Exercise 03 — Solution Reference

I2DL — Introduction to Deep Learning

exercise_code/data/dataloader.py

```
################################################################################
# TODO:                                                                        #
# Define an iterable function that samples batches from the dataset.           #
# Each batch should be a dict containing numpy arrays of length                #
# batch_size (except for the last batch if drop_last=True)                     #
# Hints:                                                                       #
#   - np.random.permutation(n) can be used to get a list of all                #
#     numbers from 0 to n-1 in a random order                                  #
#   - To load data efficiently, you should try to load only those              #
#     samples from the dataset that are needed for the current batch.          #
#     An easy way to do this is to build a generator with the yield            #
#     keyword, see https://wiki.python.org/moin/Generators                     #
#   - Have a look at the "DataLoader" notebook first. This function is         #
#     supposed to combine the functions:                                       #
#       - combine_batch_dicts                                                  #
#       - batch_to_numpy                                                       #
#       - build_batch_iterator                                                 #
#     in section 1 of the notebook.                                            #
################################################################################

def combine_batch_dicts(batch):
    """
    Combines a given batch (list of dicts) to a dict of numpy arrays
    :param batch: batch, list of dicts
        e.g. [{k1: v1, k2: v2, ...}, {k1:, v3, k2: v4, ...}, ...]
    :returns: dict of numpy arrays
        e.g. {k1: [v1, v3, ...], k2: [v2, v4, ...], ...}
    """
    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):
    """Transform all values of the given batch dict to numpy arrays"""
    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))

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

```
#############################################################################
# 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 (self.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                               #
#############################################################################
```

exercise_code/data/image_folder_dataset.py

```
#############################################################################
# TODO:                                                                     #
# Return the length of the dataset (number of images)                       #
#############################################################################
length = len(self.images)
#############################################################################
#                            END OF YOUR CODE                               #
#############################################################################
```

```
#############################################################################
# TODO:                                                                     #
# Create a dict of the data at the given index in your dataset              #
# The dict should be of the following format:                               #
# {"image": <i-th image>,                                                   #
# "label": <label of i-th image>}                                           #
#                                                                           #
```

```
# Hint 1:                                                                 #
#   use self.load_image_as_numpy() to load an image from a                #
#   file path. Note: you have to use "self.load_image_as_numpy()"         #
#   and not "ImageFolderDataset.load_image_as_numpy()", as it will        #
#  cause a bug, when using  MemoryImageFolderDataset                      #
#                                                                         #
# Hint 2:                                                                 #
#   If applicable (Task 4: 'Transforms and Image Preprocessing'),         #
#   make sure to apply self.transform to the image if self.transform      #
#   is defined (not None):                                                #
#               image_transformed = self.transform(image)                 #
#                                                                         #
# Hint 3:                                                                 #
#   The labels are supposed to be numbers, in the range of [0, 9],        #
#   not strings.                                                          #
#                                                                         #
# Hint 4: the labels and images are already prepared and stored in        #
#  self.labels and self.images. DO NOT call self.make_dataset() again! #
###########################################################################

label = self.labels[index]
path = self.images[index]
image = self.load_image_as_numpy(path)

image = self.transform(image)
data_dict = {
    "image": image,
    "label": label,
}
###########################################################################
#                           END OF YOUR CODE                              #
###########################################################################
```

exercise_code/data/transforms.py

```
###########################################################################
# TODO:                                                                   #
# Rescale the given images:                                               #
#   - from (self._data_min, self._data_max)                               #
#   - to (self.min, self.max)                                             #
# Hint 1:                                                                 #
#       Google the following algorithm:                                   #
#       "convert-a-number-range-to-another-range-maintaining-ratio"       #
# Hint 2:                                                                 #
#       Don't change the image in-place (directly in the memory),         #
#       but return a copy with ret_image                                  #
###########################################################################

ret_image = image - self._data_min  # normalize to (0, data_max-data_min)
ret_image /= (self._data_max - self._data_min)  # normalize to (0, 1)
ret_image *= (self.max - self.min)  # norm to (0, target_max-target_min)
ret_image += self.min  # normalize to (target_min, target_max)

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

```
###############################################################################
# TODO:                                                                       #
# Calculate the per-channel mean and standard deviation of the images         #
# Hint 1: You can use numpy to calculate the mean and standard                #
# deviation.                                                                  #
#                                                                             #
# Hint 2: Make sure that the shapes of the resulting arrays are (C,)          #
# and not [1, C], [C, 1] or anything else. Use print(mean.shape) to           #
# test yourself.                                                              #
###############################################################################

# mean = [np.mean(images[:, :, :, c]) for c in range(images.shape[3])]
# std = [np.std(images[:, :, :, c]) for c in range(images.shape[3])]

# # Alternative - slower:
# mean = np.mean(images, axis=(0, 1, 2))
# std = np.std(images, axis=(0, 1, 2))
images = images.reshape(-1, images.shape[-1])
mean = np.mean(images, axis=0)
std = np.std(images, axis=0)

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

```
###############################################################################
# TODO:                                                                       #
# normalize the given images:                                                 #
#   - substract the mean of dataset                                           #
#   - divide by standard deviation                                            #
###############################################################################
images = (images - self.mean) / self.std
###############################################################################
#                           END OF YOUR CODE                                  #
###############################################################################
```