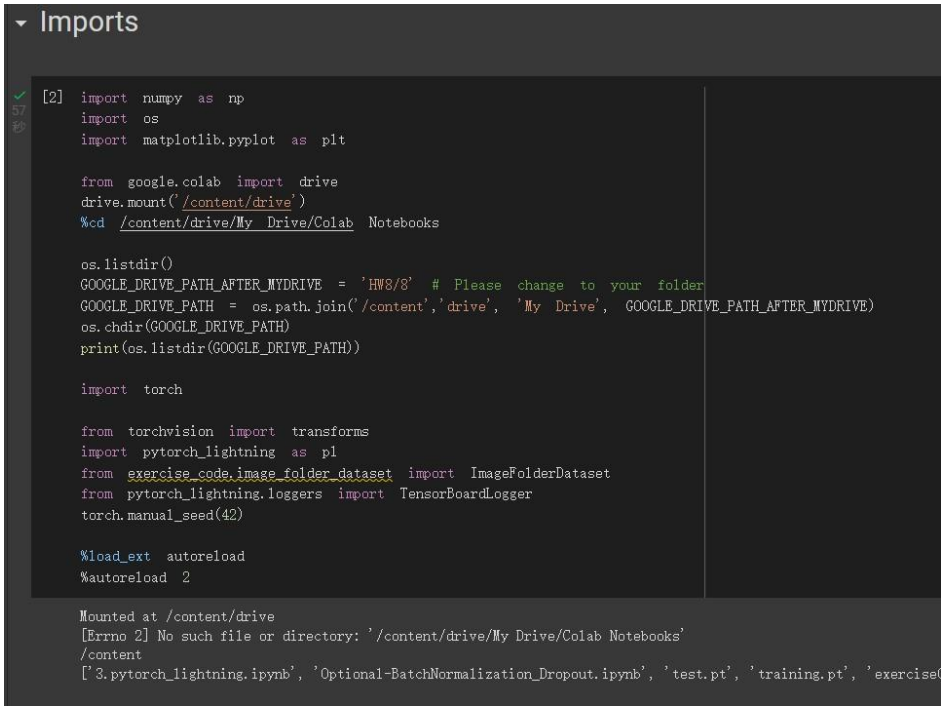
B093040024
郭晏涵

深度視覺

HW5：HW8:pylighting&mnist 資料集

notebook 執行過程

設雲端硬碟路徑



```
Imports

[2] import numpy as np
    import os
    import matplotlib.pyplot as plt

    from google.colab import drive
    drive.mount('/content/drive')
    %cd /content/drive/My Drive/Colab Notebooks

    os.listdir()
    GOOGLE_DRIVE_PATH_AFTER_MYDRIVE = 'HW8/8' # Please change to your folder
    GOOGLE_DRIVE_PATH = os.path.join('/content','drive', 'My Drive', GOOGLE_DRIVE_PATH_AFTER_MYDRIVE)
    os.chdir(GOOGLE_DRIVE_PATH)
    print(os.listdir(GOOGLE_DRIVE_PATH))

    import torch

    from torchvision import transforms
    import pytorch_lightning as pl
    from exercise_code.image_folder_dataset import ImageFolderDataset
    from pytorch_lightning.loggers import TensorBoardLogger
    torch.manual_seed(42)

    %load_ext autoreload
    %autoreload 2

    Mounted at /content/drive
    [Errno 2] No such file or directory: '/content/drive/My Drive/Colab Notebooks'
    /content
    ['3.pytorch_lightning.ipynb', 'Optional-BatchNormalization_Dropout.ipynb', 'test.pt', 'training.pt', 'exercise(
```

設定 transform、分割資料集(將 train data、test data 的 label、data 分開，選擇各一百筆資料作為 train、test、val)

```
[5] ########################################################################
    # TODO: Feel free to define transforms
    ########################################################################

    transform = None

    ########################################################################
    #                                                        END OF YOUR CODE
    ########################################################################


    i2dl_exercises_path = os.path.dirname(os.path.abspath(os.getcwd()))
    mnist_root = os.path.join(i2dl_exercises_path, "datasets", "mnist")

    print(os.getcwd())
    trainPt = torch.load(os.path.join(mnist_root, 'training.pt'))
    testPt = torch.load(os.path.join(mnist_root, 'test.pt'))

    GOOGLE_DRIVE_PATH_AFTER_MYDRIVE = 'HW8' # Please change to your folder
    GOOGLE_DRIVE_PATH = os.path.join('/content','drive', 'My Drive', GOOGLE_DRIVE_PATH_AFTER_MYDRIVE)
    os.chdir(GOOGLE_DRIVE_PATH)
    print(os.listdir(GOOGLE_DRIVE_PATH))
    print(os.getcwd())

    torch.save(trainPt[0][0:99],'datasets/mnist/train_images.pt')
    torch.save(trainPt[0][100:199],'datasets/mnist/val_images.pt')
    torch.save(trainPt[0][200:299],'datasets/mnist/test_images.pt')
    torch.save(trainPt[1][0:99],'datasets/mnist/train_labels.pt')
    torch.save(trainPt[1][100:199],'datasets/mnist/val_labels.pt')
    torch.save(trainPt[1][200:299],'datasets/mnist/test_labels.pt')
```

接續分割資料及設定路徑

```
torch.save(trainPt[0][200:299],'datasets/mnist/test_images.pt')
torch.save(trainPt[1][0:99],'datasets/mnist/train_labels.pt')
torch.save(trainPt[1][100:199],'datasets/mnist/val_labels.pt')
torch.save(trainPt[1][200:299],'datasets/mnist/test_labels.pt')
torch.save(trainPt[0][300:399],'datasets/mnist/unlabeled_train_images.pt')
torch.save(trainPt[1][300:399],'datasets/mnist/unlabeled_val_images.pt')

GOOGLE_DRIVE_PATH_AFTER_MYDRIVE = 'HW8/8'  # Please change to your folder
GOOGLE_DRIVE_PATH = os.path.join('/content','drive', 'My Drive', GOOGLE_DRIVE_PATH_AFTER_MYDRIVE)
os.chdir(GOOGLE_DRIVE_PATH)


i2dl_exercises_path = os.path.dirname(os.path.abspath(os.getcwd()))
mnist_root = os.path.join(i2dl_exercises_path, "datasets", "mnist")
print(os.getcwd())

train = ImageFolderDataset(root=mnist_root,images='train_images.pt',labels='train_labels.pt',force_download=False,verbose=True,transform=transform)
val = ImageFolderDataset(root=mnist_root,images='val_images.pt',labels='val_labels.pt',force_download=False,verbose=True,transform=transform)
test = ImageFolderDataset(root=mnist_root,images='test_images.pt',labels='test_labels.pt',force_download=False,verbose=True,transform=transform)


# We also set up the unlabeled images which we will use later
unlabeled_train = ImageFolderDataset(root=mnist_root,images='unlabeled_train_images.pt',force_download=False,verbose=True,transform=transform)
unlabeled_val = ImageFolderDataset(root=mnist_root,images='unlabeled_val_images.pt',force_download=False,verbose=True,transform=transform)
```

```
/content/drive/My Drive/HW8/8
['8', 'datasets']
/content/drive/My Drive/HW8
/content/drive/My Drive/HW8/8
Found dataset folder. Skipped downloading. If you face issues, please re-download the dataset using
'--force_download=True'
http://i2dl.vc.in.tum.de/static/data/mnist.zip
Found dataset folder. Skipped downloading. If you face issues, please re-download the dataset using
'--force_download=True'
```

測試資料集輸出結果

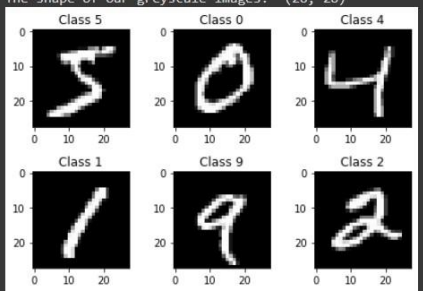
Let's turn a few of the images into numpy arrays, to look at their shape and visualize them and see if the labels we paid for are correct.

```
plt.rcParams['figure.figsize'] = (6,6) # Make the figures a bit bigger

for i in range(9):
        image = np.array(train[i][0].squeeze()) # get the image of the data sample
        label = train[i][1] # get the label of the data sample
        plt.subplot(3,3,i+1)
        plt.imshow(image, cmap='gray', interpolation='none')
        plt.title("Class {}".format(label))

plt.tight_layout()
print('The shape of our greyscale images: ', image.shape)
```

```
The shape of our greyscale images:  (28, 28)
```



測試分類器、Encoder

Task: Implement

Implement the `configure_optimizers` method of the `Classifier` in `exercise_code/models.py`.

```
from exercise_code.models import Encoder
from exercise_code.models import Classifier

################################################################################
# TODO: Define your hyper parameters here!                                     #
################################################################################

hparams = {'n_hidden':10, 'batch_size':10, 'layer_1_dim':128}

################################################################################
#                                                          END OF YOUR CODE
################################################################################

encoder = Encoder(hparams)
classifier = Classifier(hparams, encoder, train, val, test)
```

Encoder 初始化

```python
class Encoder(nn.Module):

    def __init__(self, hparams, input_size=28 * 28, latent_dim=20):
        super().__init__()

        # set hyperparams
        self.latent_dim = latent_dim
        self.input_size = input_size
        self.hp = hparams
        self.encoder = None

        ########################################################################
        # TODO: Initialize your encoder!
        ########################################################################

        self.encoder = nn.Sequential(
            nn.Linear(input_size, self.hp["n_hidden"]),
            nn.ReLU(),
            nn.Linear(self.hp["n_hidden"], 784)
        )

        ########################################################################
        #                                                       END OF YOUR CODE
        ########################################################################

    def forward(self, x):
        # feed x into encoder!
        return self.encoder(x)
```

Classifier 分類器初始化

```python
class Classifier(pl.LightningModule):

    def __init__(self, hparams, encoder, train_set=None, val_set=None, test_set=None):
        super().__init__()
        # set hyperparams
        self.hp = hparams
        self.encoder = encoder
        self.model = nn.Identity()
        self.data = {'train': train_set,
                     'val': val_set,
                     'test': test_set}
        ########################################################################
        # TODO: Initialize your classifier!
        # Remember that it must have the same inputsize as the outputsize      #
        # of your encoder
        ########################################################################

        self.data['train'] = self.forward(torch.flatten(self.data['train'].images, 1).float())
        self.data['val'] = self.forward(torch.flatten(self.data['val'].images, 1).float())
        self.data['test'] = self.forward(torch.flatten(self.data['test'].images, 1).float())

        ########################################################################
        #                                                       END OF YOUR CODE
        ########################################################################
    def configure_optimizers(self):

        optim = None
        ########################################################################
        # TODO: Define your optimizer.
        ########################################################################

        optim = torch.optim.SGD([{'params': self.model.parameters()}], lr=0.01, momentum=0.9)

        ########################################################################
        #                                                       END OF YOUR CODE
        ########################################################################
        return optim

```

測試 trainer

```
import copy
trainer = None

trainer = pl.Trainer(
        max_epochs=100,
        gpus=1 if torch.cuda.is_available() else None
)

trainer.fit(classifier) # train the standard classifier
print("Validation accuracy when training from scratch: {}%".format(classifier.getAcc(classifier.val_dataloader())[1]*100))
```

```
GPU available: True, used: True
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs
Missing logger folder: /content/drive/My Drive/HW8/8/lightning_logs
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]

  | Name    | Type     | Params
-----------------------------------
0 | encoder | Encoder  | 16.5 K
1 | model   | Identity | 0
-----------------------------------
16.5 K    Trainable params
0         Non-trainable params
16.5 K    Total params
0.066     Total estimated model params size (MB)
```

測試 Autoencoder

3.2 Autoencoder Training

Now, we can train the full autoencoder consisting of both en- and decoder.

```
[9] from exercise_code.models import Autoencoder, Encoder, Decoder

    ########################################################################
    # TODO: Define your hyperparameters here!                              #
    ########################################################################

    hparams = {'n_hidden':10, 'batch_size':10, 'layer_1_dim':128}

    ########################################################################
    #                                                  END OF YOUR CODE
    ########################################################################

    encoder_pretrained = Encoder(hparams)
    decoder = Decoder(hparams)
    ae_logger = TensorBoardLogger(save_dir='lightning_logs')
    autoencoder = Autoencoder(hparams, encoder_pretrained, decoder, unlabeled_train, unlabeled_val, ae_logger)
```

Some tests to check whether we'll accept your model.

```
[10] from exercise_code.Util import printModelInfo, load_model
     _ = printModelInfo(autoencoder)

     FYI: Your model has 0.024 mio. params.
     Model accepted!
```

Autoencoder forward

```
class Autoencoder(pl.LightningModule):

    def __init__(self, hparams, encoder, decoder, train_set, val_set, logger):
        super().__init__()
        self.hp = hparams
        # set hyperparams
        self.encoder = encoder
        self.decoder = decoder
        self.train_set = train_set
        self.val_set = val_set
        self.log = logger

    def forward(self, x):
        reconstruction = None
        ########################################################################
        # TODO: Feed the input image to your encoder to generate the latent
        #    vector. Then decode the latent vector and get your reconstruction
        #    of the input.
        ########################################################################


        self.model = nn.Sequential(
            nn.Linear(self.hp["n_hidden"], 10),
            nn.ReLU(),
            nn.Linear(10, self.hp["n_hidden"])
        )

        ########################################################################
        #                                                  END OF YOUR CODE
        ########################################################################
        return reconstruction
```

Training Autoencoder

```
ae_trainer = None

################################################################################
# TODO: Define your trainer! Don't forget the logger.                          #
################################################################################


ae_trainer = pl.Trainer(gpus=1)


################################################################################
#                                                              END OF YOUR CODE
################################################################################
ae_trainer.fit(autoencoder)
```

```
/usr/local/lib/python3.7/dist-packages/pytorch_lightning/loops/utilities.py:94: PossibleUserWarning: `
  category=PossibleUserWarning,
GPU available: True, used: True
TPU available: False, using: 0 TPU cores
IPU available: False, using: 0 IPUs
HPU available: False, using: 0 HPUs
Missing logger folder: /content/drive/My Drive/HW8/8/lightning_logs
LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES: [0]
```

設定 hyper parameter、training classifier

```
[13] ################################################################################
     # TODO: Define your hyper parameters here!                                     #
     ################################################################################

     hparams = {}

     ################################################################################
     #                                                              END OF YOUR CODE
     ################################################################################
     classifier_pretrained = Classifier(hparams, encoder_pretrained, train, val, test)
```

Now specify another trainer that we will use the pretrained classifier to compare its performance with the classifier we t
labeled data. You might need to optimize the parameters defined above in order to achieve a reasonable result.

```
trainer = None

################################################################################
# TODO: Define your trainer! Don't forget the logger.                          #
# Hint: Choose an appropriate logging frequency in your trainer.               #
################################################################################

trainer = pl.Trainer(gpus=1)

################################################################################
#                                                              END OF YOUR CODE
################################################################################

trainer.fit(classifier_pretrained) # train the standard classifier
```

```
/usr/local/lib/python3.7/dist-packages/pytorch_lightning/loops/utilities.py:94: PossibleUserWarning: `max
  category=PossibleUserWarning,
GPU available: True, used: True
TPU available: False, using: 0 TPU cores
```

完成訓練、輸出最後格式

Let's have a look at the validation accuracy of the two different classifiers and compare them. And don't forget that you can also monitor your training in TensorBoard.

We will only look at the test accuracy and compare our two classifiers with respect to that in the very end.

```
[15] print("Validation accuracy when training from scratch: {}%".format(classifier.getAcc(classifier.val_dataloader())[1]*100))
     print("Validation accuracy with pretraining: {}%".format(classifier_pretrained.getAcc(classifier.val_dataloader())[1]*100))
```

```
Validation accuracy when training from scratch: 100%
Validation accuracy with pretraining: 68%
```

Now that everything is working, feel free to play around with different architectures. As you've seen, it's really easy to define your model or do changes there.

```
from exercise_code.Util import test_and_save

print("Test accuracy when training from scratch: {}%".format(classifier.getAcc()[1]*100))
print('\nNow to the pretrained classifier:')
test_and_save(classifier_pretrained)
```

```
[17] # Now zip the folder for upload
     from exercise_code.submit import submit_exercise

     submit_exercise('exercise08')
```

```
relevant folders: ['exercise_code']
notebooks files: ['3.pytorch_lightning.ipynb', 'Optional-BatchNormalization_Dropout.ipynb', '1_Autoencoder_PyTorch_Lightning.ipynb']
Adding folder exercise_code
Adding notebook 3.pytorch_lightning.ipynb
Adding notebook Optional-BatchNormalization_Dropout.ipynb
Adding notebook 1_Autoencoder_PyTorch_Lightning.ipynb
Zipping successful! Zip is stored under: /content/drive/My Drive/HW8/8/exercise08.zip
```

Congrats! You've now finished your first autoencoder and transferred the weights to a classifier! Much easier than in plain numpy, right? But wait, to complete the exercise

Submission Goals

- Goal: Successfully implement a fully connected autoencoder for MNIST with Pytorch Lightning and transfer the encoder weights to a classifier.

decoder

```
class Decoder(nn.Module):

    def __init__(self, hparams, latent_dim=20, output_size=28 * 28):
        super().__init__()

        # set hyperparams
        self.hp = hparams
        self.decoder = None

        ########################################################################
        # TODO: Initialize your decoder!
        ########################################################################

        self.encoder = nn.Sequential(
            nn.Linear(self.hp["n_hidden"], 10),
            nn.ReLU(),
            nn.Linear(output_size, self.hp["n_hidden"])
        )

        ########################################################################
        #                                                       END OF YOUR CODE
        ########################################################################
    def configure_optimizers(self):

        optim = None
        ########################################################################
        # TODO: Define your optimizer.
        ########################################################################

        self.train_set = torch.tensor(self.train_set)
        self.val_set = torch.tensor(self.val_set)
        optim = torch.optim.Adam([self.train_set, self.val_set], lr=0.0001)

        ########################################################################
        #                                                   END OF YOUR CODE
        ########################################################################
        return optim
```