Term Project

Deep Learning

한양대학교
HANYANG UNIVERSITY

DSLAB
데이터사이언스 연구실

Task

- EL-Image Classification
  - Electroluminescence (EL) images are used to detect defects or evaluate quality of solar panels and LEDs.
  - We are going to categorize these images into normal and fault types by training a neural network.

- Dataset
  - Input size: 100x200
  - Classes: Normal/Fault

	Normal	Fault
Train	8000	2014
Test	2000	503

Normal cell

Fault cell

Submission

- Due: 24.11.29 (Fri) 23:59
- Submit file
  - Report (studentid_name.pdf)
  - Source code and state_dict(model.pth)
    - Must submit in student_id.zip / ex) 2024111111.zip
    - Size of state_dict ≤ 60MB
  - Don't forget to remove all the training images when you submit the zip file!

Test Process

- Execute run.py
  - Test data will be in test_data folder.
  - You must designate trained model path in argument.

```python
if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='2024 DL Term Project')
    parser.add_argument('--load-model', default='./checkpoints/model.pth', help="Model's state_dict")
    parser.add_argument('--batch-size', default=1, help='test loader batch size')
    parser.add_argument('--fault-dir', default='./test_data/fault_image', help='Directory for fault images')
    parser.add_argument('--normal-dir', default='./test_data/normal_image', help='Directory for normal images')
```

- Result.txt will be generated.
- We will rank based on the result.txt file
- Do not modify run.py

Skeleton code

- Term_Project
  - checkpoints
  - term_project_train_data
  - utils
    - _utils.py
  - model.py
  - test.py
  - train.py
  - run.py

execute run.py

- Term_Project
  - checkpoints
  - term_project_train_data
  - utils
    - _utils.py
  - model.py
  - test.py
  - train.py
  - run.py
  - result.txt

Grading

- 100 points total
  - Performance: 60 points
    - Points: 60-(your rank)
  - Report: 40 points

  - Non-runnable code will be big minus point!
  - The run.py file must only perform inference, not training => huge minus!

Google Colaboratory

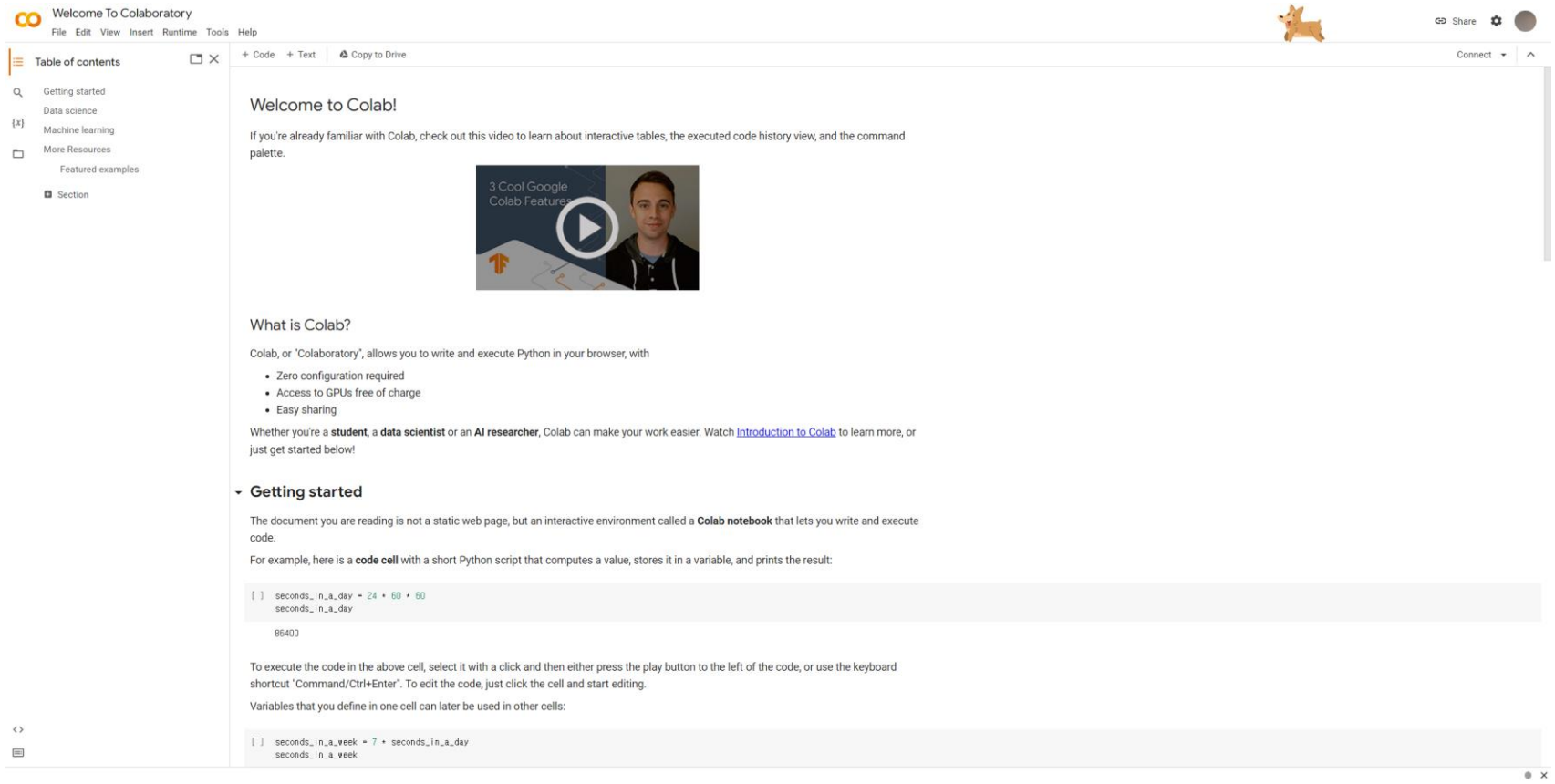
Deep Learning

한양대학교
HANYANG UNIVERSITY

DSLAB
데이터사이언스 연구실

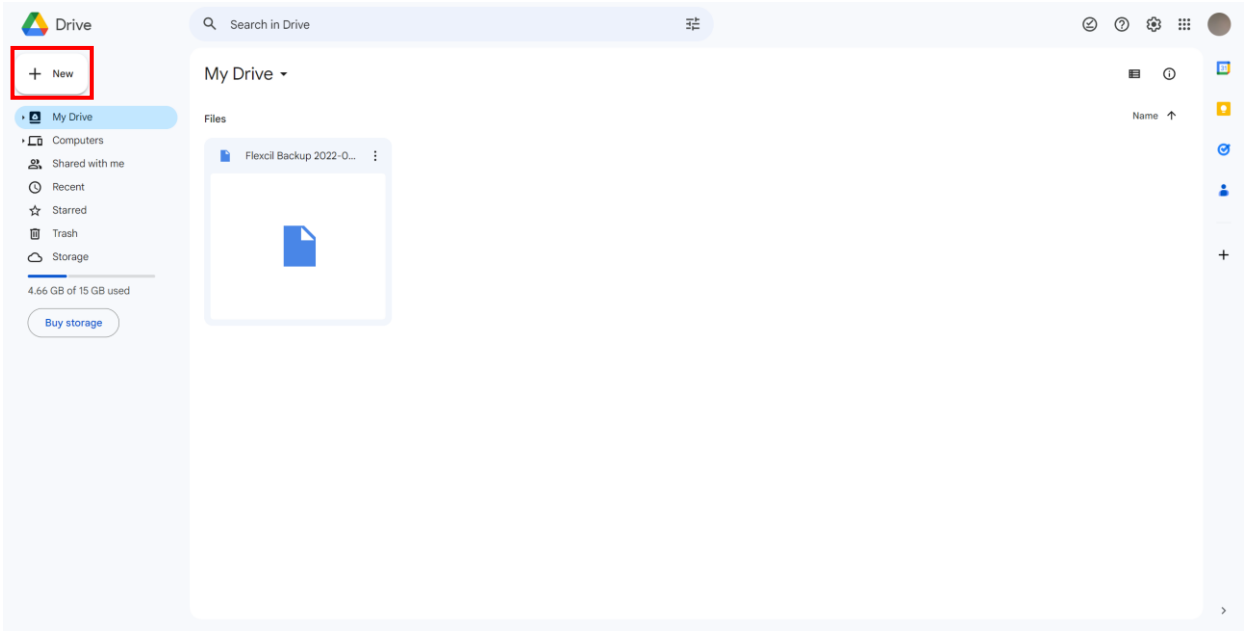
Colab

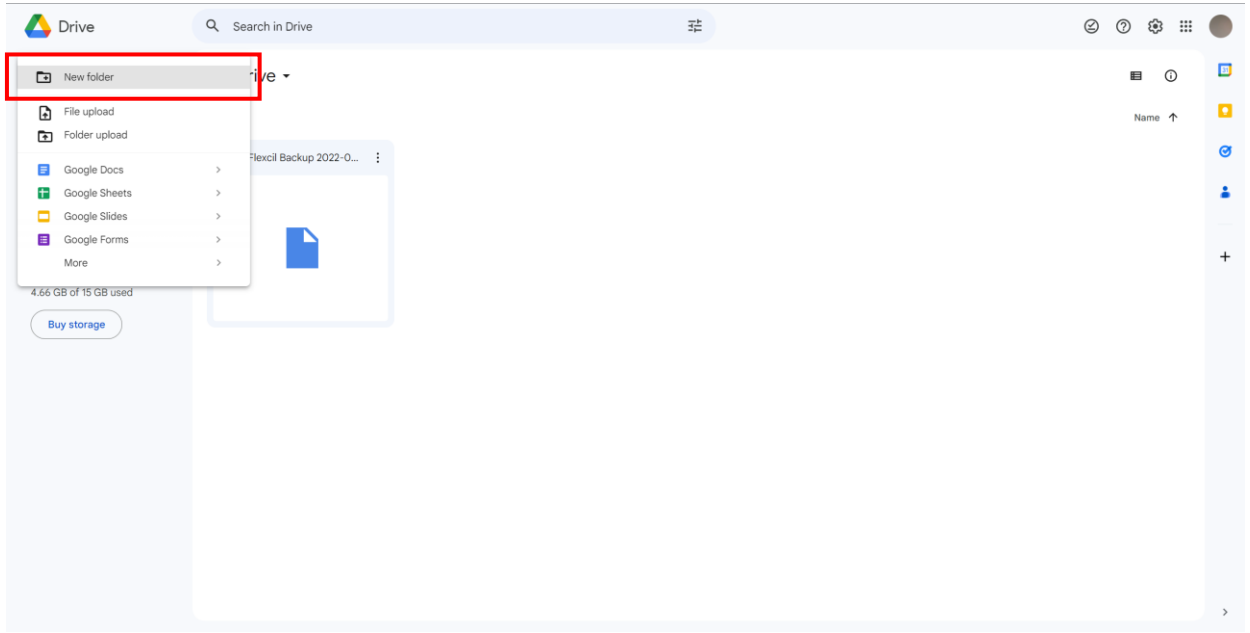
- Colab, or "Colaboratory", allows you to write and execute Python in your browser, with
  - Zero configuration required
  - Access to GPUs free of charge



Term Project

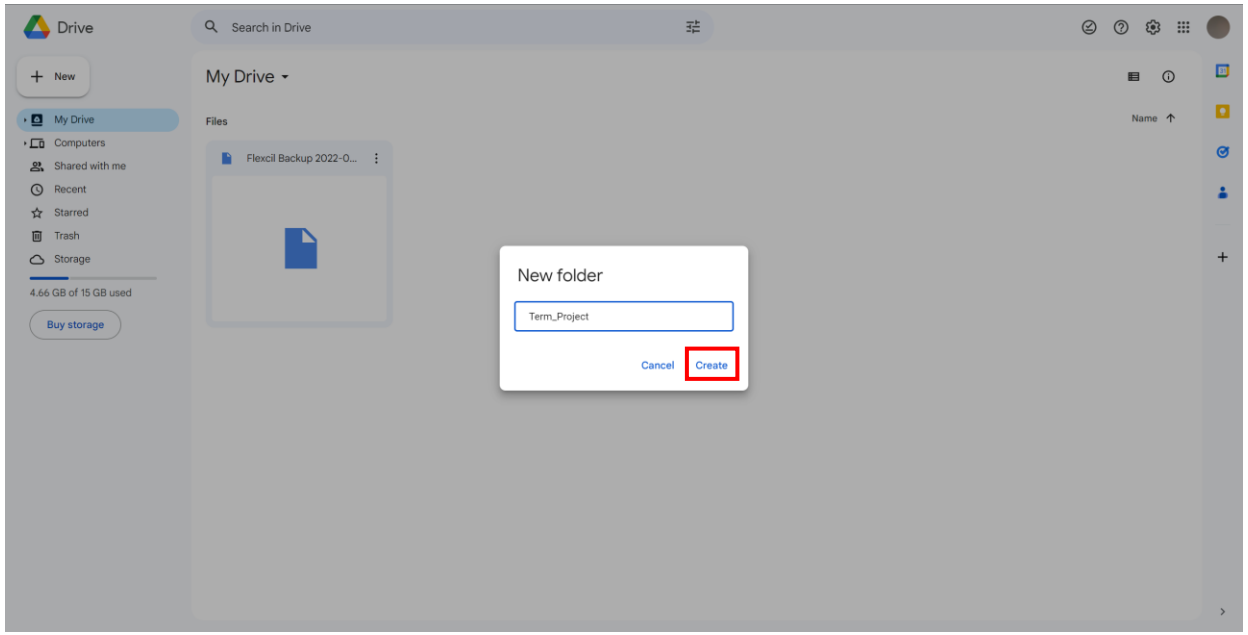
- Get into your Google Drive.
- Make a new folder for term project.

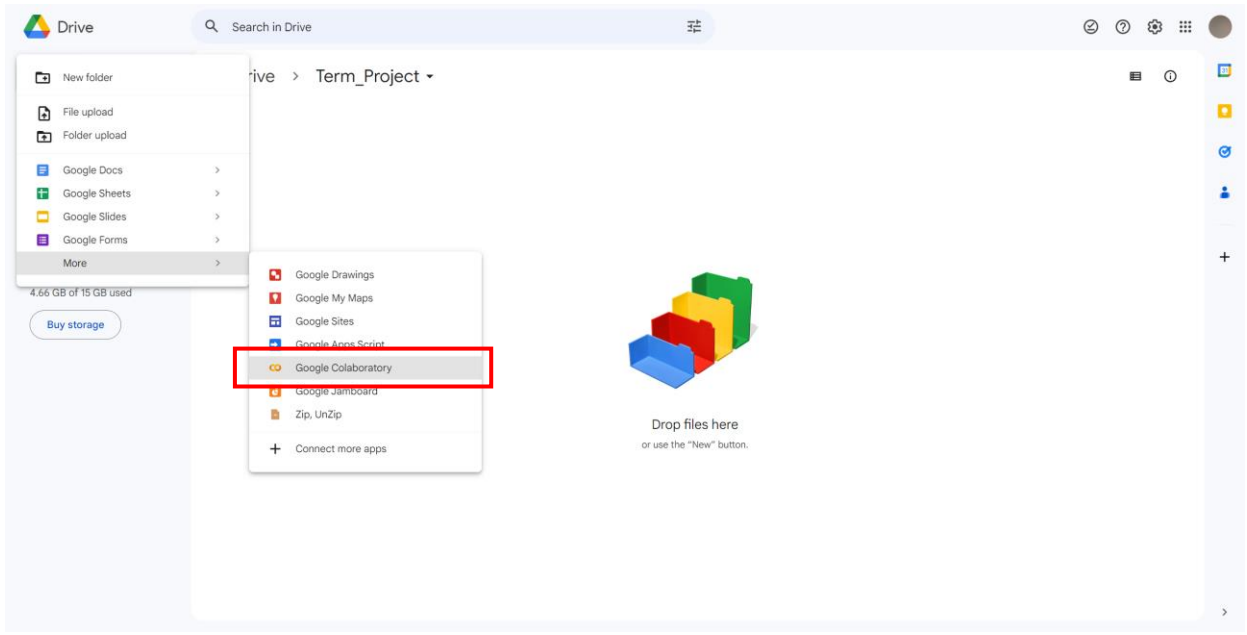




Term Project

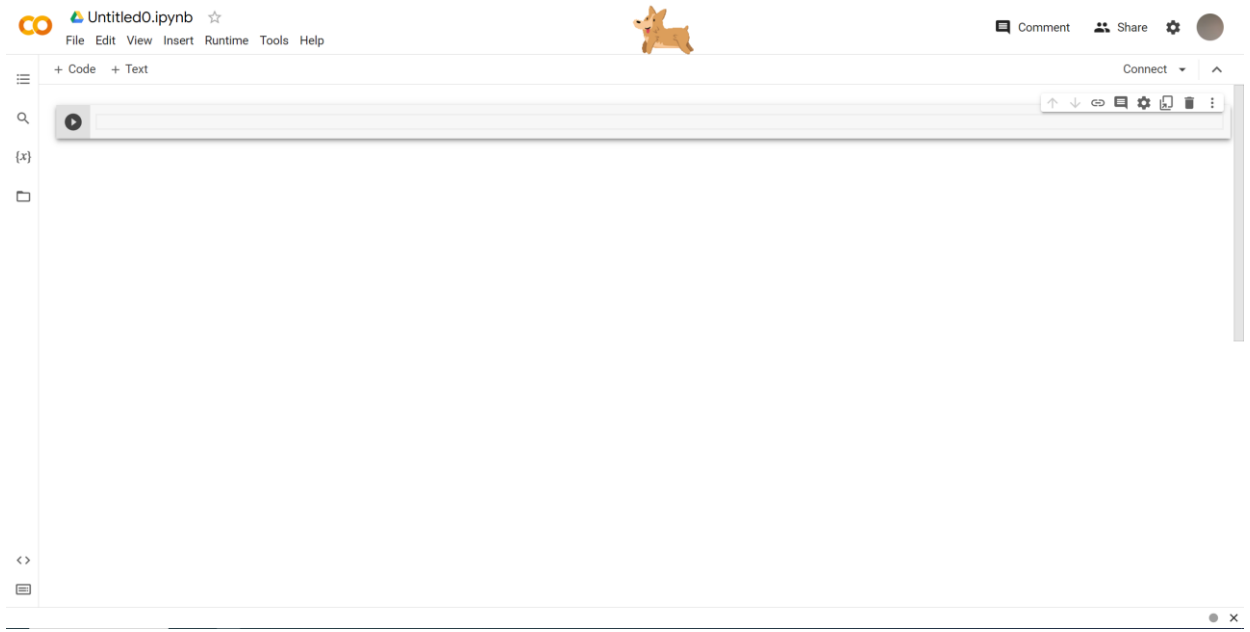
- Make a new folder for term project.
- Make a new google colaboratory file inside of the folder.





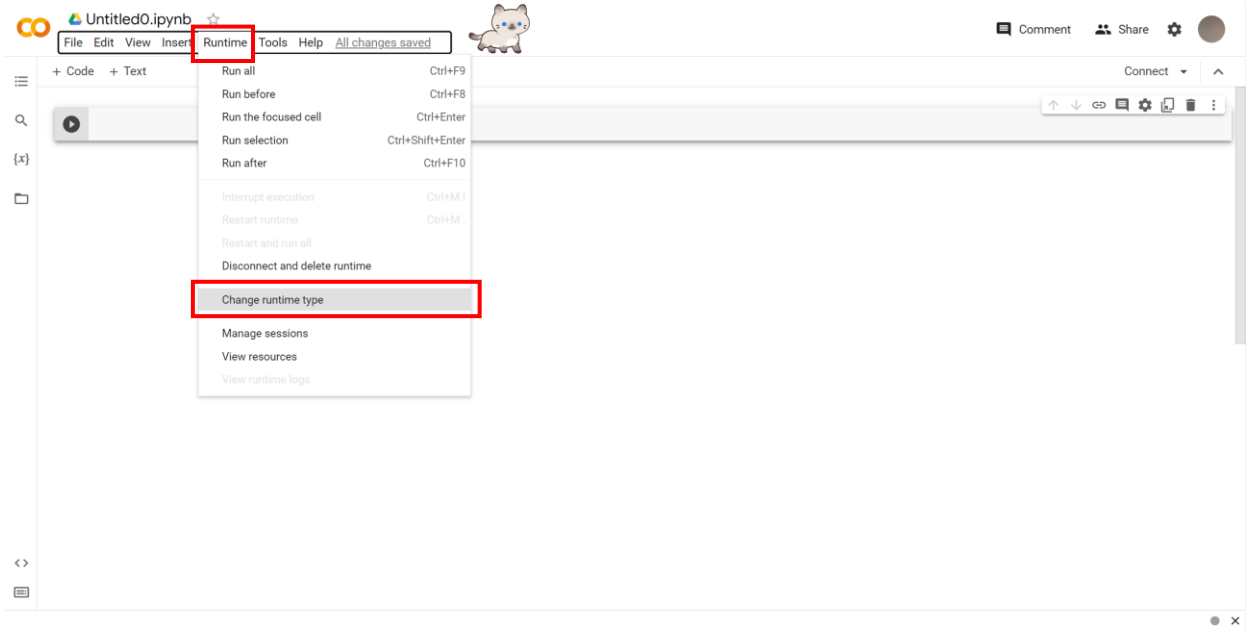
Term Project

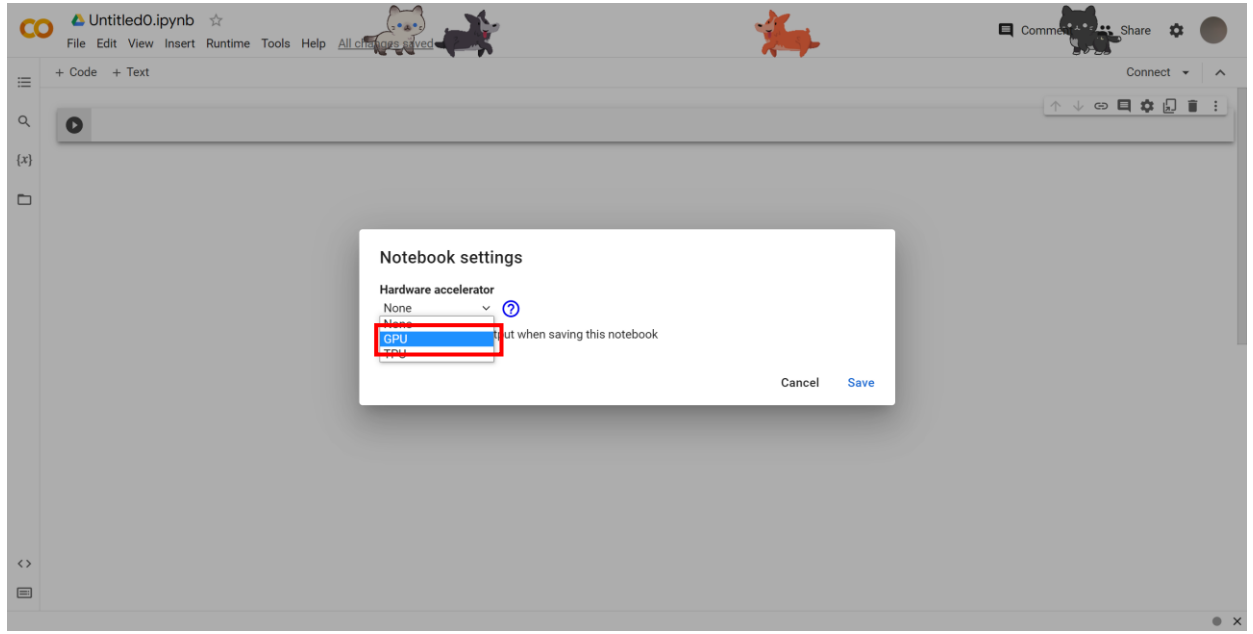
- Main page of colab.



Term Project

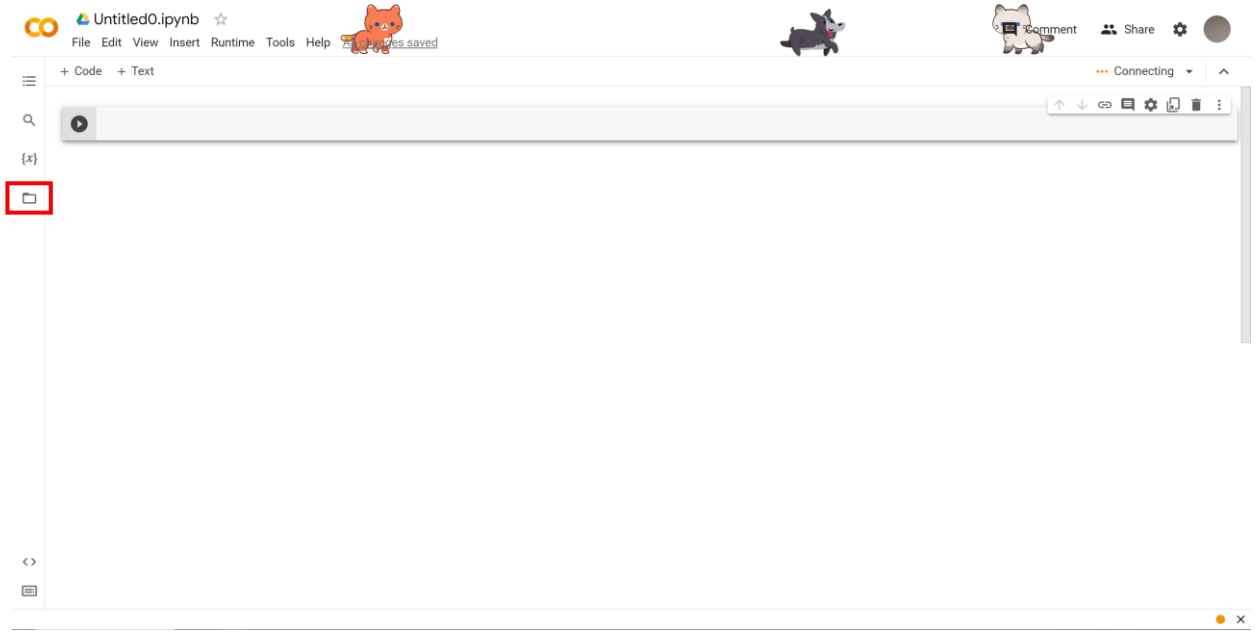
- Change runtime for using GPU.
  - You can use Tesla T4 GPU!!

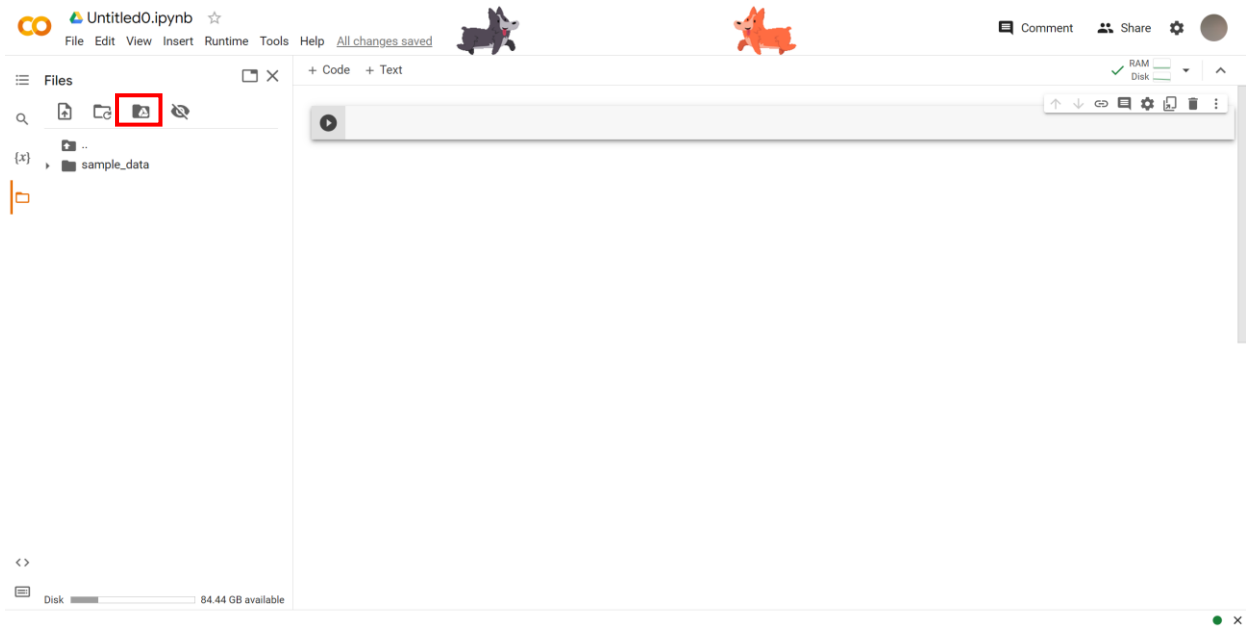




Related Work

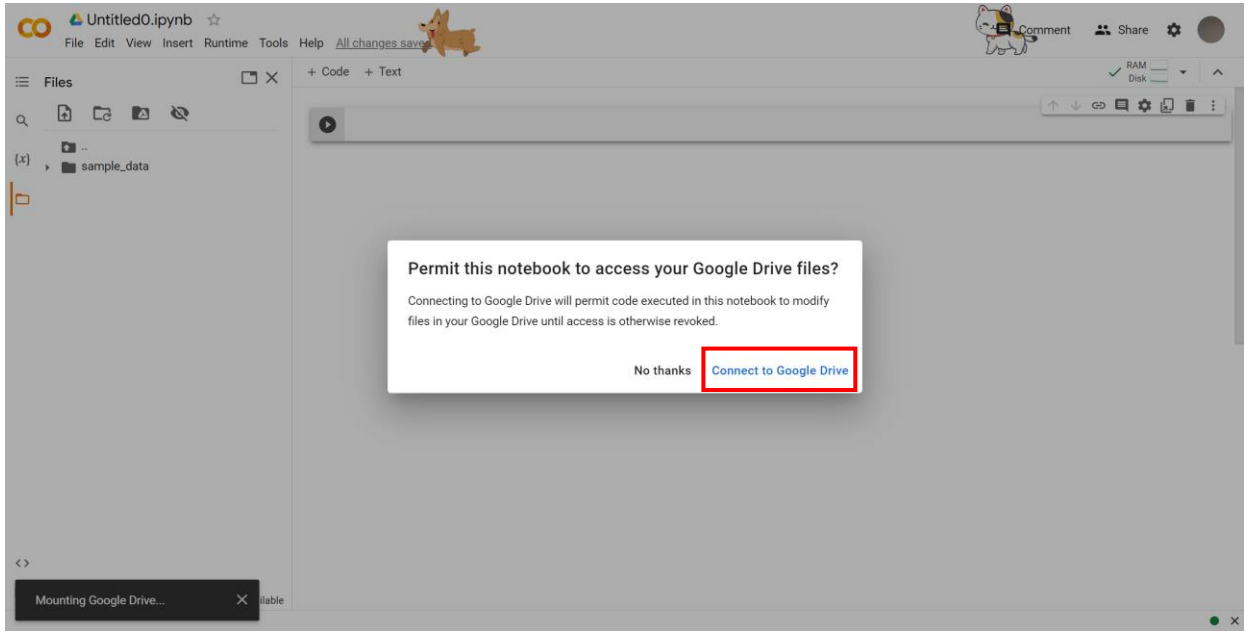
- Mount google drive.
  - Or you will lose all results when runtime is over.

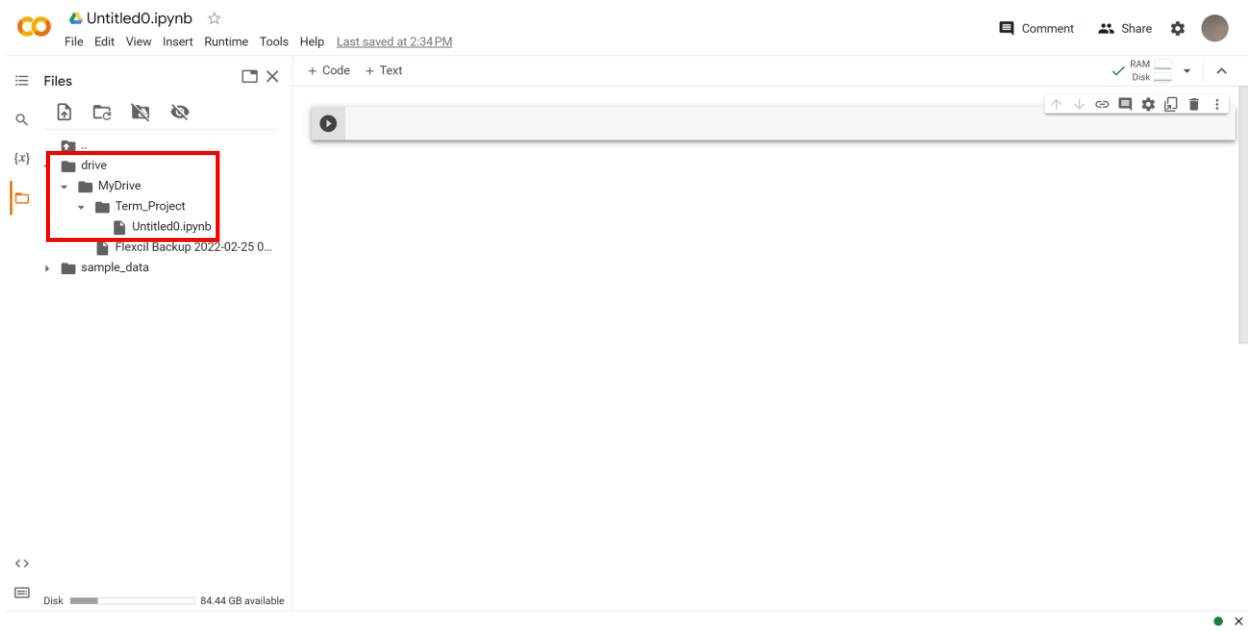




Term Project

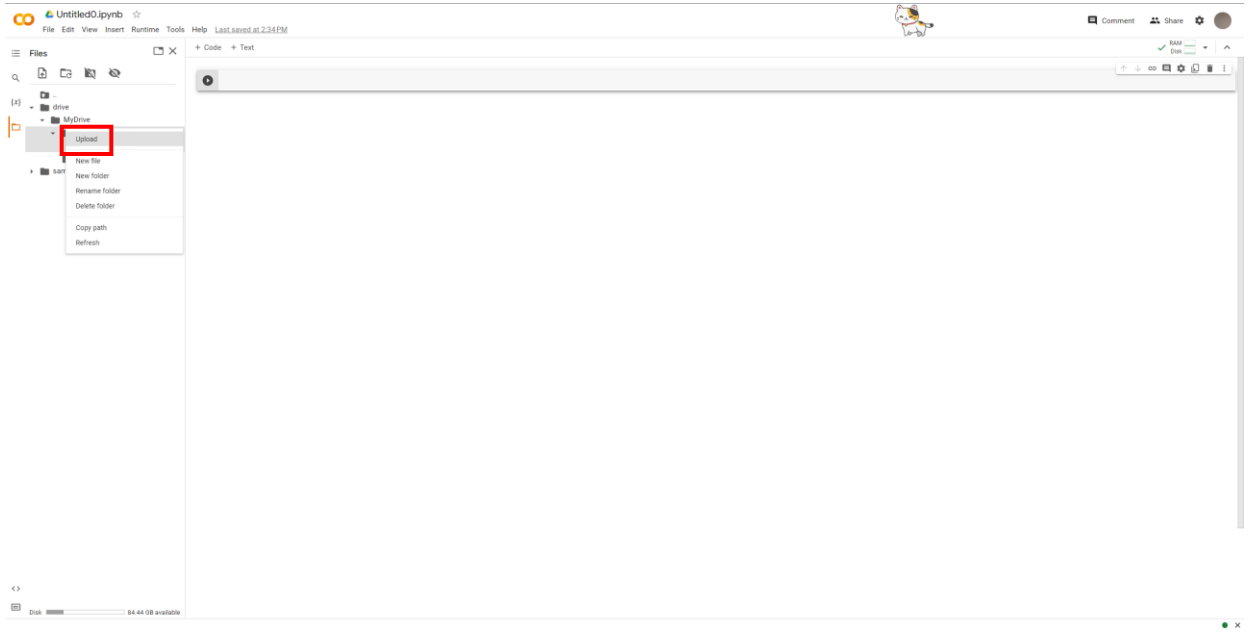
- Mount google drive.
  - Or you will lose all results when runtime is over.

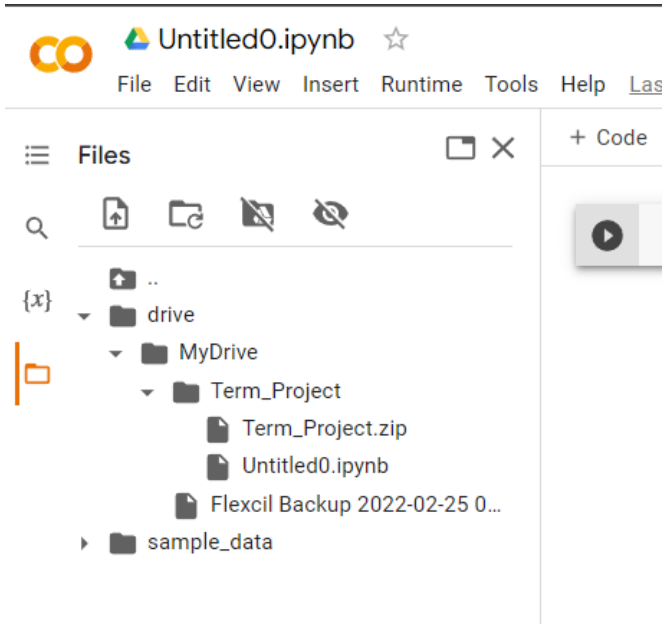




Term Project

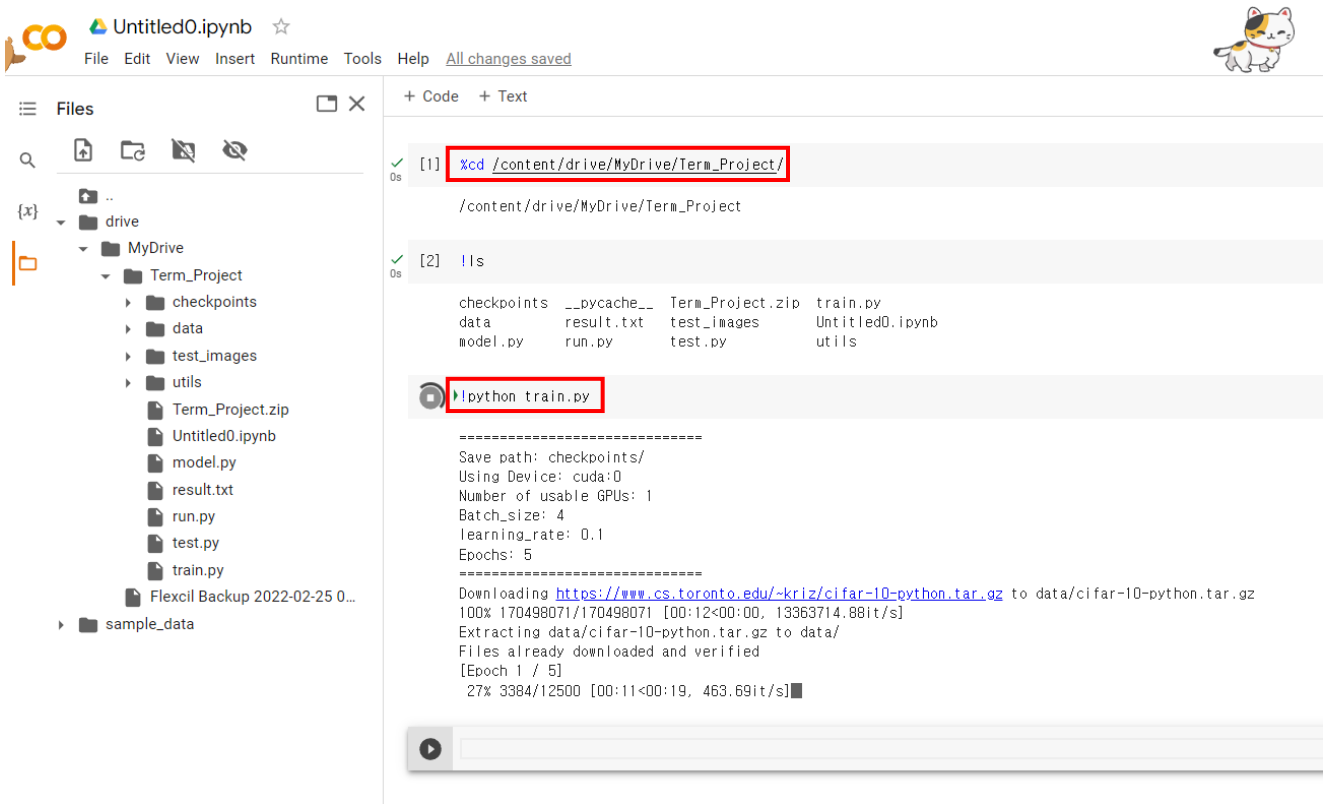
- Upload project folder.





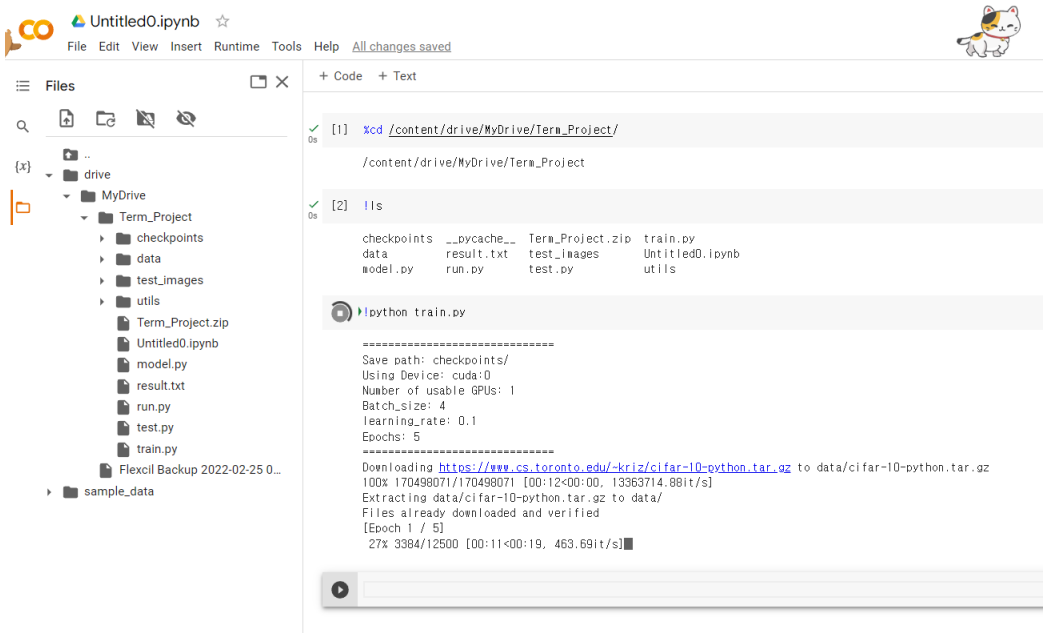
Term Project

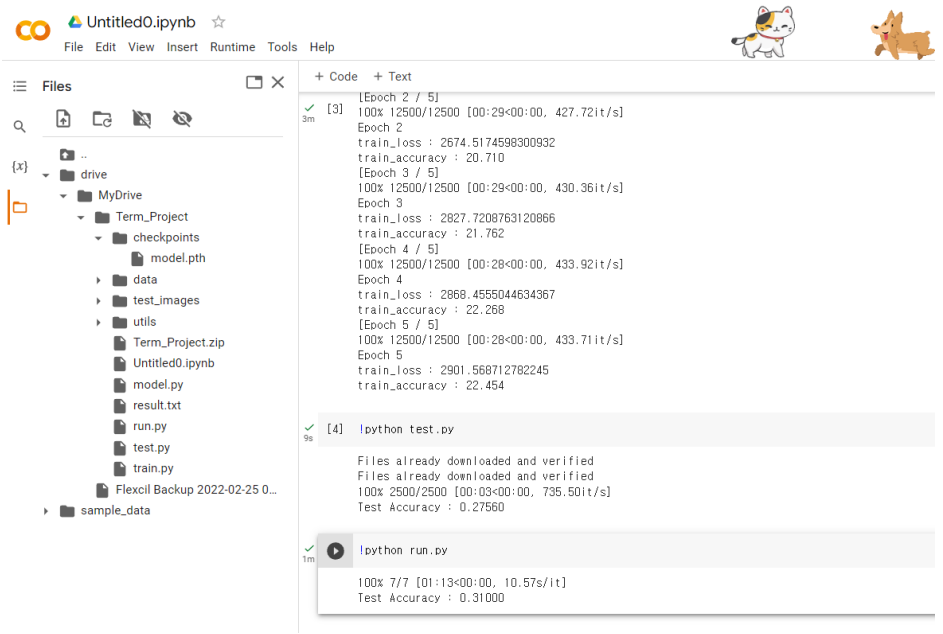
- Unzip folder (!unzip Term_Project.zip).
- Move to project folder.
- You can excute python file with "!python".
  - Also terminal commands!
- Ctrl + enter will execute current shell.



Term Project

- You can excute python file with "!python".
  - Also terminal commands!





Term Project

- Basic commands
  - !cd => move to designated folder
  - !ls => list files in current folder
  - !nvidia-smi => current gpu status
  - !python file.py => excute python file

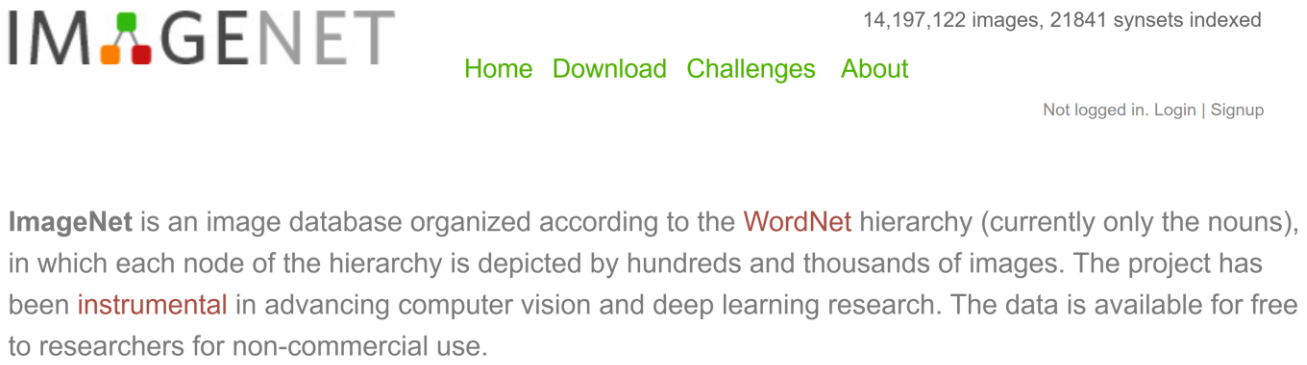
Pretrained model

Deep Learning

Pretrained model

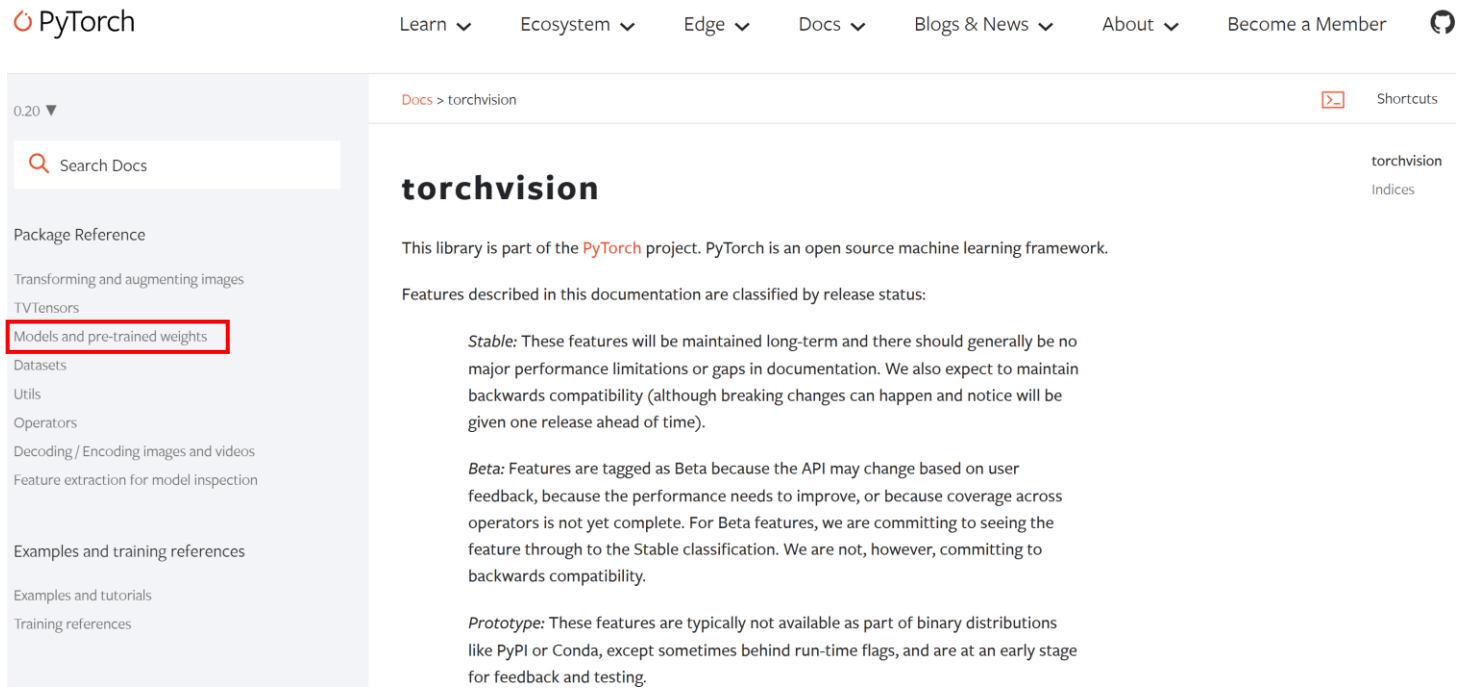
- NN model with pre-learned weights from training on large datasets
  - During its initial training phase, it learns from a vast amount of images, text, or data and captures general features and patterns.

- Training can be performed quickly
- Better test accuracy in samll dataset
- Better generalization performance



torchvision

- Torchvision is a robust library for image tasks, efficiently supporting the entire process from data loading and preprocessing to model construction
- The *torchvision.models* model offers pre-trained model architectures, facilitating the straightforward construction of image classification models



torchvision — Torchvision 0.20 documentation

torchvision

- Pretrained weights for various models are available

Classification

The following classification models are available, with or without pre-trained weights:

- AlexNet
- ConvNeXt
- DenseNet
- EfficientNet
- EfficientNetV2
- GoogLeNet
- Inception V3
- MaxVit
- MNASNet
- MobileNet V2
- MobileNet V3
- RegNet
- ResNet
- ResNeXt
- ShuffleNet V2
- SqueezeNet
- SwinTransformer
- VGG
- VisionTransformer
- Wide ResNet

Table of all available classification weights

Accuracies are reported on ImageNet-1K using single crops:

Weight	Acc@1	Acc@5	Params	GFLOPS	Recipe
AlexNet_Weights.IMAGENET1K_V1	56.522	79.066	61.1M	0.71	link
ConvNeXt_Base_Weights.IMAGENET1K_V1	84.062	96.87	88.6M	15.36	link
ConvNeXt_Large_Weights.IMAGENET1K_V1	84.414	96.976	197.8M	34.36	link
ConvNeXt_Small_Weights.IMAGENET1K_V1	83.616	96.65	50.2M	8.68	link
ConvNeXt_Tiny_Weights.IMAGENET1K_V1	82.52	96.146	28.6M	4.46	link
DenseNet121_Weights.IMAGENET1K_V1	74.434	91.972	8.0M	2.83	link
DenseNet161_Weights.IMAGENET1K_V1	77.138	93.56	28.7M	7.73	link
DenseNet169_Weights.IMAGENET1K_V1	75.6	92.806	14.1M	3.36	link
DenseNet201_Weights.IMAGENET1K_V1	76.896	93.37	20.0M	4.29	link
EfficientNet_B0_Weights.IMAGENET1K_V1	77.692	93.532	5.3M	0.39	link
EfficientNet_B1_Weights.IMAGENET1K_V1	78.642	94.186	7.8M	0.69	link
EfficientNet_B1_Weights.IMAGENET1K_V2	79.838	94.934	7.8M	0.69	link
EfficientNet_B2_Weights.IMAGENET1K_V1	80.608	95.31	9.1M	1.09	link
EfficientNet_B3_Weights.IMAGENET1K_V1	82.008	96.054	12.2M	1.83	link
EfficientNet_B4_Weights.IMAGENET1K_V1	83.384	96.594	19.3M	4.39	link
EfficientNet_B5_Weights.IMAGENET1K_V1	83.444	96.628	30.4M	10.27	link
EfficientNet_B6_Weights.IMAGENET1K_V1	84.008	96.916	43.0M	19.07	link
EfficientNet_B7_Weights.IMAGENET1K_V1	84.122	96.908	66.3M	37.75	link
EfficientNet_V2_L_Weights.IMAGENET1K_V1	85.808	97.788	118.5M	56.08	link
EfficientNet_V2_M_Weights.IMAGENET1K_V1	85.112	97.156	54.1M	24.58	link
EfficientNet_V2_S_Weights.IMAGENET1K_V1	84.228	96.878	21.5M	8.37	link

[torchvision — Torchvision 0.20 documentation]

torchvision

- Several usage methods

```python
from torchvision.models import resnet50, ResNet50_Weights

# Old weights with accuracy 76.130%
resnet50(weights=ResNet50_Weights.IMAGENET1K_V1)

# New weights with accuracy 80.858%
resnet50(weights=ResNet50_Weights.IMAGENET1K_V2)

# Best available weights (currently alias for IMAGENET1K_V2)
# Note that these weights may change across versions
resnet50(weights=ResNet50_Weights.DEFAULT)

# Strings are also supported
resnet50(weights="IMAGENET1K_V2")

# No weights - random initialization
resnet50(weights=None)
```

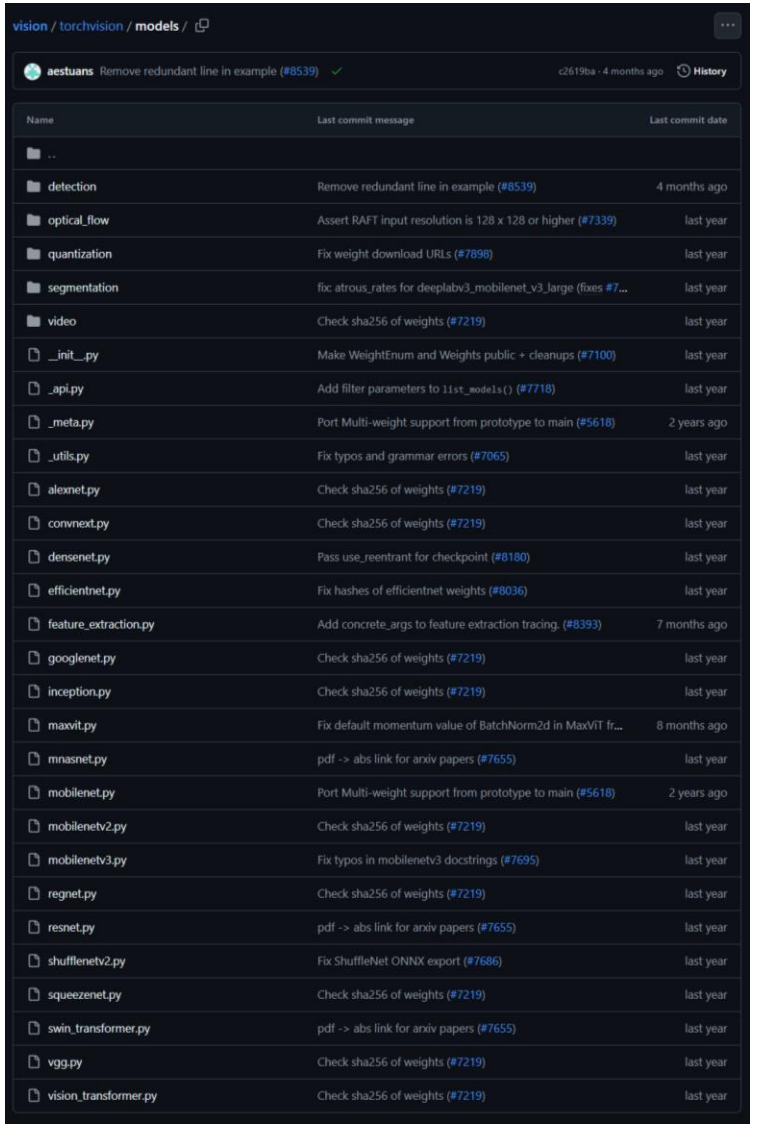
- train.py

```python
    # Print Hyperparameter
    print("Batch_size:", args.batch_size)
    print("learning_rate:", args.learning_rate)
    print("Epochs:", args.epochs)
    print("==============================")

    # Make Data loader and Model
    train_loader, _ = make_data_loader(args)

    # custom model
    # model = BaseModel()

    # torchvision model
    model = resnet18(weights=ResNet18_Weights)

    # you have to change num_classes to 2
    num_features = model.fc.in_features # edited
    model.fc = nn.Linear(num_features, num_classes) # edited
    model.to(device)
    print(model)

    # Training The Model
    train(args, train_loader, model)
```

[torchvision — Torchvision 0.20 documentation]

torchvision

- You can download the source code of model from the official GitHub repository



torchvision

- You can also modify the code of the installed library

```python
    # Print Hyperparameter
    print("Batch_size:", args.batch_size)
    print("learning_rate:", args.learning_rate)
    print("Epochs:", args.epochs)
    print("==============================")

    # Make Data loader and Model
    train_loader, _ = make_data_loader(args)

    # custom model
    # model = BaseModel()

    # torchvision model
    model = resnet18(weights=ResNet18_Weights)

    # you have to change num_classes to 2
    num_features = model.fc.in_features # edited
    model.fc = nn.Linear(num_features, num_classes) # edited
    model.to(device)
    print(model)

    # Training The Model
    train(args, train_loader, model)
```

```python
@register_model()
@handle_legacy_interface(weights=("pretrained", ResNet18_Weights.IMAGENET1K_V1))
def resnet18(*, weights: Optional[ResNet18_Weights] = None, progress: bool = True, **kwargs: Any) -> ResNet:
    """ResNet-18 from `Deep Residual Learning for Image Recognition <https://arxiv.org/pdf/1512.03385.pdf>`__.

    Args:
        weights (:class:`~torchvision.models.ResNet18_Weights`, optional): The
            pretrained weights to use. See
            :class:`~torchvision.models.ResNet18_Weights` below for
            more details, and possible values. By default, no pre-trained
            weights are used.
        progress (bool, optional): If True, displays a progress bar of the
            download to stderr. Default is True.
        **kwargs: parameters passed to the ``torchvision.models.resnet.ResNet``
            base class. Please refer to the `source code
            <https://github.com/pytorch/vision/blob/main/torchvision/models/resnet.py>`_
            for more details about this class.

    .. autoclass:: torchvision.models.ResNet18_Weights
        :members:
    """
    weights = ResNet18_Weights.verify(weights)

    return _resnet(BasicBlock, [2, 2, 2, 2], weights, progress, **kwargs)
```

Term Project

- Feel free to email TA if the bug isn't fixed or if you have any questions!
- Email: seokjinoh@hanyang.ac.kr
  wjdeodbs386@hanyang.ac.kr

- Please try searching on Google before asking any questions regarding the bug.
- Please specify which class you are taking and who you are when sending an email.