the code and re running it.*

To run the code in your notebook, add the %%cu extension at the beginning of your code.

CPP

 %%cu  
 #include <iostream>  
 int  
main()  
 {  
std::cout << "Welcome To GeeksforGeeks\n";  
return 0;  
}

Output:



Welcome To GeeksforGeeks

I suggests you to try program of find maximum element from vector to check that everything works properly.

CPP



%%cu



```
#include <cstdio>
```

```
#include <iostream>
```

```
using namespace std;
```

```
__global__ void maxi(int* a, int* b, int n)  
{
```

```
    int block = 256 * blockIdx.x;
```

```
    int max = 0;
```

```
    for (int i = block; i < min(256 + block, n); i++) {
```

```
        if (max < a[i]) {
```

```
            max = a[i];
```

```
        }
```

```
    }
```

```
    b[blockIdx.x] = max;
```

```
}
```

```
int main()
```

```
{
```

```

int n;
n = 3 >> 2;
int a[n];

for (int i = 0; i < n; i++) {
    a[i] = rand() % n;
    cout << a[i] << "\t";
}

cudaEvent_t start, end;
int *ad, *bd;
int size = n * sizeof(int);
cudaMalloc(&ad, size);
cudaMemcpy(ad, a, size, cudaMemcpyHostToDevice);
int grids = ceil(n * 1.0f / 256.0f);
cudaMalloc(&bd, grids * sizeof(int));

dim3 grid(grids, 1);
dim3 block(1, 1);

cudaEventCreate(&start);
cudaEventCreate(&end);
cudaEventRecord(start);

while (n > 1) {
    maxi<<<grids, block>>>(ad, bd, n);
    n = ceil(n * 1.0f / 256.0f);
    cudaMemcpy(ad, bd, n * sizeof(int), cudaMemcpyDeviceToDevice);
}

cudaEventRecord(end);
cudaEventSynchronize(end);

float time = 0;
cudaEventElapsedTime(&time, start, end);

```

```
int ans[2];
cudaMemcpy(ans, ad, 4, cudaMemcpyDeviceToHost);

cout << "The maximum element is : " << ans[0] << endl;

cout << "The time required : ";
cout << time << endl;
}
```

Output:

```
The maximum element is : 1338278816
The time required : 0.003392
```