

Tutorials for Google Colab and PyTorch

CIS 4930/5930 Spring 2026



Today's Outline

- Pre-requisite
- Google Colab and Alternatives
- PyTorch Basic
- CNN Example: Image Classification
- Presenting your model: Gradio and Streamlit

Pre-requisite

1. Basic python knowledge
2. Google account for Colab (OR PC/Mac with a GPU OR Other online platform)
 - - Why you need a GPU for deep learning

Python Basic

- Python is a high-level, interpreted, general-purpose programming language known for its simplicity and readability. Unlike C++, Python is a scripting language that does not need to be compile.

Run by executing file:
`'python main.py'`

```
import random

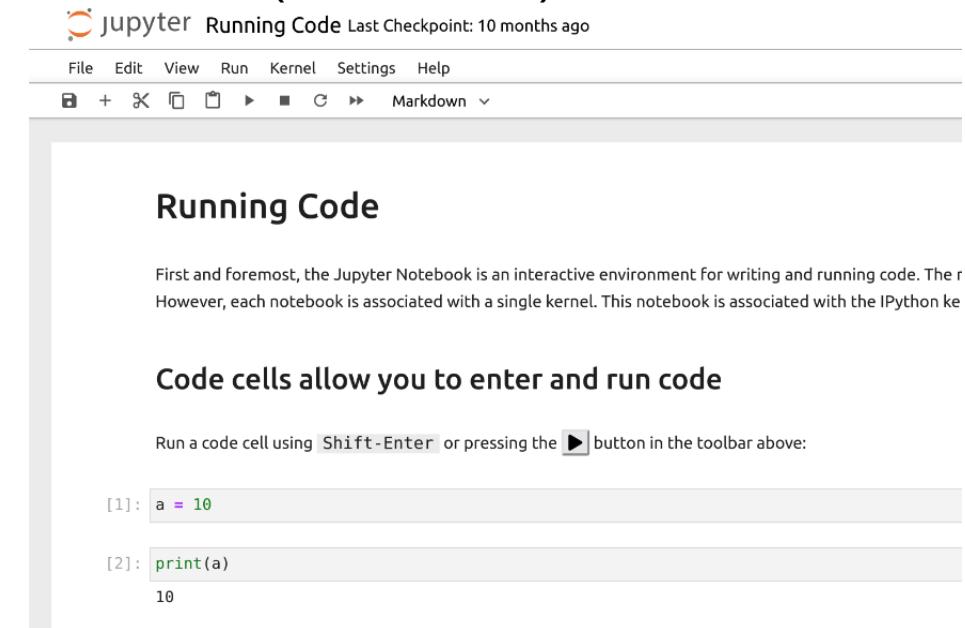
# Generate a random number between 1 and 10
random_number = random.randint(1, 10)

print("Random number:", random_number)
```

Run in console
interactively:

```
>>> print("Hello, I'm Python!")
Hello, I'm Python!
```

Run in Jupyter notebook –
like tools (In our class):



Python Basic (con't)

- In this class, we assume that you already know some of python usages. Like basic "if", "for", how to install and import package, etc.

Below are some free tutorials you can find online. if you are not familiar with python environment and python language, you may find they are useful.

Tutorial: How to install Python

windows: <https://www.youtube.com/watch?v=TNAu6DvB9Ng>

mac: <https://www.youtube.com/watch?v=nhv82tvFfkM>

Tutorial: Create Anaconda Virtual Environment for Jupyter Notebook

windows: <https://www.youtube.com/watch?v=sv0ca-6liM8>

mac: https://www.youtube.com/watch?v=SF_w7LBV_Zo

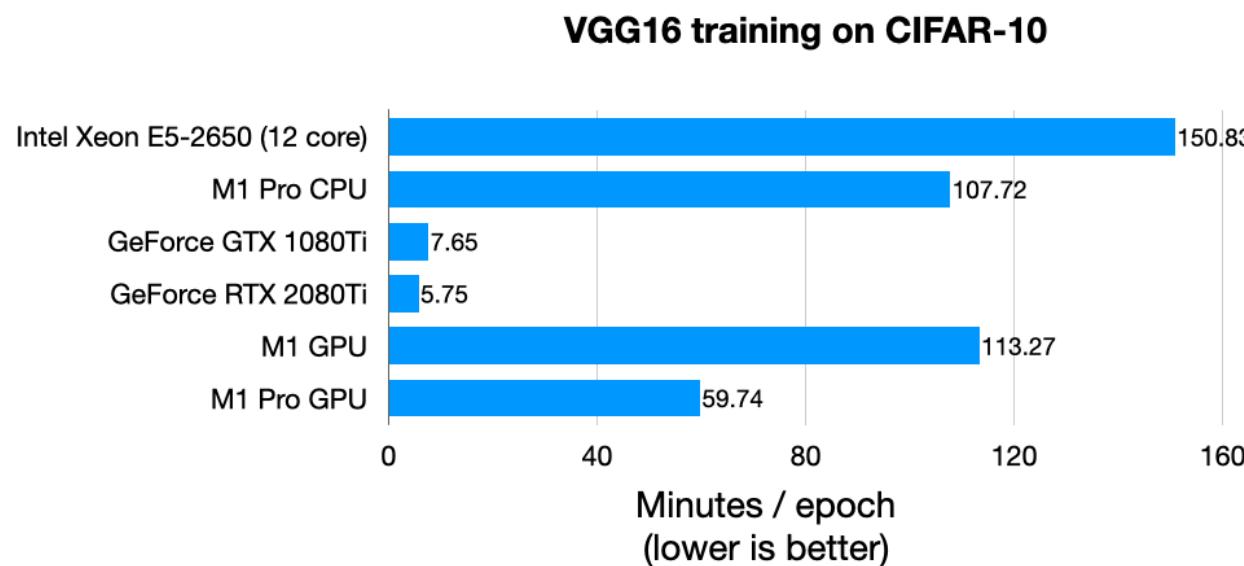
Tutorial (Python for beginners) (about 1 hour):

<https://www.youtube.com/watch?v=kqtD5dpn9C8>

A Machine with GPU

- In this class you need to have a machine with GPU in order to train your neural network.

Why you need a GPU?



A Machine with GPU (con't)

- You can use your local computer with Nvidia GPU. (For mac, you can use MPS with not promising performance)

Correct corresponding version:
Nvidia driver, CUDA toolkit, cuDNN libray

cuDNN Package ¹	CUDA Toolkit Version ²	Supports static linking? ²	NVIDIA Driver Version for Linux	NVIDIA Driver Version for Windows
cuDNN 9.5.0 for CUDA 12.x	> 12.6 > 12.5 > 12.4 > 12.3 > 12.2 > 12.1 > 12.0	Yes	>=525.60.13	>=527.41

<https://docs.nvidia.com/deeplearning/cudnn/latest/reference/support-matrix.html>

Command line to check if you already have CUDA toolkit installed

```
(base) C:\Users\User>nvcc -V
nvcc: NVIDIA (R) Cuda compiler driver
Copyright (c) 2005-2024 NVIDIA Corporation
Built on Tue_Feb_27_16:28:36_Pacific_Standard_Time_2024
Cuda compilation tools, release 12.4, V12.4.99
Build cuda_12.4.r12.4/compiler.33961263_0
```

How to install cuda toolkit on windows: <https://wiki.cci.arts.ac.uk/books/how-to-guides/page/how-to-install-cuda-toolkit-on-your-personal-windows-pc>

Accelerated PyTorch training on Mac: <https://developer.apple.com/metal/pytorch/>

Create Running Environment

- Each Python project can have its own unique running environment
- Recommended tool: Anaconda and Pip
 - Allows users to create and manage isolated environments easily
 - different projects can use different versions of Python and packages
 - Example: Numpy, Sklearn, PyTorch, Pandas, etc.
 - ensures consistent package versions and environment behavior across Windows, macOS, and Linux
 - Useful tutorials:
 - <https://www.anaconda.com/docs/getting-started/getting-started>
 - <https://www.youtube.com/watch?v=1nDNhiRg4DY>
- Other choice: uv
 - Documentations: <https://docs.astral.sh/uv/>

Google Colab and Alternatives

1. Getting start with Colab:
 - Runing Code, GPU Runtime, File Manipulation, Linux Commands on Colab, Tips
2. Alternatives:
 - PC(Recommended)/MAC with a GPU: Jupyter notebook with Conda virtual environment(optional)
 - Paid options: Colab Pro, Paperspace, etc.

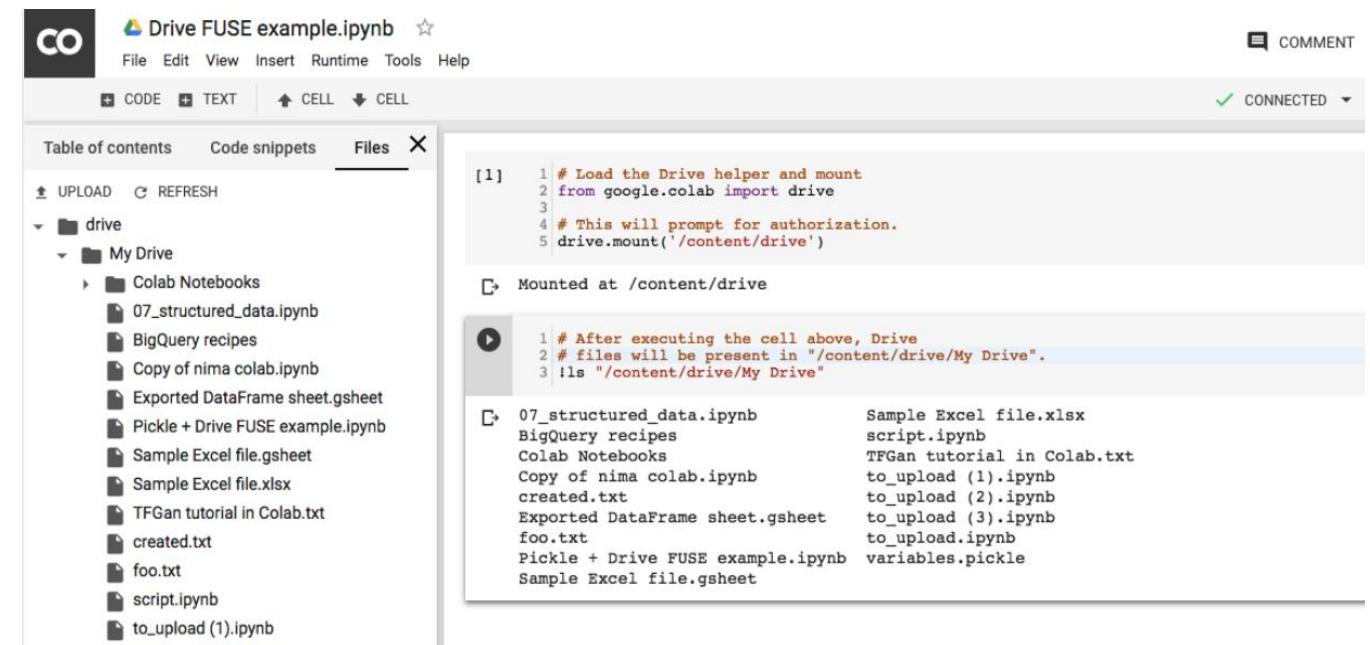
Google Colab

- A ready-to-use python environment with free GPU hours

You can see it as an online version of Jupyter notebook. I recommend you to use this since it already comes with some common packages and python environment if you do not have a decent GPU on your local computer

<https://colab.research.google.com/>

You need a free google account in order to use it.



The screenshot shows the Google Colab interface. On the left, there's a sidebar with a 'Files' tab selected, showing a tree view of files and folders. Inside the 'My Drive' folder, there are several notebooks and other files like 'Sample Excel file.xlsx'. On the right, a main workspace shows a code cell with the following Python code:

```
[1] 1 # Load the Drive helper and mount
2 from google.colab import drive
3
4 # This will prompt for authorization.
5 drive.mount('/content/drive')
```

Below the code cell, there's a preview pane showing the output of the code, which includes the path '/content/drive/My Drive' and a list of files from the 'My Drive' folder. The preview pane also shows a play button icon, indicating the code has been run.

Add and Run Code

Click "+Code" to add python code

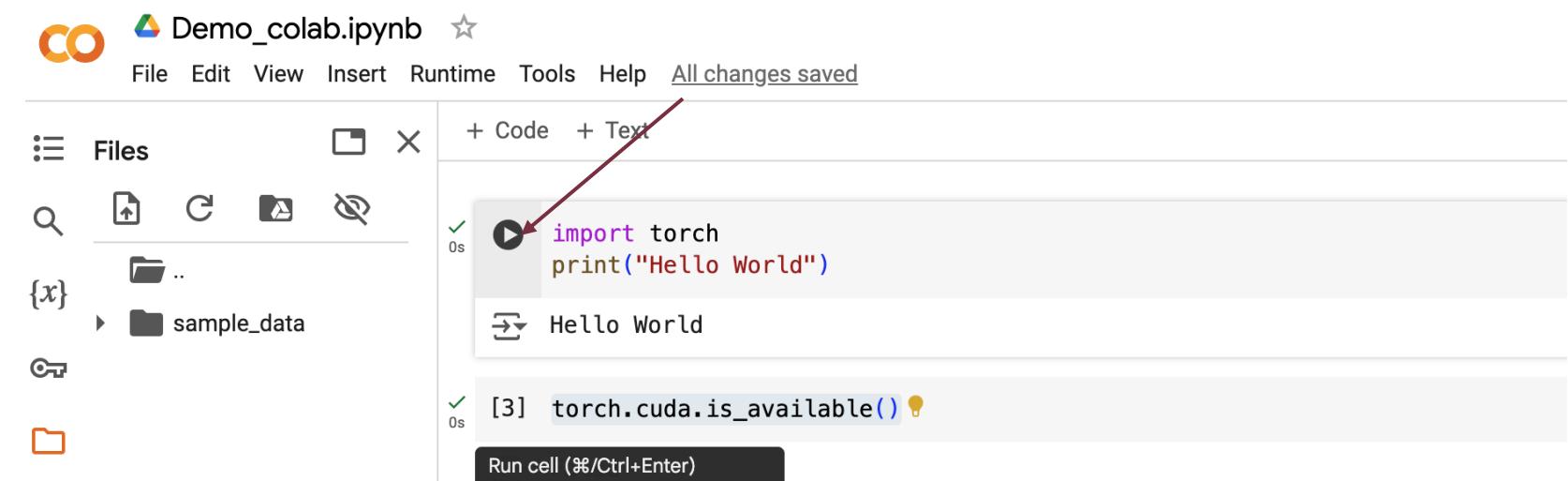
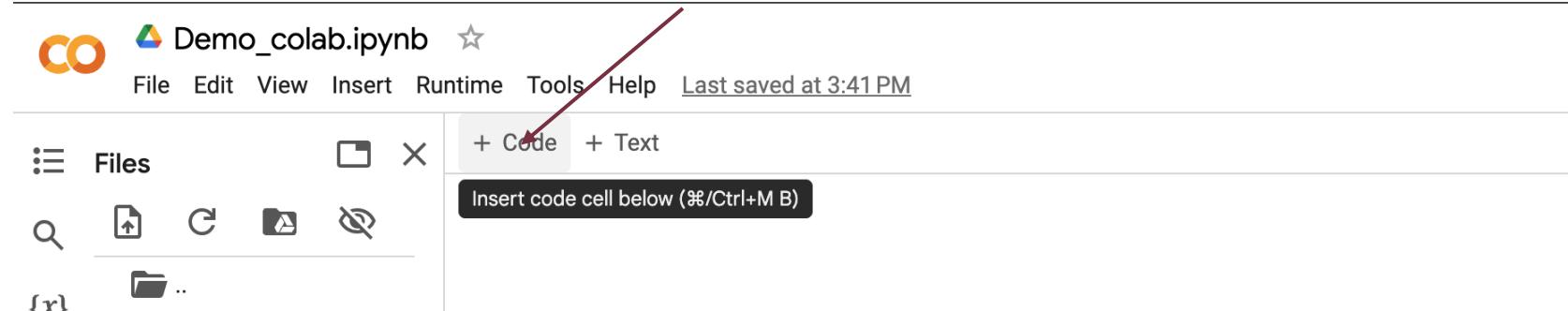
OR

Click "+Text" to add plain text(markdown)

Click the run button to run the code

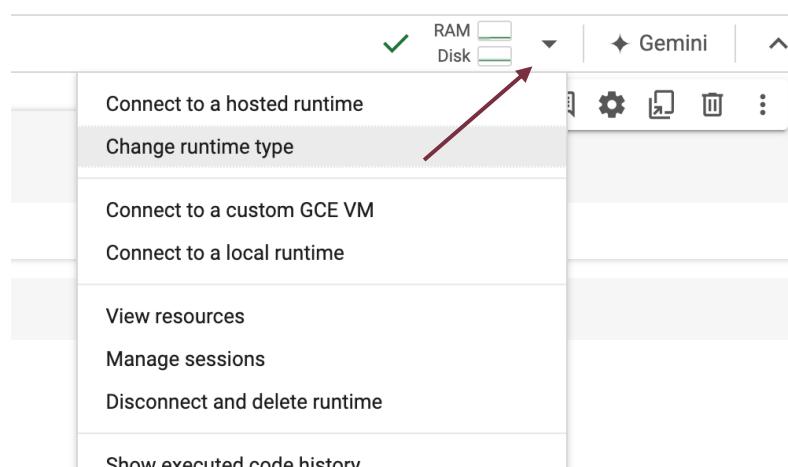
OR

You can select the "Runtime" in the menu to run multiple blocks of code

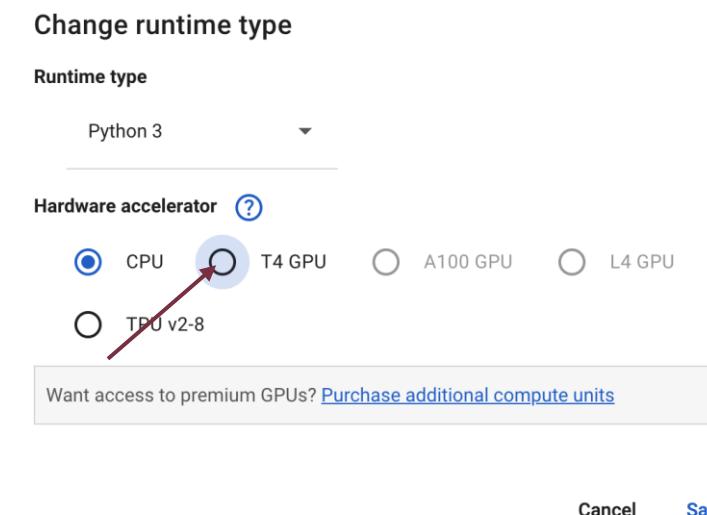


Select GPU environment and Verify

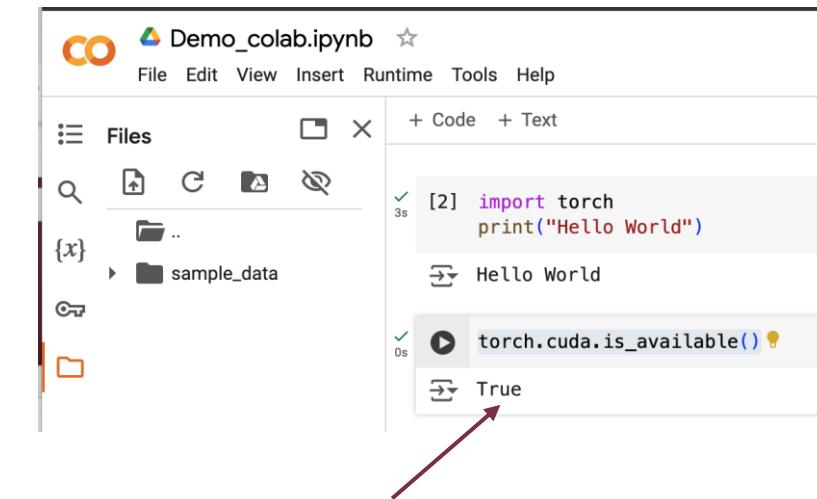
Select the button next to resource monitor icon and choose "Change runtime type"



Choose "T4 GPU" which is free to use and save



You can run the `torch.cuda.is_available()` again to verify your configuration

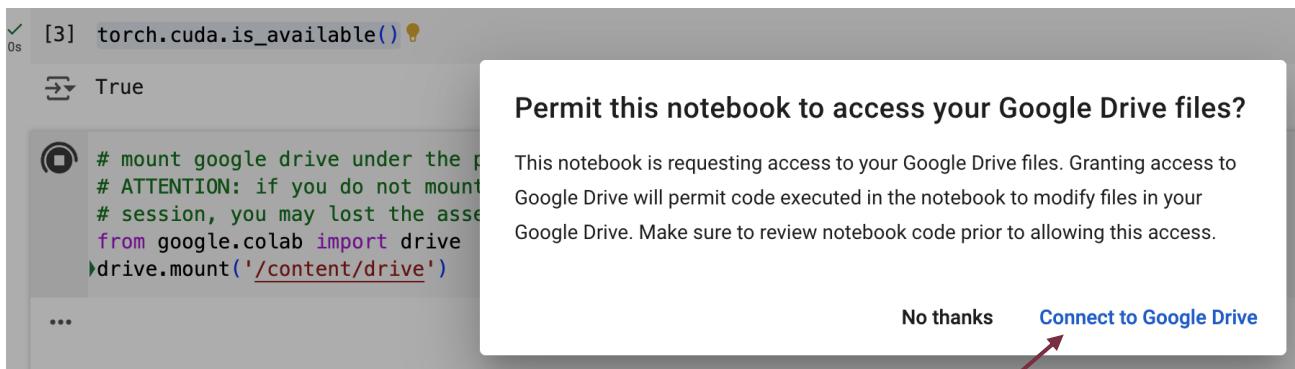


Mount Google Drive

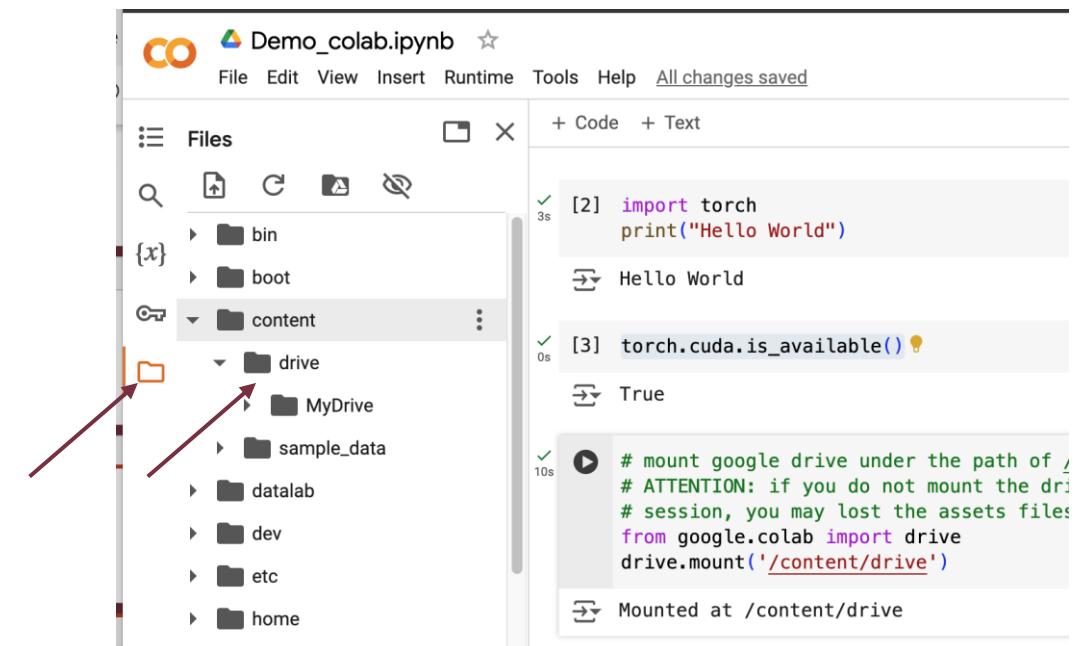
ALERT: the files in your temporary session will be lost after you restart your notebook

Use the code below to mount your google drive(15GB free) and grant the permission

```
from google.colab import drive  
drive.mount('/content/drive')
```



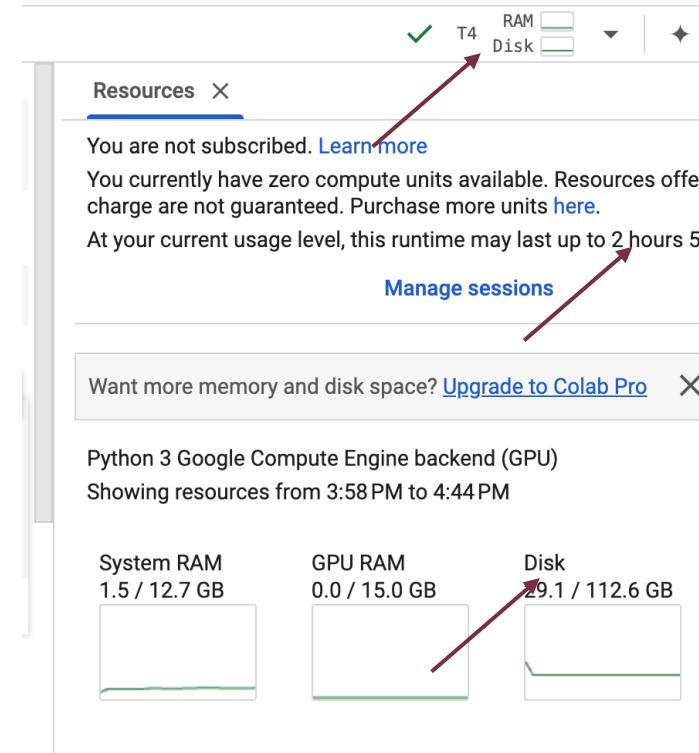
You can find the mounted folder in the side bar, where you can upload your files



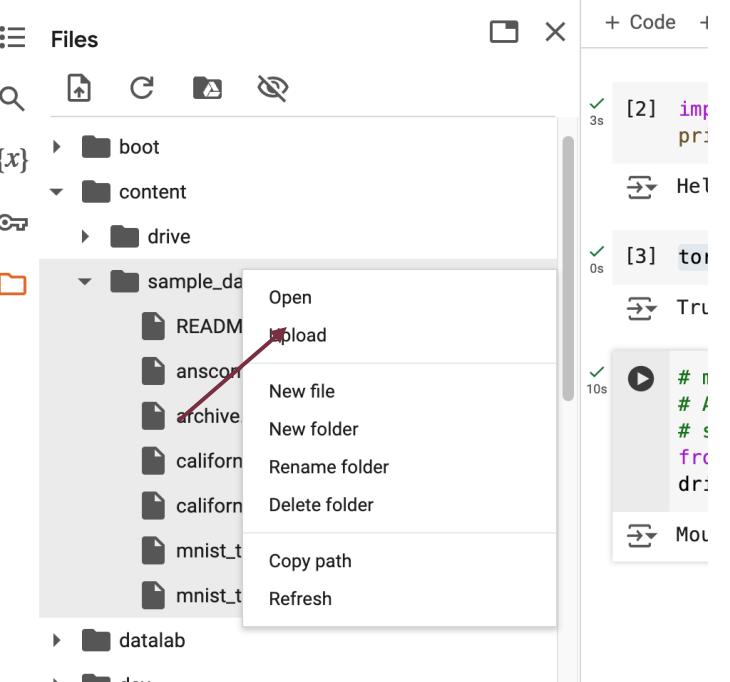
Upload or Download dataset

If you have a large dataset, you can also upload or download it directly to your temporary session

View your usage here
(Notice: you do have a time limitation to use the GPU session, after that, you will lost all your temporary data)



Right click the folder and select upload



Upload or Download dataset

- Or, you can download your dataset from URL to your session.
- Google Colab supports linux commands in the code block with a suffix "!" (use "%cd" to change directory), for example, use "!wget" to download files

```
!wget https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz -P /content/sample_data/
```

```
✓ 7s  ⏴ !wget https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz -P /content/sample_data/
→ --2024-10-15 21:03:40-- https://www.cs.toronto.edu/~kriz/cifar-100-python.tar.gz
Resolving www.cs.toronto.edu (www.cs.toronto.edu)... 128.100.3.30
Connecting to www.cs.toronto.edu (www.cs.toronto.edu)|128.100.3.30|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 169001437 (161M) [application/x-gzip]
Saving to: '/content/sample_data/cifar-100-python.tar.gz'

cifar-100-python.ta 100%[=====] 161.17M  30.4MB/s    in 5.9s

2024-10-15 21:03:46 (27.5 MB/s) - '/content/sample_data/cifar-100-python.tar.gz' saved [169001437/169001437]
```

Read files

Use "%cd" enter into the folder contains your dataset,

Use "!unzip" to unzip the file

Use matplotlib to read and show image

The screenshot shows a Jupyter Notebook environment. On the left is a file browser titled "Files" showing a directory structure under "/content". The "sample_data" folder contains a "PokemonData" subfolder which in turn contains numerous subfolders corresponding to different Pokémon species. At the bottom of the file browser, it says "Disk" and "82.47 GB available". To the right of the file browser is a code editor and output pane. The code editor shows two code cells:

```
[13]: %cd sample_data/  
!unzip archive.zip
```

Cell 13 runs the command to change the current directory to "sample_data" and then unzips the "archive.zip" file. Below this cell is a button labeled "Show hidden output".

```
[16]: !pwd
```

Cell 16 runs the command to print the current working directory, which is "/content/sample_data".

Below the code cells is a play button icon followed by Python code:

```
import matplotlib.pyplot as plt  
import matplotlib.image as mpimg  
import os  
  
dir_path = '/content/sample_data/PokemonData/Pikachu/'  
  
img = mpimg.imread(os.path.join(dir_path, '00000000.jpg'))  
plt.imshow(img)
```

This code imports matplotlib and numpy, sets the directory path to "Pikachu" within "PokemonData", and then reads the first image file ("00000000.jpg") and displays it using plt.imshow(). The resulting image is a yellow Pikachu character.

Getting Free one-year Pro Subscription

- Visit <https://colab.research.google.com/signup>

Choose the Colab plan that's right for you

Whether you're a student, a hobbyist, or a ML researcher, Colab has you covered

Colab is always free of charge to use, but as your computing needs grow there are paid options to meet them.

[Restrictions apply, learn more here](#)

The screenshot shows the Colab subscription selection page. It features four main plans: 'Pay As You Go', 'Colab Pro', 'Colab Pro+', and 'Colab Enterprise'. The 'Colab Pro' plan is highlighted with a yellow border and a red box around the text 'No cost for students and educators'. Other visible text includes '\$9.99 per month' for Colab Pro, '\$49.99 per month' for Colab Pro+, and 'Pay for what you use' for Colab Enterprise. The 'Pay As You Go' plan shows compute unit costs of '\$9.99 for 100 Compute Units' and '\$49.99 for 500 Compute Units'.

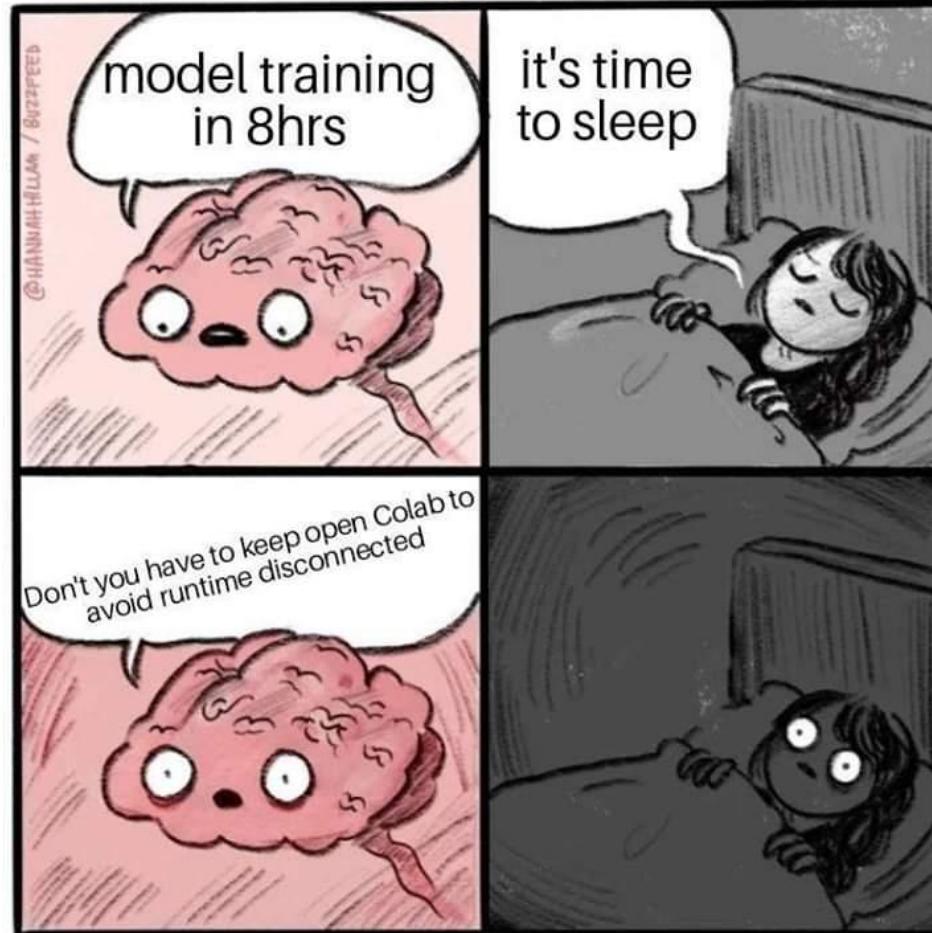
Verify your institution

You will be redirected to SheerID to verify your student or faculty affiliation with a US-based educational higher education institution. Information provided will be processed by SheerID in accordance with their [privacy policy](#). Read more in our [FAQ](#).

I confirm that [REDACTED] is the account I wish to receive my non-transferable Colab Pro for Education subscription upon successful identity verification.

[Cancel](#) [Continue as Teacher](#) [Continue as Student](#)

Attention:



- Google Colab notebooks have an idle timeout of 90 minutes and absolute timeout of 12 hours.
- This means, if user does not interact with his Google Colab notebook for more than 90 minutes, its instance is automatically terminated.
- Maximum lifetime of a Colab instance: 12 hours.
- To keep your colab active: try
<https://stackoverflow.com/questions/57113226/how-can-i-prevent-google-colab-from-disconnecting>
- There is a cooldown period, so you may have to use multiple free accounts

Alternatives

- Paperspace (Plan or on-demand instance): for example, free plan or \$8 month for faster free GPU: <https://www.paperspace.com>
- VastAI - Similar to above: <https://vast.ai>
- Google Cloud, PyTorch lighting, or AWS, etc...
- Notice: If you have a good gaming PC that have RTX 4090, etc. You can always use your own PC for training. (For environment setup, please refer to the pre-requisite part)

PyTorch Basic

1. Introduction:
 - Basic Concepts, Build your network, Pipelines for training/testing/saving/loading model
2. PyTorch Documentation

PyTorch Introduction



- <https://thenewstack.io/the-ultimate-guide-to-machine-learning-frameworks/>

A popular machine learning framework

- For training machine learning model
- Developed at Facebook AI and Research lab
- Implementing a neural network in PyTorch is simpler and intuitive than other frameworks
- **Converting NumPy objects to tensors is natively integrated with PyTorch's core data structures**

✓ [1] import torch
8 秒

Key Concepts:

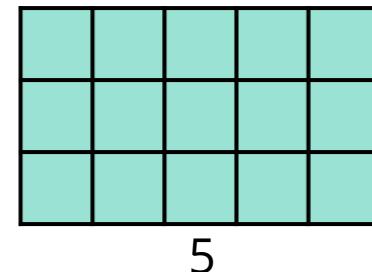
Tensor: A tensor is a mathematical object that generalizes scalars, vectors, and matrices into higher-dimensional spaces.



5

1-D tensor
e.g. audio

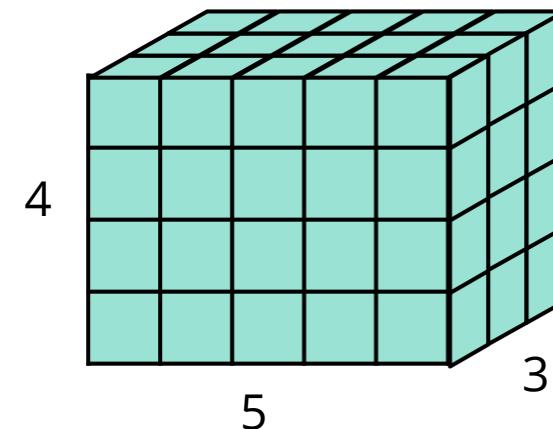
(5,) -> dim0



5

2-D tensor
e.g. black&white
images

(3, 5) -> dim0, dim1



5

3-D tensor
e.g. RGB
images

(4, 5, 3) -> dim0, dim1, dim2

Create tensors

```
tensor_0 = torch.zeros([2,3])  
tensor_0
```

```
tensor([[0., 0., 0.],  
       [0., 0., 0.]])
```

```
tensor_1 = torch.tensor([0,1], [1,0])  
tensor_1
```

```
tensor([[0, 1],  
       [1, 0]])
```

Check tensor's shape:

```
tensor_1.shape
```

```
torch.Size([2, 2])
```

Key Concepts:

Tensor operations example:

Transpose:

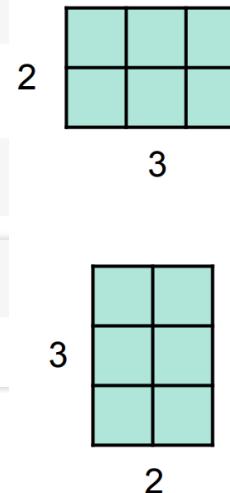
```
tensor_0.shape
```

```
torch.Size([2, 3])
```

```
tensor_0 = tensor_0.transpose(0, 1)
```

```
tensor_0.shape
```

```
torch.Size([3, 2])
```



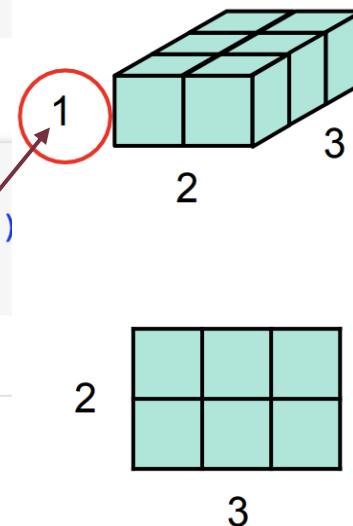
Remove Dim:

```
tensor3.shape
```

```
torch.Size([1, 2, 3])
```

```
# remove first dim  
tensor3 = tensor3.squeeze(0)  
tensor3.shape
```

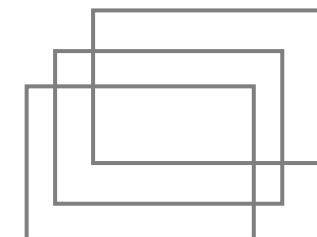
```
torch.Size([2, 3])
```



Expand Dim:

```
tensor3 = tensor3.unsqueeze(0)  
tensor3.shape
```

```
torch.Size([1, 2, 3])
```



ML Steps in PyTorch



```
# Prepare data
xs = np.array([-1.0, 0.0, 1.0, 2.0, 3.0, 4.0], dtype=np.float32)
ys = np.array([-3.0, -1.0, 1.0, 3.0, 5.0, 7.0], dtype=np.float32)

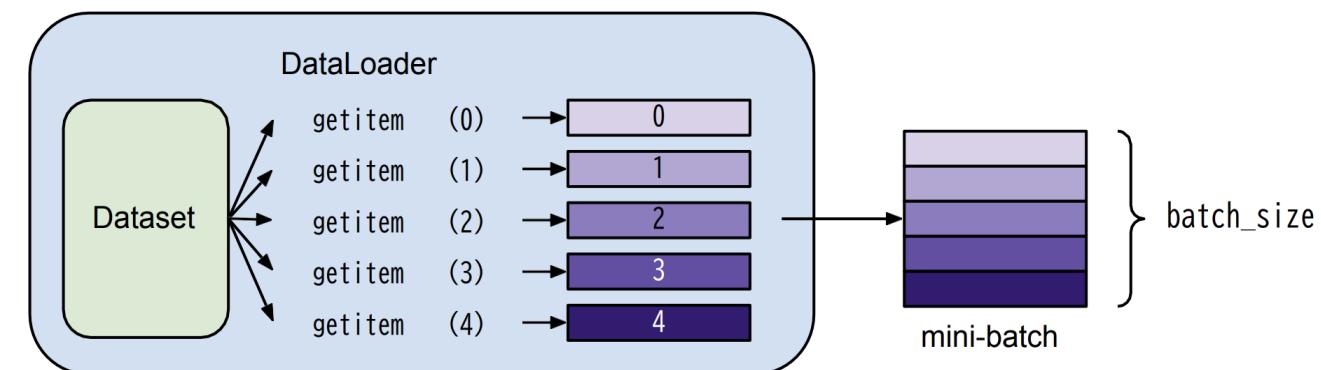
# Convert numpy arrays to torch tensors
xs = torch.tensor(xs).view(-1, 1) # Reshape to (n_samples, n_features)
ys = torch.tensor(ys).view(-1, 1)

print(xs)
print(ys)

tensor([[-1.],
       [ 0.],
       [ 1.],
       [ 2.],
       [ 3.],
       [ 4.]])
tensor([[-3.],
       [-1.],
       [ 1.],
       [ 3.],
       [ 5.],
       [ 7.]])
```

In a real example, you may use the code below to load data:

```
dataset = MyDataset('data.csv')
dataloader = DataLoader(dataset, batch_size=4, shuffle=True)
```



ML Steps in PyTorch



nn.Linear (fully connected layer)

```
# Define the model
class SimpleModel(nn.Module):
    def __init__(self):
        super(SimpleModel, self).__init__()
        self.linear = nn.Linear(1, 1) # 1 input, 1 output

    def forward(self, x):
        return self.linear(x)

model = SimpleModel()
```

Under the hood, you may define your own layer manually

```
class SimpleLinearLayer(nn.Module):
    def __init__(self, in_features, out_features):
        super(SimpleLinearLayer, self).__init__()
        self.in_features = in_features
        self.out_features = out_features

        # Initialize weights and bias
        self.weights = nn.Parameter(torch.randn(out_features, in_features) * 0.01) # Small random weights
        self.bias = nn.Parameter(torch.zeros(out_features)) # Bias initialized to zeros

    def forward(self, x):
        return x @ self.weights.t() + self.bias
```

$$\text{Output} = (\text{Weight} * \text{Input}) + \text{Bias}$$
$$Y = AX + B \text{ (Linear)}$$

ML Steps in PyTorch



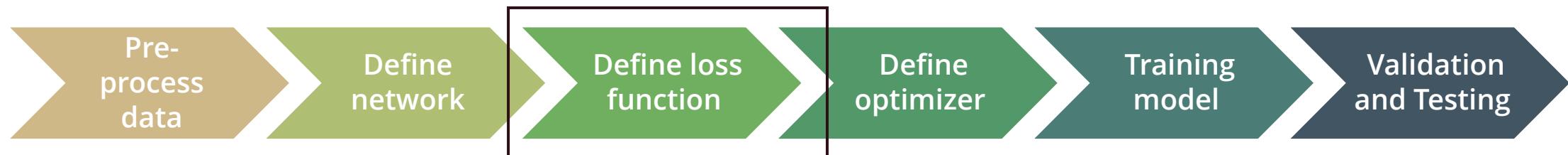
More layers in a container

```
# Define the model
class SimpleModel2(nn.Module):
    def __init__(self):
        super(SimpleModel2, self).__init__()
        self.net = nn.Sequential(
            nn.Linear(1, 32), # input layer, 1 input, 32 neurons (32 features)
            nn.Sigmoid(), # hidden layer, sigmoid function
            nn.Linear(32, 1) # output layer, output 1
        )

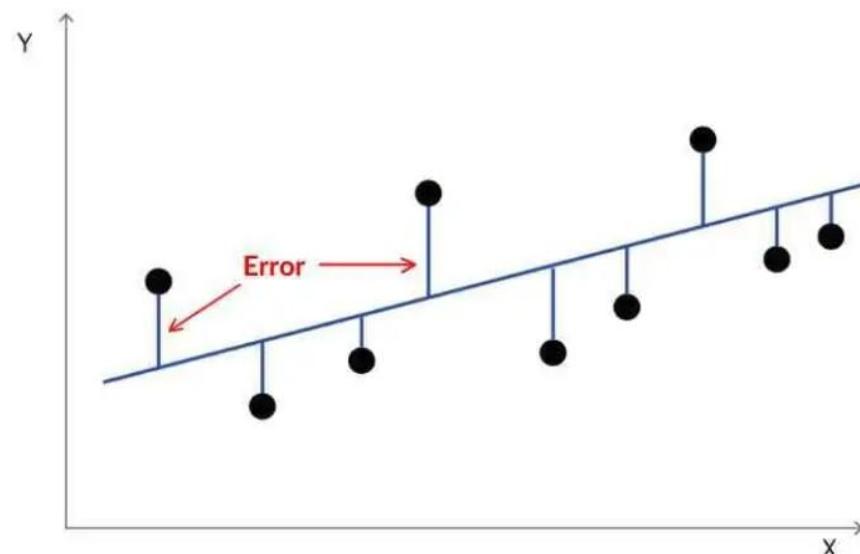
    def forward(self, x):
        return self.net(x)
```

Whole Example:
https://colab.research.google.com/drive/1_ru4CdNgz_MrtiPs5qUOaRoW_5q1R6cl?usp=sharing

ML Steps in PyTorch



```
# Define loss  
criterion = nn.MSELoss()
```



Under the hood, it just something like this:

```
def loss(predictions, targets):  
    return ((predictions - targets) ** 2).mean()
```

Other Example:

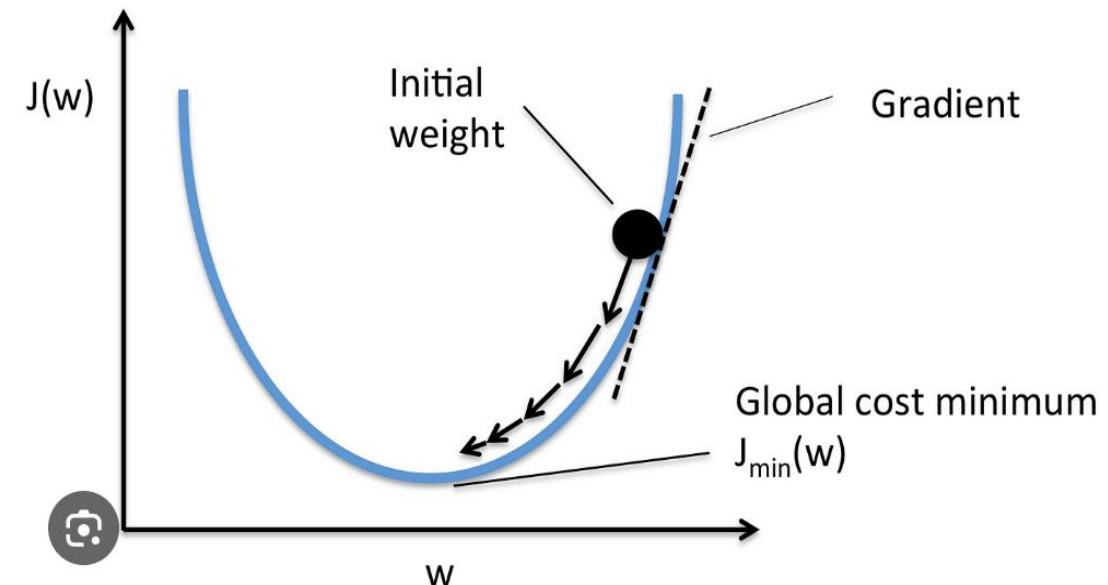
```
# Cross Entropy (for classification tasks)  
criterion = nn.CrossEntropyLoss()
```

ML Steps in PyTorch

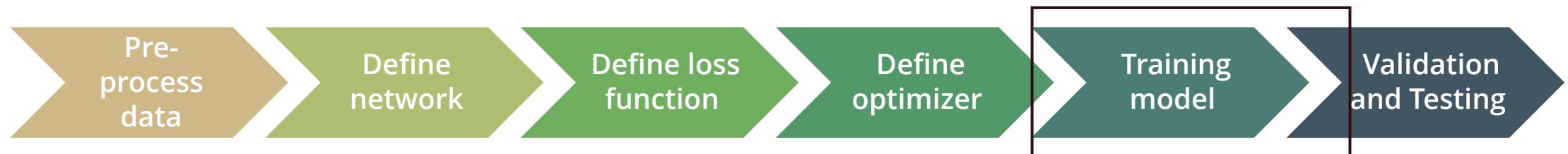


Here we use Stochastic Gradient Descent Optimizer

```
# Define optimizer  
optimizer = optim.SGD(model.parameters(), lr=0.01)
```



ML Steps in PyTorch



```
] # Train the model
epochs = 10
for epoch in range(epochs):
    # Forward pass
    outputs = model(xs)
    loss = criterion(outputs, ys)

    # Backward pass and optimization
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

- Take the input: put xs to model,
- Calculate the loss
- Zero the gradients before the backward pass in current iteration
 optimizer.zero_grad()
- loss.backward() # Compute gradients
- Optimizer.step() # # Update parameters based on gradients

ML Steps in PyTorch



```
# Predict
with torch.no_grad():
    prediction = model(torch.tensor([[10.0]]))
    print(prediction.item())
```

18.317289352416992

torch.no_grad():

Prevents calculations from being added into gradient computation graph. Usually used to prevent accidental training on validation/testing data.

model.eval(): put model in evaluation mode

ML Steps in PyTorch



Use GPU: move input and model to GPU

```
[1] !nvidia-smi
Thu Oct 17 18:16:48 2024
+-----+
| NVIDIA-SMI 535.104.05      Driver Version: 535.104.05 CUDA Version: 12.2 |
|=================================================================================================|
| GPU  Name        Persistence-M | Bus-Id     Disp.A  Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap | Memory-Usage | GPU-Util  Compute M. |
|=================================================================================================|
|   0  Tesla T4           Off  00000000:00:04.0 Off           0% Default      N/A |
|   N/A  49C   P8          10W / 70W    0MiB / 15360MiB     0%      Default      N/A |
+-----+
+-----+
| Processes:                               PID  Type  Process name          GPU Memory Usage |
| GPU  GI CI      ID  ID                                         Usage |
|=====================================================================
| No running processes found
+-----+
```

```

# Check if GPU is available, else fallback to CPU
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

xs = torch.tensor(xs).view(-1, 1).to(device) # Move to GPU (if available)
ys = torch.tensor(ys).view(-1, 1).to(device)

# Initialize the model and move it to the selected device
model = SimpleModel().to(device)

with torch.no_grad():
    # Move input tensor to GPU
    prediction = model(torch.tensor([[10.0]], device=device))
    print(prediction.item())
  
```

"mps" if mac

ML Steps in PyTorch



Save and load model

```
# Save the model state
torch.save(model.state_dict(), 'model_state.pth')

# Load the model state
loaded_model = SimpleModel().to(device)
loaded_model.load_state_dict(torch.load('model_state.pth'))
loaded_model.eval() # Set to evaluation mode

# Predict using the loaded model
with torch.no_grad():
    prediction = loaded_model(torch.tensor([[10.0]], device=device))
print(f"Prediction: {prediction.item()}")
```

PyTorch document

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

The screenshot shows the PyTorch documentation website. On the left, there's a sidebar with links like 'Community', 'Developer Notes', 'Language Bindings', and 'Python API'. Below these are links for various torch modules: torch, torch.nn, torch.nn.functional, torch.Tensor, Tensor Attributes, Tensor Views, torch.amp, torch.autograd, torch.library, torch.cpu, torch.cuda, Understanding CUDA Memory Usage, Generating a Snapshot, Using the visualizer, Snapshot API Reference, torch.mps, torch.xpu, torch.mtia, Meta device, torch.backends, torch.export, torch.distributed, and torch.distributed.tensor. A 'Screenshot' button is at the bottom of the sidebar.

The main content area has a breadcrumb navigation 'Docs > torch.nn > Sequential' and a title 'Sequential'. It contains two code snippets for the 'Sequential' class:

```
CLASS torch.nn.Sequential(*args: Module) [SOURCE]
```

```
CLASS torch.nn.Sequential(arg: OrderedDict[str, Module])
```

A descriptive text follows, stating: "A sequential container. Modules will be added to it in the order they are passed in the constructor. Alternatively, an `OrderedDict` of modules can be passed in. The `forward()` method of `Sequential` accepts any input and forwards it to the first module it contains. It then ‘chains’ outputs to inputs sequentially for each subsequent module, finally returning the output of the last module." Another paragraph explains: "The value a `Sequential` provides over manually calling a sequence of modules is that it allows treating the whole container as a single module, such that performing a transformation on the `Sequential` applies to each of the modules it stores (which are each a registered submodule of the `Sequential`)."

Below this, another text block discusses the difference between `Sequential` and `ModuleList`: "What's the difference between a `Sequential` and a `torch.nn.ModuleList`? A `ModuleList` is exactly what it sounds like—a list for storing `Module`s! On the other hand, the layers in a `Sequential` are connected in a cascading way."

Under the heading 'Example:', there is a block of Python code:

```
# Using Sequential to create a small model. When 'model' is run,
# input will first be passed to 'Conv2d(1,20,5)'. The output of
# 'Conv2d(1,20,5)' will be used as the input to the first
# 'ReLU'; the output of the first 'ReLU' will become the input
# for 'Conv2d(20,64,5)'. Finally, the output of
# 'Conv2d(20,64,5)' will be used as input to the second 'ReLU'
model = nn.Sequential(
    nn.Conv2d(1,20,5),
    nn.ReLU(),
    nn.Conv2d(20,64,5),
    nn.ReLU()
```

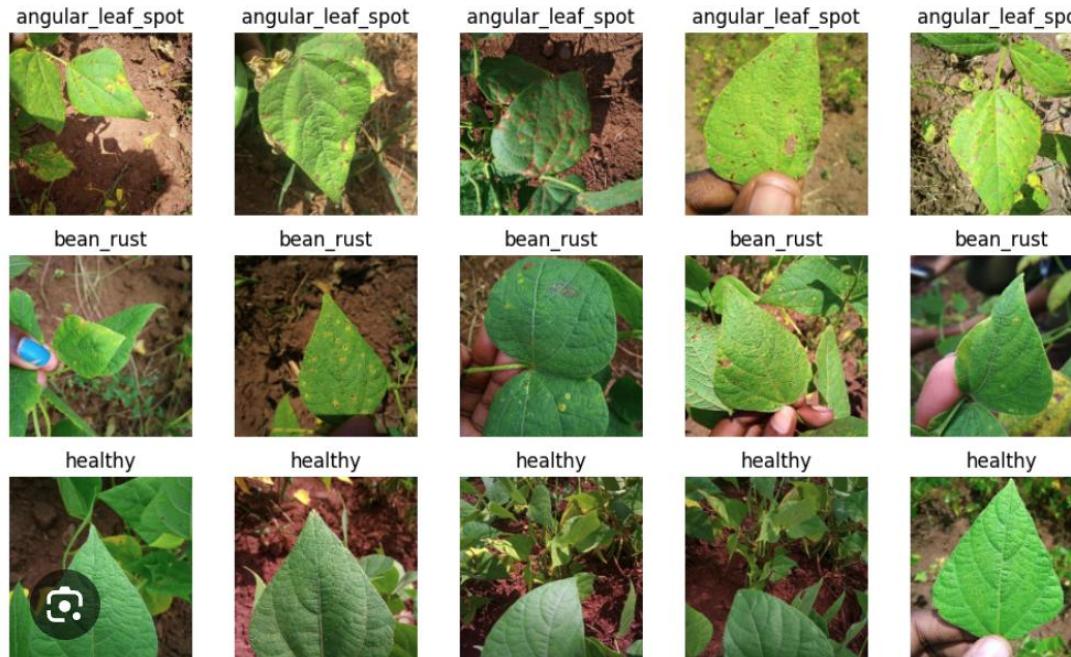
CNN Example

A simple image classification. From load data to testing model

- (Optional) Transfer Learning Example

CNN Example - Image classification

- Classify the bean's category
- <https://www.kaggle.com/datasets/therealoiise/bean-disease-dataset>

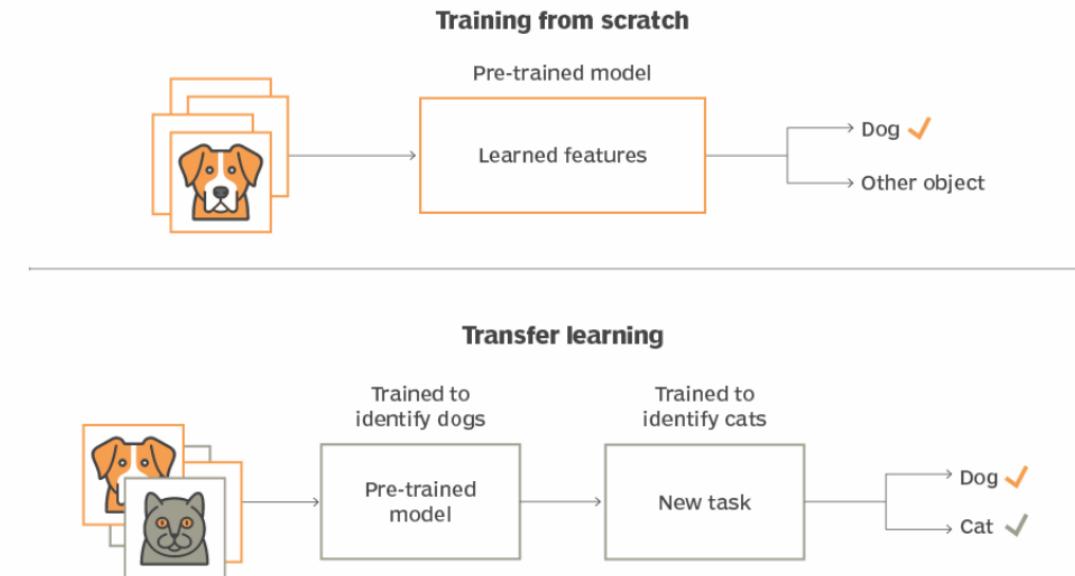


https://colab.research.google.com/drive/1i_77s0GqXkcqnqBeuvb5A0u-REQ4X6bD?usp=sharing

Transfer Learning

- **Transfer learning** is a machine learning technique where model developed for one task is reused as the starting point for a model on a second, related task.
- Useful links:
- Tutorials: <https://www.scaler.com/topics/pytorch/transfer-learning-pytorch/>
- Models Repository: <https://huggingface.co/>

How transfer learning works



Presenting your model

(Optional, just some suggestions)

1. Gradio (others: Streamlit)

Gradio

<https://www.gradio.app/guides/quickstart>

Gradio is an open-source Python package that allows you to quickly build a demo or web application for your machine learning model, API, or any arbitrary Python function. You can then share your demo with a public link in seconds using Gradio's built-in sharing features. No JavaScript, CSS, or web hosting experience needed!

```
## Source: https://www.gradio.app/guides/quickstart
import gradio as gr

def greet(name, intensity):
    return "Hello, " + name + "!" * int(intensity)

demo = gr.Interface(
    fn=greet,
    inputs=["text", "slider"],
    outputs=["text"],
)
demo.launch()
```

The screenshot shows a Gradio interface with two main sections. On the left, there is a form with a text input field containing 'Bryce' and a slider labeled 'intensity' set to 3. Below the form are two buttons: 'Clear' and 'Submit'. On the right, there is a panel titled 'output' displaying the text 'Hello, Bryce!!!' and a button labeled 'Flag'.

Tutorial: <https://www.youtube.com/watch?v=eE7CamOE-PA&t=395s>

Thank you