Exercise 3: Solution

Dummy Dataset Forward Pass

```
31
32  class DummyNetwork(Network):
33      """
34      A Dummy network which takes in an input numpy array and computes its sigmoid
35      """
36
37      def __init__(self, model_name="dummy_model"): ...
42
43      def forward(self, x):
44          """
45          :param x: The input to the network
46          :return: results of computation of sigmoid on the input
47          """
48          ##########################################################################
49          # TODO                                                                   #
50          # Implement the sigmoid function.                                        #
51          #                                                                        #
52          ##########################################################################
53          x = 1 / (1 + np.exp(-x))
54          ##########################################################################
55          #                             END OF YOUR CODE                           #
56          ##########################################################################
57
58          return x
59
60      def __repr__(self):
61          return "A dummy class that would compute sigmoid function"
62
63      def save_model(self, data=None): ...
69
```

ImageFolderDataset make_dataset()

```python
    @staticmethod
    def make_dataset(directory, class_to_idx):
        """
        Create the image dataset by preparaing a list of samples
        :param directory: root directory of the dataset
        :param class_to_idx: A dict that maps classes to labels
        :returns: (images, labels) where:
            - images is a list containing paths to all images in the dataset
            - labels is a list containing one label per image
        """
        images, labels = [], []
        ############################################################################
        # TODO:                                                                    #
        # Construct a list of all images in the dataset                            #
        # and a list of corresponding labels.                                      #
        # Hints:                                                                   #
        #   - have a look at the "CIFAR-10: Image Dataset" notebook first          #
        #   - class_idx contains all classes and corresponding labels              #
        #   - images should only contain file paths, NOT the actual images         #
        #   - sort images by class and file name (ascending)                       #
        ############################################################################

        for target_class in sorted(class_to_idx.keys()):
            label = class_to_idx[target_class]
            target_dir = os.path.join(directory, target_class)
            for root, _, fnames in sorted(os.walk(target_dir)):
                for fname in sorted(fnames):
                    path = os.path.join(root, fname)
                    images.append(path)
                    labels.append(label)

        ############################################################################
        #                             END OF YOUR CODE                             #
        ############################################################################
        assert len(images) == len(labels)
        return images, labels
```

Hints

- class_to_idx is a dictionary mapping the name of a label to an index.
- All images having the same label are found in a directory named just as the label.

Code

- We iterate through the labels using class_to_idx.
- We obtain the path of the directory containing all images with a certain label (target_dir) by concatenating the parent directory path (directory) with the name of the label (label).
- We get the names of all files in the target directory and add them to the images list.

ImageFolderDataset __len__()

```
77
78      def __len__(self):
79          length = None
80          ############################################################################
81          # TODO:                                                                    #
82          # Return the length of the dataset (number of images)                      #
83          ############################################################################
84
85          length = len(self.images)
86
87          ############################################################################
88          #                             END OF YOUR CODE                             #
89          ############################################################################
90          return lengthscr
91
```

ImageFolderDataset __getitem__()

```python
def __getitem__(self, index):
    data_dict = None
    ###########################################################################
    # 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>}                 #
    # Hints:                                                                  #
    #    - use load_image_as_numpy() to load an image from a file path        #
    #    - Make sure to apply self.transform to the image                     #
    ###########################################################################

    label = self.labels[index]
    path = self.images[index]
    image = self.load_image_as_numpy(path)
    if self.transform is not None:
        image = self.transform(image)
    data_dict = {
        "image": image,
        "label": label,
    }

    ###########################################################################
    #                           END OF YOUR CODE                              #
    ###########################################################################
    return data_dict
```

Hints

- self.images[index] contains the full name of the image we want to retrieve (we don't want to keep all images in memory at the same time - we only read them when it's required)
- Self.labels[index] contains the label of the image we want to retrieve
- We only apply the transformation if it's not None

RescaleTransform __call__()

```
161     def __call__(self, images):
162         ###########################################################################
163         # TODO:                                                                   #
164         # Rescale the given images:                                               #
165         #   - from (self._data_min, self._data_max)                               #
166         #   - to (self.min, self.max)                                             #
167         ###########################################################################
168 
169         images = images - self._data_min  # normalize to (0, data_max-data_min)
170         images /= (self._data_max - self._data_min)  # normalize to (0, 1)
171         images *= (self.max - self.min)  # norm to (0, target_max-target_min)
172         images += self.min  # normalize to (target_min, target_max)
173 
174         ###########################################################################
175         #                             END OF YOUR CODE                            #
176         ###########################################################################
177         return images
178 
```

compute_image_mean_and_std()

```
124
125  def compute_image_mean_and_std(images):
126      """
127      Calculate the per-channel image mean and standard deviation of given images
128      :param images: numpy array of shape NxHxWxC
129          (for N images with C channels of spatial size HxW)
130      :returns: per-channels mean and std; numpy array of shape C
131      """
132      mean, std = None, None
133      ################################################################################
134      # TODO:                                                                        #
135      # Calculate the per-channel mean and standard deviation of the images          #
136      # Hint: You can use numpy to calculate mean and standard deviation             #
137      ################################################################################
138
139      mean = np.mean(images, axis=(0, 1, 2))
140      std = np.std(images, axis=(0, 1, 2))
141
142      ################################################################################
143      #                              END OF YOUR CODE                                #
144      ################################################################################
145      return mean, std
146
```

Dataloader: Iterator Helpers

```python
    def __iter__(self):
        ###########################################################################
        # 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                      #
        ###########################################################################
        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
```

Hints

We create two helper functions: one for merging a batch of dictionaries as well as a convenient way to convert those dictionaries to numpy arrays which we will then feed to our networks later

Dataloader: Iterator Implementation

```
63
64          if self.shuffle:
65              index_iterator = iter(np.random.permutation(len(self.dataset)))
66          else:
67              index_iterator = iter(range(len(self.dataset)))
68
69          batch = []
70          for index in index_iterator:
71              batch.append(self.dataset[index])
72              if len(batch) == self.batch_size:
73                  yield batch_to_numpy(combine_batch_dicts(batch))
74                  batch = []
75
76          if len(batch) > 0 and not self.drop_last:
77              yield batch_to_numpy(combine_batch_dicts(batch))
78          ########################################################################
79          #                            END OF YOUR CODE                          #
80          ########################################################################
```

Hints

- Shuffling is implemented here using numpy's random permutation but there are multiple possible solutions
- We iterate over the dataset and use yield to properly invoke our iterator
- Finally we have to check for the last batch size in order to account for "drop_last".

Dataloader: Length

```
81
82      def __len__(self):
83          length = None
84          ############################################################################
85          # TODO:                                                                    #
86          # Return the length of the dataloader                                      #
87          # Hint: this is the number of batches you can sample from the dataset      #
88          ############################################################################
89          if self.drop_last:
90              length = len(self.dataset) // self.batch_size
91          else:
92              length = int(np.ceil(len(self.dataset) / self.batch_size))
93          ############################################################################
94          #                             END OF YOUR CODE                             #
95          ############################################################################
96          return length
```

Questions? Moodle ☺