Introduction:

In this template, methods are provided to get you started on the task at hand (please see project description). Please implement your solution in the code cells marked with **TODO**. Most of the other code cells are hidden, feel free to explore and change these. These cells implement a basic pipeline for training your model but you may want to explore more complex procedures. **Make sure you run all cells before trying to implement your own solution!**

Imports and definitions:

```
#@title
import numpy as np
import requests
import io
from torch.utils.data import TensorDataset
import matplotlib.pyplot as plt
import torch
from scipy.ndimage import rotate
%matplotlib inline
from torch.utils.data import Dataset
import random
import torch
import torchvision.transforms as transforms
from sklearn.model_selection import train_test_split
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from torchsummary import summary
from tqdm import tqdm
import itertools



class BatchSampler():
  """
  Implements an iterable which given a torch dataset and a batch_size
  will produce batches of data of that given size. The batches are
  returned as tuples in the form (images, labels).
  Can produce balanced batches, where each batch will have an equal
  amount of samples from each class in the dataset. If your dataset is hea
  imbalanced, this might mean throwing away a lot of samples from
  over-represented classes!
```

```
      # Counting the ocurrence of the class labels:
      unique, counts = np.unique(self.dataset.targets, return_counts=True)
      indexes = []
      # Sampling an equal amount from each class:
      for i in range(len(unique)):
        indexes.append(np.random.choice(np.where(self.dataset.targets == i)
      # Setting the indexes we will sample from later:
      self.indexes = np.concatenate(indexes)
    else:
      # Setting the indexes we will sample from later (all indexes):
      self.indexes = [i for i in range(len(dataset))]


  def __len__(self):
    return (len(self.indexes) // self.batch_size) + 1

  def shuffle(self):
    # We do not need to shuffle if we use the balanced sampling method.
    # Shuffling is already done when making the balanced samples.
    if not self.balanced:
      random.shuffle(self.indexes)

  def __iter__(self):
    remaining = False
    self.shuffle()
    # Go over the datset in steps of 'self.batch_size':
    for i in range(0, len(self.indexes), self.batch_size):
        imgs, labels = [], []
        # If our current batch is larger than the remaining data, we quit:
        if i + self.batch_size > len(self.indexes):
          remaining = True
          break
        # If not, we yield a complete batch:
        else:
          # Getting a list of samples from the dataset, given the indexes
          X_batch = [self.dataset[self.indexes[k]][0] for k in range(i, i +
          Y_batch = [self.dataset[self.indexes[k]][1] for k in range(i, i +
          # Stacking all the samples and returning the target labels as a t
          yield torch.stack(X_batch).float(), torch.tensor(Y_batch).long()
    # If there is still data left that was not a full batch:
    if remaining:
```

```
 92
 93
 94   def __init__(self, x, y, transform=None, target_transform=None):
 95     self.targets = y
 96     self.imgs = x
 97     self.transform = transform
 98     self.target_transform = target_transform
 99
100   def __len__(self):
101     return len(self.targets)
102
103   def __getitem__(self, idx):
104     image = torch.from_numpy(self.imgs[idx] / 255).float()
105     label = self.targets[idx]
106     return image, label
107
108 def load_numpy_arr_from_url(url):
109     """
110     Loads a numpy array from surfdrive.
111
112     Input:
113     url: Download link of dataset
114
115     Outputs:
116     dataset: numpy array with input features or labels
117     """
118
119     response = requests.get(url)
120     response.raise_for_status()
121
122     return np.load(io.BytesIO(response.content))
123
124
125 class_labels = {0: 'Atelectasis',
126                 1: 'Effusion',
127                 2: 'Infiltration',
128                 3: 'No Finding',
```

```
#@title
# Downloading the images:
train_x = load_numpy_arr_from_url('https://surfdrive.surf.nl/files/index.pl
test_x = load_numpy_arr_from_url('https://surfdrive.surf.nl/files/index.php
```

changes on data set

Data Augmentation (not being used in base model)

```
# data augentation

# defining transfrmation for data points

preprocess_aug = transforms.Compose([

    #preprocess takes tensor x
    transforms.Lambda(lambda x: x.repeat(1, 3, 1, 1)),
    transforms.Resize(299),
    transforms.CenterCrop(299),
    transforms.RandomResizedCrop(299),
    transforms.RandomHorizontalFlip(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.
    transforms.RandomResizedCrop((128,128))
])
```

```
#example of transformed data point
x_eg = torch.cuda.FloatTensor(test_x[0].reshape(128,128))
```

```
# Function used for previous testing of model

def sizedSet(set_x, set_y):
    """Given a data set x this function makes all the labeled (by y) sub se
    the same size as the smallest subset of x by removing the other points
    """

    # working out arrays to be used below

    set_arr = np.array([[test_x[i],test_y[i]] for i in range(len(test_y))])
    set_arr = set_arr[set_arr[:,1].argsort()]
    label_sizes = np.array([np.count_nonzero(test_y==i) for i in class_labe
    size = int(min(label_sizes))


    # taking care of indeces for new set

    new_arr=np.array([])
    index = 0
    for l in range(np.size(label_sizes)):
        for i in range(size):
            new_arr = np.append(new_arr , [set_arr[index+i,0],set_arr[index
        index += label_sizes[l]
```

```
 6 xval=  np.array([train_x[i] for i in xval_index])
 7 yval=  np.array([train_y[i] for i in xval_index])
 8
 9
10
11 train_index= [i for i in indexs if i not in xval_index]
12 train_xx= np.array([train_x[i] for i in train_index])
13 train_yy= np.array([train_y[i] for i in train_index])
14
15 val_dataset = ImageDataset(xval, yval)  #test
16
17 # Comment out not to use validation split
18 train_x = train_xx
19 train_y = train_yy
20
```

Plotting the data distribution:

```
2 test_dataset = ImageDataset(test_x, test_y)
```

Defining our model as a neural network:

TODO define your own model here, follow the structure as presented in the Pytorch tutorial (or see below as an example).

```
1 class Net(nn.Module):
2     def __init__(self, n_classes):
3         super(Net, self).__init__()
```

```
46
47 # Make sure your model instance is assigned to a variable 'model':
48 model = Net(n_classes = 6)
```

Moving model to CUDA, verifying model structure and