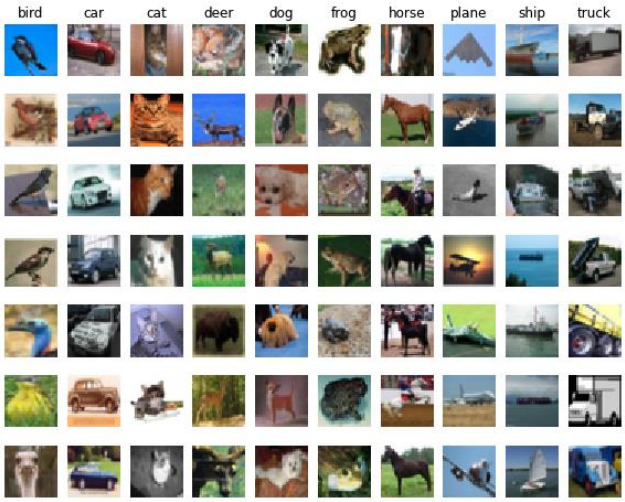
HW3: CIFAR10 and DataLoader
B083040029 邱品諺

1. CIFAR10-image-dataset

CIFAR10 為包含了 50000 張 32x32 的 RGB 圖片並分類成 10 個 class 的 dataset。下圖為以每類 7 張圖片示意：



(1) Dataset Download

宣告變數 dataset 作為 class ImageFolderDataset 的 object，並從此網站下載 CIFAR10 dataset 至指定的 google drive 路徑，其中 ImageFolderDataset 繼承了 class Dataset，且 Dataset 繼承了 Abstract Base Method (ABC)，使 Dataset 作為 Abstract class。在 Dataset 中指定了兩個 abstract method: __getitem__(), __len__() 使其 child class 自行定義函式內容。在 class ImageFolderDataset 中定義了四個變數：classes, class_to_idx, images, labels。其中_find_classes()回傳變數 classes 及 class_to_idx，前者為以 10 個種類名稱所形成的 list，並以字母順序排序，後者為每個種類所對應的 label 為何，函式中以數字 0-9 為代表，並以 dictionary 儲存，像是 {'bird': 0, 'car': 1, …}。而 make_dataset()回傳的變數 images 為 dataset 中各張圖片的路徑，而 labels 為各張圖片所對應的 label。

(2) ImageFolderDataset Implementation

__len__(): 此函式回傳 dataset 的長度，而 CIFAR10 的 dataset 為 50000 筆資料，因此長度為 50000。此函式可使 ImageFolderDataset 的 Instance 可直接使用 len()獲得 dataset 長度。

__getitem__(): 此函數回傳 dataset 中的指定項目，首先利用 load_image_as_numpy 將路徑的圖片轉換成長度為 32x32x3 的 NumPy array，並將該圖片對應的 label 以一個 pair 的方式存入 dictionary 並回傳。

(3) Transform and Image Preprocessing

此部分實作了三種調整數據比例的方式：

a. Rescale

以圖片為例，每個 pixel 的亮暗程度以 0-255 表示，若想 rescale 至 0-1 之間，以下為計算公式：

$$\frac{Xoriginal - 0}{255 - 0} = \frac{Xtarget - 0}{1 - 0}$$

```
def __call__(self, images):
    ###########################################################################
    # TODO:                                                                   #
    # Rescale the given images:                                               #
    #   - from (self._data_min, self._data_max)                               #
    #   - to (self.min, self.max)                                             #
    ###########################################################################
    images = ((self.max - self.min) / (self._data_max - self._data_min)) \
        * (images - self._data_min) + self.min

    ###########################################################################
    #                             END OF YOUR CODE                            #
    ###########################################################################
    return images
```

b. Normalization

此部分須對每個 channel 執行，也就是分別對 RGB 三個軸 normalize，計算為後數值範圍將落在[0, 1]，以下為計算公式：

$$z = \frac{x - \min(x)}{\max(x) - \min(x)}$$

c. Standardization

此部分也須對每個 channel 執行，計算完畢後每個 channel 的平均值為 0、標準差為 1，以下為計算公式：

$$z = \frac{x_i - \mu}{\sigma}$$

最後，利用 ComposeTransform 傳入欲對資料集依序進行的前處理方式以進行資料前處理，再轉換成長度為 32x32x3 的 NumPy array 及其對應的 label 以 dictionary 輸出。

2. DataLoader

由於深度學習在資料載入及模型訓練上，資料通常是以 batch 為單位，就像是有好多個一樣大小的子集所組合成的資料集，因此此部分我們將自己實作 DataLoader 以改變資料讀取的方式。

(1) Iterating over a dataset

首先此部分將利用 1-100 的偶數作為 dataset，且每一筆資料的型態為 dictionary，如：{'data': number}。而此部分每個 batch 的大小為 3，首先我們可以用建立一個 list 持續 append，如下圖：

```
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}]
```

但我們希望可以將每個 batch 的數值都放在同個 dictionary，因此我們可以遞迴每個 batch，並將其中每筆資料放進 list 後再放入一個新的 dictionary，最後再轉換成 NumPy array。

```
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 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])}
```

但實際上在載入 dataset 時，除了可以決定是否要 shuffle 外，通常不會一次將所有資料載入，因為這樣需要許多記憶體。因此為了節省記憶體的使用，利用 generator 並搭配 for 迴圈可以每次載入一部分的資料直到 raise StopIteration 並結束資料載入。

(2) DataLoader Class Implementation

此部分為自己實作 class DataLoader，主要為實作__len__()、__iter__()，不過和上個部分不同的地方為需要考慮每個 batch 是否能剛好裝到預設的 batch 大小。

__len__():

此函式決定 batch 的數量，因此只需將 dataset 的長度除以每個 batch 的大小，並考慮是否需要將最後未能裝滿的 batch 作為一個 batch 即可。

```
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!                                     #
    ############################################################################

    if self.drop_last:
        length = len(self.dataset) // self.batch_size
    else:
        length = int(np.ceil(len(self.dataset) / self.batch_size))

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

__iter__():

此部分其實就是將第一部分實作於此函式中，首先先決定資料載入順序的 index，並決定是否需要 shuffle 載入，接下來以設定的 batch 大小為單位形成一個個 batch，並透過 generator 和 for 迴圈將每個 batch 回傳。最後如果需要將未能裝滿的 batch 回傳，只需看最後欲裝進 batch 的 list 是否為空便可判斷。

```
def __iter__(self):
    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

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

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

    batch = []
    for index in index_iterator:
        batch.append(self.dataset[index])
        if len(batch) == self.batch_size:
            yield batch_to_numpy(combine_batch_dicts(batch))
            batch = []

    if len(batch) > 0 and not self.drop_last:
        yield batch_to_numpy(combine_batch_dicts(batch))
```