```
# ## install syft package to use Private Aggregation of Teacher Ensembles (PATE)
#!pip install syft==0.2.9


  # import our libraries
import numpy as np
import pandas as pd
import torch
from torchvision import datasets, transforms,models
from torch.utils.data import Dataset, Subset, DataLoader
from torch import nn, optim
import torch.nn.functional as F
from PIL import Image
import time, os, random


# libary from pysyft needed to perform pate analysis
from syft.frameworks.torch.dp import pate

# we'll train on GPU if it is available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")



## authorize access to google drive
from google.colab import drive
drive.mount('/content/drive')

# navigate to project directory
%cd '/content/drive/My Drive/Colab Notebooks/'
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mou
/content/drive/My Drive/Colab Notebooks
```

```
# Custom dataset
#from https://github.com/UCSD-AI4H/COVID-CT/blob/master/baseline%20methods/DenseNet169/DenseN
class CovidCTDataset(Dataset):
    def __init__(self, root_dir, txt_COVID, txt_NonCOVID, transform=None):
        """
        Args:
            txt_path (string): Path to the txt file with annotations.
            root_dir (string): Directory with all the images.
            transform (callable, optional): Optional transform to be applied
                on a sample.
        File structure:
        - root_dir
            - CT_COVID
                - img1.png
                - img2.png
```

```
                - ......
            - CT_NonCOVID
                - img1.png
                - img2.png
                - ......
        """
        self.root_dir = root_dir
        self.txt_path = [txt_COVID,txt_NonCOVID]
        self.classes = ['CT_COVID', 'CT_NonCOVID']
        self.num_cls = len(self.classes)
        self.img_list = []
        for c in range(self.num_cls):
            cls_list = [[os.path.join(self.root_dir,self.classes[c],item), c] for item in rea
            self.img_list += cls_list
        self.transform = transform

    def __len__(self):
        return len(self.img_list)

    def __getitem__(self, idx):
        if torch.is_tensor(idx):
            idx = idx.tolist()

        img_path = self.img_list[idx][0]
        image = Image.open(img_path).convert('RGB')

        if self.transform:
            image = self.transform(image)
        label = int(self.img_list[idx][1])
        return image, label



def read_txt(txt_path):
    with open(txt_path) as f:
        lines = f.readlines()
    txt_data = [line.strip() for line in lines]
    return txt_data



batchsize=16
path = './data/images'

# Transforms used for datasets
data_transforms = transforms.Compose([
    transforms.Resize(224),
    transforms.RandomResizedCrop((224),scale=(0.5,1.0)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])
])
```

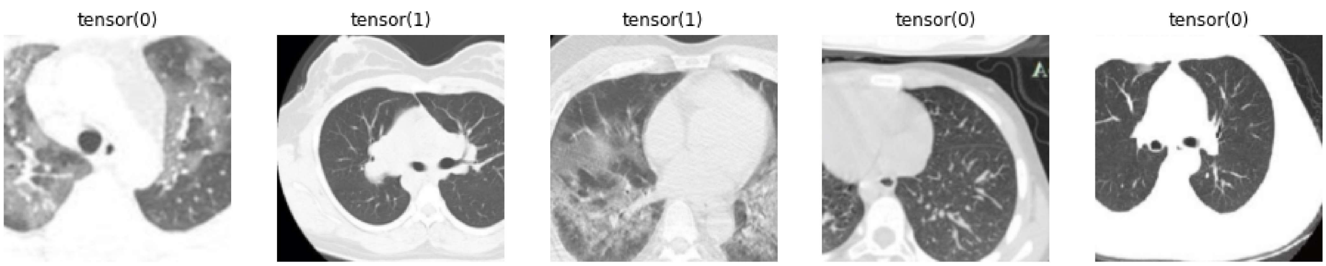
```
# divided among teachers
trainset = CovidCTDataset(root_dir=f'{path}',
                              txt_COVID='./data/labels/COVID/trainCT_COVID.txt',
                              txt_NonCOVID='./data/labels/NonCOVID/trainCT_NonCOVID.txt',
                              transform= data_transforms)

# used as student valid set
validset = CovidCTDataset(root_dir=f'{path}',
                              txt_COVID='./data/labels/COVID/valCT_COVID.txt',
                              txt_NonCOVID='./data/labels/NonCOVID/valCT_NonCOVID.txt',
                              transform= data_transforms)

# used as student train set
testset = CovidCTDataset(root_dir=f'{path}',
                              txt_COVID='./data/labels/COVID/testCT_COVID.txt',
                              txt_NonCOVID='./data/labels/NonCOVID/testCT_NonCOVID.txt',
                              transform= data_transforms)

print("Number of Classes: ",len(trainset.classes))
len(trainset), len(testset), len(validset)
```

```
Number of Classes:  2
(425, 203, 118)
```

```
data_loader = DataLoader(trainset, batch_size=batchsize, shuffle=True)

import matplotlib.pyplot as plt

## Method to display Image for Tensor
def imshow(image, ax=None, title=None, normalize=True):
    """Imshow for Tensor."""
    if ax is None:
        fig, ax = plt.subplots()
    #print(type(image))
    image = image.numpy().transpose((1, 2, 0))

    if normalize:
        mean = np.array([0.485, 0.456, 0.406])
        std = np.array([0.229, 0.224, 0.225])
        image = std * image + mean
        image = np.clip(image, 0, 1)

    ax.imshow(image)
    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
    ax.spines['left'].set_visible(False)
    ax.spines['bottom'].set_visible(False)
    ax.tick_params(axis='both', length=0)
    ax.set_xticklabels('')
    ax.set_yticklabels('')
```

```
    ax.set_yticklabels('')
    return ax


# Displaying Images and other info about the train set
images, labels = next(iter(data_loader))
print(" Image Size",images.size())
#print(" Image Size",images[ii].size())

fig, axes = plt.subplots(figsize=(16,5), ncols=5)
for ii in range(5):
    ax = axes[ii]
    ax.set_title(labels[ii])
    imshow(images[ii], ax=ax, normalize=True)
```

```
 Image Size torch.Size([16, 3, 224, 224])
```




```
# TEACHERS
#divide train set among teachers and create dataloaders for valid and trainsets
num_teachers = 5
valid_per = 0.2 #20% for validation
batch_size = 32

def teacher_dataloaders(transet=trainset, num_teachers=num_teachers, batch_size=batch_size, v
  trainloaders = []
  validloaders = []
  teacher_data_len = len(trainset) // num_teachers

  # create a list of shuffled indices
  my_list = random.sample(range(1,len(trainset)), len(trainset)-1)
  random.shuffle(my_list)

  for i in range(num_teachers):
    # get particular subset of data
    indice = my_list[i*teacher_data_len: (i+1)*teacher_data_len]
    data_subset = Subset(trainset, indice)

    # split into train and validation set
    valid_size = int(len(data_subset) * valid_per)
    train_size = len(data_subset) - valid_size
    train_subset, valid_subset = torch.utils.data.random_split(data_subset, [train_size,valic
```

```
    #create data loaders
    trainloader = DataLoader(train_subset, batch_size=batch_size, shuffle=True, num_workers=1
    validloader = DataLoader(valid_subset, batch_size=batch_size, shuffle=False, num_workers=

    #add dataloaders to list
    trainloaders.append(trainloader)
    validloaders.append(validloader)

  return trainloaders, validloaders

# creating dataloaders
trainloaders, validloaders = teacher_dataloaders()
len(trainloaders), len(validloaders)
```

```
(5, 5)
```

```
#  # STUDENT
# split into train and validation set
valid_size = int(len(testset) * 0.2)
train_size = len(testset) - valid_size
student_train_subset, student_valid_subset = torch.utils.data.random_split(testset, [train_si

#create data loaders
student_train_loader = DataLoader(student_train_subset, batch_size=batch_size, shuffle=False,
student_valid_loader = DataLoader(student_valid_subset, batch_size=batch_size, shuffle=False,

len(student_train_loader), len(student_valid_loader)
```

```
(6, 2)
```

```
class SimpleCNN(torch.nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__() # b, 3, 32, 32
        layer1 = torch.nn.Sequential()
        layer1.add_module('conv1', torch.nn.Conv2d(3, 32, 3, 1, padding=1))

        #b, 32, 32, 32
        layer1.add_module('relu1', torch.nn.ReLU(True))
        layer1.add_module('pool1', torch.nn.MaxPool2d(2, 2))
        self.layer1 = layer1
        layer4 = torch.nn.Sequential()
        layer4.add_module('fc1', torch.nn.Linear(401408, 2))
        self.layer4 = layer4

    def forward(self, x):
        conv1 = self.layer1(x)
        fc_input = conv1.view(conv1.size(0), -1)
        fc_out = self.layer4(fc_input)

        return fc_out
```

```
def train(n_epochs, trainloader, validloader, model, optimizer, criterion, use_cuda, save_pat
    """returns trained model"""
    # # initialize tracker for minimum validation loss
    valid_loss_min = np.Inf

    for epoch in range(1, n_epochs+1):
        # initialize variables to monitor training and validation loss
        train_loss = 0.0
        valid_loss = 0.0
        train_correct = 0.0
        train_total = 0.0
        valid_correct =0.0
        valid_total = 0.0
        # train the model #
        model.train()
        for batch_idx, (data, target) in enumerate(trainloader):
            # move to GPU
            if use_cuda:
                data, target = data.cuda(), target.cuda()
            # initialize weights to zero
            optimizer.zero_grad()
            output = model(data)
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()
            train_loss = train_loss + ((1 / (batch_idx + 1)) * (loss.data - train_loss))

            # convert output probabilities to predicted class
            pred = output.data.max(1, keepdim=True)[1]
            # compare predictions to true label
            train_correct += np.sum(np.squeeze(pred.eq(target.data.view_as(pred))).cpu().nump
            train_total += data.size(0)
            train_acc = 100. * train_correct / train_total

        # validate the model
        model.eval()
        for batch_idx, (data, target) in enumerate(validloader):
            # move to GPU
            if use_cuda:
                data, target = data.cuda(), target.cuda()
            output = model(data)
            loss = criterion(output, target)
            valid_loss = valid_loss + ((1 / (batch_idx + 1)) * (loss.data - valid_loss))

            pred = output.data.max(1, keepdim=True)[1]
            # compare predictions to true label
            valid_correct += np.sum(np.squeeze(pred.eq(target.data.view_as(pred))).cpu().nump
            valid_total += data.size(0)
            valid_acc = 100. * valid_correct / valid_total
```

```
        # print training/validation statistics
        print('Epoch: {} \n\tTrain Loss: {:.6f} \tTrain Acc: {:.6f} \n\tValid Loss: {:.6f} \t
            epoch,train_loss,train_acc,valid_loss,valid_acc ))

        ## save the student model if validation loss has decreased
        if is_not_teacher:
          if valid_loss < valid_loss_min:
              torch.save(model.state_dict(), save_path)
              print('\tValidation loss decreased ({:.6f} --> {:.6f}).  Saving model ...'.form
              valid_loss_min,
              valid_loss))
              valid_loss_min = valid_loss

    return model


# instantiate model and move it to GPU if available
model = SimpleCNN()
model.to(device)

#define hyperparameters
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters() , lr=0.001)
epochs = 50


# Training teachers
teacher_models = []
i = 1
for trainloader, validloader in zip(trainloaders, validloaders):
  print(" Training Teacher {}".format(i))
  teacher_model = train(epochs, trainloader, validloader, model, optimizer, criterion, True)
  teacher_models.append(teacher_model)
  i+=1
  print("="*40)
```

```
    Epoch: 32
            Train Loss: 0.587310    Train Acc: 75.000000
            Valid Loss: 0.260758    Valid Acc: 87.500000
    Epoch: 33
            Train Loss: 0.479838    Train Acc: 75.000000
            Valid Loss: 1.103393    Valid Acc: 56.250000
    Epoch: 34
            Train Loss: 0.437322    Train Acc: 72.058824
            Valid Loss: 0.972711    Valid Acc: 56.250000
    Epoch: 35
            Train Loss: 0.545287    Train Acc: 75.000000
            Valid Loss: 0.940666    Valid Acc: 68.750000
    Epoch: 36
            Train Loss: 0.338666    Train Acc: 76.470588
            Valid Loss: 0.782138    Valid Acc: 75.000000
    Epoch: 37
            Train Loss: 0.312768    Train Acc: 85.294118
```

```
        Valid Loss: 0.590699    Valid Acc: 68.750000
Epoch: 38
        Train Loss: 0.486671    Train Acc: 72.058824
        Valid Loss: 0.573031    Valid Acc: 68.750000
Epoch: 39
        Train Loss: 0.265417    Train Acc: 85.294118
        Valid Loss: 0.903816    Valid Acc: 62.500000
Epoch: 40
        Train Loss: 0.572715    Train Acc: 72.058824
        Valid Loss: 0.923169    Valid Acc: 68.750000
Epoch: 41
        Train Loss: 0.471427    Train Acc: 79.411765
        Valid Loss: 1.065939    Valid Acc: 68.750000
Epoch: 42
        Train Loss: 0.642316    Train Acc: 83.823529
        Valid Loss: 0.564738    Valid Acc: 81.250000
Epoch: 43
        Train Loss: 0.467015    Train Acc: 82.352941
        Valid Loss: 0.629469    Valid Acc: 75.000000
Epoch: 44
        Train Loss: 0.353315    Train Acc: 80.882353
        Valid Loss: 0.398938    Valid Acc: 75.000000
Epoch: 45
        Train Loss: 0.475742    Train Acc: 77.941176
        Valid Loss: 0.455273    Valid Acc: 75.000000
Epoch: 46
        Train Loss: 0.355838    Train Acc: 79.411765
        Valid Loss: 0.901842    Valid Acc: 62.500000
Epoch: 47
        Train Loss: 0.367591    Train Acc: 80.882353
        Valid Loss: 0.988729    Valid Acc: 75.000000
Epoch: 48
        Train Loss: 0.395747    Train Acc: 77.941176
        Valid Loss: 0.608627    Valid Acc: 62.500000
Epoch: 49
        Train Loss: 0.492842    Train Acc: 85.294118
        Valid Loss: 0.299718    Valid Acc: 81.250000
Epoch: 50
        Train Loss: 0.507246    Train Acc: 82.352941
        Valid Loss: 0.510288    Valid Acc: 75.000000
========================================
```

```
# get private labels
def student_train_labels(teacher_models, dataloader):
  student_labels = []

  # get label from each teacher
  for model in teacher_models:
    student_label = []
    for images,_ in dataloader:
      with torch.no_grad():
        images = images.cuda()
        outputs = model(images)
        preds = torch.argmax(torch.exp(outputs), dim=1)
```

```
      student_label.append(preds.tolist())

    # add all teacher predictions to student_labels
    student_label = sum(student_label, [])
    student_labels.append(student_label)
  return student_labels

predicted_labels = student_train_labels(teacher_models, student_train_loader)
predicted_labels = np.array([np.array(p) for p in predicted_labels]).transpose(1, 0)

# We see here that we have 5 labels for each image in our dataset
print(predicted_labels.shape)
# See labels of 3rd Image Scan
print(predicted_labels[3])
```

```
(163, 5)
[1 1 1 1 1]
```

```
# Get private labels with the most votes count and add noise them
def add_noise(predicted_labels, epsilon=0.1):
  noisy_labels = []
  for preds in predicted_labels:

    # get labels with max votes
    label_counts = np.bincount(preds, minlength=2)

    # add laplacian noise to label
    epsilon = epsilon
    beta = 1/epsilon
    for i in range(len(label_counts)):
      label_counts[i] += np.random.laplace(0, beta, 1)

    # after adding noise we get labels with max counts
    new_label = np.argmax(label_counts)
    noisy_labels.append(new_label)

  #return noisy_labels
  return np.array(noisy_labels)



# # Open File
# # resultFyle = open("output.csv",'w')

labels_with_noise = add_noise(predicted_labels, epsilon=0.1)
print(labels_with_noise)
print(labels_with_noise.shape)
```

```
[0 1 0 1 1 1 0 0 1 1 1 0 1 0 1 0 0 0 1 0 1 1 1 0 1 1 0 0 1 1 1 1 1 1 0 1 0
 0 1 0 1 1 0 0 0 0 0 1 0 0 1 1 0 1 0 1 0 1 0 0 1 0 1 0 0 1 1 1 1 0 0 0 1 0
 0 1 0 0 0 1 1 0 1 0 1 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 1 0 1 1 0 0 1 1 1 1 0
```

```
 1 0 0 1 0 1 0 0 1 0 1 1 0 1 0 0 1 1 1 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 1 1 0
 1 1 0 1 0 1 1 0 0 0 1 1 0 0 0]
(163,)
```

```
#write to csv file
import csv
def write_csv(data):
    with open('labels.csv', 'a') as outfile:
        writer = csv.writer(outfile)
        writer.writerow(data)

write_csv(labels_with_noise)
```

```
# Performing PATE analysis
data_dep_eps, data_ind_eps = pate.perform_analysis(teacher_preds=predicted_labels.T, indices=
print('Data dependent epsilon:', data_dep_eps)
print('Data independent epsilon:', data_ind_eps)
```

```
Data dependent epsilon: 15.536462732485106
Data independent epsilon: 15.536462732485116
```

```
# We have to create a new training dataloader for the student with the newly created
# labels with noise. We have to replace the old labels with the new labels
def new_student_data_loader(dataloader, noisy_labels, batch_size=32):
  image_list = []
  for image,_ in dataloader:
    image_list.append(image)

  data = np.vstack(image_list)
  new_dataset = list(zip(data, noisy_labels))
  new_dataloader = DataLoader(new_dataset, batch_size, shuffle=False)

  return new_dataloader

labeled_student_trainloader = new_student_data_loader(student_train_loader, labels_with_noise
len(labeled_student_trainloader),len(student_valid_loader)
```

```
(6, 2)
```

```
student_model = train(epochs, labeled_student_trainloader, student_valid_loader, model, optim
```

```
        Valid Loss: 1.022140    Valid Acc: 57.500000
Epoch: 32
        Train Loss: 0.015681    Train Acc: 100.000000
        Valid Loss: 0.748233    Valid Acc: 60.000000
Epoch: 33
        Train Loss: 0.014858    Train Acc: 100.000000
        Valid Loss: 0.835062    Valid Acc: 65.000000
Epoch: 34
```

```
        Train Loss: 0.014168    Train Acc: 100.000000
        Valid Loss: 0.759211    Valid Acc: 62.500000
Epoch: 35
        Train Loss: 0.013462    Train Acc: 100.000000
        Valid Loss: 1.135166    Valid Acc: 47.500000
Epoch: 36
        Train Loss: 0.012831    Train Acc: 100.000000
        Valid Loss: 0.735590    Valid Acc: 60.000000
Epoch: 37
        Train Loss: 0.012260    Train Acc: 100.000000
        Valid Loss: 0.836232    Valid Acc: 55.000000
Epoch: 38
        Train Loss: 0.011713    Train Acc: 100.000000
        Valid Loss: 1.161770    Valid Acc: 50.000000
Epoch: 39
        Train Loss: 0.011226    Train Acc: 100.000000
        Valid Loss: 1.347101    Valid Acc: 47.500000
Epoch: 40
        Train Loss: 0.010727    Train Acc: 100.000000
        Valid Loss: 1.009625    Valid Acc: 42.500000
Epoch: 41
        Train Loss: 0.010303    Train Acc: 100.000000
        Valid Loss: 0.737544    Valid Acc: 52.500000
Epoch: 42
        Train Loss: 0.009874    Train Acc: 100.000000
        Valid Loss: 1.663713    Valid Acc: 50.000000
Epoch: 43
        Train Loss: 0.009492    Train Acc: 100.000000
        Valid Loss: 0.807792    Valid Acc: 55.000000
Epoch: 44
        Train Loss: 0.009114    Train Acc: 100.000000
        Valid Loss: 1.100018    Valid Acc: 52.500000
Epoch: 45
        Train Loss: 0.008773    Train Acc: 100.000000
        Valid Loss: 0.859432    Valid Acc: 57.500000
Epoch: 46
        Train Loss: 0.008460    Train Acc: 100.000000
        Valid Loss: 1.014665    Valid Acc: 50.000000
Epoch: 47
        Train Loss: 0.008143    Train Acc: 100.000000
        Valid Loss: 1.030685    Valid Acc: 52.500000
Epoch: 48
        Train Loss: 0.007865    Train Acc: 100.000000
        Valid Loss: 1.011487    Valid Acc: 57.500000
Epoch: 49
        Train Loss: 0.007588    Train Acc: 100.000000
        Valid Loss: 1.012489    Valid Acc: 60.000000
Epoch: 50
        Train Loss: 0.007338    Train Acc: 100.000000
        Valid Loss: 0.811221    Valid Acc: 65.000000
```

```
# Normal DL Training
normal_model = train(epochs, student_train_loader, student_valid_loader, model, optimizer, cr
```

```
Epoch: 32
```

```
Epoch: 32
        Train Loss: 0.435684    Train Acc: 79.141104
        Valid Loss: 0.603301    Valid Acc: 70.000000
Epoch: 33
        Train Loss: 0.432372    Train Acc: 74.846626
        Valid Loss: 0.450745    Valid Acc: 75.000000
Epoch: 34
        Train Loss: 0.475231    Train Acc: 76.073620
        Valid Loss: 0.636019    Valid Acc: 72.500000
Epoch: 35
        Train Loss: 0.378335    Train Acc: 84.662577
        Valid Loss: 0.493334    Valid Acc: 72.500000
Epoch: 36
        Train Loss: 0.422642    Train Acc: 79.754601
        Valid Loss: 0.449700    Valid Acc: 77.500000
Epoch: 37
        Train Loss: 0.404586    Train Acc: 79.141104
        Valid Loss: 0.722434    Valid Acc: 80.000000
Epoch: 38
        Train Loss: 0.360791    Train Acc: 82.208589
        Valid Loss: 0.495759    Valid Acc: 77.500000
Epoch: 39
        Train Loss: 0.393794    Train Acc: 81.595092
        Valid Loss: 0.517766    Valid Acc: 75.000000
Epoch: 40
        Train Loss: 0.403322    Train Acc: 80.368098
        Valid Loss: 0.554805    Valid Acc: 75.000000
Epoch: 41
        Train Loss: 0.376831    Train Acc: 77.300613
        Valid Loss: 0.715183    Valid Acc: 72.500000
Epoch: 42
        Train Loss: 0.370465    Train Acc: 82.208589
        Valid Loss: 0.446818    Valid Acc: 82.500000
Epoch: 43
        Train Loss: 0.470671    Train Acc: 76.687117
        Valid Loss: 0.534479    Valid Acc: 72.500000
Epoch: 44
        Train Loss: 0.331112    Train Acc: 83.435583
        Valid Loss: 0.593995    Valid Acc: 72.500000
Epoch: 45
        Train Loss: 0.306164    Train Acc: 84.662577
        Valid Loss: 0.624620    Valid Acc: 67.500000
Epoch: 46
        Train Loss: 0.300856    Train Acc: 85.889571
        Valid Loss: 0.390725    Valid Acc: 80.000000
        Validation loss decreased (0.395889 --> 0.390725).  Saving model ...
Epoch: 47
        Train Loss: 0.278320    Train Acc: 84.049080
        Valid Loss: 0.567250    Valid Acc: 75.000000
Epoch: 48
        Train Loss: 0.357786    Train Acc: 84.049080
        Valid Loss: 0.452256    Valid Acc: 82.500000
Epoch: 49
        Train Loss: 0.309117    Train Acc: 80.368098
        Valid Loss: 0.618512    Valid Acc: 75.000000
Epoch: 50
        Train Loss: 0.312820    Train Acc: 84.662577
```

```
        Valid Loss: 0.500149    Valid Acc: 85.000000
```

```
# Create a dataloader for the test Dataset
batch_size=16
print(len(validset))
dataloader = DataLoader(validset, batch_size=batchsize, shuffle=False)
```

```
118
```

```
# We set a seed for the dataset to prevent it from producing different values every time it i
seed = 3
random.seed(seed)
np.random.seed(seed)
torch.manual_seed(seed)
torch.backends.cudnn.deterministic = True
torch.backends.cudnn.benchmark = False
```

```
def test(dataloader, model, criterion, use_cuda):

    # monitor test loss and accuracy
    test_loss = 0.
    correct = 0.
    total = 0.

    model.eval()
    for batch_idx, (data, target) in enumerate(dataloader):
        # move to GPU
        if use_cuda:
            data, target = data.cuda(), target.cuda()

        # forward pass: compute predicted outputs by passing inputs to the model
        output = model(data)

        # calculate the loss
        loss = criterion(output, target)

        # update average test loss
        test_loss = test_loss + ((1 / (batch_idx + 1)) * (loss.data - test_loss))

        # convert output probabilities to predicted class
        pred = output.data.max(1, keepdim=True)[1]

        # compare predictions to true label
        correct += np.sum(np.squeeze(pred.eq(target.data.view_as(pred))).cpu().numpy())
        total += data.size(0)

    print('\tTest Loss: {:.6f}'.format(test_loss))
    print('\tTest Accuracy: %2d%% (%2d/%2d)' % (
```

```
        100. * correct / total, correct, total))

# call test function
print("Student Model")
test(dataloader, student_model, criterion, True)

print("\n======================\nNormal Model")
test(dataloader, normal_model, criterion, True)
```

```
Student Model
        Test Loss: 1.292508
        Test Accuracy: 59% (70/118)

=======================
Normal Model
        Test Loss: 1.116361
        Test Accuracy: 65% (77/118)
```