B093040024
郭晏涵

深度視覺

HW3：CIFAR10 與 dataloader

notebook 執行過程

1 cifar10-image-dataset：

建立雲端硬碟存放路徑

CIFAR-10: Image Dataset

Throughout this course, we will teach you all basic skills and how to use all neccessary tools that you need to implement deep neural networks, which is the main focus of this class. However, you should also be proficient with handling data and know how to prepare it for your specific task. In fact, most of the jobs that involve deep learning in industry are very data related so this is an important skill that you have to pick up.

Therefore, we will take a deep dive into data preparation this week by implementing our own datasets and dataloader. In this notebook, we will focus on the image dataset CIFAR-10. The CIFAR-10 dataset consists of 50000 32x32 colour images in 10 classes, which are *plane*, *car*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship*, *truck*.

```
[1] import os
    print(os.getcwd())

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

```
/content
Mounted at /content/drive
```

```
[2] os.listdir()
```

```
['.config', 'drive', 'sample_data']
```

Let's start by importing some libraries that you will need along the way, as well as some code files that you will work on throughout this notebook.

Let's start by importing some libraries that you will need along the way, as well as some code files that you will work on throughout this notebook.

```
[3] %load_ext autoreload
    %autoreload 2
    !pip list
```

```
Package                       Version
----------------------------- --------------------
absl-py                       1.0.0
alabaster                     0.7.12
albumentations                0.1.12
altair                        4.2.0
appdirs                       1.4.4
argon2-cffi                   21.3.0
argon2-cffi-bindings          21.2.0
arviz                         0.11.4
astor                         0.8.1
astropy                       4.3.1
astunparse                    1.6.3
```

```
[4] import sys
    sys.path.append('/content/drive/MyDrive/HW3/3')
    %pwd
```

```
'/content'
```

```
[5]  import os
     import pickle

     import numpy as np
     from PIL import Image
     import matplotlib.pyplot as plt
     from tqdm import tqdm

     from exercise_code.data import (
             ImageFolderDataset,
             RescaleTransform,
             NormalizeTransform,
             ComposeTransform,
             compute_image_mean_and_std,
     )
     from exercise_code.tests import (
             test_image_folder_dataset,
             test_rescale_transform,
             test_compute_image_mean_and_std,
             test_len_dataset,
             test_item_dataset,
             test_transform_dataset,
             save_pickle
     )

     %load_ext autoreload
     %autoreload 2
     %matplotlib inline

     plt.rcParams['figure.figsize'] = (10.0, 8.0) # set default size of plots
     plt.rcParams['image.interpolation'] = 'nearest'
     plt.rcParams['image.cmap'] = 'gray'

     The autoreload extension is already loaded. To reload it, use:
       %reload_ext autoreload
```

下載資料集

1. Dataset Download

Let us get started by downloading the data. In `exercise_code/data/image_folder_dataset.py` you can find a class `ImageFolderDataset`, which you will have to complete throughout this notebook.

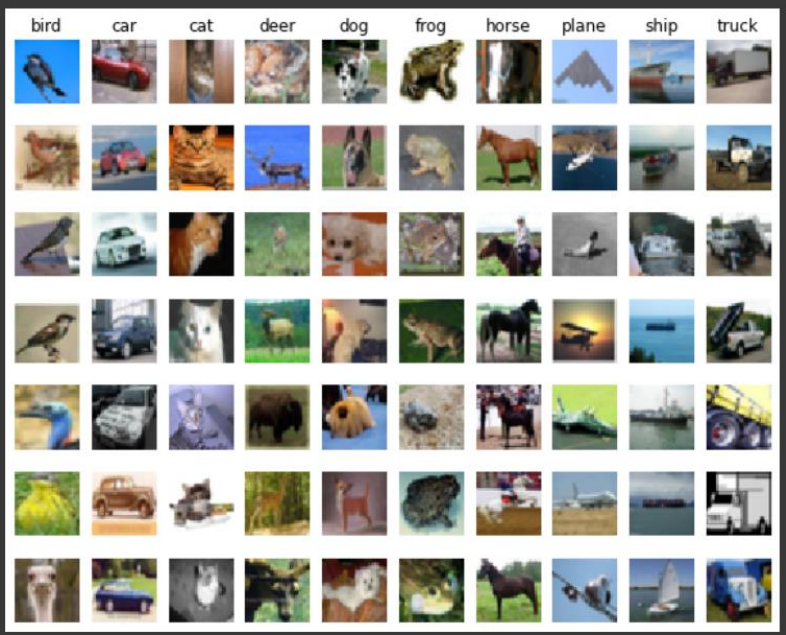
This class automatically downloads the raw data for you. To do so, simply initialize the class as below:

```
[7]  %cd /content
     %pwd

     /content
     '/content'
```

```
[8]  # Set up the output dataset folder
     i2dl_exercises_path = os.path.dirname(os.path.abspath(os.getcwd()))
     cifar_root = os.path.join(i2dl_exercises_path, "datasets", "cifar10")
     import urllib.request
     # Init the dataset and display downloading information this one time
     dataset = ImageFolderDataset(
             root=cifar_root,
             force_download=False,
             verbose=True
     )

     Downloading cifar10.zip
     Downloading https://i2dl.vc.in.tum.de/static/data/cifar10.zip to /datasets/cifar10/cifar10.zip
     120725504it [00:07, 17193406.21it/s]
     Extracting cifar10.zip
     Dataset successfully downloaded! Stored under: /datasets/cifar10
```

檢視資料

```
[9]  def load_image_as_numpy(image_path):
         return np.asarray(Image.open(image_path), dtype=float)

     classes = [
         'plane', 'car', 'bird', 'cat', 'deer',
         'dog', 'frog', 'horse', 'ship', 'truck',
     ]
     num_classes = len(classes)
     samples_per_class = 7
     for label, cls in enumerate(sorted(classes)):
         for i in range(samples_per_class):
             image_path = os.path.join(
                 cifar_root,
                 cls,
                 str(i+1).zfill(4) + ".png"
             )   # e.g. cifar10/plane/0001.png
             image = np.asarray(Image.open(image_path))   # open image as numpy array
             plt_idx = i * num_classes + label + 1   # calculate plot location in the grid
             plt.subplot(samples_per_class, num_classes, plt_idx)
             plt.imshow(image.astype('uint8'))
             plt.axis('off')
             if i == 0:
                 plt.title(cls)   # plot class names above columns
     plt.show()
```



測試資料長度：實現__len__(self) in image_folder_dataset.py

```
73      def __len__(self):
74          length = None
75          ############################################################################
76          # TODO:
77          # Return the length of the dataset (number of images)
78          ############################################################################
79
80          length = len(self.images)
81
82          ############################################################################
83          #                                                          END OF YOUR CODE
84          ############################################################################
85          return length
86
```

Task: Implement

Implement the `__len__(self)` method in `exercise_code/data/image_folder_dataset.py` and test your implementation by running the following cell.

```
[10] from exercise_code.data.image_folder_dataset import ImageFolderDataset

     dataset = ImageFolderDataset(
            root=cifar_root,
     )

     _ = test_len_dataset(dataset)
```

```
LenTestInt passed.
LenTestCorrect passed.
Method __len__() correctly implemented. Tests passed: 2/2
Score: 100/100
```

檢視資料：實現__getitem__(self) in image_folder_dataset.py

```
92      def __getitem__(self, index):
93          data_dict = None
94          ###########################################################################
95          # TODO:
96          # create a dict of the data at the given index in your dataset
97          # The dict should be of the following format:
98          # {"image": <i-th image>,
99          # "label": <label of i-th image>}
100         # Hints:
101         #     - use load_image_as_numpy() to load an image from a file path
102         #     - If applicable (Task 4: 'Transforms and Image Preprocessing'),
103         #       make sure to apply self.transform to the image:
104         #       image_transformed = self.transform(image)
105         ###########################################################################
106
107         if(index>=len(self)):
108             raise IndexError
109
110         image = self.load_image_as_numpy(self.images[index])
111         if(self.transform):
112             image = self.transform(image)
113
114         data_dict = {"image":image, "label":self.labels[index]}
115
116         ###########################################################################
117         #                                                       END OF YOUR CODE
118         ###########################################################################
119         return data_dict
```

Task: Implement

Implement the `__getitem__(self, index)` method in `exercise_code/data/image_folder_dataset.py` and test your implementation by running the following cell.

Hint: You may want to reuse parts of the '2. Data Visualization' code above in your implementation of `__getitem__()`.

```
[11] from exercise_code.data.image_folder_dataset import ImageFolderDataset

     dataset = ImageFolderDataset(
            root=cifar_root,
     )

     _ = test_item_dataset(dataset)
```

```
GetItemTestType passed.
GetItemTestImageShape passed.
Method __getitem__() correctly implemented. Tests passed: 2/2
Score: 100/100
```

3.3 Dataset Usage

Now that you have implemented all required parts of the ImageFolderDataset, using the `__getitem__()` method, you can now access our dataset as conveniently as you would access a list:

```
[27] sample_item = dataset[0]
     sample_image = sample_item["image"]
     sample_label = sample_item["label"]

     print('Sample image shape:', sample_image.shape)
     print('Sample label:', sample_label)
     print('Sample image first values:', sample_image[0][0])
```

```
Sample image shape: (32, 32, 3)
Sample label: 0
Sample image first values: [ 3. 125. 233.]
```

As you can see, the images are represented as uint8 values for each of the three RGB color channels. The data type and scale will be important later.

As you have implemented both `__len__()` and `__getitem__()`, you can now even iterate over the dataset with a simple for loop!

```
[13] num_samples = 0
     for sample in tqdm(dataset):
         num_samples += 1

     print("Number of samples:", num_samples)
```

```
100%|██████████| 50000/50000 [00:12<00:00, 3947.40it/s]Number of samples: 50000
```

資料轉換

Task: Implement

Modify the `__getitem__(self, index)` method in `exercise_code/data/image_folder_dataset.py` such that it applies `self.transform`. With this change you can simply define the transforms during dataset creation and apply those automatically for each `__getitem__` call. Make sure not to break it though ;).

```
[14] from exercise_code.data.image_folder_dataset import ImageFolderDataset

     dataset = ImageFolderDataset(
         root=cifar_root,
     )

     _ = test_transform_dataset(dataset)
```

```
GetItemTestType passed.
GetItemTestImageShape passed.
GetItemTestTransformApplied passed.
Method __getitem__() correctly implemented. Tests passed: 3/3
Score: 100/100
```

轉換資料 scale：實現 RescaleTransform in transforms.py

```
23      def __call__(self, images):
24          ###########################################################################
25          # TODO:
26          # Rescale the given images:
27          #     - from (self._data_min, self._data_max)
28          #     - to (self.min, self.max)
29          ###########################################################################
30
31          images = (images-self._data_min)*(self.max-self.min)/(self._data_max-self._data_min)+self.min
32
33          ###########################################################################
34          #                                                      END OF YOUR CODE
35          ###########################################################################
36          return images
37
```

Task: Implement

Implement the `__call__()` method of `RescaleTransform` in `exercise_code/data/transforms.py` and test your implementation by running the following cell.

```
[15] from exercise_code.data.image_folder_dataset import ImageFolderDataset
     from exercise_code.data.transforms import RescaleTransform

     rescale_transform = RescaleTransform()
     dataset_rescaled = ImageFolderDataset(
            root=cifar_root,
            transform=rescale_transform
     )

     _ = test_rescale_transform(dataset_rescaled)
```

```
100%|██████████| 50000/50000 [00:16<00:00, 3011.85it/s]
RescaleTransformTestMin passed.
100%|██████████| 50000/50000 [00:16<00:00, 3046.59it/s]RescaleTransformTestMax passed.
Class RescaleTransform correctly implemented. Tests passed: 2/2
Score: 100/100
```

If you look at the first image, you should now see that all values are between 0 and 1.

```
[16] sample_item = dataset_rescaled[0]
     sample_label = sample_item["label"]
     sample_image = sample_item["image"]

     print("Max value:", np.max(sample_image))
     print("Min value:", np.min(sample_image))
     print('Sample rescaled image first values:', sample_image[0][0])
```

```
Max value: 1.0
Min value: 0.00392156862745098
Sample rescaled image first values: [0.01176471 0.49019608 0.91372549]
```

正規化-計算平均與標準差：實現 NormalizeTransform in transforms.py

```
def compute_image_mean_and_std(images):
    """
    Calculate the per-channel image mean and standard deviation of given images
    :param images: numpy array of shape NxHxWxC
        (for N images with C channels of spatial size HxW)
    :returns: per-channels mean and std; numpy array of shape C
    """
    mean, std = None, None
    ###########################################################################
    # TODO:                                                                   #
    # Calculate the per-channel mean and standard deviation of the images    #
    # Hint: You can use numpy to calculate the mean and standard deviation #
    ###########################################################################

    mean = np.mean(images, axis=(0, 1, 2))
    std = np.std(images, axis=(0, 1, 2))

    ###########################################################################
    #                                                      END OF YOUR CODE
    ###########################################################################
    return mean, std
```

Have a look at the code in `exercise_code/data/transforms.py`.

The next step is to normalize the CIFAR-10 **images channel-wise** to standard normal. To do so, you need to calculate the **per-channel image mean and standard deviation** first, which you can then provide to `NormalizeTransform` to normalize the data accordingly.

```
[17] # You first have to load all rescaled images
     rescaled_images = []
     for sample in tqdm(dataset_rescaled):
         rescaled_images.append(sample["image"])
     rescaled_images = np.array(rescaled_images)
```

```
100%|██████████| 50000/50000 [00:16<00:00, 3014.35it/s]
```

Task: Implement

Implement the `compute_image_mean_and_std()` method and the `__call__()` method of `NormalizeTransform` in `exercise_code/data/transforms.py`. Compute the rescaled dataset's mean and variance by running the following cell.

```
[18] from exercise_code.data.transforms import compute_image_mean_and_std

     cifar_mean, cifar_std = compute_image_mean_and_std(rescaled_images)
     print("Mean:\t", cifar_mean, "\nStd:\t", cifar_std)
```

```
Mean:    [0.49191375 0.48235852 0.44673872]
Std:     [0.24706447 0.24346213 0.26147554]
```

Rescale：

To test your implementation, run the following code:

```
[19] _ = test_compute_image_mean_and_std(cifar_mean, cifar_std)
```

```
CIFARImageMeanTest passed.
CIFARImageStdTest passed.
Method compute_image_mean_and_std() correctly implemented. Tests passed: 2/2
Score: 100/100
```

```
[20] # The rescaled images will be deleted now from your ram as they are no longer needed
     try:
         del rescaled_images
     except NameError:
         pass
```

Now you can use the mean and standard deviation you computed to normalize the loaded data. This can be done by simply adding the `NormalizeTransform` to the list of transformations our dataset applies in `__getitem__()`.

```
80      def __call__(self, images):
81          ###########################################################################
82          # TODO:
83          # normalize the given images:
84          #     - substract the mean of dataset
85          #     - divide by standard deviation
86          ###########################################################################
87
88          for i in range(0, 32):
89              for j in range(0, 32):
90                  for k in range(0, 3):
91                      images[i][j][k] = (images[i][j][k]-self.mean[k])/self.std[k]
92
93          ###########################################################################
94          #                                                           END OF YOUR CODE
95          ###########################################################################
96          return images
```

Task: Check Code

Please check out the `ComposeTransform` in `transforms.py`. Later on, we will most often use multiple transforms and chain them together. Remember that the order is of importance here!

```
[21] from exercise_code.data.image_folder_dataset import ImageFolderDataset
     from exercise_code.data.transforms import RescaleTransform, NormalizeTransform, ComposeTransform

     # Set up both transforms using the parameters computed above
     rescale_transform = RescaleTransform()
     normalize_transform = NormalizeTransform(
             mean=cifar_mean,
             std=cifar_std
     )

     final_dataset = ImageFolderDataset(
             root=cifar_root,
             transform=ComposeTransform([rescale_transform, normalize_transform])
     )
```

You can now check out the results of the transformed samples:

```
[22] sample_item = final_dataset[0]
     sample_label = sample_item["label"]
     sample_image = sample_item["image"]

     print('Sample normalized image shape:', sample_image.shape)
     print('Sample normalized image first values:', sample_image[0][0])
```

```
Sample normalized image shape: (32, 32, 3)
Sample normalized image first values: [-1.94341602  0.03219212  1.7859673 ]
```

儲存轉換後的資料

5. Save your Dataset

Now save your dataset and transforms using the following cell. This will save it to a pickle file `models/cifar_dataset.p`. We will use this dataset for the next notebook and this will count for the submission.

```
[23] %cd '/content/drive/MyDrive/HW3/3'
     %pwd
```

```
/content/drive/MyDrive/HW3/3
'/content/drive/MyDrive/HW3/3'
```

```
[24] save_pickle(
             data_dict={
                     "dataset": final_dataset,
                     "cifar_mean": cifar_mean,
                     "cifar_std": cifar_std,
             },
             file_name="cifar_dataset.p"
     )
```

2 dataloader：

建立雲端硬碟路徑及 import modules

```
[1]  import os
     print(os.getcwd())

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

```
/content
Mounted at /content/drive
```

```
[2]  import sys
     sys.path.append('/content/drive/MyDrive/HW3/3')
     %pwd
```

```
'/content'
```

```
[3]  import numpy as np

     from exercise_code.data import DataLoader, DummyDataset
     from exercise_code.tests import (
             test_dataloader,
             test_dataloader_len,
             test_dataloader_iter,
             save_pickle,
             load_pickle
     )

     %load_ext autoreload
     %autoreload 2
```

檢視資料

1. Iterating over a Dataset

Throughout this notebook a dummy dataset will be used that contains all even numbers from 2 to 100. Similar to the dataset you have implemented before, the dummy dataset has a `__len__()` method that allows us to call `len(dataset)`, as well as a `__getitem__()` method, which allows you to call `dataset[i]` and returns a dict `{"data": val}` where `val` is the i-th even number. If you would like to see the code, have a look at `DummyDataset` in `exercise_code/data/base_dataset.py`.

Let's start by defining the dataset, and calling its methods to get a better feel for it.

```
[4]  from exercise_code.data.base_dataset import DummyDataset

     dataset = DummyDataset(
             root=None,
             divisor=2,
             limit=100
     )
     print(
             "Dataset Length:\t", len(dataset),
             "\nFirst Element:\t", dataset[0],
             "\nLast Element:\t", dataset[-1],
     )
```

```
Dataset Length:  50
First Element:   {'data': 2}
Last Element:    {'data': 100}
```

In the following, you will write some code to iterate over the dataset in mini-batches, similarly to what a dataloader is supposed to do. The number of samples to load per mini-batch is called **batch size**. For the remainder of this notebook, the batch size is 3.

```
[5]  batch_size = 3
```

建立 batch 及轉換資料型態

Let us now define a simple function that iterates over the dataset and groups samples into mini-batches:

```
[6]  def build_batches(dataset, batch_size):
         batches = []   # list of all mini-batches
         batch = []   # current mini-batch
         for i in range(len(dataset)):
             batch.append(dataset[i])
             if len(batch) == batch_size:   # if the current mini-batch is full,
                 batches.append(batch)   # add it to the list of mini-batches,
                 batch = []   # and start a new mini-batch
         return batches

     batches = build_batches(
         dataset=dataset,
         batch_size=batch_size
     )
```

Let's have a look at the mini-batches:

```
[7]  def print_batches(batches):
         for i, batch in enumerate(batches):
             print("mini-batch %d:" % i, str(batch))

     print_batches(batches)
```

```
mini-batch 0: [{'data': 2}, {'data': 4}, {'data': 6}]
mini-batch 1: [{'data': 8}, {'data': 10}, {'data': 12}]
mini-batch 2: [{'data': 14}, {'data': 16}, {'data': 18}]
mini-batch 3: [{'data': 20}, {'data': 22}, {'data': 24}]
mini-batch 4: [{'data': 26}, {'data': 28}, {'data': 30}]
mini-batch 5: [{'data': 32}, {'data': 34}, {'data': 36}]
mini-batch 6: [{'data': 38}, {'data': 40}, {'data': 42}]
mini-batch 7: [{'data': 44}, {'data': 46}, {'data': 48}]
```

```
[8]  def combine_batch_dicts(batch):
         batch_dict = {}
         for data_dict in batch:
             for key, value in data_dict.items():
                 if key not in batch_dict:
                     batch_dict[key] = []
                 batch_dict[key].append(value)
         return batch_dict

     combined_batches = [combine_batch_dicts(batch) for batch in batches]
     print_batches(combined_batches)
```

```
mini-batch 0: {'data': [2, 4, 6]}
mini-batch 1: {'data': [8, 10, 12]}
mini-batch 2: {'data': [14, 16, 18]}
mini-batch 3: {'data': [20, 22, 24]}
mini-batch 4: {'data': [26, 28, 30]}
mini-batch 5: {'data': [32, 34, 36]}
mini-batch 6: {'data': [38, 40, 42]}
mini-batch 7: {'data': [44, 46, 48]}
mini-batch 8: {'data': [50, 52, 54]}
mini-batch 9: {'data': [56, 58, 60]}
mini-batch 10: {'data': [62, 64, 66]}
mini-batch 11: {'data': [68, 70, 72]}
mini-batch 12: {'data': [74, 76, 78]}
mini-batch 13: {'data': [80, 82, 84]}
mini-batch 14: {'data': [86, 88, 90]}
mini-batch 15: {'data': [92, 94, 96]}
```

This looks much more organized.

To perform operations more efficiently later, we would also like the values of the mini-batches to be contained in a numpy array instead of a simple list. Let's briefly write a function for that:

```
[9] def batch_to_numpy(batch):
        numpy_batch = {}
        for key, value in batch.items():
            numpy_batch[key] = np.array(value)
        return numpy_batch

    numpy_batches = [batch_to_numpy(batch) for batch in combined_batches]
    print_batches(numpy_batches)
```

```
mini-batch 0: {'data': array([2, 4, 6])}
mini-batch 1: {'data': array([ 8, 10, 12])}
mini-batch 2: {'data': array([14, 16, 18])}
mini-batch 3: {'data': array([20, 22, 24])}
mini-batch 4: {'data': array([26, 28, 30])}
mini-batch 5: {'data': array([32, 34, 36])}
mini-batch 6: {'data': array([38, 40, 42])}
mini-batch 7: {'data': array([44, 46, 48])}
mini-batch 8: {'data': array([50, 52, 54])}
mini-batch 9: {'data': array([56, 58, 60])}
mini-batch 10: {'data': array([62, 64, 66])}
mini-batch 11: {'data': array([68, 70, 72])}
mini-batch 12: {'data': array([74, 76, 78])}
mini-batch 13: {'data': array([80, 82, 84])}
mini-batch 14: {'data': array([86, 88, 90])}
mini-batch 15: {'data': array([92, 94, 96])}
```

視參數決定是否亂序

```
[10] def build_batch_iterator(dataset, batch_size, shuffle):
        if shuffle:
            index_iterator = iter(np.random.permutation(len(dataset)))  # define indices as iterator
        else:
            index_iterator = iter(range(len(dataset)))  # define indices as iterator

        batch = []
        for index in index_iterator:  # iterate over indices using the iterator
            batch.append(dataset[index])
            if len(batch) == batch_size:
                yield batch  # use yield keyword to define a iterable generator
                batch = []

    batch_iterator = build_batch_iterator(
        dataset=dataset,
        batch_size=batch_size,
        shuffle=True
    )
    batches = []
    for batch in batch_iterator:
        batches.append(batch)

    print_batches(
        [batch_to_numpy(combine_batch_dicts(batch)) for batch in batches]
    )
```

```
mini-batch 0: {'data': array([36, 42, 86])}
mini-batch 1: {'data': array([26, 96,  8])}
mini-batch 2: {'data': array([46,  4, 22])}
mini-batch 3: {'data': array([74, 70, 88])}
mini-batch 4: {'data': array([ 2, 92, 14])}
mini-batch 5: {'data': array([30, 98, 44])}
mini-batch 6: {'data': array([84, 18, 16])}
```

測試資料長度：實現__len__(self) in dataloader.py

```python
    def __len__(self):
        length = None
        ############################################################################
        # TODO:
        # Return the length of the dataloader
        # Hint: this is the number of batches you can sample from the dataset.  #
        # Don't forget to check for drop last!
        ############################################################################

        length = len(self.dataset)//self.batch_size
        if(not self.drop_last and len(self.dataset)%self.batch_size):
            length += 1

        ############################################################################
        #                                                          END OF YOUR CODE
        ############################################################################
        return length
```

Task: Implement

Implement the `__len__(self)` method in `exercise_code/data/dataloader.py`.

Hint: Don't forget to think about drop_last! We will test for both modes.

```
[11] from exercise_code.data.dataloader import DataLoader

     dataloader = DataLoader(
             dataset=dataset,
             batch_size=batch_size,
             shuffle=True
     )

     _ = test_dataloader_len(
             dataloader=dataloader
     )
```

```
LenTestInt passed.
LenTestCorrect passed.
Method __len__() using drop_last=True correctly implemented. Tests passed: 2/2

LenTestInt passed.
LenTestCorrect passed.
Method __len__() using drop_last=False correctly implemented. Tests passed: 2/2

Method __len__() correctly implemented. Tests passed: 4/4
Score: 100/100
```

測試資料迭代：實現__iter__(self) in dataloader.py

```python
            if (self.shuffle):
                index_iterator = iter(np.random.permutation(len(self.dataset)))
            else:
                index_iterator = iter(range(len(self.dataset)))

            for _ in range(0, len(self)):

                batch = []
                for _ in range(0, self.batch_size):
                    try:
                        index = next(index_iterator)
                        batch.append(self.dataset[index])
                    except StopIteration:
                        break

                batch_dict = {}
                for data_dict in batch:
                    for key, value in data_dict.items():
                        if key not in batch_dict:
                            batch_dict[key] = []
                        batch_dict[key].append(value)

                numpy_batch_dict = {}
                for key, value in batch_dict.items():
                    numpy_batch_dict[key] = np.array(value)

                yield numpy_batch_dict

```

```
[12]
from exercise_code.data.dataloader import DataLoader

dataloader = DataLoader(
        dataset=dataset,
        batch_size=batch_size,
        shuffle=True,
)

_ = test_dataloader_iter(
        dataloader=dataloader
)
```

```
IterTestIterable passed.
IterTestItemType passed.
IterTestBatchSize passed.
IterTestNumBatches passed.
IterTestValuesUnique passed.
IterTestValueRange passed.
IterTestShuffled passed.
IterTestNonDeterministic passed.
Method __iter__() using drop_last=True correctly implemented. Tests passed: 8/8

IterTestIterable passed.
IterTestItemType passed.
IterTestBatchSize passed.
IterTestNumBatches passed.
IterTestValuesUnique passed.
IterTestValueRange passed.
IterTestShuffled passed.
IterTestNonDeterministic passed.
Method __iter__() using drop_last=False correctly implemented. Tests passed: 8/8

Method __iter__() correctly implemented. Tests passed: 16/16
Score: 100/100
```

```
[13] from exercise_code.data.dataloader import DataLoader

     dataloader = DataLoader(
             dataset=dataset,
             batch_size=batch_size,
             shuffle=True,
             drop_last=False,        # Change here if you want to see the impact of drop last and check out the last batch
     )
     for batch in dataloader:
             print(batch)
```

```
{'data': array([62, 20, 90])}
{'data': array([74, 22, 76])}
{'data': array([92, 86, 52])}
{'data': array([82, 48, 60])}
{'data': array([ 4, 54, 72])}
{'data': array([40, 64, 14])}
{'data': array([42, 50, 18])}
{'data': array([ 6, 56, 16])}
{'data': array([78, 70,  8])}
{'data': array([100,   2,  32])}
{'data': array([88, 10, 66])}
{'data': array([38, 68, 26])}
{'data': array([58, 96, 28])}
{'data': array([84, 80, 30])}
{'data': array([98, 94, 34])}
{'data': array([44, 24, 46])}
{'data': array([36, 12])}
```

設定儲存路徑

```
[14] %cd '/content/drive/MyDrive/HW3/3'
     %pwd
```

```
/content/drive/MyDrive/HW3/3
'/content/drive/MyDrive/HW3/3'
```

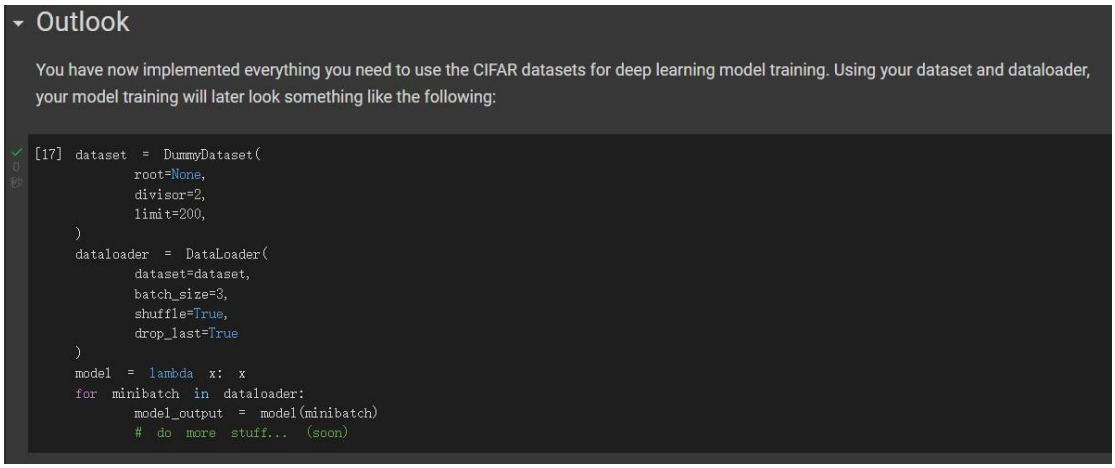
儲存檔案

```
[15] from exercise_code.data.dataloader import DataLoader

     dataloader = DataLoader(
             dataset=dataset,
             batch_size=batch_size,
             shuffle=True,
             drop_last=True,
     )

     dataset = load_pickle("cifar_dataset.p")  # load dataset from the pickle file saved in notbook 1

     save_pickle(
             data_dict={
                     "dataset": dataset['dataset'],
                     "cifar_mean": dataset['cifar_mean'],
                     "cifar_std": dataset['cifar_std'],
                     "dataloader": dataloader
             },
             file_name="cifar_dataset_and_loader.p"
     )
```

```
[16] from exercise_code.submit import submit_exercise

     submit_exercise('exercise03')
```

```
relevant folders: ['exercise_code', 'models']
notebooks files: ['2_dataloader.ipynb', '1_cifar10-image-dataset.ipynb']
Adding folder exercise_code
Adding folder models
Adding notebook 2_dataloader.ipynb
Adding notebook 1_cifar10-image-dataset.ipynb
Zipping successful! Zip is stored under: /content/drive/MyDrive/HW3/3/exercise03.zip
```

Outlook


Outlook
You have now implemented everything you need to use the CIFAR datasets for deep learning model training. Using your dataset and dataloader, your model training will later look something like the following:
[17] dataset = DummyDataset(
root=None,
divisor=2,
limit=200,
)
dataloader = DataLoader(
dataset=dataset,
batch_size=3,
shuffle=True,
drop_last=True
)
model = lambda x: x
for minibatch in dataloader:
model_output = model(minibatch)
# do more stuff... (soon)