Unit – V

Neural Networks and Deep Learning

Introduction to Artificial Neural Networks with Keras:

Artificial Neural Networks:

“Artificial Neural Networks or ANN is an information processing paradigm that is inspired by the way the biological nervous system such as brain process information. It is composed of large number of highly interconnected processing elements(neurons) working in unison to solve a specific problem.”

Keras :

Deep learning is one of the major subfield of machine learning framework. Machine learning is the study of design of algorithms, inspired from the model of human brain. Deep learning is becoming more popular in data science fields like robotics, artificial intelligence(AI), audio & video recognition and image recognition. Artificial neural network is the core of deep learning methodologies. Deep learning is supported by various libraries such as Theano, TensorFlow, Caffe, Mxnet etc., Keras is one of the most powerful and easy to use python library, which is built on top of popular deep learning libraries like TensorFlow, Theano, etc., for creating deep learning models.

Implementing MLPs with Keras

1. Multi-layer perception is the basic type of algorithm used in deep learning it is also known as an artificial neural network and they are the most useful type of neural network.
2. They are connected to multiple layers in a directed graph a perceptron is a single neuron model that was a precursor to large neural Nets it is a field of study that investigates how simple models of the biological brain can be used to solve different computational tasks.
3. The goal is to create robust algorithms and data structures that can be used to model difficult problems MLP utilizes a supervised learning technique called backpropagation.

4. MLP is used to solve regression and classification problems we use MLP in speech and image processing computer vision time series prediction and machine translation.

5. In MLP neurons are arranged into networks a row of neurons is called a layer and one network can have multiple layers any network model starts with an input layer that takes input from the data set layers after the input layer are called hidden layers.

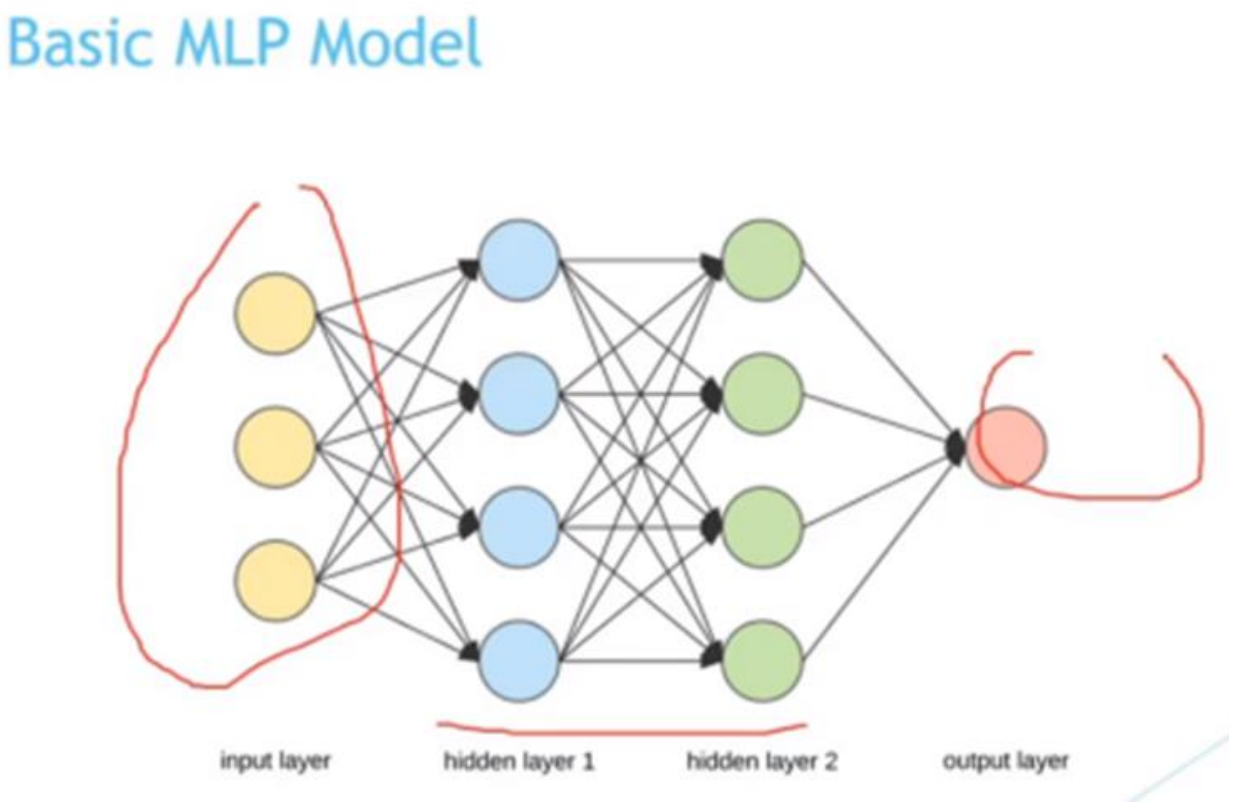


Figure-1

1. In the above example, we have two hidden layers that are not directly exposed to the input the planning can refer to have many hidden layers in our network model usually if we increase the number of hidden layers the model will work more efficiently.

2. The final layer is called the output layer and it is responsible for the final outcome the choice of activation function in the output layer strongly depends on the type of problem.

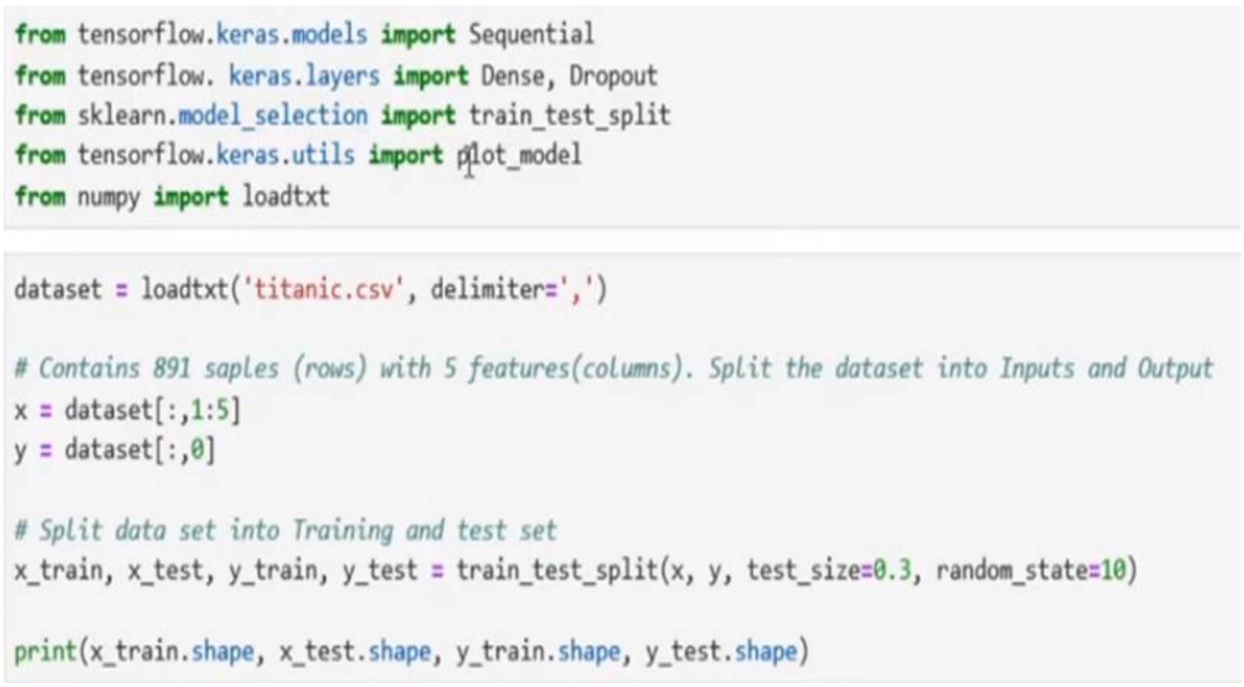
3. If we are modeling a regression problem may have a single output neuron and the neuron may have no activation function a binary classification

problem may have a single output neuron and use a sigmoid activation function to output a value between zero and one

4. A multi-class classification problem may have multiple neurons in the final output layer one for each class in this scenario softmax activation function may be used to output our probability of the network.

DATASET

Data is comma-separated it has a total of 891 samples of five variables first one is survival column zero means dead one means survived this is our target variable next is passenger class so we have three classes 1 2 & 3 then 6 after that age and the final one is fair these four are our predictors using predictors we will predict the class survival which contains 0 & 1.



```
from tensorflow.keras.models import Sequential
from tensorflow. keras.layers import Dense, Dropout
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import plot_model
from numpy import loadtxt
```

```
dataset = loadtxt('titanic.csv', delimiter=',')

# Contains 891 saples (rows) with 5 features(columns). Split the dataset into Inputs and Output
x = dataset[:,1:5]
y = dataset[:,0]

# Split data set into Training and test set
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=10)

print(x_train.shape, x_test.shape, y_train.shape, y_test.shape)
```

Figure-2

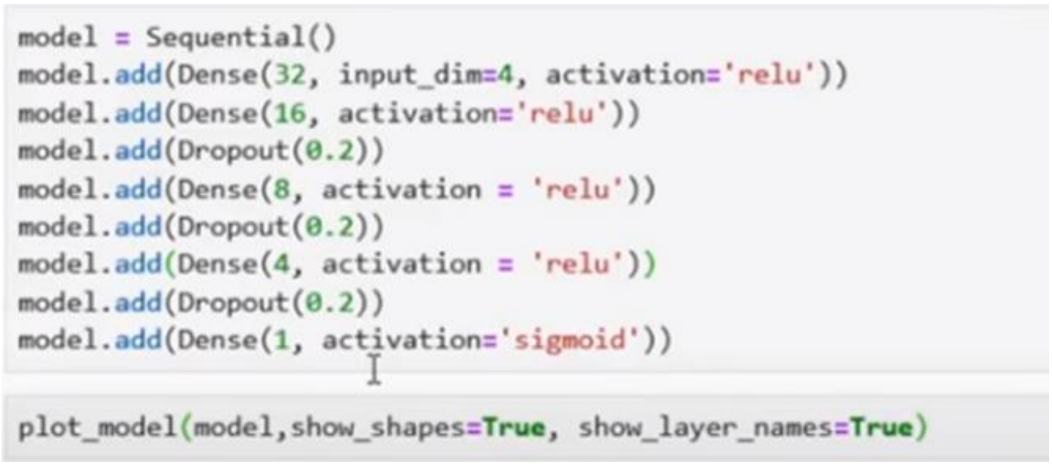
1. Import the data set from our working directory first we split this data set into inputs and the output this X contains the four columns second to fifth as inputs and y contains the output that survival column next we split our data set into training and test set using this training test speed function we select this size as 0.3.

2. 70% data will be in the training set and 30% will be in the test set this final line will display the shape of the training and test set so let’s execute

it in our inputs we have 4 variables so in the model the input dimension will be four.

3. Now we construct a sequential model with dense and dropout layers first we construct a dense layer with 32 neurons as this is the first layer we have to specify the input dimension which is 4.

4. So in the first hidden layer there will be 4 inputs and 32 outputs we use RELU as our activation function the next one is another dense layer with 16 neurons then dropout layer with 0.2 dropout is a technique used to prevent a model from overfitting this dropout will use 20% inputs at the time of model.



```
model = Sequential()
model.add(Dense(32, input_dim=4, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(8, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(4, activation = 'relu'))
model.add(Dropout(0.2))
model.add(Dense(1, activation='sigmoid'))
```

```
plot_model(model,show_shapes=True, show_layer_names=True)
```

Figure-3

1. Finally, we have a dense output layer with the activation function sigmoid as our target variable contains only zero and one sigmoid

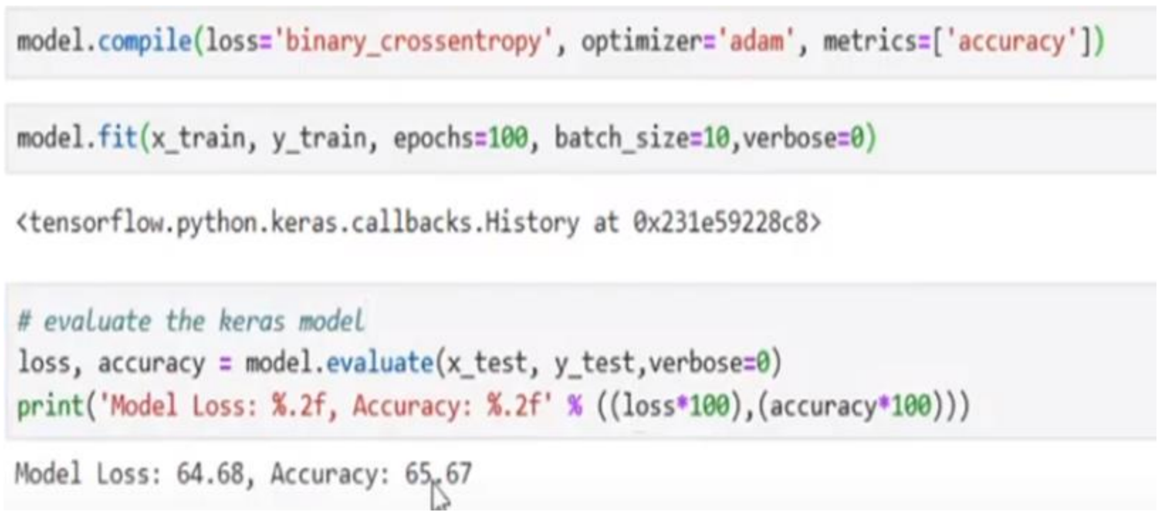
2. Then we compile our model as this is a binary classification we will use binary cross-entropy as a loss function we set Adam as optimizer it calculates an exponential moving average of the gradient and the square gradient

3. Parameters control the decay rates of the moving averages we are also using accuracy as a metric function let's compile it.

4. it is time to train our model with training data with a batch size of 10 as our training set contains 623 samples there will be 63 batches of ten samples

5. Finally model evaluation we will evaluate our model using test data set this evaluation function will return the loss and accuracy of the model this

final line will display them in percentage let's run it so our model accuracy is 65.67.



```
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
model.fit(x_train, y_train, epochs=100, batch_size=10,verbose=0)
```

```
<tensorflow.python.keras.callbacks.History at 0x231e59228c8>
```

```
# evaluate the keras model
loss, accuracy = model.evaluate(x_test, y_test,verbose=0)
print('Model Loss: %.2f, Accuracy: %.2f' % ((loss*100),(accuracy*100)))
```

```
Model Loss: 64.68, Accuracy: 65.67
```

Introduction to TensorFlow

TensorFlow is an open-source software library for machine learning, created by Google. It was initially released on November 28, 2015, and it's now used across many fields including research in the sciences and engineering.

The idea behind TensorFlow is to make it quick and simple to train deep neural networks that use a diversity of mathematical models. These networks are then able to learn from data without human intervention or supervision, making them more efficient than conventional methods. The library also offers support for processing on multiple machines simultaneously with different operating systems and GPUs.

TensorFlow applications

TensorFlow is a library for deep learning built by Google, it's been gaining a lot of traction ever since its introduction early last year. The main features include automatic differentiation, convolutional neural networks (CNN), and recurrent neural networks (RNN). It's written in C++ and Python, for high performance it uses a server called a "Cloud TensorFlow" that runs on Google Cloud Platform. It doesn't require a GPU, which is one of its main features.

The newest release of Tensorflow also supports data visualization through matplotlib. This visualization library is very popular, and it's often used in data science coursework, as well as by artists and engineers to do data visualizations using MATLAB or Python / R / etc.

Installing TensorFlow 2:

Windows

Prerequisite

- Python 3.6–3.8
- Windows 7 or later (with C++ redistributable)
- Check https://www.tensorflow.org/install/so… For latest version information

Steps

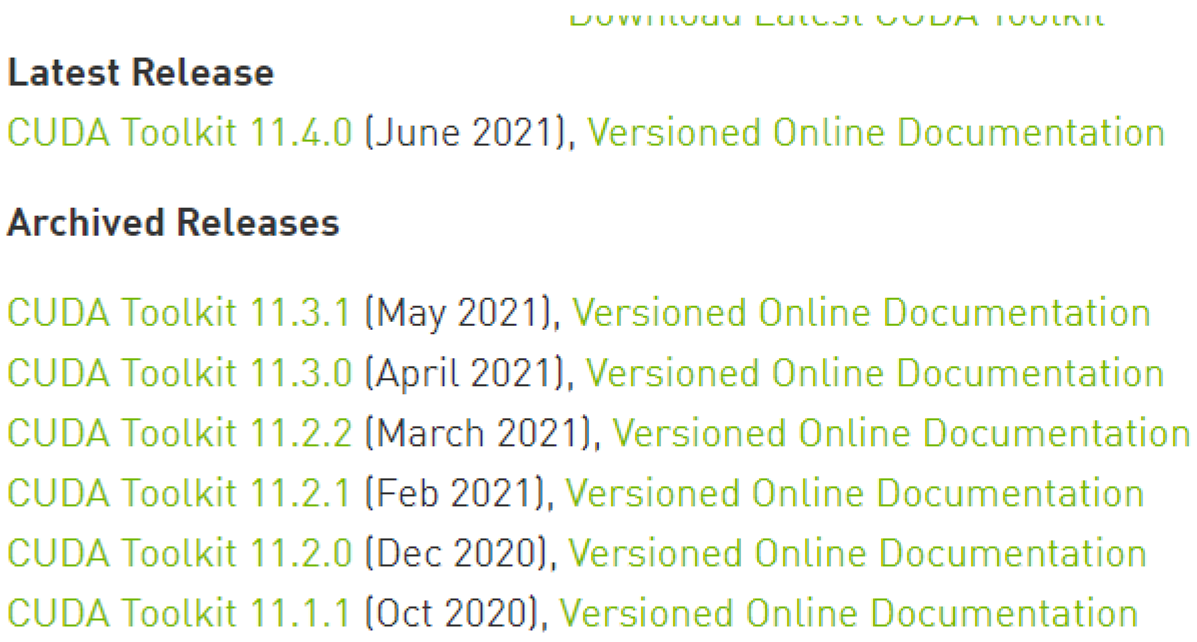
1) Download Microsoft Visual Studio from:

https://visualstudio.microsoft.com/vs…

2) Install the NVIDIA CUDA Toolkit (https://developer.nvidia.com/cuda-too…), check the version of software and hardware requirements, we'll be using :

Version	**Python version**	**Compiler**	**Build tools**	**cuDNN**	**CUDA**
tensorflow-2.5.0	3.6-3.9	GCC 7.3.1	Bazel 3.7.2	8.1	11.2

We will install CUDA version 11.2, but make sure you install the latest or updated version (for example – 11.2.2 if it's available).



Download Latest CUDA Toolkit

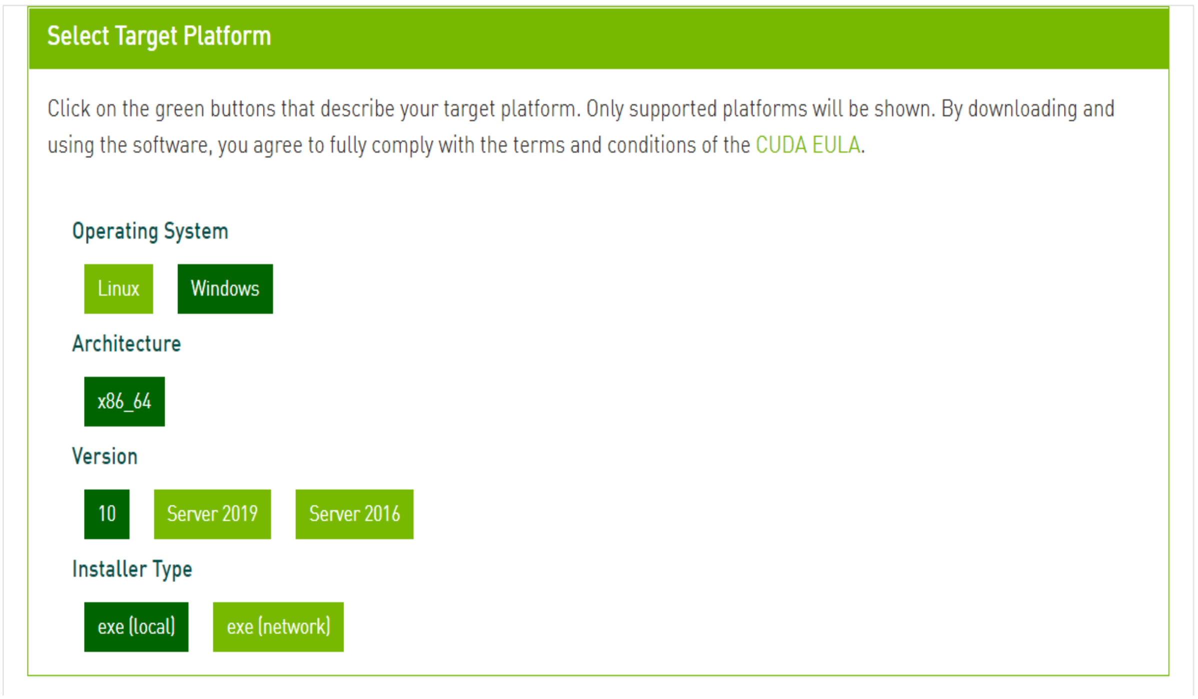
Latest Release

CUDA Toolkit 11.4.0 (June 2021), Versioned Online Documentation

Archived Releases

CUDA Toolkit 11.3.1 (May 2021), Versioned Online Documentation
CUDA Toolkit 11.3.0 (April 2021), Versioned Online Documentation
CUDA Toolkit 11.2.2 (March 2021), Versioned Online Documentation
CUDA Toolkit 11.2.1 (Feb 2021), Versioned Online Documentation
CUDA Toolkit 11.2.0 (Dec 2020), Versioned Online Documentation
CUDA Toolkit 11.1.1 (Oct 2020), Versioned Online Documentation

Click on the newest version and a screen will pop up, where you can choose from a few options, so follow the below image and choose these options for Windows.



Select Target Platform

Click on the green buttons that describe your target platform. Only supported platforms will be shown. By downloading and using the software, you agree to fully comply with the terms and conditions of the CUDA EULA.

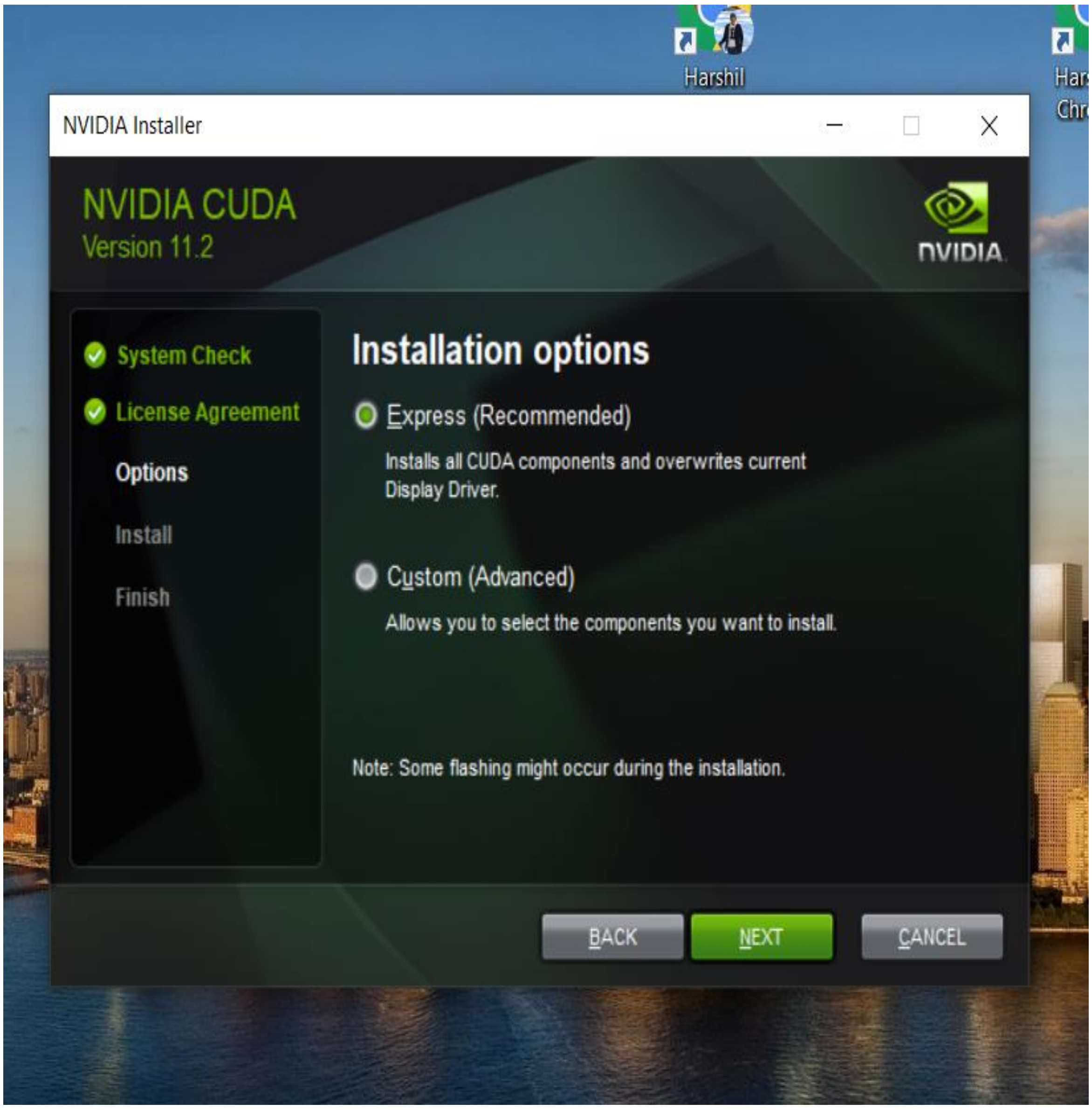
Operating System
Linux [✓ Windows]

Architecture
[✓ x86_64]

Version
[✓ 10] Server 2019 Server 2016

Installer Type
[✓ exe (local)] exe (network)

Once you choose the above options, wait for the download to complete.


NVIDIA Installer

NVIDIA CUDA
Version 11.2

System Check
License Agreement
Options
Install
Finish

Installation options

[✓ Express (Recommended)]
Installs all CUDA components and overwrites current Display Driver.

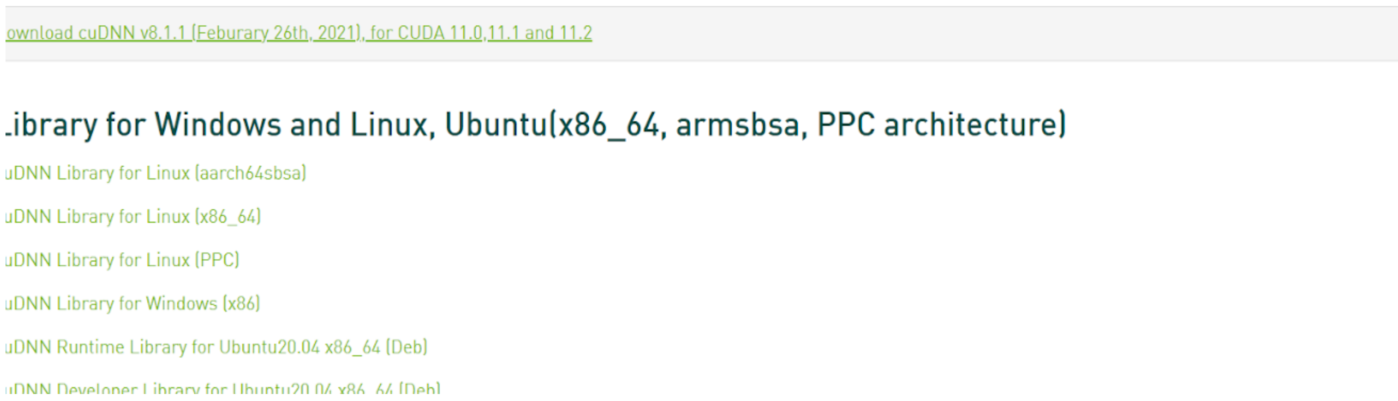
Custom (Advanced)
Allows you to select the components you want to install.

Note: Some flashing might occur during the installation.

Install it with the Express (Recommended) option, it will take a while to install on your machine.

3) **Now we'll download NVIDIA cuDNN, https://developer.nvidia.com/cudnn**

Check the version code from the TensorFlow site.


ownload cuDNN v8.1.1 (Feburary 26th, 2021), for CUDA 11.0,11.1 and 11.2

.ibrary for Windows and Linux, Ubuntu(x86_64, armsbsa, PPC architecture)

uDNN Library for Linux (aarch64sbsa)

uDNN Library for Linux (x86_64)

uDNN Library for Linux (PPC)

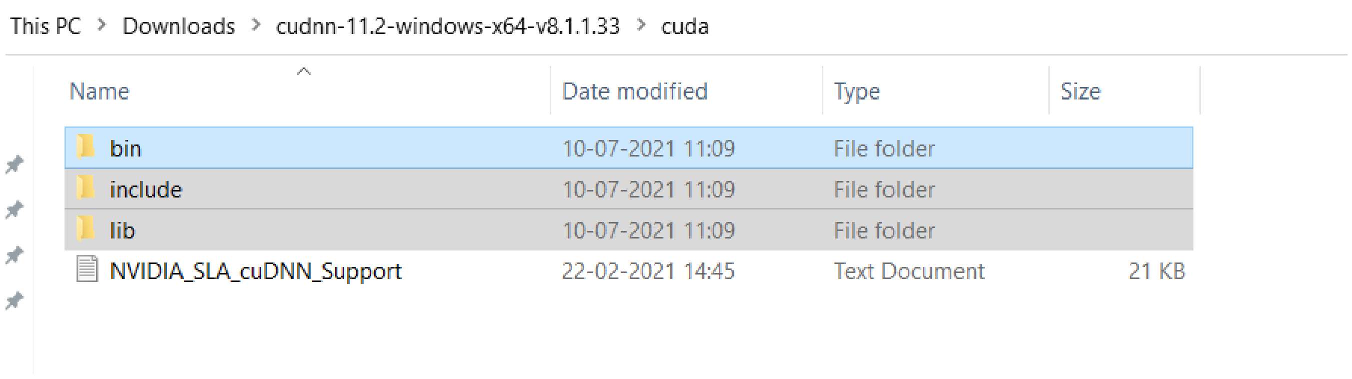
uDNN Library for Windows (x86)

uDNN Runtime Library for Ubuntu20.04 x86_64 (Deb)

uDNN Developer Library for Ubuntu20.04 x86_64 (Deb)

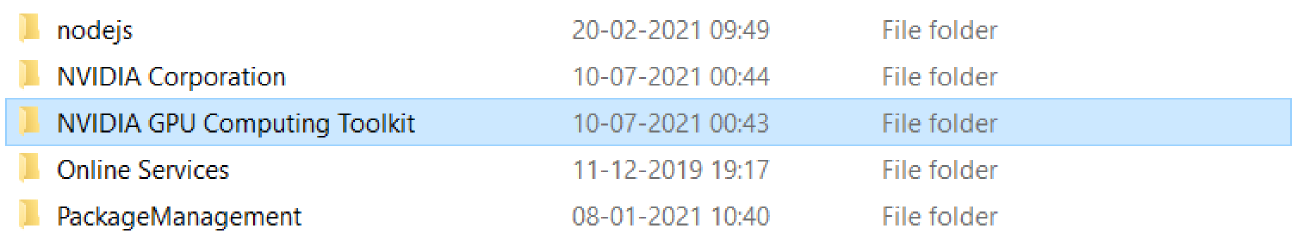
Now, check versions for CUDA and cuDNN, and click download for your operating system. If you can’t find your desired version, click on cuDNN Archive and download from there.

Once the download is complete, extract the files.


This PC > Downloads > cudnn-11.2-windows-x64-v8.1.1.33 > cuda

Name	Date modified	Type	Size
bin	10-07-2021 11:09	File folder	
include	10-07-2021 11:09	File folder	
lib	10-07-2021 11:09	File folder	
NVIDIA_SLA_cuDNN_Support	22-02-2021 14:45	Text Document	21 KB

Now, copy these 3 folders (bin, include, lib). Go to C Drive>Program Files, and search for NVIDIA GPU Computing Toolkit.

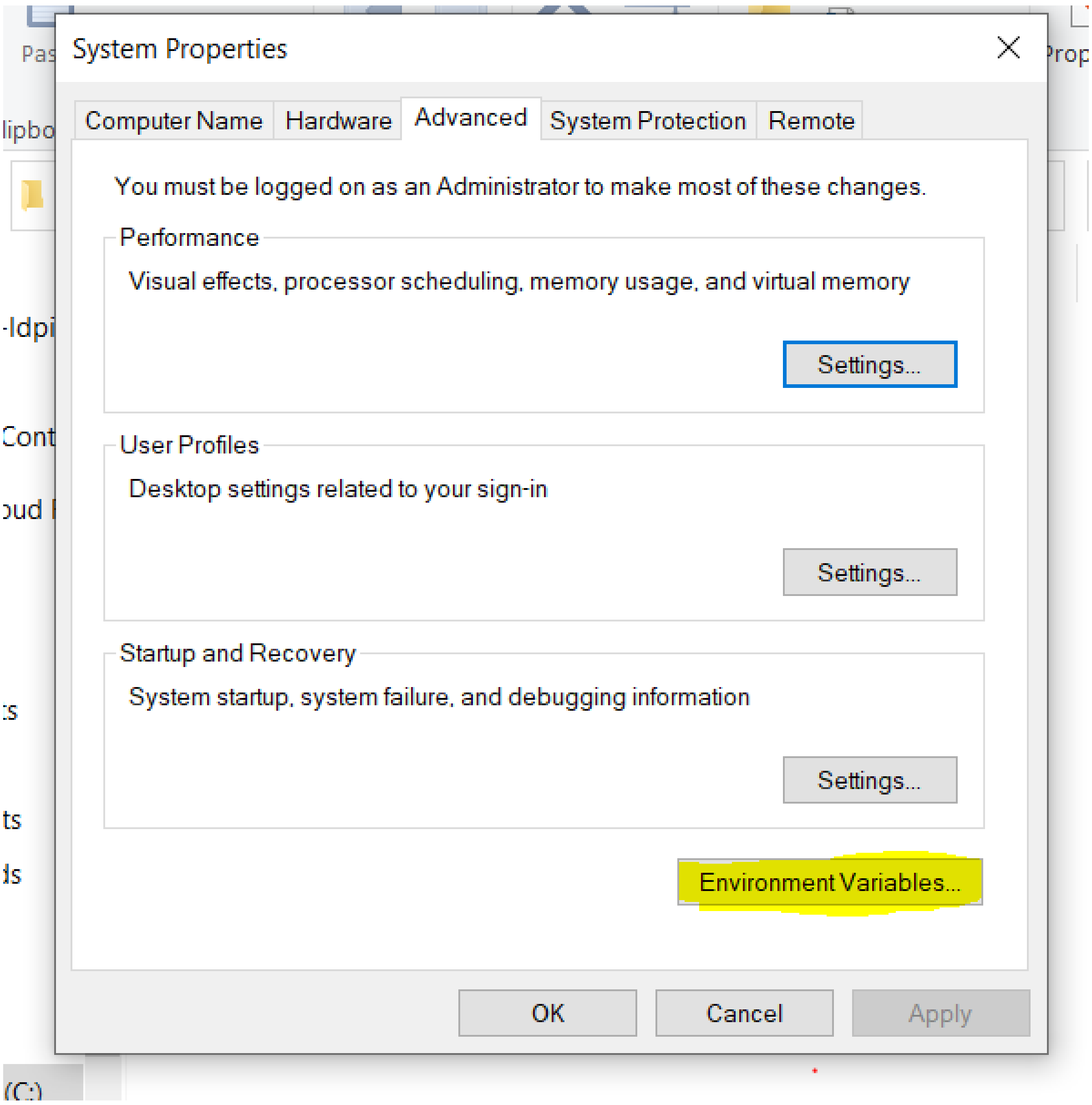


nodejs	20-02-2021 09:49	File folder
NVIDIA Corporation	10-07-2021 00:44	File folder
NVIDIA GPU Computing Toolkit	10-07-2021 00:43	File folder
Online Services	11-12-2019 19:17	File folder
PackageManagement	08-01-2021 10:40	File folder

Open the folder, select CUDA > Version Name, and replace (paste) those copied files.

Now click on the bin folder and copy the path. It should look like this: **C:Program FilesNVIDIA GPU Computing ToolkitCUDAv11.2bin.**

On your PC, search for Environment variables, as shown below.



System Properties

You must be logged on as an Administrator to make most of these changes.

Performance

Visual effects, processor scheduling, memory usage, and virtual memory

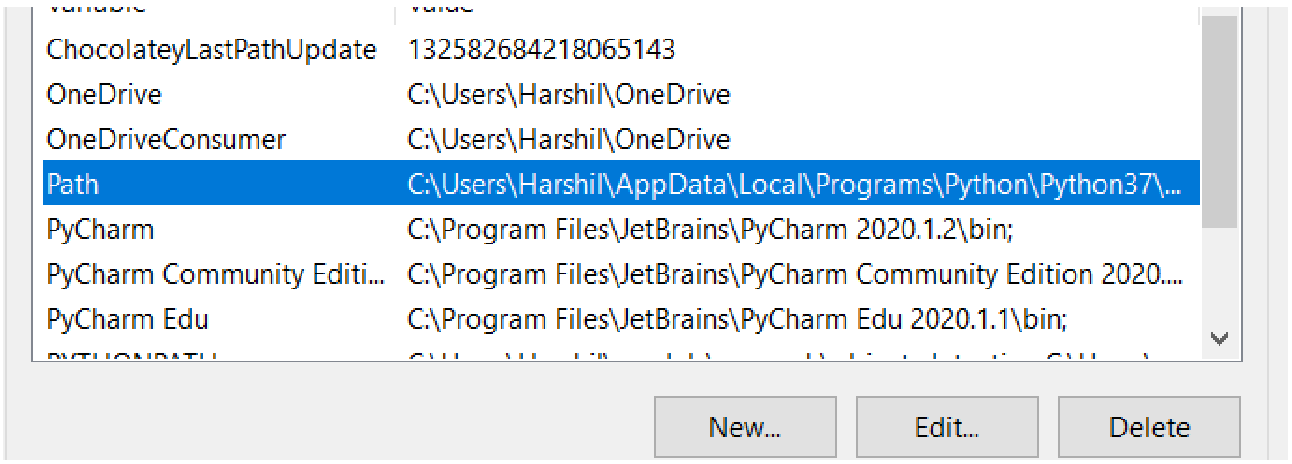
User Profiles

Desktop settings related to your sign-in

Startup and Recovery

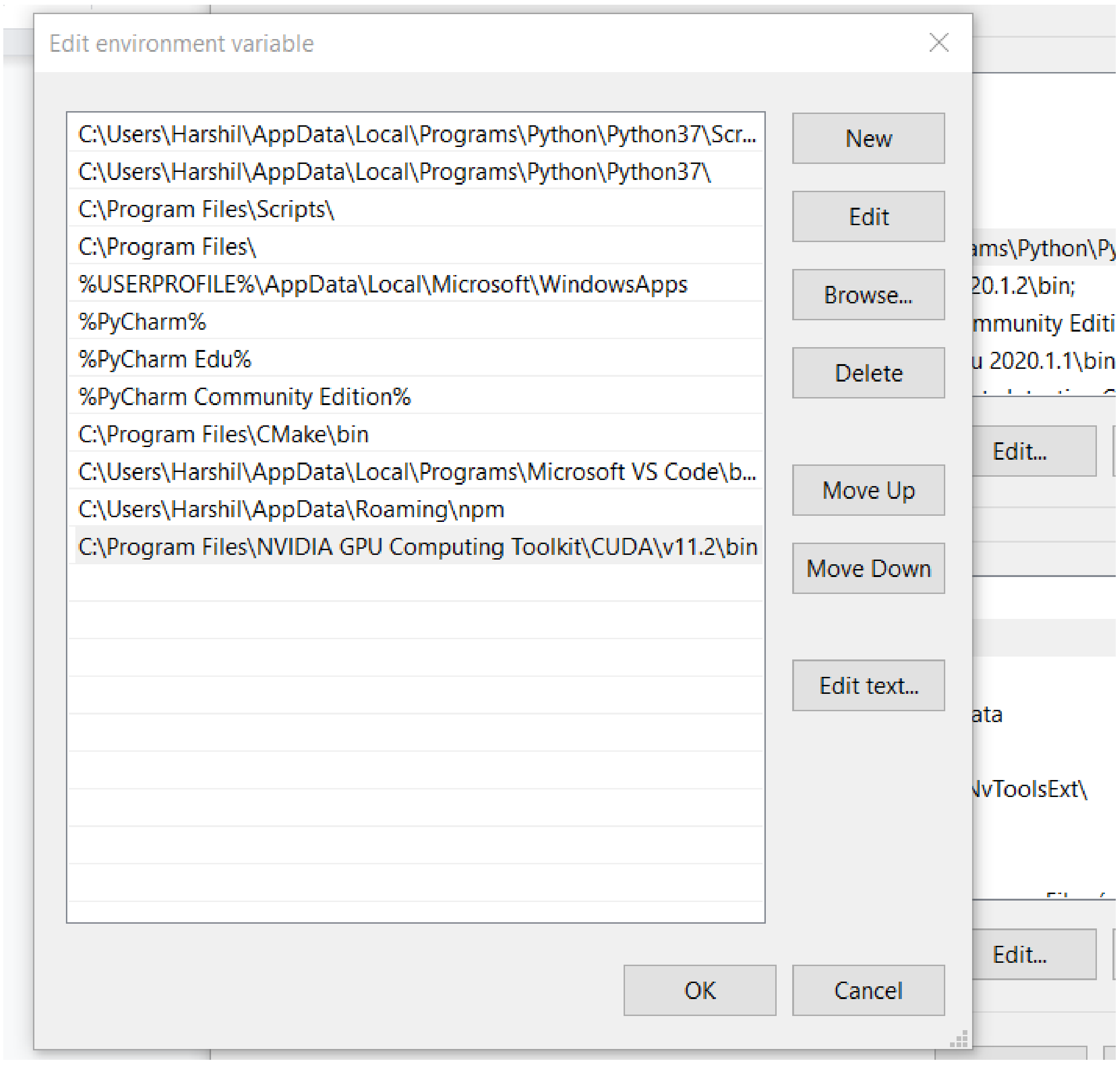
System startup, system failure, and debugging information

Click on **Environment Variables** on the bottom left. Now click on the link which states **PATH**.



Variable	Value
ChocolateyLastPathUpdate	132582684218065143
OneDrive	C:\Users\Harshil\OneDrive
OneDriveConsumer	C:\Users\Harshil\OneDrive
Path	C:\Users\Harshil\AppData\Local\Programs\Python\Python37\...
PyCharm	C:\Program Files\JetBrains\PyCharm 2020.1.2\bin;
PyCharm Community Editi...	C:\Program Files\JetBrains\PyCharm Community Edition 2020....
PyCharm Edu	C:\Program Files\JetBrains\PyCharm Edu 2020.1.1\bin;

Once you click on the PATH, you will see something like this.



```
C:\Users\Harshil\AppData\Local\Programs\Python\Python37\Scr...
C:\Users\Harshil\AppData\Local\Programs\Python\Python37\
C:\Program Files\Scripts\
C:\Program Files\
%USERPROFILE%\AppData\Local\Microsoft\WindowsApps
%PyCharm%
%PyCharm Edu%
%PyCharm Community Edition%
C:\Program Files\CMake\bin
C:\Users\Harshil\AppData\Local\Programs\Microsoft VS Code\b...
C:\Users\Harshil\AppData\Roaming\npm
C:\Program Files\NVIDIA GPU Computing Toolkit\CUDA\v11.2\bin
```

Now click on **New (Top Left), and paste the bin path here. Go to the CUDA folder, select libnvvm folder, and copy its path.** Follow the same process and paste that path into the system path. Next, just restart your PC.

4) **Installing Tensorflow**

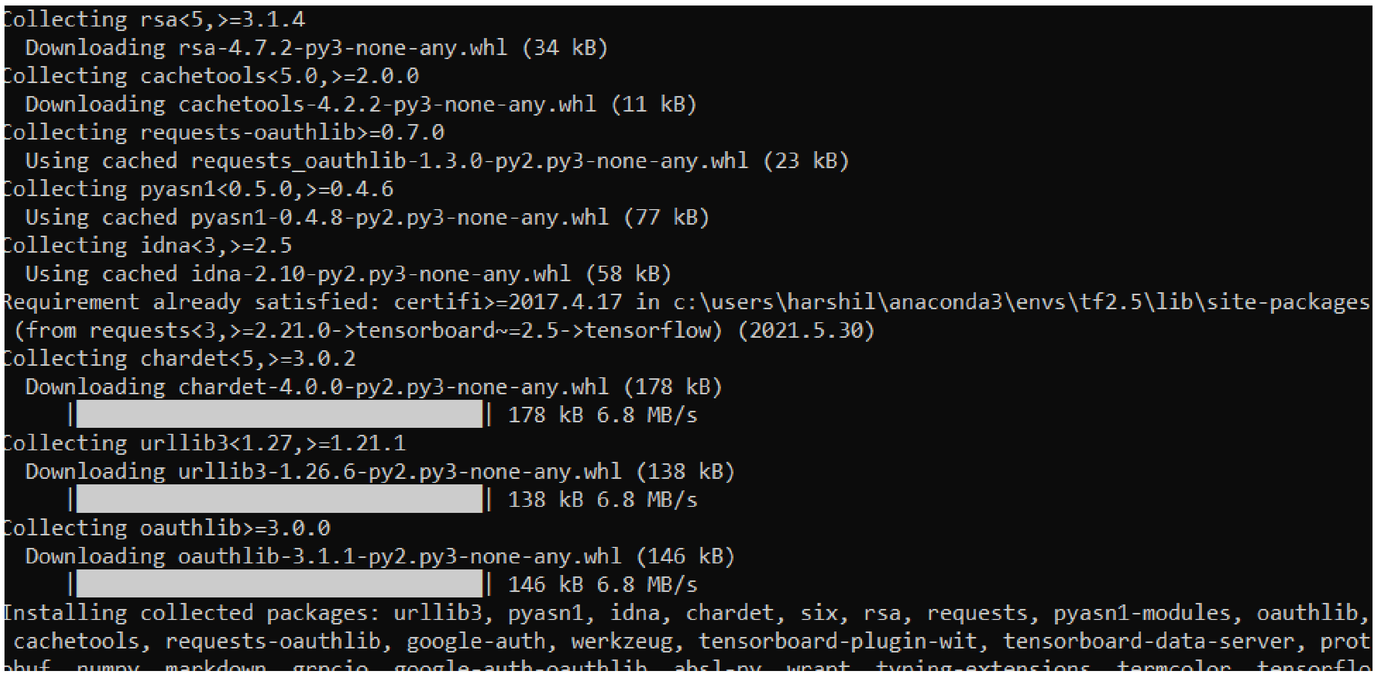
Open conda prompt. If not installed, get it here → https://www.anaconda.com/products/individual.

Now copy the below commands and paste them into the prompt (Check for the versions).

conda create --name tf2.5 python==3.8

conda activate tf2.5 (version)

pip install tensorflow (With GPU Support) //Install TensorFlow GPU command, pip install --upgrade tensorflow-gpu



```
Collecting rsa<5,>=3.1.4
  Downloading rsa-4.7.2-py3-none-any.whl (34 kB)
Collecting cachetools<5.0,>=2.0.0
  Downloading cachetools-4.2.2-py3-none-any.whl (11 kB)
Collecting requests-oauthlib>=0.7.0
  Using cached requests_oauthlib-1.3.0-py2.py3-none-any.whl (23 kB)
Collecting pyasn1<0.5.0,>=0.4.6
  Using cached pyasn1-0.4.8-py2.py3-none-any.whl (77 kB)
Collecting idna<3,>=2.5
  Using cached idna-2.10-py2.py3-none-any.whl (58 kB)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\harshil\anaconda3\envs\tf2.5\lib\site-packages
 (from requests<3,>=2.21.0->tensorboard~=2.5->tensorflow) (2021.5.30)
Collecting chardet<5,>=3.0.2
  Downloading chardet-4.0.0-py2.py3-none-any.whl (178 kB)
     |████████████████████████████████| 178 kB 6.8 MB/s
Collecting urllib3<1.27,>=1.21.1
  Downloading urllib3-1.26.6-py2.py3-none-any.whl (138 kB)
     |████████████████████████████████| 138 kB 6.8 MB/s
Collecting oauthlib>=3.0.0
  Downloading oauthlib-3.1.1-py2.py3-none-any.whl (146 kB)
     |████████████████████████████████| 146 kB 6.8 MB/s
Installing collected packages: urllib3, pyasn1, idna, chardet, six, rsa, requests, pyasn1-modules, oauthlib,
 cachetools, requests-oauthlib, google-auth, werkzeug, tensorboard-plugin-wit, tensorboard-data-server, prot
obuf, numpy, markdown, grpcio, google-auth-oauthlib, absl-py, wrapt, typing-extensions, termcolor, tensorflo
```

You'll see an installation screen like this. If you see any errors, Make sure you're using the correct version and don't miss any steps.

We've installed everything, so let's test it out in Pycharm.

Test

To test the whole process we'll use Pycharm. If not installed, get the community edition → https://www.jetbrains.com/pycharm/download/#section=windows.

First, to check if TensorFlow GPU has been installed properly on your machine, run the below code:

# importing the tensorflow package

import tensorflow **as** tf

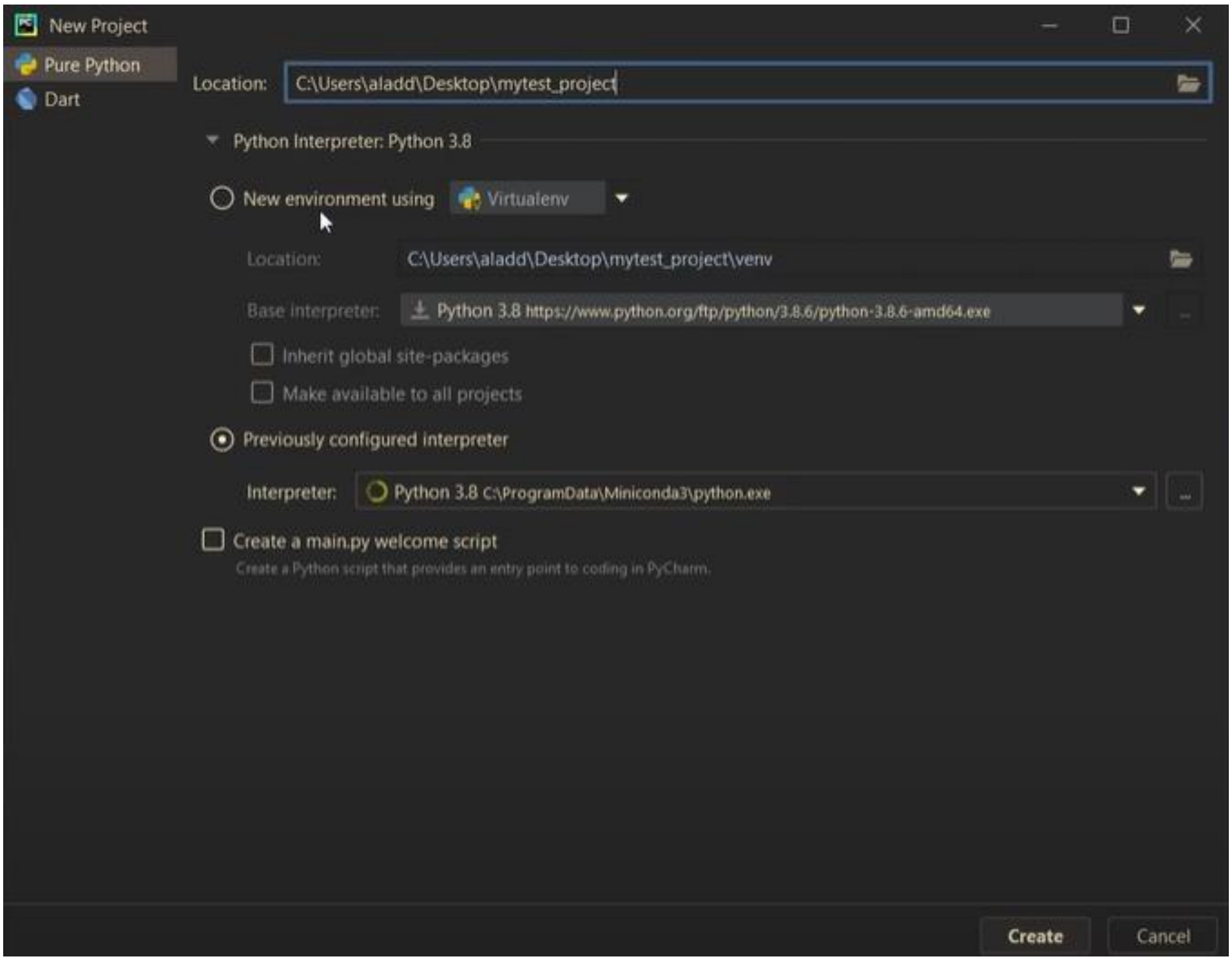
tf.test.is_built_with_cuda()

tf.test.is_gpu_available(cuda_only=**False**, min_cuda_compute_capability=**None**)

It should show TRUE as output. If it's FALSE or some error, look at the steps.

Now let's run some code.

For a simple demo, we train it on the MNIST dataset of handwritten digits. We’ll see through how to create the network as well as initialize a loss function, check accuracy, and more.



Location: C:\Users\aladd\Desktop\mytest_project

Python Interpreter: Python 3.8

New environment using Virtualenv

Location: C:\Users\aladd\Desktop\mytest_project\venv

Base interpreter: Python 3.8 https://www.python.org/ftp/python/3.8.6/python-3.8.6-amd64.exe

☐ Inherit global site-packages

☐ Make available to all projects

[✓ Previously configured interpreter]

Interpreter: Python 3.8 C:\ProgramData\Miniconda3\python.exe

☐ Create a main.py welcome script

Create a Python script that provides an entry point to coding in PyCharm.

Configure the env, create a new Python file, and paste the below code:

```
# Imports

import torch

import torchvision

import torch.nn.functional as F

import torchvision.datasets as datasets

import torchvision.transforms as transforms

from torch import optim
```

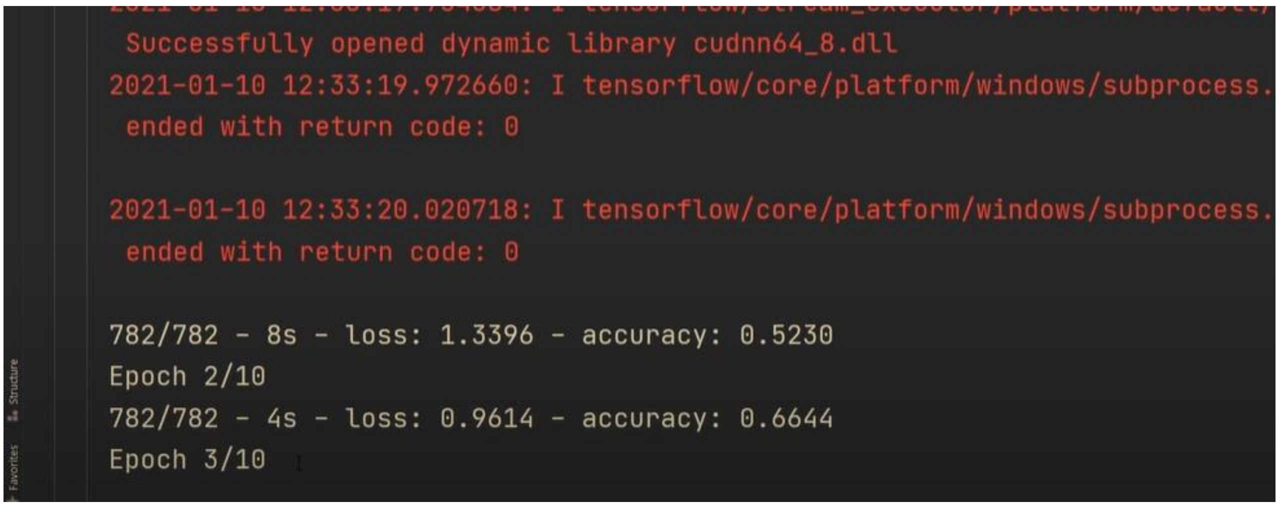
from torch **import** nn

from torch.utils.data **import** DataLoader

from tqdm **import** tqdm

Check the rest of the code here -> https://github.com/aladdinpersson/Machine-Learning-Collection/blob/master/ML/Pytorch/Basics/pytorch_simple_CNN.py.

When you run the code, look for ***successfully opened cuda(versioncode).***



```
 Successfully opened dynamic library cudnn64_8.dll
2021-01-10 12:33:19.972660: I tensorflow/core/platform/windows/subprocess.
 ended with return code: 0

2021-01-10 12:33:20.020718: I tensorflow/core/platform/windows/subprocess.
 ended with return code: 0

782/782 - 8s - loss: 1.3396 - accuracy: 0.5230
Epoch 2/10
782/782 - 4s - loss: 0.9614 - accuracy: 0.6644
Epoch 3/10
```

Once the training started, all the steps were successful!

Loading and Preprocessing Data with TensorFlow.

Deep learning systems are often trained on very large datasets that will not fit in RAM. With TensorFlow Data API, it makes easy to get the data, load and transform it. TensorFlow takes care of all implementation details, such as multithreading, queueing, batching and prefetching. Moreover, the Data API works seamlessly with tf.keras.

The Data API can read data from text files (such as CSV files), binary files with fixed-size records and binary files that use TensorFlow’s TFRecord format. In this write up, I will cover the Data API.

The Data API

The Data API revolves around the concept of a dataset that represents a sequence of data items.

Let's create a simple dataset using this below method.

```
X = tf.range(10)
dataset = tf.data.Dataset.from_tensor_slices(X)
dataset
```

```
<TensorSliceDataset shapes: (), types: tf.int32>
```

The method from_tensor_slices() takes a tensor and create tf.data.Dataset. Let's iterate the dataset's item.

```
dataset = tf.data.Dataset.range(10)
for item in dataset:
    print(item)
```

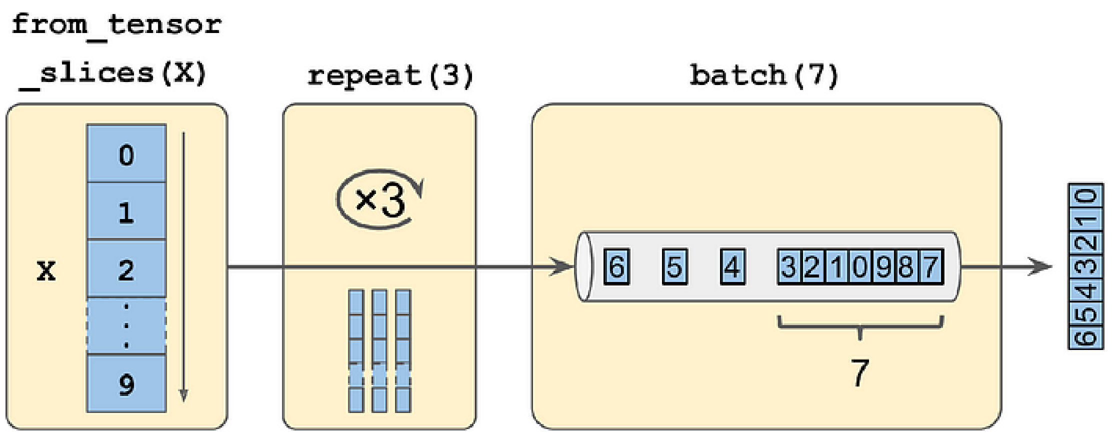
```
tf.Tensor(0, shape=(), dtype=int64)
tf.Tensor(1, shape=(), dtype=int64)
tf.Tensor(2, shape=(), dtype=int64)
tf.Tensor(3, shape=(), dtype=int64)
tf.Tensor(4, shape=(), dtype=int64)
tf.Tensor(5, shape=(), dtype=int64)
tf.Tensor(6, shape=(), dtype=int64)
tf.Tensor(7, shape=(), dtype=int64)
tf.Tensor(8, shape=(), dtype=int64)
tf.Tensor(9, shape=(), dtype=int64)
```

Transformations

Once we have a dataset, we can apply all sorts of transformations by calling the its methods.

```
dataset = dataset.repeat(3).batch(7)
for item in dataset:
    print(item)
```

```
tf.Tensor([0 1 2 3 4 5 6], shape=(7,), dtype=int64)
tf.Tensor([7 8 9 0 1 2 3], shape=(7,), dtype=int64)
tf.Tensor([4 5 6 7 8 9 0], shape=(7,), dtype=int64)
tf.Tensor([1 2 3 4 5 6 7], shape=(7,), dtype=int64)
tf.Tensor([8 9], shape=(2,), dtype=int64)
```



Chaining dataset transformations

We can also transform the items by calling the map() method.

```
dataset = dataset.map(lambda x: x * 2)
```

```
for item in dataset:
    print(item)
```

```
tf.Tensor([ 0  2  4  6  8 10 12], shape=(7,), dtype=int64)
tf.Tensor([14 16 18  0  2  4  6], shape=(7,), dtype=int64)
tf.Tensor([ 8 10 12 14 16 18  0], shape=(7,), dtype=int64)
tf.Tensor([ 2  4  6  8 10 12 14], shape=(7,), dtype=int64)
tf.Tensor([16 18], shape=(2,), dtype=int64)
```

We can also apply a filter on the dataset using filter() method.

```
dataset = dataset.filter(lambda x: x < 10)  # keep only items < 10
for item in dataset.take(3):
    print(item)
```

```
tf.Tensor(0, shape=(), dtype=int32)
tf.Tensor(1, shape=(), dtype=int32)
tf.Tensor(2, shape=(), dtype=int32)
```

Shuffle

Using shuffle() method, the following codes creates and displays a dataset containing the integers 0 to 9, repeated 3 times, shuffled using a buffer size of 3, a random seed 42 and batched with a batch size of 9.

```
tf.random.set_seed(42)

dataset = tf.data.Dataset.range(10).repeat(3)
dataset = dataset.shuffle(buffer_size=3, seed=42).batch(7)
for item in dataset:
    print(item)
```

```
tf.Tensor([1 3 0 4 2 5 6], shape=(7,), dtype=int64)
tf.Tensor([8 7 1 0 3 2 5], shape=(7,), dtype=int64)
tf.Tensor([4 6 9 8 9 7 0], shape=(7,), dtype=int64)
tf.Tensor([3 1 4 5 2 8 7], shape=(7,), dtype=int64)
tf.Tensor([6 9], shape=(2,), dtype=int64)
```

Deep learning systems are often trained on very large datasets that will not fit in RAM. With TensorFlow Data API, it makes easy to get the data, load and transform it. TensorFlow takes care of all implementation details, such as multithreading, queueing, batching and prefetching. Moreover, the Data API works seamlessly with tf.keras.

The Data API can read data from text files (such as CSV files), binary files with fixed-size records and binary files that use TensorFlow's TFRecord format. In this write up, I will cover the Data API.

The Data API

The Data API revolves around the concept of a dataset that represents a sequence of data items.

Let's create a simple dataset using this below method.

```
X = tf.range(10)
dataset = tf.data.Dataset.from_tensor_slices(X)
dataset
```

```
<TensorSliceDataset shapes: (), types: tf.int32>
```

The method from_tensor_slices() takes a tensor and create tf.data.Dataset. Let's iterate the dataset's item.

```
dataset = tf.data.Dataset.range(10)
for item in dataset:
    print(item)
```

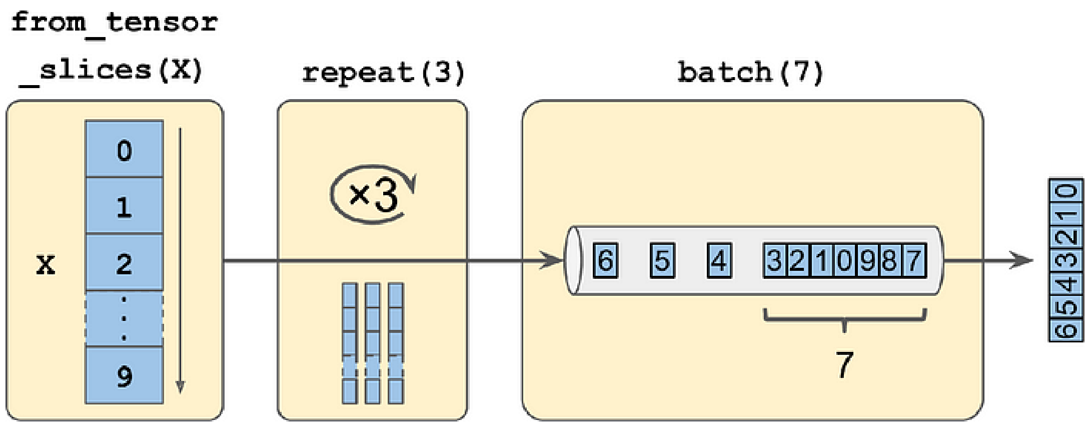
```
tf.Tensor(0, shape=(), dtype=int64)
tf.Tensor(1, shape=(), dtype=int64)
tf.Tensor(2, shape=(), dtype=int64)
tf.Tensor(3, shape=(), dtype=int64)
tf.Tensor(4, shape=(), dtype=int64)
tf.Tensor(5, shape=(), dtype=int64)
tf.Tensor(6, shape=(), dtype=int64)
tf.Tensor(7, shape=(), dtype=int64)
tf.Tensor(8, shape=(), dtype=int64)
tf.Tensor(9, shape=(), dtype=int64)
```

Transformations

Once we have a dataset, we can apply all sorts of transformations by calling the its methods.

```
dataset = dataset.repeat(3).batch(7)
for item in dataset:
    print(item)
```

```
tf.Tensor([0 1 2 3 4 5 6], shape=(7,), dtype=int64)
tf.Tensor([7 8 9 0 1 2 3], shape=(7,), dtype=int64)
tf.Tensor([4 5 6 7 8 9 0], shape=(7,), dtype=int64)
tf.Tensor([1 2 3 4 5 6 7], shape=(7,), dtype=int64)
tf.Tensor([8 9], shape=(2,), dtype=int64)
```



Chaining dataset transformations

We can also transform the items by calling the [map()](#) method.

```
dataset = dataset.map(lambda x: x * 2)
```

```
for item in dataset:
    print(item)
```

```
tf.Tensor([ 0  2  4  6  8 10 12], shape=(7,), dtype=int64)
tf.Tensor([14 16 18  0  2  4  6], shape=(7,), dtype=int64)
tf.Tensor([ 8 10 12 14 16 18  0], shape=(7,), dtype=int64)
tf.Tensor([ 2  4  6  8 10 12 14], shape=(7,), dtype=int64)
tf.Tensor([16 18], shape=(2,), dtype=int64)
```

We can also apply a filter on the dataset using filter() method.

```
dataset = dataset.filter(lambda x: x < 10)  # keep only items < 10
for item in dataset.take(3):
    print(item)
```

```
tf.Tensor(0, shape=(), dtype=int32)
tf.Tensor(1, shape=(), dtype=int32)
tf.Tensor(2, shape=(), dtype=int32)
```

Shuffle

Using shuffle() method, the following codes creates and displays a dataset containing the integers 0 to 9, repeated 3 times, shuffled using a buffer size of 3, a random seed 42 and batched with a batch size of 9.

```
tf.random.set_seed(42)

dataset = tf.data.Dataset.range(10).repeat(3)
dataset = dataset.shuffle(buffer_size=3, seed=42).batch(7)
for item in dataset:
    print(item)
```

```
tf.Tensor([1 3 0 4 2 5 6], shape=(7,), dtype=int64)
tf.Tensor([8 7 1 0 3 2 5], shape=(7,), dtype=int64)
tf.Tensor([4 6 9 8 9 7 0], shape=(7,), dtype=int64)
tf.Tensor([3 1 4 5 2 8 7], shape=(7,), dtype=int64)
tf.Tensor([6 9], shape=(2,), dtype=int64)
```