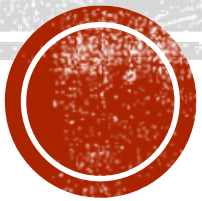
IE4424 Machine Learning Design and Application

Week 2: AI Resources and Programming



Dr Yap Kim Hui

Email: ekhyap@ntu.edu.sg

References

- Python Tutorial. https://docs.python.org/3/contents.html
- Anaconda installation. https://docs.conda.io/projects/conda/en/latest/user-guide/install/index.html
- Github. https://github.com/
- Hugging Face. https://huggingface.co/

Outline

- This session will cover the following:
  - Introduction to programming platforms / environments:
    - Lab PC (Linux)
    - Home PC / notebook (Windows)
    - Google Colab
  - Familiarization of Python, Pytorch, Jupyter Notebook, Anaconda.
  - Introduction to Github / Hugging Face.
  - Hands-on Exercises.

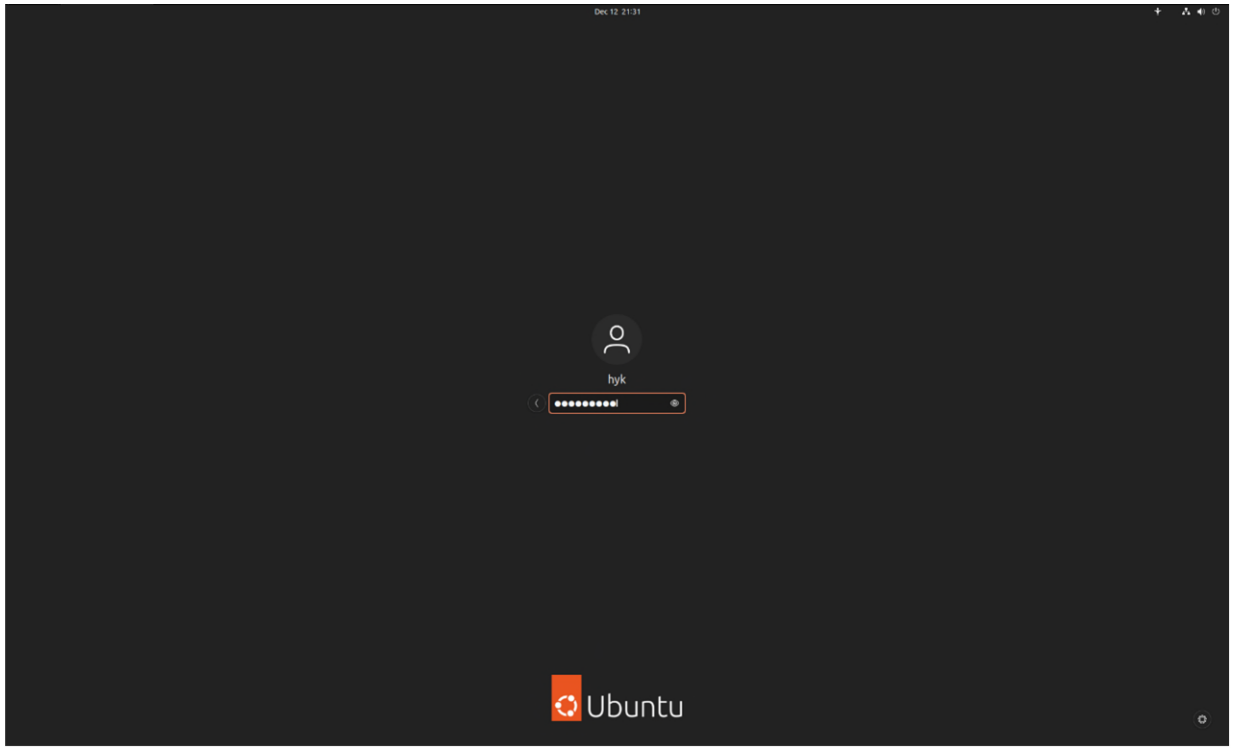
Linux Environment Guide

Linux Environment Guide

- Login with your account.
- Create a folder for codes you want to run.
- Open a terminal.
- Activate the conda environment.
- Launch Jupyter Notebook.
- Open and run your code.

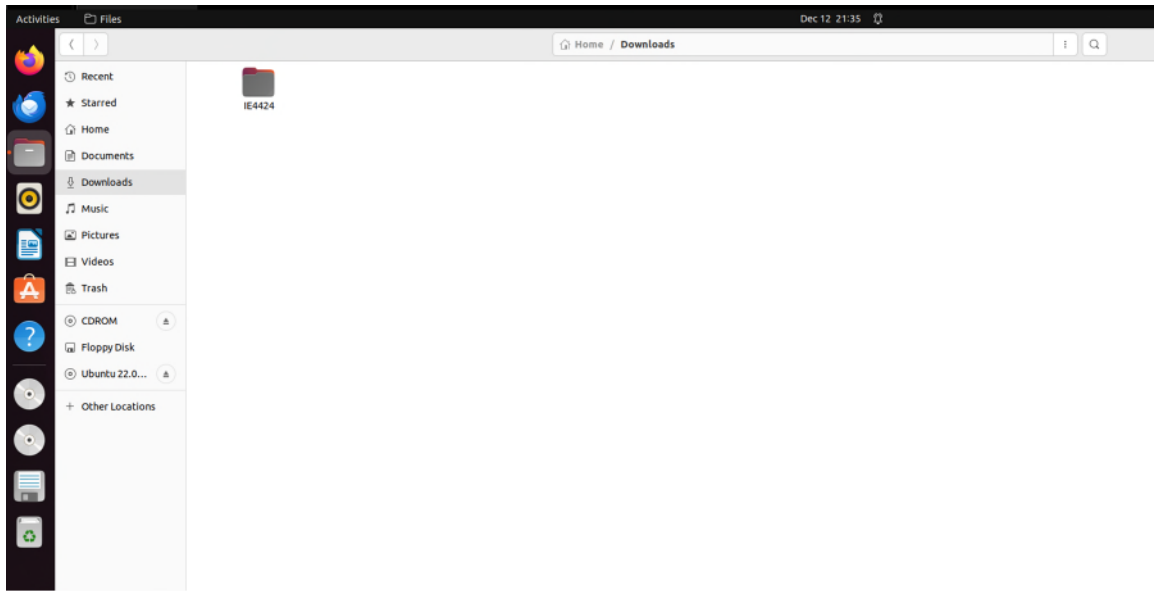
Linux Environment Guide

- Login with your account and password



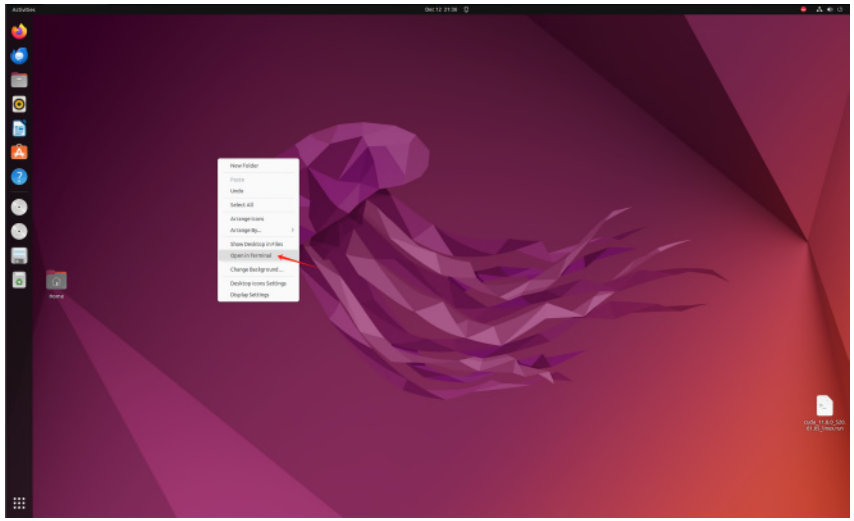
Linux Environment Guide

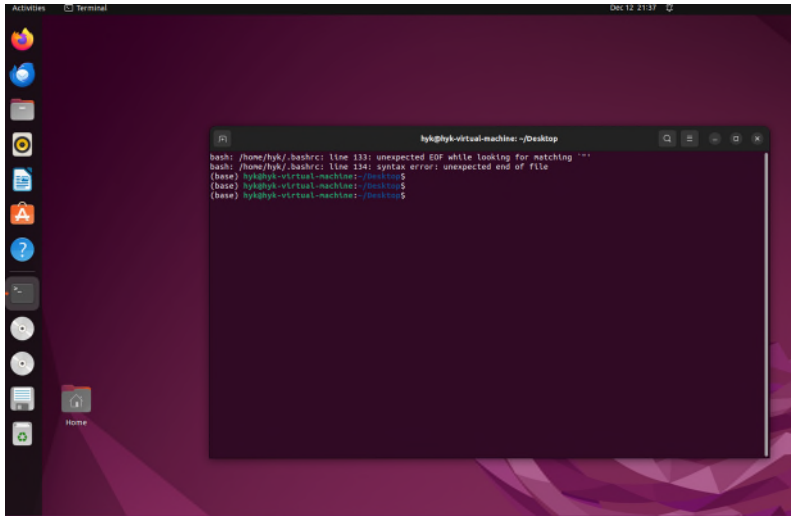
- Create a folder for codes you want to run.
- Use thumb drive to copy your codes or download your codes from online resource to the computer.



Linux Environment Guide

- Open a terminal.





Linux Environment Guide

- Activate the conda environment

```
(base) hyk@hyk-virtual-machine:~/Desktop$ conda activate ie4424
(ie4424) hyk@hyk-virtual-machine:~/Desktop$
```

- Run the following command to register ie4424 as a Jupyter notebook kernel:

  python -m ipykernel install --user --name ie4424 --display-name "ie4424"

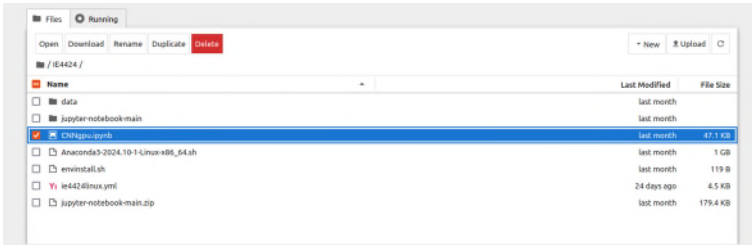
```
(ie4424) C:\Users\HYKKK>python -m ipykernel install --user --name ie4424 --display-name "ie4424"
```

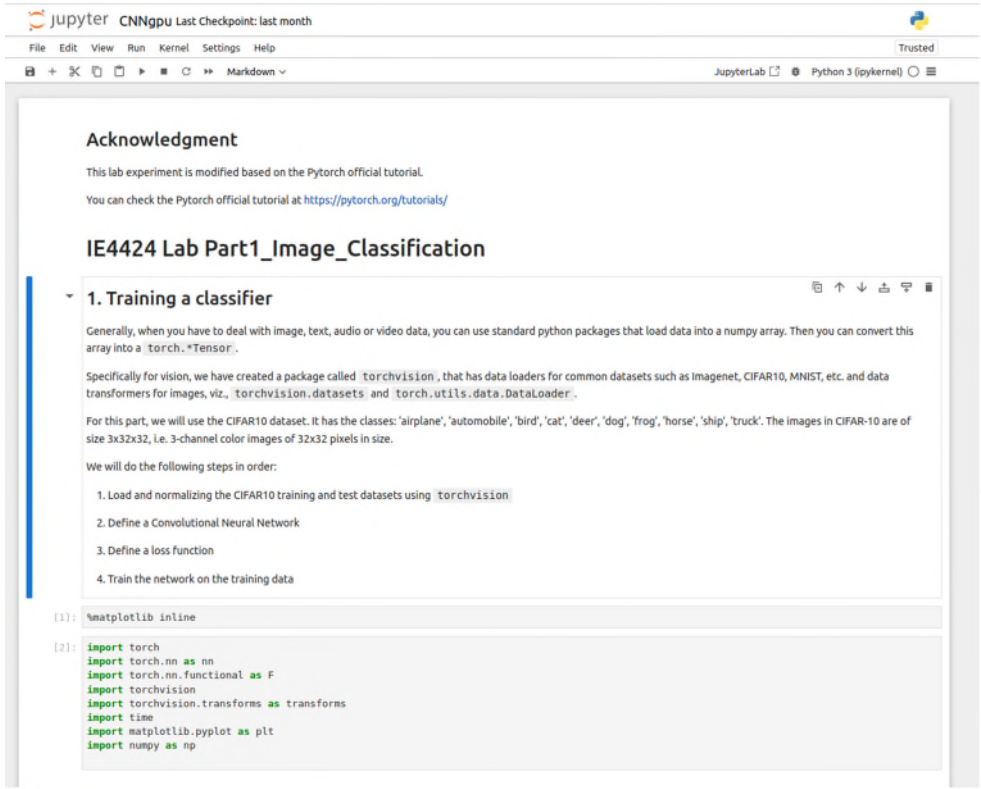
- Launch Jupyter Notebook

```
(ie4424) hyk@hyk-virtual-machine:~/Desktop$ jupyter notebook
[I 2024-12-12 21:39:12.832 ServerApp] jupyter_lsp | extension was successfully linked.
[I 2024-12-12 21:39:12.839 ServerApp] jupyter_server_terminals | extension was successfully linked.
[I 2024-12-12 21:39:12.847 ServerApp] jupyterlab | extension was successfully linked.
[I 2024-12-12 21:39:12.855 ServerApp] notebook | extension was successfully linked.
[I 2024-12-12 21:39:13.386 ServerApp] notebook_shim | extension was successfully linked.
[I 2024-12-12 21:39:13.480 ServerApp] notebook_shim | extension was successfully loaded.
```

Linux Environment Guide

- Open and run your code.





Environment Installation on Home PC / Notebook

Environment Installation on Home PC / Notebook

- Step 0 – Install conda
  (Guide: https://docs.conda.io/projects/conda/en/latest/user-guide/install/index.html).
- Step 1 – Download the environment file (ie4424_windows.yml)
- Step 2 – Open Anaconda Prompt, run the following command to create a python environment named ie4424 from the environment file.

  conda env create -f ie4424_windows.yml -n ie4424

```
C:\Users\HYKKK>conda env create -f ie4424_windows.yml -n ie4424
```

- Step 3 – Run the following command to activate the virtual environment just created.

  conda activate ie4424

Environment Installation on Home PC / Notebook

- Step 4 – Run the following command to register the newly created environment as a Jupyter notebook kernel:

  python -m ipykernel install --user --name ie4424 --display-name "ie4424"

```
C:\Users\HYKKK>conda activate ie4424

(ie4424) C:\Users\HYKKK>python -m ipykernel install --user --name ie4424 --display-name "ie4424"
```

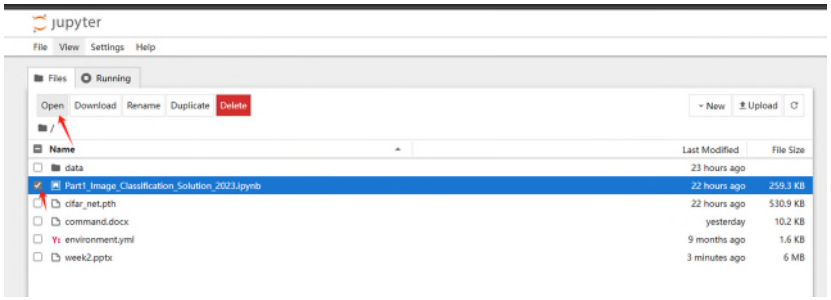
- Step 5 – Open the jupyter notebook:

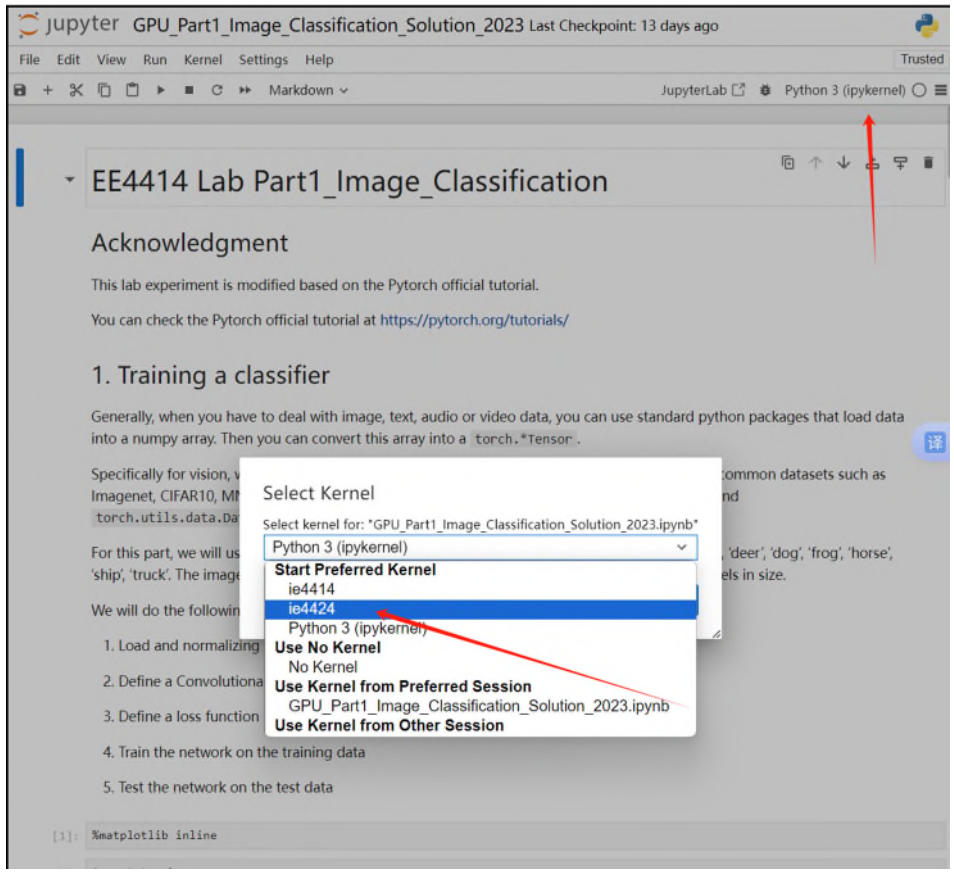
  jupyter notebook

```
(ie4424) C:\Users\HYKKK>jupyter notebook
```

Environment Installation on Home PC / Notebook

- Step 6 – Open the .ipynb file and verify that the kernel is created successfully and then select the kernel ‘ie4424’.





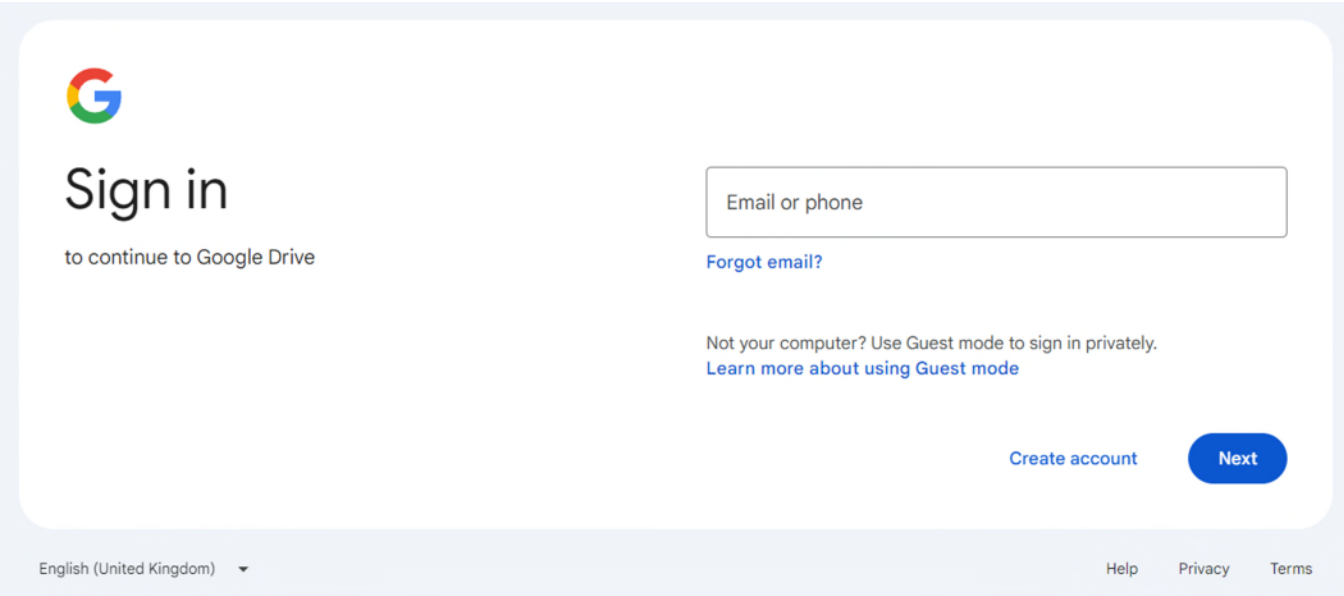
Google Colab Guide

Setting up Google Colab

1. Create Google account
2. Copy files to IE4424 folder locally.
3. Upload files / folder to Google Drive.
4. Open your Jupyter notebook in Google Colab.
5. Change runtime type.
6. (Optional) Mount Google Drive.
7. (Optional) Change working directory.
8. Run the .ipynb file.

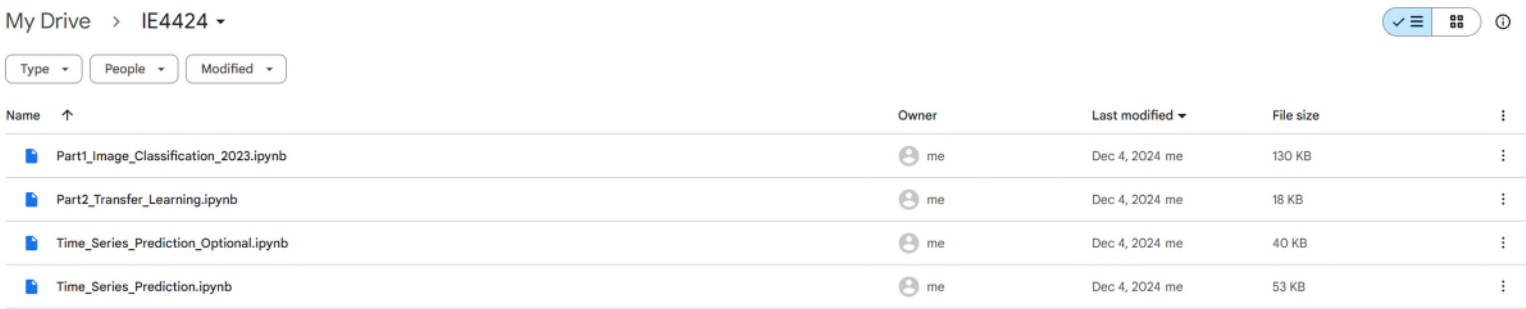
Create Google Account

- Create a Google account if you do not have one yet. See link of How to create Google account?
- Login to your own Google account and open Google Drive -> My Drive



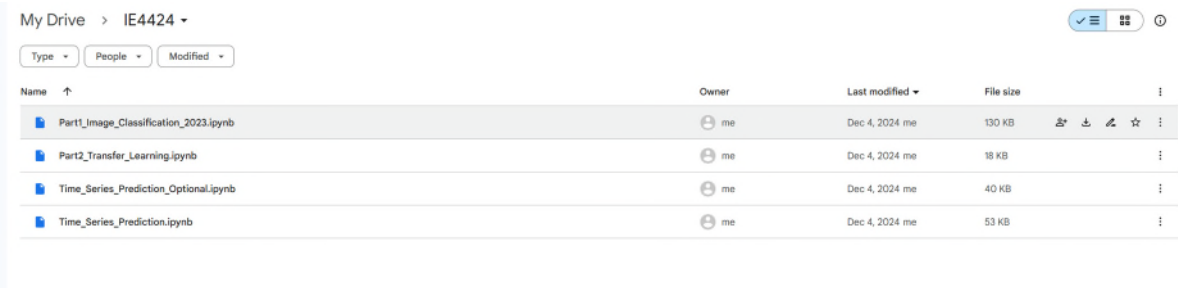
Upload Files / Folder to Google Drive

- Drag local ‘IE4424’ folder to Google Drive to upload.
- Upload ‘IE4424’ folder to ‘MyDrive’ as shown below

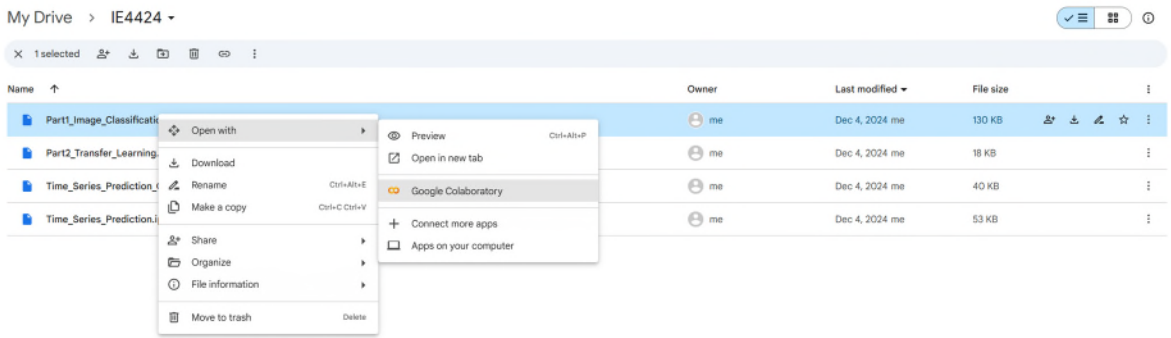


Open Jupyter Notebook in Colab

- Find your jupyter notebook file on Google Drive.

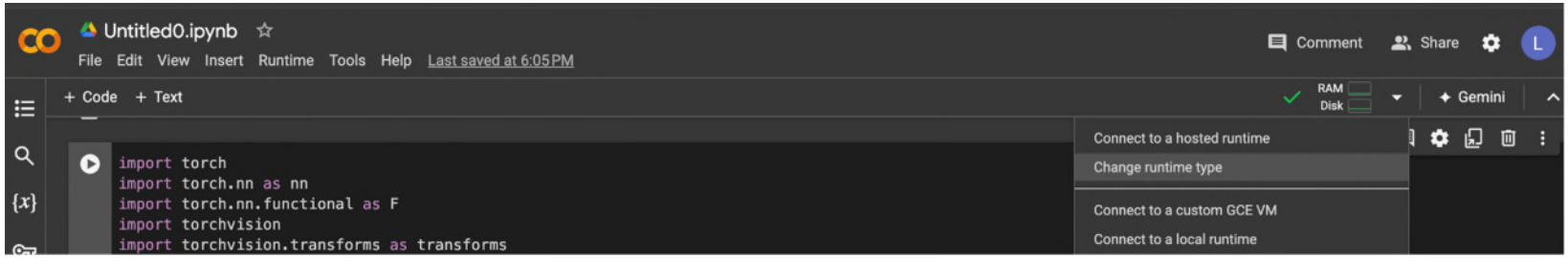


- Open the file using Google Colab.

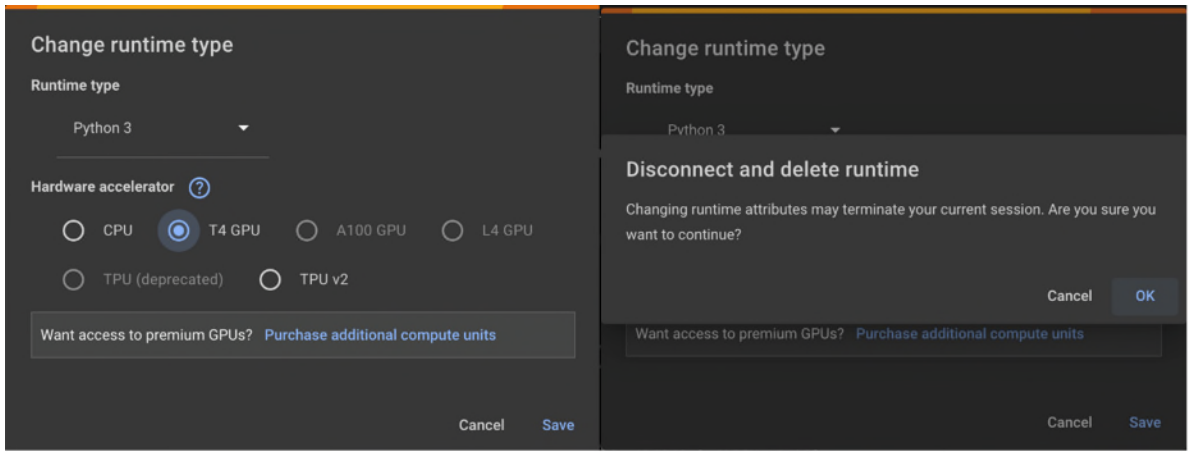


Change Runtime Type

- Open .ipynb file, select Connect -> Change runtime type.



- Choose T4 GPU -> OK -> Save.

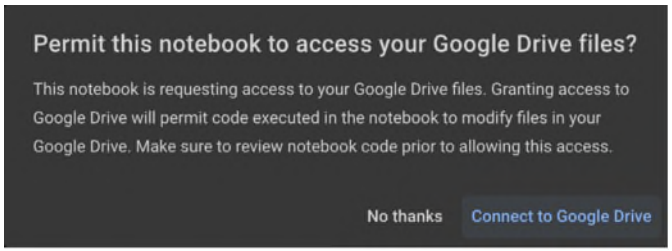


(Optional) Mount Google Drive

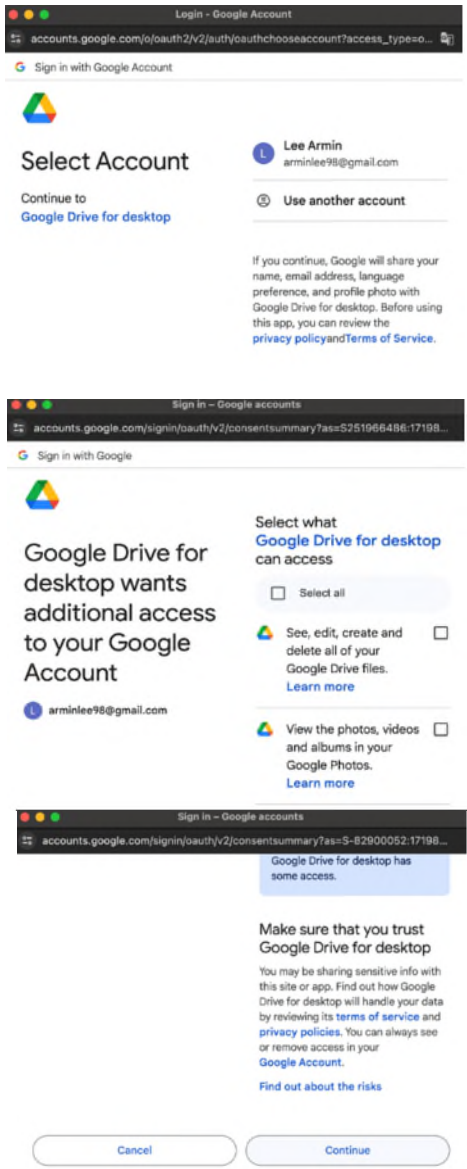
- Insert the following as the first cell and run it:

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

- Connect to Google Drive -> Continue



- No need to select any box, just scroll down to the bottom and click continue (if there is error like ‘MessageError: Error: credential propagation was unsuccessful’, you can click ‘select all’ to solve this problem)



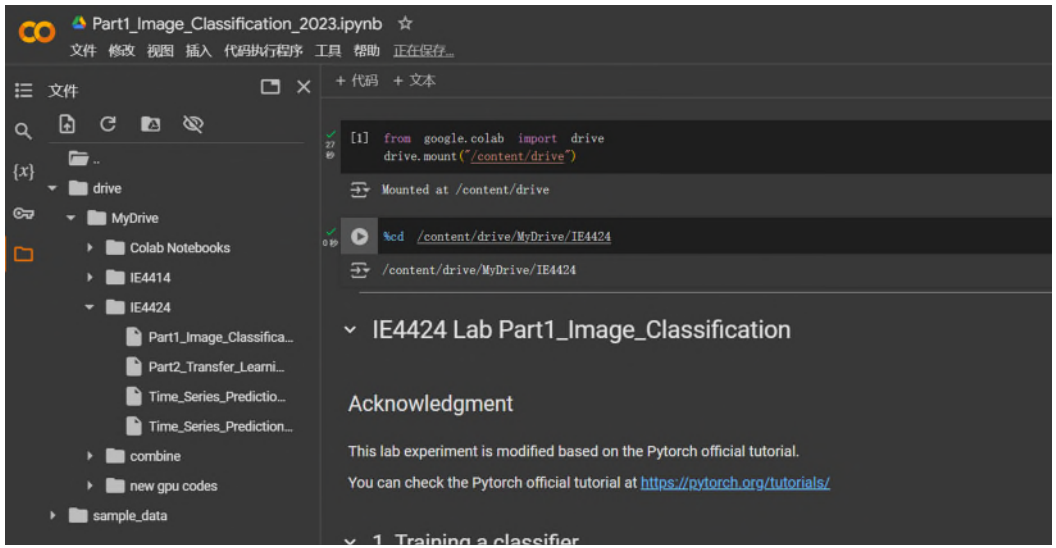
(Optional) Change Working Directory

- Insert the following as second cell and run it.
- Change current directory to IE4424 folder using following command:

```
%cd /content/drive/MyDrive/ie4424
```

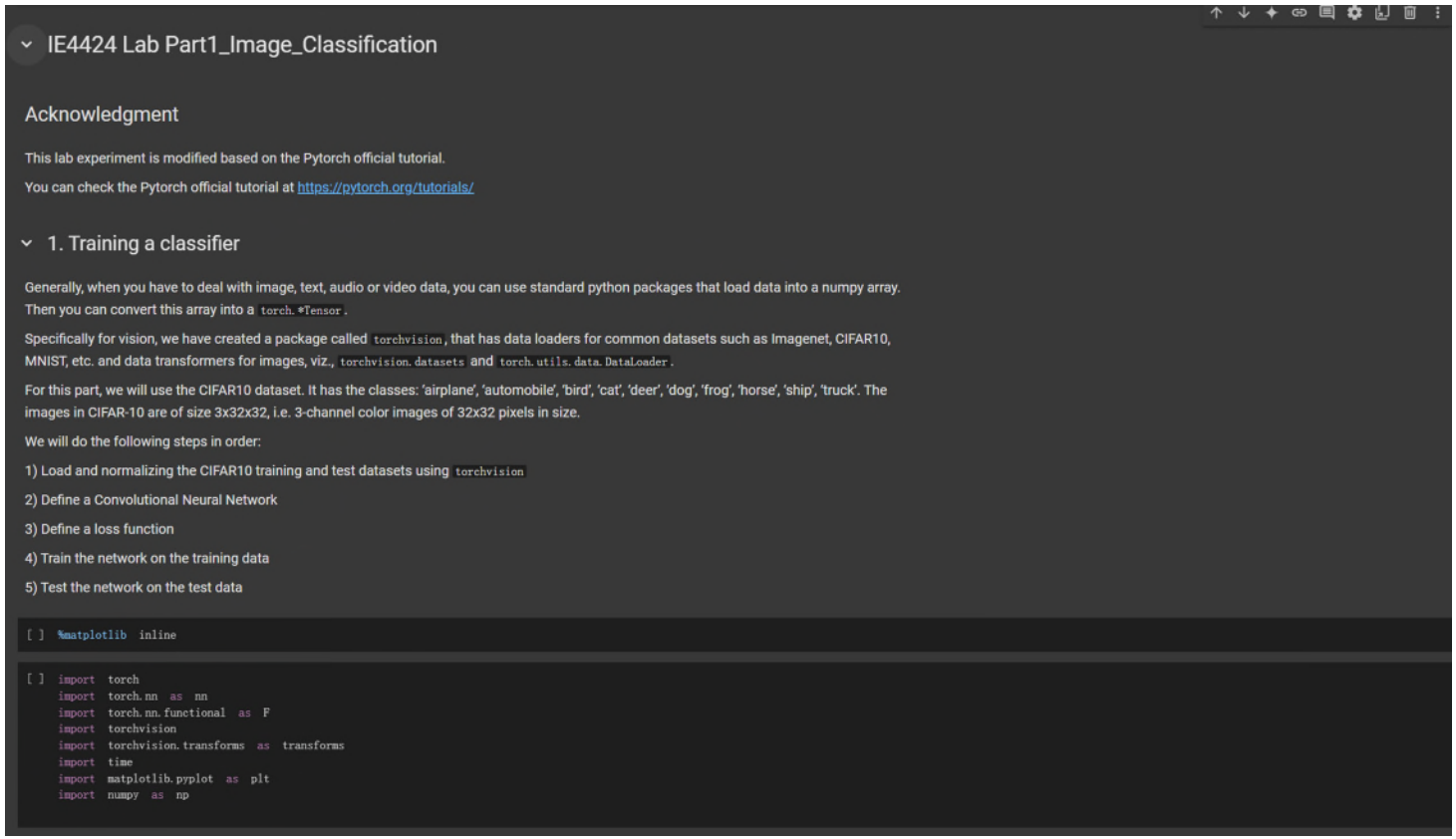
- If the directory is not the same with your own Drive, right click your working folder in Colab to copy the path and replace as:

```
%cd /your/own/path/to/ie4424
```



Running Code in Colab

- Using ‘Shift + Enter’ or menu to run the code.



Jupyter Notebook Guide

Jupyter Notebook

- Codes in Jupyter Notebook is written in cells.

[1]:
```
# 1. import library
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
# 2. creat data
x = np.linspace(0, 10, 100)  # generate 100 points from 0 to 10
y = np.sin(x)  # calculate sin(x) for all the points
```

[3]:
```
# 3. draw figure
plt.figure(figsize=(10, 5))
plt.plot(x, y, label='y = sin(x)', color='blue')
plt.title('Sine Wave')
plt.xlabel('x')
plt.ylabel('y')
plt.axhline(0, color='black',linewidth=0.5, ls='--')
plt.axvline(0, color='black',linewidth=0.5, ls='--')
plt.grid()
plt.legend()
plt.show()  # display
```

Jupyter Notebook Cheat Sheet

Cheatography

Jupyter Notebook Keyboard Shortcuts
by weidadeyue via cheatography.com/26788/cs/7602/

Command Mode (press Esc to enable)	
Enter	enter edit mode
Shift-Enter	run cell, select below
Ctrl-Enter	run cell
Alt-Enter	run cell, insert below
Y	to code
M	to markdown
R	to raw
1	to heading 1
2,3,4,5,6	to heading 2,3,4,5,6
Up/K	select cell above
Down/J	select cell below
A/B	insert cell above/below
X	cut selected cell
C	copy selected cell
Shift-V	paste cell above
V	paste cell below
Z	undo last cell deletion
D,D	delete selected cell
Shift-M	merge cell below
Ctrl-S	Save and Checkpoint
L	toggle line numbers
O	toggle output
Shift-O	toggle output scrolling
Esc	close pager
H	show keyboard shortcut help dialog
I,I	interrupt kernel
0,0	restart kernel
Space	scroll down
Shift-Space	scroll up
Shift	ignore

Edit Mode (press Enter to enable)	
Tab	code completion or indent
Shift-Tab	tooltip
Ctrl-]	indent
Ctrl-[	dedent
Ctrl-A	select all
Ctrl-Z	undo
Ctrl-Shift-Z	redo
Ctrl-Y	redo
Ctrl-Home	go to cell start
Ctrl-Up	go to cell start
Ctrl-End	go to cell end
Ctrl-Down	go to cell end
Ctrl-Left	go one word left
Ctrl-Right	go one word right
Ctrl-Backspace	delete word before
Ctrl-Delete	delete word after
Esc	command mode
Ctrl-M	command mode
Shift-Enter	run cell, select below
Ctrl-Enter	run cell
Alt-Enter	run cell, insert below
Ctrl-Shift-Subtract	split cell
Ctrl-Shift--	split cell
Ctrl-S	Save and Checkpoint
Up	move cursor up or previous cell
Down	move cursor down or next cell
Ctrl-/	toggle comment on current or selected lines

By **weidadeyue**
cheatography.com/weidadeyue/

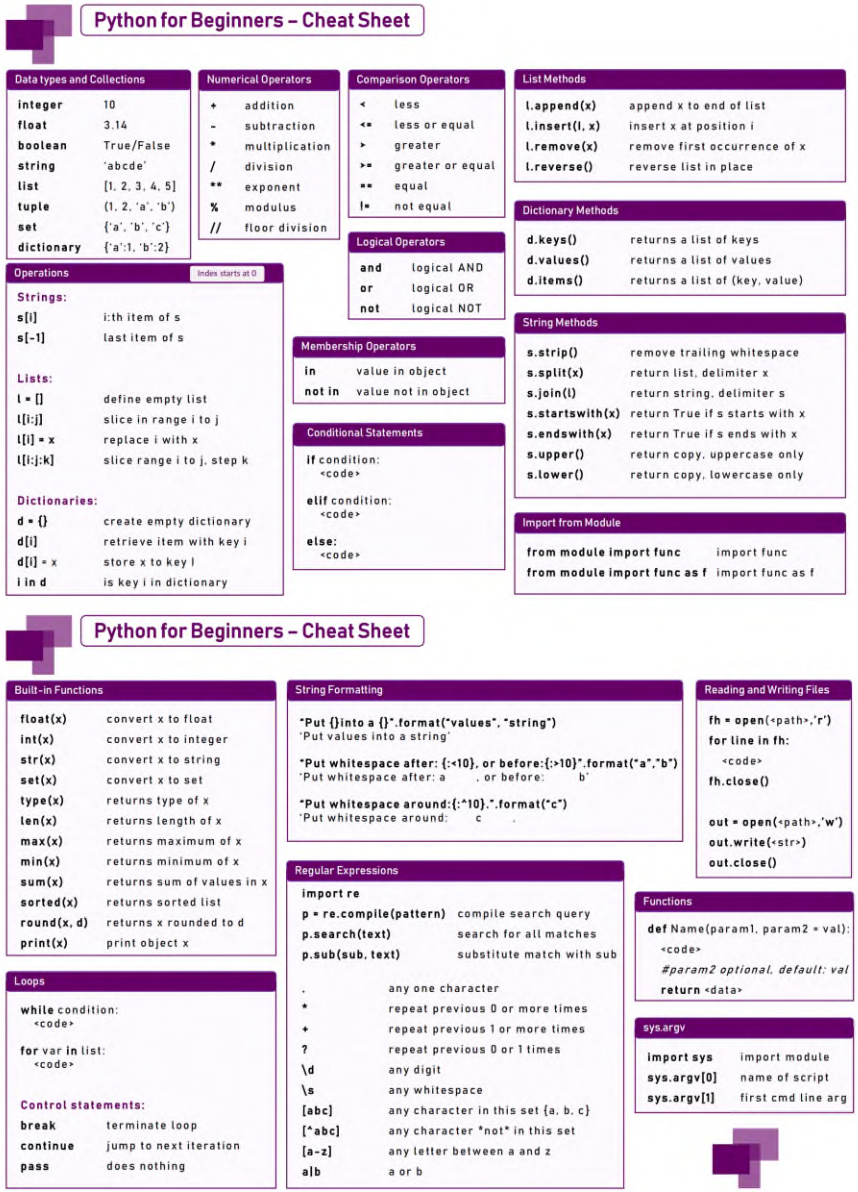
Published 21st March, 2016.
Last updated 21st March, 2016.
Page 1 of 1.

Python & Pytorch

What is Python?

- Python is an easy-to-read, high-level programming language.
- Key features:
  - Interpreted: Code is executed line by line, making debugging easier.
  - Versatile: Used in web development, data analysis, artificial intelligence, scientific computing, and more.
  - Rich Libraries: Extensive standard libraries and frameworks for various applications.
  - Community Support: Strong community with abundant resources and documentation.
- Python Tutorial: https://docs.python.org/3/contents.html

Python Cheat Sheet



Python for Beginners – Cheat Sheet

Data types and Collections

integer	10
float	3.14
boolean	True/False
string	'abcde'
list	[1, 2, 3, 4, 5]
tuple	(1, 2, 'a', 'b')
set	{'a', 'b', 'c'}
dictionary	{'a':1, 'b':2}

Numerical Operators

+	addition
-	subtraction
*	multiplication
/	division
**	exponent
%	modulus
//	floor division

Comparison Operators

<	less
<=	less or equal
>	greater
>=	greater or equal
==	equal
!=	not equal

List Methods

l.append(x)	append x to end of list
l.insert(I, x)	insert x at position i
l.remove(x)	remove first occurrence of x
l.reverse()	reverse list in place

Operations

Index starts at 0

Strings:

s[i]	i:th item of s
s[-1]	last item of s

Lists:

l = []	define empty list
l[i:j]	slice in range i to j
l[i] = x	replace i with x
l[i:j:k]	slice range i to j, step k

Dictionaries:

d = {}	create empty dictionary
d[i]	retrieve item with key i
d[i] = x	store x to key I
i in d	is key i in dictionary

Logical Operators

and	logical AND
or	logical OR
not	logical NOT

Dictionary Methods

d.keys()	returns a list of keys
d.values()	returns a list of values
d.items()	returns a list of (key, value)

Membership Operators

in	value in object
not in	value not in object

String Methods

s.strip()	remove trailing whitespace
s.split(x)	return list, delimiter x
s.join(l)	return string, delimiter s
s.startswith(x)	return True if s starts with x
s.endswith(x)	return True if s ends with x
s.upper()	return copy, uppercase only
s.lower()	return copy, lowercase only

Conditional Statements

```
if condition:
    <code>

elif condition:
    <code>

else:
    <code>
```

Import from Module

from module import func	import func
from module import func as f	import func as f

Python for Beginners – Cheat Sheet

Built-in Functions

float(x)	convert x to float
int(x)	convert x to integer
str(x)	convert x to string
set(x)	convert x to set
type(x)	returns type of x
len(x)	returns length of x
max(x)	returns maximum of x
min(x)	returns minimum of x
sum(x)	returns sum of values in x
sorted(x)	returns sorted list
round(x, d)	returns x rounded to d
print(x)	print object x

String Formatting

```
"Put {}into a {}".format("values", "string")
'Put values into a string'

"Put whitespace after: {:<10}, or before:{:>10}".format("a","b")
'Put whitespace after: a         , or before:          b'

"Put whitespace around:{:^10}.".format("c")
'Put whitespace around:     c      .
```

Reading and Writing Files

```
fh = open(<path>,'r')
for line in fh:
    <code>
fh.close()

out = open(<path>,'w')
out.write(<str>)
out.close()
```

Regular Expressions

```
import re
```

p = re.compile(pattern)	compile search query
p.search(text)	search for all matches
p.sub(sub, text)	substitute match with sub

.	any one character
*	repeat previous 0 or more times
+	repeat previous 1 or more times
?	repeat previous 0 or 1 times
\d	any digit
\s	any whitespace
[abc]	any character in this set {a, b, c}
[^abc]	any character *not* in this set
[a-z]	any letter between a and z
a|b	a or b

Functions

```
def Name(param1, param2 = val):
    <code>
    #param2 optional, default: val
    return <data>
```

Loops

```
while condition:
    <code>

for var in list:
    <code>
```

Control statements:

break	terminate loop
continue	jump to next iteration
pass	does nothing

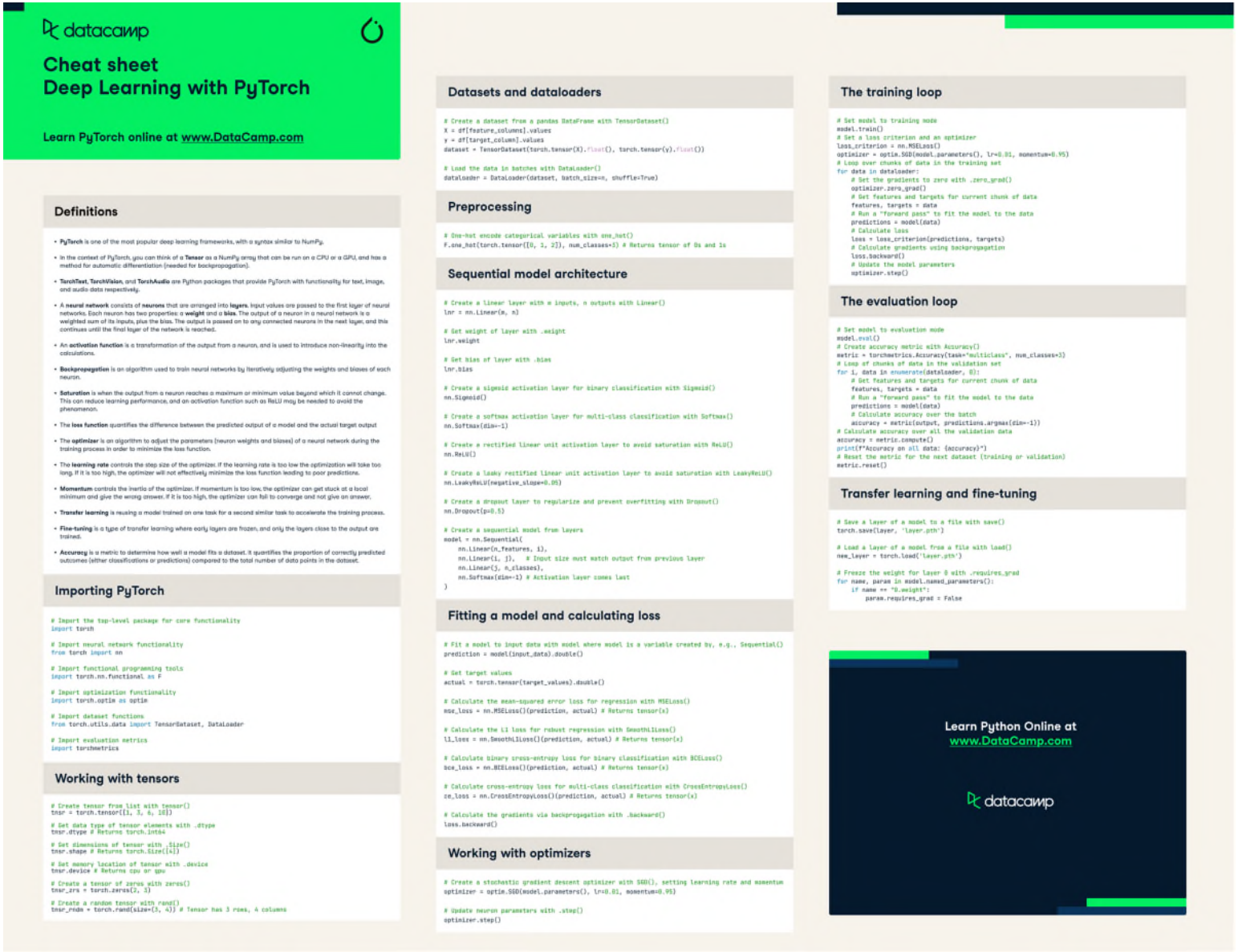
sys.argv

import sys	import module
sys.argv[0]	name of script
sys.argv[1]	first cmd line arg

What is Pytorch?

- PyTorch is an open-source machine learning library.
- Key features:
  - Dynamic Computation: Supports dynamic computation graphs for flexibility in model building.
  - Deep Learning: Primarily used for deep learning applications.
  - GPU Acceleration: Utilize GPU for faster computation.
  - Community and Ecosystem: Strong community support and a rich ecosystem of tools and libraries.
- Pytorch Tutorial. https://pytorch.org/tutorials/

Pytorch Cheat Sheet



datacamp

Cheat sheet
Deep Learning with PyTorch

Learn PyTorch online at www.DataCamp.com

Definitions

- **PyTorch** is one of the most popular deep learning frameworks, with a syntax similar to NumPy.
- In the context of PyTorch, you can think of a **Tensor** as a NumPy array that can be run on a CPU or a GPU, and has a method for automatic differentiation (needed for backpropagation).
- **TorchText**, **TorchVision**, and **TorchAudio** are Python packages that provide PyTorch with functionality for text, image, and audio data respectively.
- A **neural network** consists of **neurons** that are arranged into **layers**. Input values are passed to the first layer of neural networks. Each neuron has two properties: a **weight** and a **bias**. The output of a neuron in a neural network is a weighted sum of its inputs, plus the bias. The output is passed on to any connected neurons in the next layer, and this continues until the final layer of the network is reached.
- An **activation function** is a transformation of the output from a neuron, and is used to introduce non-linearity into the calculations.
- **Backpropagation** is an algorithm used to train neural networks by iteratively adjusting the weights and biases of each neuron.
- **Saturation** is when the output from a neuron reaches a maximum or minimum value beyond which it cannot change. This can reduce learning performance, and an activation function such as ReLU may be needed to avoid the phenomenon.
- The **loss function** quantifies the difference between the predicted output of a model and the actual target output
- The **optimizer** is an algorithm to adjust the parameters (neuron weights and biases) of a neural network during the training process in order to minimize the loss function.
- The **learning rate** controls the step size of the optimizer. If the learning rate is too low the optimization will take too long. If it is too high, the optimizer will not effectively minimize the loss function leading to poor predictions.
- **Momentum** controls the inertia of the optimizer. If momentum is too low, the optimizer can get stuck at a local minimum and give the wrong answer. If it is too high, the optimizer can fail to converge and not give an answer.
- **Transfer learning** is reusing a model trained on one task for a second similar task to accelerate the training process.
- **Fine-tuning** is a type of transfer learning where early layers are frozen, and only the layers close to the output are trained.
- **Accuracy** is a metric to determine how well a model fits a dataset. It quantifies the proportion of correctly predicted outcomes (either classifications or predictions) compared to the total number of data points in the dataset.

Importing PyTorch

```
# Import the top-level package for core functionality
import torch

# Import neural network functionality
from torch import nn

# Import functional programming tools
import torch.nn.functional as F

# Import optimization functionality
import torch.optim as optim

# Import dataset functions
from torch.utils.data import TensorDataset, DataLoader

# Import evaluation metrics
import torchmetrics
```

Working with tensors

```
# Create tensor from list with tensor()
tnsr = torch.tensor([1, 3, 6, 10])

# Get data type of tensor elements with .dtype
tnsr.dtype # Returns torch.int64

# Get dimensions of tensor with .size()
tnsr.shape # Returns torch.Size([4])

# Get memory location of tensor with .device
tnsr.device # Returns cpu or gpu

# Create a tensor of zeros with zeros()
tnsr_zrs = torch.zeros(2, 3)

# Create a random tensor with rand()
tnsr_rnds = torch.rand(size=(3, 4)) # Tensor has 3 rows, 4 columns
```

Datasets and dataloaders

```
# Create a dataset from a pandas DataFrame with TensorDataset()
X = df[feature_columns].values
y = df[target_column].values
dataset = TensorDataset(torch.tensor(X).float(), torch.tensor(y).float())

# Load the data in batches with DataLoader()
dataLoader = DataLoader(dataset, batch_size=n, shuffle=True)
```

Preprocessing

```
# One-hot encode categorical variables with one_hot()
F.one_hot(torch.tensor([0, 1, 2]), num_classes=3) # Returns tensor of 0s and 1s
```

Sequential model architecture

```
# Create a linear layer with m inputs, n outputs with Linear()
lnr = nn.Linear(m, n)

# Get weight of layer with .weight
lnr.weight

# Get bias of layer with .bias
lnr.bias

# Create a sigmoid activation layer for binary classification with Sigmoid()
nn.Sigmoid()

# Create a softmax activation layer for multi-class classification with Softmax()
nn.Softmax(dim=-1)

# Create a rectified linear unit activation layer to avoid saturation with ReLU()
nn.ReLU()

# Create a leaky rectified linear unit activation layer to avoid saturation with LeakyReLU()
nn.LeakyReLU(negative_slope=0.05)

# Create a dropout layer to regularize and prevent overfitting with Dropout()
nn.Dropout(p=0.5)

# Create a sequential model from layers
model = nn.Sequential(
    nn.Linear(n_features, 1),
    nn.Linear(1, 1),   # Input size must match output from previous layer
    nn.Linear(1, n_classes),
    nn.Softmax(dim=-1) # Activation layer comes last
)
```

Fitting a model and calculating loss

```
# Fit a model to input data with model where model is a variable created by, e.g., Sequential()
prediction = model(input_data).double()

# Get target values
actual = torch.tensor(target_values).double()

# Calculate the mean-squared error loss for regression with MSELoss()
mse_loss = nn.MSELoss()(prediction, actual) # Returns tensor(x)

# Calculate the L1 loss for robust regression with SmoothL1Loss()
l1_loss = nn.SmoothL1Loss()(prediction, actual) # Returns tensor(x)

# Calculate binary cross-entropy loss for binary classification with BCELoss()
bce_loss = nn.BCELoss()(prediction, actual) # Returns tensor(x)

# Calculate cross-entropy loss for multi-class classification with CrossEntropyLoss()
ce_loss = nn.CrossEntropyLoss()(prediction, actual) # Returns tensor(x)

# Calculate the gradients via backpropagation with .backward()
loss.backward()
```

Working with optimizers

```
# Create a stochastic gradient descent optimizer with SGD(), setting learning rate and momentum
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.95)

# Update neuron parameters with .step()
optimizer.step()
```

The training loop

```
# Set model to training mode
model.train()
# Set a loss criterion and an optimizer
loss_criterion = nn.MSELoss()
optimizer = optim.SGD(model.parameters(), lr=0.01, momentum=0.95)
# Loop over chunks of data in the training set
for data in dataloader:
    # Set the gradients to zero with .zero_grad()
    optimizer.zero_grad()
    # Get features and targets for current chunk of data
    features, targets = data
    # Run a "forward pass" to fit the model to the data
    predictions = model(data)
    # Calculate loss
    loss = loss_criterion(predictions, targets)
    # Calculate gradients using backpropagation
    loss.backward()
    # Update the model parameters
    optimizer.step()
```

The evaluation loop

```
# Set model to evaluation mode
model.eval()
# Create accuracy metric with Accuracy()
metric = torchmetrics.Accuracy(task="multiclass", num_classes=3)
# Loop of chunks of data in the validation set
for i, data in enumerate(dataloader, 0):
    # Get features and targets for current chunk of data
    features, targets = data
    # Run a "forward pass" to fit the model to the data
    predictions = model(data)
    # Calculate accuracy over the batch
    accuracy = metric(output, predictions.argmax(dim=-1))
# Calculate accuracy over all the validation data
accuracy = metric.compute()
print(f"Accuracy on all data: {accuracy}")
# Reset the metric for the next dataset (training or validation)
metric.reset()
```

Transfer learning and fine-tuning

```
# Save a layer of a model to a file with save()
torch.save(layer, 'layer.pth')

# Load a layer of a model from a file with load()
new_layer = torch.load('layer.pth')

# Freeze the weight for layer 0 with .requires_grad
for name, param in model.named_parameters():
    if name == "0.weight":
        param.requires_grad = False
```

Learn Python Online at
www.DataCamp.com

datacamp

Github & Hugging Face

What is Github?

- GitHub is a developer platform that allows developers to create, store, manage, and share their code.
- Key features:
  - Version Control: Track changes in code.
  - Collaboration: Multiple users can work together on projects.
  - Open Source: Host many open-source projects.
  - Project Management: Tools for issue tracking and planning.
  - Integration: Works with various development tools.
- Github. https://github.com/

What is Hugging Face?

- Hugging Face is an open-source community and platform focused on natural language processing (NLP) and deep learning.
- Key features:
  - Model Hub: Offer pre-trained models, especially for NLP (e.g., Transformers).
  - User-Friendly: Easy-to-use APIs for quick deployment.
  - Community Support: Active community sharing models and datasets.
  - Research Focus: Drive innovation in NLP and machine learning.
  - Multilingual: Support various languages and tasks like translation and sentiment analysis.
- Hugging Face. https://huggingface.co/

Hands On Exercises

Familiarization Exercises

- Two hands-on exercises
  - Basic python code.
  - Training CNN for image classification.

Basic Python Code

- The exercise covers:
  - Plotting graph
  - Basic arithmetic and function definition
  - Data visualization
  - Array manipulation

Basic Python Code: Plotting Graph

[1]:
```python
# 1. import library
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```python
# 2. creat data
x = np.linspace(0, 10, 100)  # generate 100 points from 0 to 10
y = np.sin(x)  # calculate sin(x) for all the points
```

[3]:
```python
# 3. draw figure
plt.figure(figsize=(10, 5))
plt.plot(x, y, label='y = sin(x)', color='blue')
plt.title('Sine Wave')
plt.xlabel('x')
plt.ylabel('y')
plt.axhline(0, color='black',linewidth=0.5, ls='--')
plt.axvline(0, color='black',linewidth=0.5, ls='--')
plt.grid()
plt.legend()
plt.show()  # display
```

Basic Python Code: Basic Arithmetic and Function Definition

```
# 4. basic calculation
a = 10
b = 5
sum_ab = a + b
product_ab = a * b
```

```
print("Sum:", sum_ab)
print("Product:", product_ab)
```

```
# 5. define a function
def greet(name):
    return f"Hello, {name}!"
```

```
# Call the function
print(greet("World"))
```

Basic Python Code: Data Visualization

```python
import pandas as pd

data = {
    'name': ['Alice', 'Bob', 'Charlie', 'David'],
    'age': [24, 30, 22, 35],
    'city': ['Beijing', 'Shanghai', 'Guangzhou', 'Shenzhen']
}

df = pd.DataFrame(data)

print("Content: ")
print(df)
```

```python
# draw a bar chart
df.plot(kind='bar', x='name', y='age', color='orange')
plt.title('Data')
plt.ylabel('Age')
plt.show()
```

Basic Python Code: Array Manipulation

```
# create array
array1 = np.array([1, 2, 3, 4, 5])
array2 = np.array([6, 7, 8, 9, 10])
```

```
print("array 1:", array1)
print("array 2:", array2)
```

```
array 1: [1 2 3 4 5]
array 2: [ 6  7  8  9 10]
```

```
sum_array = array1 + array2
print("sum result:", sum_array)
```

```
sum result: [ 7  9 11 13 15]
```

```
product_array = array1 * array2
print("product result:", product_array)
```

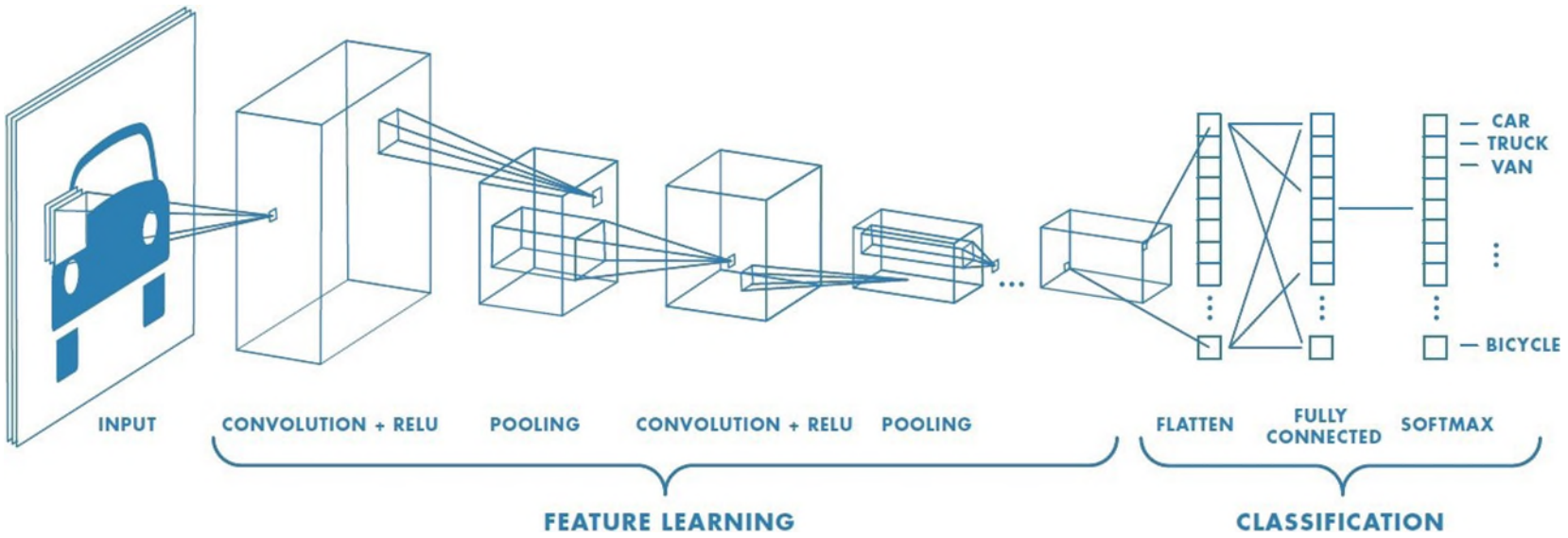
```
product result: [ 6 14 24 36 50]
```

```
squared_array = array1 ** 2
print("squared result:", squared_array)
```

Training CNN for Image Classification

- This exercise covers:
  - Data loader
  - Pytorch CNN implementation
  - Model training

CNN Architecture



Source: “A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way”, towardsdatascience.com

Data Loader

- Define Dataset

```
transform = transforms.Compose(
    [transforms.ToTensor(),
     transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5))])

trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
```

- Construct Dataloader

```
trainloader = torch.utils.data.DataLoader(trainset, batch_size=4,
                                          shuffle=True, num_workers=0)
```

Pytorch CNN Implementation

```
import torch.nn as nn
import torch.nn.functional as F
```

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 16, 3, padding=1)
        self.fc1 = nn.Linear(16 * 8 * 8, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
```

Layer Definition

```
    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = x.view(-1, 16 * 8 * 8)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Feed Forward Logic

```
net = Net().to(device)  #2024 modification CPU > GPU
```

Optimizer

```
import torch.optim as optim

criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), lr=0.001, momentum=0.9)
```

Model Training

```
t1 = time.time()
loss_list = []
for epoch in range(2):  # loop over the dataset multiple times

    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        # get the inputs; data is a list of [inputs, labels]
        inputs, labels = data

        # zero the parameter gradients
        optimizer.zero_grad()

        # forward + backward + optimize
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()

        # print statistics
        running_loss += loss.item()
        if i % 2000 == 1999:    # print every 2000 mini-batches
            print('[%d, %5d] loss: %.3f' %
                  (epoch + 1, i + 1, running_loss / 2000))
            loss_list.append(running_loss/2000)
            running_loss = 0.0

t2 = time.time()
print('Finished Training')
print('Training time:'+str(t2-t1))
```

```
[1,  2000] loss: 2.224
[1,  4000] loss: 1.901
[1,  6000] loss: 1.699
[1,  8000] loss: 1.554
[1, 10000] loss: 1.471
[1, 12000] loss: 1.391
[2,  2000] loss: 1.294
[2,  4000] loss: 1.272
[2,  6000] loss: 1.262
[2,  8000] loss: 1.228
[2, 10000] loss: 1.208
[2, 12000] loss: 1.192
Finished Training
Training time:92.16319823265076
```

Summary

- This session covers the following:
  - Introduction to programming platforms / environments:
    - Lab PC (Linux)
    - Home PC / notebook (Windows)
    - Google Colab
  - Familiarization of Python, Pytorch, Jupyter Notebook, Anaconda.
  - Introduction to Github / Hugging Face.
  - Hands-on Exercises.