DEEP LEARNING for MSc COURSEWORK 2021

Identify different types of cell nuclei in a colon cancer sample

Student: Elissaios Baxevanis

StudentID: 2588922B

Student Mail: 2588922B@student.glasgow.ac.uk

In [1]:
```python
#SET UP SECTION
#IMPORTS - SEEDS - DEVICE CHECK
#SEEDS: USED THEM IN ORDER TO HAVE CONSISTENT RESULTS

#IMPORTS
import torchvision
import torch
import os
import pandas as pd
import numpy as np
from torch.utils.data import Dataset
from sklearn.preprocessing import OrdinalEncoder
from torchvision import datasets, transforms
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import random
from torchvision.io import read_image
%matplotlib inline
%config InlineBackend.figure_format = 'retina'
import matplotlib.pyplot as plt
from torch.utils.data import Dataset, DataLoader
from sklearn.preprocessing import OrdinalEncoder
from sklearn.metrics import multilabel_confusion_matrix
import collections
from sklearn.metrics import confusion_matrix,f1_score,
plot_confusion_matrix,classification_report
from matplotlib.ticker import MaxNLocator
#!pip3 install ax-platform
from ax import optimize
from ax.plot.contour import plot_contour
from ax.plot.trace import optimization_trace_single_method
from ax.service.managed_loop import optimize
from ax.utils.notebook.plotting import render, init_notebook_plotting
from ax.utils.tutorials.cnn_utils import train, evaluate


#SEEDS
torch.manual_seed(42)
torch.cuda.manual_seed(42)
np.random.seed(42)
random.seed(42)
torch.backends.cudnn.benchmark = False
torch.backends.cudnn.deterministic=True

#DEVICE CHECK
dtype = torch.float
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(device)
print(torch.cuda.get_device_name(0))
```

```
cuda
Tesla V100-SXM2-16GB
```

DATA AUGMENTATION THE FOLLOWIN FUNCTION CREATED IN ORDER TO HANDLE THE CLASS IMBALANCE IN THE GIVEN TRAINING DATA SET, WHERE IMMUNE AND CONNECTIVE SAMPLES WERE DOMINANT IN COMPARISSON TO CANCER AND NORMAL ONES, AS FOLLOWS:

- CANCER: 546
- CONNECTIVE: 726
- IMMUNE: 729
- NORMAL: 189

THE TECHNIQUE I USED WAS DATA AUGMENTATION BY CREATING NEW SAMPLES BASED ON THE TWO UNDERSAMPLED CLASSES(CANCER AND NORMAL). IN ORDER TO PRODUCE THOSE IMAGES I USED 5 DIFFERENT TRANSFORMATIONS (3 ROTATIONS AND 2 FLIPS) IN ORDER TO MAINTAIN THE COLOURS AND THE SIZES. MORE OVER I RANDOMLY PICKED THE CORRESPONDING OF IMAGES TO BE TRANSFORMED PER CLASS. AFTER THE AUGMENTATION THE TRAIN SET HAD A MORE OR LESS EQUAL DISTRIBUTION OF SAMPLES , AS FOLLOWS:

- CANCER: 726
- CONNECTIVE: 726
- IMMUNE: 729
- NORMAL: 724

THE FINAL TRAINING SET SIZE BECAME 2905 IMG-LABEL PAIRS THAT ALLOWED ME TO TRAIN AND EVALUATE MY MODEL PROPERLY

THE FOLLOWING PROCEDURE WAS DONE LOCALLY IN MY COMPUTER AND UPLOADED TO DRIVE, THUS THE PATHS ARE THE ONES IN MY PC AND I HAVE SET FLAG OUT TO AVOID ERRORS AS I AM WORKING ON GOOLGE COLAB PRO.

In [2]:
```python
#flag commented out, as it was implemented once
#flag = 1
def data_augment(flag=0):
  #idetify cancer and normal id's , extract their index
  #add them in lists in order to sample from them
  if flag==1:
    labels = pd.read_csv('train.csv', sep=',')
    print(labels.groupby('Type').count())
    c=[] #c is the list of cancer imgs
    n=[] #n is the list of normal imgs
    for i in range(len(labels)):
        if labels['Type'][i] == 'Cancer' :
            c.append(i)
        if labels['Type'][i] == 'Normal':
            n.append(i)
    print(f"Cancer:{len(c)}")
    print(f"Normal:{len(n)}")

    from PIL import Image
    from PIL import ImageFilter
    #balance aprox on 725 images per class
    #to balance we need
    #725-546=180 cancer , 5 transforamtions to new images-> 180/5= 36
    for_csv = []
    counter = len(labels) #counter starts at 2190 (the last index of imgs)

    #random samples from the list of indeces
    samples_c = random.choices(c,k=36)

    for i in samples_c:
        image = Image.open(f'./train_transpose_only/{i}.png')

        #first image: rotate 90 degrees, append the img-label pair in csv file
        counter+=1
        img1 = image.transpose(Image.ROTATE_90)
        img1.save('./train_transpose_only/'+str(counter)+'.png','png')
        labels = labels.append({'Id':counter,'Type':'Cancer'},ignore_index=True)

        #second image: rotate 180 degrees, append the img-label pair in csv file
        counter+=1
        img2 = image.transpose(Image.ROTATE_180)
        img2.save('./train_transpose_only/'+str(counter)+'.png','png')
        labels =labels.append({'Id':counter,'Type':'Cancer'},ignore_index=True)

        #third image: rotate 270 degrees, append the img-label pair in csv file
        counter+=1
        img3 = image.transpose(Image.ROTATE_270)
        img3.save('./train_transpose_only/'+str(counter)+'.png','png')
        labels =labels.append({'Id':counter,'Type':'Cancer'},ignore_index=True)

        #fourth image: flip left-right , append the img-label pair in csv file
        counter+=1
        img4 = image.transpose(Image.FLIP_LEFT_RIGHT)
        img4.save('./train_transpose_only/'+str(counter)+'.png','png')
        labels =labels.append({'Id':counter,'Type':'Cancer'},ignore_index=True)

        #fifth image: flip up-down , append the img-label pair in csv file
        counter+=1
        img5 = image.transpose(Image.FLIP_TOP_BOTTOM)
        img5.save('./train_transpose_only/'+str(counter)+'.png','png')
        labels =labels.append({'Id':counter,'Type':'Cancer'},ignore_index=True)

    #balance aprox on 725 images per class
    #to balance we need
    #725-189=536 normal , 5 transforamtions to new images-> 536/5=107.2 = 107

    #same process like above
    samples_n = random.choices(n,k=107)
    for i in samples_n:
        image = Image.open(f'./train_transpose_only/{i}.png')


        counter+=1
        img1 = image.transpose(Image.ROTATE_90)
        img1.save('./train_transpose_only/'+str(counter)+'.png','png')
        labels =labels.append({'Id':counter,'Type':'Normal'},ignore_index=True)

        counter+=1
        img2 = image.transpose(Image.ROTATE_180)
        img2.save('./train_transpose_only/'+str(counter)+'.png','png')
        labels =labels.append({'Id':counter,'Type':'Normal'},ignore_index=True)

        counter+=1
        img3 = image.transpose(Image.ROTATE_270)
        img3.save('./train_transpose_only/'+str(counter)+'.png','png')
        labels =labels.append({'Id':counter,'Type':'Normal'},ignore_index=True)

        counter+=1
        img4 = image.transpose(Image.FLIP_LEFT_RIGHT)
        img4.save('./train_transpose_only/'+str(counter)+'.png','png')
        labels =labels.append({'Id':counter,'Type':'Normal'},ignore_index=True)

        counter+=1
        img5 = image.transpose(Image.FLIP_TOP_BOTTOM)
        img5.save('./train_transpose_only/'+str(counter)+'.png','png')
        labels =labels.append({'Id':counter,'Type':'Normal'},ignore_index=True)


    #create new csv file, from the pandas dataframe
    #with all the img-label pairs for the final training set
    #also a small sanity check about classes and successfull creation of new csv
    labels.to_csv('new_csv_transpose_only.csv',header=True,index=False)
    a = pd.read_csv('new_csv_transpose_only.csv', sep=',')
    a.groupby('Type').count()
```

CUSTOM DATA LOADER CLASS

AS IMAGES ARE NOT ASSOCIATED DIRECTLY WITH THEIR LABELS I CAN NOT USE A PRE-DEFINED LOADER, THUS I CREATE THIS CLASS IN ORDER TO ASSOCIATE IMAGE-LABEL PAIRS AND LOAD THE DATA IN ORDER TO PROCESS THEM. ALSO, IMAGES IN CSV FILE ARE MISSING THE .PNG EXTENSION THUS IT NEEDS TO BE ADDED, ALSO I MAKE USE OF AN ORDINAL ENCODER IN ORDER TO TRANSFORM CATEGORICAL LABELS TO NUMERICAL.

In [3]:
```python
#class to match images with corresponding label and load them

class MyImgDset(Dataset):
    def __init__(self, annotations_file, img_dir, transform=None):
        self.img_labels = pd.read_csv(annotations_file,sep=',') #read labels
        self.img_labels['Id_ext'] = self.img_labels['Id'].apply(lambda x: f"{x}.png") #create
new column with .png extension
        encoder = OrdinalEncoder() #ordinal encoder to transform categorical labels
        self.img_labels['Type_ord'] =
encoder.fit_transform(np.array(self.img_labels['Type']).reshape(-1,1)) #transformation of labels
        self.img_dir = img_dir #get image path
        self.transform = transform

    def __len__(self):
        return len(self.img_labels) #function to get the len of the loaded dataset

    def __getitem__(self, idx):
        img_path = os.path.join(self.img_dir, self.img_labels.iloc[idx, 2]) #get next pair
image-label amd join them, notice i use the new created column with .png extension
        image = read_image(img_path) #read next image
        label = torch.tensor(int(self.img_labels.iloc[idx, 3])) #transform labels to tensors,
notice its the column with numerical values
        if self.transform:
            image = self.transform(image) #if there is transform(s) to be applied
        return image,label #return image-label pair
```

LEARNING CURVES PLOT FUNCTION

IN ORDER TO MONITOR THE PROCESS AND GET INSIGHT ON HOW MY MODEL IS DOING ON THE GIVEN DATA SET I AM CREATING THE FOLLOWING FUNCTION THAT PLOTS TRAIN AND VALIDATION ACCURACIES AND LOSSES OVER THE NUMBER OF EPOCHS

In [4]:
```python
def plot_func(total_valid_acc, total_valid_loss, total_train_loss, total_train_acc, n_epoch):
  fig, ax = plt.subplots()
  ax.plot(total_valid_acc, label="Valid Acc")
  ax.plot(total_train_acc, label="Train Acc")
  ax.legend(loc=4)
  ax.set_title("Accuracy over epochs")
  ax.set_xlabel("Epochs")
  ax.set_xlim(1,n_epochs+1)
  ax.yaxis.set_major_locator(MaxNLocator(nbins=10,integer=True))
  ax.set_ylabel("Accuracy")
  ax.set_ylim(0.0, 100.0)
  ax.xaxis.set_major_locator(MaxNLocator(nbins=10,integer=True))
  ax.grid()

  fig1, ax1 = plt.subplots()
  ax1.plot(total_valid_loss, label="Valid Loss")
  ax1.plot(total_train_loss, label="Train Loss")
  ax1.legend(loc=0)
  ax1.set_title("Loss over epochs")
  ax1.set_xlabel("Epochs")
  ax1.xaxis.set_major_locator(MaxNLocator(nbins=10,integer=True))
  ax1.set_xlim(1,n_epochs+1)
  ax1.set_ylabel("Loss")
  ax1.grid()
  plt.show()
```

In [5]:
```python
# after uploading data on google drive, load them using the loader i created earlier
from google.colab import drive
drive.mount('/content/drive/')
csv_file =  r"/content/drive/MyDrive/kaggle/new_csv_transpose_only.csv"
img_dir = r"/content/drive/MyDrive/kaggle/train_transpose_only/"

#side note, i tried normalization on the whole data set with but
#after all , my models performed better with batch-normalization
#and not with whole set-normalization or combination of them
dataset = MyImgDset(csv_file, img_dir) #load the data
```

```
Drive already mounted at /content/drive/; to attempt to forcibly remount, call drive.mount("/content/drive/", force_remount=True).
```

In [6]:
```python
batch_size = 16
#train-test split of the data -> ~80% train / 20% remain for test
#shuffle true in order to randomize selection of images, as the order on training set
#contains large blobs of each class until a different one appears, thus randomization is needed

train_set, test_set = torch.utils.data.random_split(dataset,(2300,605))
train_loader = DataLoader(dataset=train_set, batch_size=batch_size, shuffle=True,num_workers=0)
test_loader = DataLoader(dataset=test_set,batch_size=batch_size, shuffle=True,num_workers=0)
```

ARCHITECTURE OF MY CNN

AFTER IMPLEMENTING AND TESTING MANY COMBINATIONS OF ARCHITECTURES (FROM VERY SIMPLE ONES LIKE ONE CONVOLUTIONAL LAYER AND COUPLE OF FULLY CONNECTED , UNTIL 8 CONVOLUTIONAL AND 4-6 FULLY CONNECTED) I REALIZED THAT THE MORE COMPLICATED THE MODEL, THE POORER RESULTS I RECEIVED AND IN COMBINATION WITH THE CLASSIFICAITON TARGET (4 LABELS) AND THE SIZE OF TRAINING SET, I CONCLUDED TO THE FOLLOWING MODEL. AFTER EACH CONVOLUTIONAL LAYER I APPLY BATCH NORMALIZATION, FOLLOWED BY RELU AS ACTIVATION FUNCTION AND A POOLING LAYER. AFTER THREE SUCH LAYERS I END UP TO FLAT AND USE SINGLE FULLY CONNECTED LAYER TO PRODUCE THE DESIRED OUTPUT.

In [7]:
```python
model = nn.Sequential(collections.OrderedDict([
    ('conv1', nn.Conv2d(in_channels=3,out_channels=16,kernel_size = 3,padding=1)),
    ('bn1', nn.BatchNorm2d(num_features=16)),
    ('relu1', nn.ReLU()),
    ('pool1', nn.MaxPool2d(kernel_size = 2)),
    ('conv2', nn.Conv2d(in_channels=16,out_channels=32,kernel_size =3,padding=1)),
    ('bn2', nn.BatchNorm2d(num_features=32)),
    ('relu2', nn.ReLU()),
    ('pool2', nn.MaxPool2d(kernel_size = 2)),
    ('conv3', nn.Conv2d(in_channels=32,out_channels=64,kernel_size = 3,padding=1)),
    ('bn3', nn.BatchNorm2d(num_features=64)),
    ('relu3', nn.ReLU()),
    ('pool3', nn.MaxPool2d(kernel_size =2 )),
    ('flatten', nn.Flatten()),
    ('fc1', nn.Linear(in_features=4096,out_features=4)),
    ]))
```

TRAINING AND TESTING FUNCTIONS

In [8]:
```python
#function to train the model
def training_loop(n_epochs, optimizer, model, device, loss_fn, train_loader, flag=1):
    model = model.to(device)
    total_valid_acc, total_valid_loss = [], [] #lists that losses and accuracies are appended
at, to produce plots mentioned earlier
    total_train_loss, total_train_acc = [], []

    for epoch in range(1, n_epochs + 1):
        loss_train = 0.0
        correct = 0
        all_preds = torch.tensor([]).to(device) #tensors used to build confusion matrix
        all_true = torch.tensor([]).to(device)
        for imgs, labels in train_loader:
            labels = labels.to(device)
            imgs = imgs.to(device).float()

            optimizer.zero_grad()
            outputs = model(imgs)
            loss = loss_fn(outputs, labels)
            loss.backward()
            loss_train += loss.item()
            optimizer.step()
            pred = outputs.argmax(dim=1, keepdim=True)
            correct += pred.eq(labels.view_as(pred)).sum().item()

            all_preds = torch.cat((all_preds,pred))
            all_true = torch.cat((all_true, labels))

        train_acc = 100. * correct / len(train_loader.dataset)
        valid_acc , valid_loss = test_loop(model,device,test_loader) #evaluate the model per
epoch and get loss and accuracy
        total_valid_acc.append(valid_acc)
        total_valid_loss.append(valid_loss)
        total_train_loss.append(loss_train)
        total_train_acc.append(train_acc)

        #build confusion matrix and generate report for each epoch
        conf_matr = confusion_matrix(all_true.cpu().numpy(),all_preds.cpu().numpy())
        report = classification_report(all_true.cpu().numpy(),all_preds.cpu().numpy())
        #flag used to avoid over-printing in the AX hyper-param optimization process
        if flag == 1:
          print(f'Epoch: {epoch}')
          print(f'\nTrain Confusion Matrix:\n {conf_matr}\n')
          print(report)
          print("-----------------------------------------------------------------")
        #return total accuracies and losses for train and validation, in order to monitor the
process via graphs
    return total_valid_acc, total_valid_loss, total_train_loss, total_train_acc

#function to evaluate the model
def test_loop(model, device, test_loader, flag=0):
    model = model.to(device)
    model.eval()
    test_loss = 0

    correct = 0
    total = 0
    all_preds = torch.tensor([]).to(device) #tensors used to build confusion matrix
    all_true = torch.tensor([]).to(device)
    loss_fn = nn.CrossEntropyLoss()
    with torch.no_grad():
        for imgs, labels in test_loader:
            labels = labels.to(device)
            imgs  = imgs.to(device).float()

            output = model(imgs)
            test_loss += loss_fn(output, labels)
            pred = output.argmax(dim=1, keepdim=True)
            correct += pred.eq(labels.view_as(pred)).sum().item()
            total += labels.size(0)

            all_preds = torch.cat((all_preds,pred))
            all_true = torch.cat((all_true, labels))

    acc = 100. *correct/total

    #produce these reports only for the final single test
    #applied, after the train of the model has finished
    #flagged out to reduce information printed during training
    if flag == 1:
      #build confusion matrix
      conf_matr = confusion_matrix(all_true.cpu().numpy(),all_preds.cpu().numpy())
      report = classification_report(all_true.cpu().numpy(),all_preds.cpu().numpy())
      print(f'Test Confusion Matrix:\n {conf_matr}\n')
      print(report)
      print('\nTest set: Average loss: {:.4f}, Accuracy: {}/{} ({:.0f}%)\n'.format(
          test_loss, correct, len(test_loader.dataset),acc))


    return acc, test_loss #returns accuracy and test_loss
```

HYPER-PARAMETERS, OPTIMIZER & LOSS FUNCTION

AFTER AX OPTIMIZATION FINE-TUNING OF HYPERPARAMETERS WAS DONE BY HAND (AX PROCESS SHOWN AT THE LAST CELL OF THE NOTEBOOK)

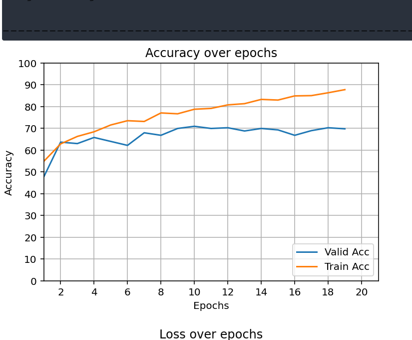
LOSS FUNCTION: AS THE TASK WAS MULTILABEL CLASSIFICATION I DECIDED TO USE CROSSENTROPYLOSS

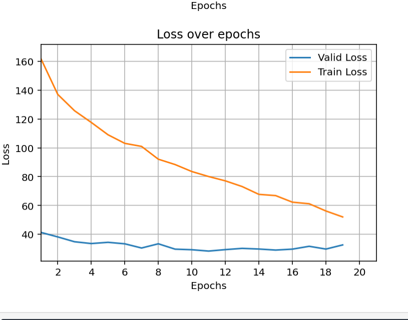
OPTIMIZER: SGD CHOOSEN AS ITS RESULTS WERE THE MOST CONSISTENT

In [9]:
```python
learning_rate = 0.001
n_epochs = 20
momentum = 0
weight_decay = 0.01
loss_fn=nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr = learning_rate,weight_decay = weight_decay,
momentum=momentum)#, nesterov=True)
```

In [10]:
```python
#initialize training and also plot the results
total_valid_acc, total_valid_loss, total_train_loss, total_train_acc = training_loop
(n_epochs=n_epochs,
                                                                                 optimizer
= optimizer,
                                                                                 model =
model,
                                                                                 device =
device,
                                                                                 loss_fn =
loss_fn,

train_loader = train_loader)
plot_func(total_valid_acc, total_valid_loss, total_train_loss, total_train_acc, n_epochs)
```

```
Epoch: 1

Train Confusion Matrix:
 [[336  95  54  89]
 [ 85 279 124  93]
 [ 51 118 299 106]
 [ 79 100 134 258]]

              precision    recall  f1-score   support

         0.0       0.61      0.59      0.60       574
         1.0       0.47      0.48      0.48       581
         2.0       0.49      0.52      0.50       574
         3.0       0.47      0.45      0.46       571

    accuracy                           0.51      2300
   macro avg       0.51      0.51      0.51      2300
weighted avg       0.51      0.51      0.51      2300

----------------------------------------------------------------
Epoch: 2

Train Confusion Matrix:
 [[365  82  35  92]
 [ 78 313 101  89]
 [ 32 115 339  88]
 [113 108 105 245]]

              precision    recall  f1-score   support

         0.0       0.62      0.64      0.63       574
         1.0       0.51      0.54      0.52       581
         2.0       0.58      0.59      0.59       574
         3.0       0.48      0.43      0.45       571

    accuracy                           0.55      2300
   macro avg       0.55      0.55      0.55      2300
weighted avg       0.55      0.55      0.55      2300

----------------------------------------------------------------
Epoch: 3

Train Confusion Matrix:
 [[386  65  43  80]
 [ 66 358  80  77]
 [ 36  97 396  45]
 [ 91  89  86 305]]

              precision    recall  f1-score   support

         0.0       0.67      0.67      0.67       574
         1.0       0.59      0.62      0.60       581
         2.0       0.65      0.69      0.67       574
         3.0       0.60      0.53      0.57       571

    accuracy                           0.63      2300
   macro avg       0.63      0.63      0.63      2300
weighted avg       0.63      0.63      0.63      2300

----------------------------------------------------------------
Epoch: 4

Train Confusion Matrix:
 [[386  67  39  82]
 [ 66 378  72  65]
 [ 23  70 433  48]
 [ 72  90  82 327]]

              precision    recall  f1-score   support

         0.0       0.71      0.67      0.69       574
         1.0       0.62      0.65      0.64       581
         2.0       0.69      0.75      0.72       574
         3.0       0.63      0.57      0.60       571

    accuracy                           0.66      2300
   macro avg       0.66      0.66      0.66      2300
weighted avg       0.66      0.66      0.66      2300

----------------------------------------------------------------
Epoch: 5

Train Confusion Matrix:
 [[403  63  26  82]
 [ 57 400  67  57]
 [ 23  77 437  37]
 [ 68  88  82 333]]

              precision    recall  f1-score   support

         0.0       0.73      0.70      0.72       574
         1.0       0.64      0.69      0.66       581
         2.0       0.71      0.76      0.74       574
         3.0       0.65      0.58      0.62       571

    accuracy                           0.68      2300
   macro avg       0.68      0.68      0.68      2300
weighted avg       0.68      0.68      0.68      2300

----------------------------------------------------------------
Epoch: 6

Train Confusion Matrix:
 [[411  57  30  76]
 [ 46 413  70  52]
 [ 21  71 450  32]
 [ 52  87  61 371]]

              precision    recall  f1-score   support

         0.0       0.78      0.72      0.74       574
         1.0       0.66      0.71      0.68       581
         2.0       0.74      0.78      0.76       574
         3.0       0.70      0.65      0.67       571

    accuracy                           0.72      2300
   macro avg       0.72      0.72      0.72      2300
weighted avg       0.72      0.72      0.72      2300

----------------------------------------------------------------
Epoch: 7

Train Confusion Matrix:
 [[424  53  28  69]
 [ 44 425  61  51]
 [ 15  59 463  37]
 [ 52  74  67 378]]

              precision    recall  f1-score   support

         0.0       0.79      0.74      0.76       574
         1.0       0.70      0.73      0.71       581
         2.0       0.75      0.81      0.78       574
         3.0       0.71      0.66      0.68       571

    accuracy                           0.73      2300
   macro avg       0.74      0.73      0.73      2300
weighted avg       0.74      0.73      0.73      2300

----------------------------------------------------------------
Epoch: 8

Train Confusion Matrix:
 [[419  70  27  58]
 [ 52 421  58  50]
 [ 18  63 461  32]
 [ 56  75  59 381]]

              precision    recall  f1-score   support

         0.0       0.77      0.73      0.75       574
         1.0       0.67      0.72      0.70       581
         2.0       0.76      0.80      0.78       574
         3.0       0.73      0.67      0.70       571

    accuracy                           0.73      2300
   macro avg       0.73      0.73      0.73      2300
weighted avg       0.73      0.73      0.73      2300

----------------------------------------------------------------
Epoch: 9

Train Confusion Matrix:
 [[437  49  27  61]
 [ 44 442  49  46]
 [ 17  48 483  26]
 [ 41  65  55 410]]

              precision    recall  f1-score   support

         0.0       0.81      0.76      0.79       574
         1.0       0.73      0.76      0.75       581
         2.0       0.79      0.84      0.81       574
         3.0       0.76      0.72      0.74       571

    accuracy                           0.77      2300
   macro avg       0.77      0.77      0.77      2300
weighted avg       0.77      0.77      0.77      2300

----------------------------------------------------------------
Epoch: 10

Train Confusion Matrix:
 [[431  54  25  64]
 [ 40 447  53  41]
 [ 15  56 472  31]
 [ 43  62  53 413]]

              precision    recall  f1-score   support

         0.0       0.81      0.75      0.78       574
         1.0       0.72      0.77      0.75       581
         2.0       0.78      0.82      0.80       574
         3.0       0.75      0.72      0.74       571

    accuracy                           0.77      2300
   macro avg       0.77      0.77      0.77      2300
weighted avg       0.77      0.77      0.77      2300

----------------------------------------------------------------
Epoch: 11

Train Confusion Matrix:
 [[435  57  22  60]
 [ 28 473  45  35]
 [ 17  45 486  26]
 [ 38  68  48 417]]

              precision    recall  f1-score   support

         0.0       0.84      0.76      0.80       574
         1.0       0.74      0.81      0.77       581
         2.0       0.81      0.85      0.83       574
         3.0       0.78      0.73      0.75       571

    accuracy                           0.79      2300
   macro avg       0.79      0.79      0.79      2300
weighted avg       0.79      0.79      0.79      2300

----------------------------------------------------------------
Epoch: 12

Train Confusion Matrix:
 [[443  52  21  58]
 [ 36 465  44  36]
 [ 15  52 490  17]
 [ 36  66  47 422]]

              precision    recall  f1-score   support

         0.0       0.84      0.77      0.80       574
         1.0       0.73      0.80      0.76       581
         2.0       0.81      0.85      0.83       574
         3.0       0.79      0.74      0.76       571

    accuracy                           0.79      2300
   macro avg       0.79      0.79      0.79      2300
weighted avg       0.79      0.79      0.79      2300

----------------------------------------------------------------
Epoch: 13

Train Confusion Matrix:
 [[451  47  23  53]
 [ 32 474  39  36]
 [ 15  48 485  26]
 [ 31  53  40 447]]

              precision    recall  f1-score   support

         0.0       0.85      0.79      0.82       574
         1.0       0.76      0.82      0.79       581
         2.0       0.83      0.84      0.84       574
         3.0       0.80      0.78      0.79       571

    accuracy                           0.81      2300
   macro avg       0.81      0.81      0.81      2300
weighted avg       0.81      0.81      0.81      2300

----------------------------------------------------------------
Epoch: 14

Train Confusion Matrix:
 [[451  47  20  56]
 [ 35 475  38  33]
 [ 12  41 500  21]
 [ 38  51  38 444]]

              precision    recall  f1-score   support

         0.0       0.84      0.79      0.81       574
         1.0       0.77      0.82      0.79       581
         2.0       0.84      0.87      0.85       574
         3.0       0.80      0.78      0.79       571

    accuracy                           0.81      2300
   macro avg       0.81      0.81      0.81      2300
weighted avg       0.81      0.81      0.81      2300

----------------------------------------------------------------
Epoch: 15

Train Confusion Matrix:
 [[464  49  13  48]
 [ 31 481  39  30]
 [ 11  41 502  20]
 [ 18  53  32 468]]

              precision    recall  f1-score   support

         0.0       0.89      0.81      0.85       574
         1.0       0.77      0.83      0.80       581
         2.0       0.86      0.87      0.87       574
         3.0       0.83      0.82      0.82       571

    accuracy                           0.83      2300
   macro avg       0.83      0.83      0.83      2300
weighted avg       0.83      0.83      0.83      2300

----------------------------------------------------------------
Epoch: 16

Train Confusion Matrix:
 [[460  47  18  49]
 [ 27 488  36  30]
 [ 10  35 512  17]
 [ 29  53  41 448]]

              precision    recall  f1-score   support

         0.0       0.87      0.80      0.84       574
         1.0       0.78      0.84      0.81       581
         2.0       0.84      0.89      0.87       574
         3.0       0.82      0.78      0.80       571

    accuracy                           0.83      2300
   macro avg       0.83      0.83      0.83      2300
weighted avg       0.83      0.83      0.83      2300

----------------------------------------------------------------
Epoch: 17

Train Confusion Matrix:
 [[478  42  16  38]
 [ 24 499  30  28]
 [  9  34 511  20]
 [ 25  49  33 464]]

              precision    recall  f1-score   support

         0.0       0.89      0.83      0.86       574
         1.0       0.80      0.86      0.83       581
         2.0       0.87      0.89      0.88       574
         3.0       0.84      0.81      0.83       571

    accuracy                           0.85      2300
   macro avg       0.85      0.85      0.85      2300
weighted avg       0.85      0.85      0.85      2300

----------------------------------------------------------------
Epoch: 18

Train Confusion Matrix:
 [[468  45  17  44]
 [ 21 508  28  24]
 [ 12  36 512  14]
 [ 34  41  29 467]]

              precision    recall  f1-score   support

         0.0       0.87      0.82      0.84       574
         1.0       0.81      0.87      0.84       581
         2.0       0.87      0.89      0.88       574
         3.0       0.85      0.82      0.83       571

    accuracy                           0.85      2300
   macro avg       0.85      0.85      0.85      2300
weighted avg       0.85      0.85      0.85      2300

----------------------------------------------------------------
Epoch: 19

Train Confusion Matrix:
 [[473  40  13  48]
 [ 23 517  25  16]
 [  9  34 517  14]
 [ 29  36  28 478]]

              precision    recall  f1-score   support

         0.0       0.89      0.82      0.85       574
         1.0       0.82      0.89      0.86       581
         2.0       0.89      0.90      0.89       574
         3.0       0.86      0.84      0.85       571

    accuracy                           0.86      2300
   macro avg       0.86      0.86      0.86      2300
weighted avg       0.86      0.86      0.86      2300

----------------------------------------------------------------
Epoch: 20

Train Confusion Matrix:
 [[485  41  12  36]
 [ 26 520  19  16]
 [ 10  27 528   9]
 [ 20  41  25 485]]

              precision    recall  f1-score   support

         0.0       0.90      0.84      0.87       574
         1.0       0.83      0.90      0.86       581
         2.0       0.90      0.92      0.91       574
         3.0       0.89      0.85      0.87       571

    accuracy                           0.88      2300
   macro avg       0.88      0.88      0.88      2300
weighted avg       0.88      0.88      0.88      2300

----------------------------------------------------------------
```





In [11]:

```python
#final test on the validation set
test_loop(model,device,test_loader, flag=1)
```

```
Test Confusion Matrix:
 [[ 95  24  13  20]
 [  6  97  37   5]
 [  1  15 139   0]
 [ 10  17  35  91]]

              precision    recall  f1-score   support

         0.0       0.85      0.62      0.72       152
         1.0       0.63      0.67      0.65       145
         2.0       0.62      0.90      0.73       155
         3.0       0.78      0.59      0.68       153

    accuracy                           0.70       605
   macro avg       0.72      0.70      0.70       605
weighted avg       0.72      0.70      0.70       605


Test set: Average loss: 32.6058, Accuracy: 422/605 (70%)
```

Out[11]:

```
(69.75206611570248, tensor(32.6058, device='cuda:0'))
```

PUBLIC TEST SET APPLICATION AND CSV GENERATION

THE FOLLOWING FUNCTION IS USED IN ORDER TO FEED THE PUBLIC TEST SET INTO THE MODEL AND PRODUCE ITS PREDICTIONS ON THEM AND ALSO TO GENERATE THE CSV FILE WITH THE PREDICTIONS FOR KAGGLE

In [12]:

```python
examp = r"/content/drive/MyDrive/kaggle/example.csv"
test_img_pth = r"/content/drive/MyDrive/kaggle/test/"
def test_for_kaggle(model,device,loader):
    for_kaggle = pd.read_csv(examp,sep=',')
    idx = 0
    model.eval()
    model = model.to(device)
    with torch.no_grad():
        for image, target in loader:
            image = image.to(device).float()
            output = model(image)
            prediction = output.argmax(dim=1, keepdim=True)
            #CHECK THE PREDICTION, MODIFY THE LABEL FOR EACH IMAGE BASED ON INDEX
            if prediction == 0 :
                for_kaggle['Type'].iloc[idx] = 'Cancer'
            if prediction == 1 :
                for_kaggle['Type'].iloc[idx] = 'Connective'
            if prediction == 2 :
                for_kaggle['Type'].iloc[idx] = 'Immune'
            if prediction == 3 :
                for_kaggle['Type'].iloc[idx] = 'Normal'
            idx+=1

    for_kaggle.to_csv(r"/content/drive/MyDrive/kaggle/elissaios.csv",header=True,index=False)
```

In [13]:

```python
#LOAD THE DATA AND FEED THEM TO THE MODEL
kaggle_testset = MyImgDset(examp, test_img_pth)
kaggle_dataloader = DataLoader(dataset=kaggle_testset,shuffle=False,num_workers=0)
test_for_kaggle(model,device,kaggle_dataloader)
```

```
/usr/local/lib/python3.7/dist-packages/pandas/core/indexing.py:670: SettingWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy
```

In [14]:

```python
#SANITY CHECK THAT PREDICTIONS ARE MADE AND ALSO GET AN INDICATION OF THE DISTRIBUTION OF PREDICTIONS
a = pd.read_csv(r"/content/drive/MyDrive/kaggle/elissaios.csv", sep=',')
print("Kaggle:\n")
print(a.groupby('Type').count())

#sidenote,the followin results are slightly different than the csv file submitted in
#kaggle due to the fact that i work on google colab i get different results every
#time i close the notebook or pc and use the same notebook (maybe because of different gpu allocatio)
#but the model and parameters are the same that generated the choosen csv file
```

```
Kaggle:

            Id
Type
Cancer     119
Connective 141
Immune      94
Normal      46
```

HYPERPARAMETER OPTIMIZATION WITH AX

USED AX TO INITIALLY GET AN INSIGHT ABOUT THE PARAMETERS

In [17]:

```python
def train_evaluate(parameterization):
    net = training_loop(model=model, train_loader=train_loader, n_epochs=1, loss_fn=loss_fn, optimizer=optimizer, device=device,flag=0)
    return evaluate(
        net=model,
        data_loader=test_loader,
        dtype=dtype,
        device=device,
    )


best_parameters, best_values, experiment, model = optimize(
        parameters=[
                    {"name": "lr", "type": "range", "bounds": [1e-6, 0.4], "log_scale": True},
                    {"name": "weight_decay", "type": "range", "bounds": [0.0, 1.0]},
                    {"name": "n_epochs", "type": "range", "bounds": [10, 100]}],
                    evaluation_function = train_evaluate,
                    objective_name = 'accuracy',
)
print("\n----------------------------------------\n")
print("AX HYPER PARAMETER SUGGESTION\n")
print(best_parameters)
```

```
[INFO 05-23 10:54:13] ax.service.utils.instantiation: Inferred value type of ParameterType.FLOAT for parameter lr. If that is not the expected value type, you can explicity specify 'value_type' ('int', 'float', 'bool' or 'str') in parameter dict.
[INFO 05-23 10:54:13] ax.service.utils.instantiation: Inferred value type of ParameterType.FLOAT for parameter weight_decay. If that is not the expected value type, you can explicity specify 'value_type' ('int', 'float', 'bool' or 'str') in parameter dict.
[INFO 05-23 10:54:13] ax.service.utils.instantiation: Inferred value type of ParameterType.INT for parameter n_epochs. If that is not the expected value type, you can explicity specify 'value_type' ('int', 'float', 'bool' or 'str') in parameter dict.
[INFO 05-23 10:54:13] ax.modelbridge.dispatch_utils: Using Bayesian Optimization generation strategy: GenerationStrategy(name='Sobol+GPEI', steps=[Sobol for 5 trials, GPEI for subsequent trials]). Iterations after 5 will take longer to generate due to  model-fitting.
[INFO 05-23 10:54:13] ax.service.managed_loop: Started full optimization with 20 steps.
[INFO 05-23 10:54:13] ax.service.managed_loop: Running optimization trial 1...
[INFO 05-23 10:54:18] ax.service.managed_loop: Running optimization trial 2...
[INFO 05-23 10:54:22] ax.service.managed_loop: Running optimization trial 3...
[INFO 05-23 10:54:27] ax.service.managed_loop: Running optimization trial 4...
[INFO 05-23 10:54:31] ax.service.managed_loop: Running optimization trial 5...
[INFO 05-23 10:54:35] ax.service.managed_loop: Running optimization trial 6...
[INFO 05-23 10:54:40] ax.service.managed_loop: Running optimization trial 7...
[INFO 05-23 10:54:45] ax.service.managed_loop: Running optimization trial 8...
[INFO 05-23 10:54:46] ax.modelbridge.modelbridge_utils: Untransformed parameter 0.40000000000000013 greater than upper bound 0.4, clamping
[INFO 05-23 10:54:50] ax.service.managed_loop: Running optimization trial 9...
[INFO 05-23 10:54:55] ax.service.managed_loop: Running optimization trial 10...
[INFO 05-23 10:55:00] ax.service.managed_loop: Running optimization trial 11...
[INFO 05-23 10:55:06] ax.service.managed_loop: Running optimization trial 12...
[INFO 05-23 10:55:12] ax.service.managed_loop: Running optimization trial 13...
[INFO 05-23 10:55:17] ax.service.managed_loop: Running optimization trial 14...
[INFO 05-23 10:55:23] ax.service.managed_loop: Running optimization trial 15...
[INFO 05-23 10:55:29] ax.service.managed_loop: Running optimization trial 16...
[INFO 05-23 10:55:35] ax.service.managed_loop: Running optimization trial 17...
[INFO 05-23 10:55:41] ax.service.managed_loop: Running optimization trial 18...
[INFO 05-23 10:55:47] ax.service.managed_loop: Running optimization trial 19...
[INFO 05-23 10:55:49] ax.modelbridge.modelbridge_utils: Untransformed parameter 0.40000000000000013 greater than upper bound 0.4, clamping
[INFO 05-23 10:55:53] ax.service.managed_loop: Running optimization trial 20...
[INFO 05-23 10:55:55] ax.modelbridge.modelbridge_utils: Untransformed parameter 0.40000000000000013 greater than upper bound 0.4, clamping
----------------------------------------

AX HYPER PARAMETER SUGGESTION

{'lr': 0.0009571964270774816, 'weight_decay': 4.741933135132015e-18, 'n_epochs': 65}
```

In [ ]: