CMPUT 466/566 - CNN [25 marks]

RESOURCES

This assignment requires some basic knowledge of Pytorch which can be found in the following links:

1. Tensors
2. Build the Neural Network
3. Optimizing Model Parameters
4. torch.nn
5. ResNet

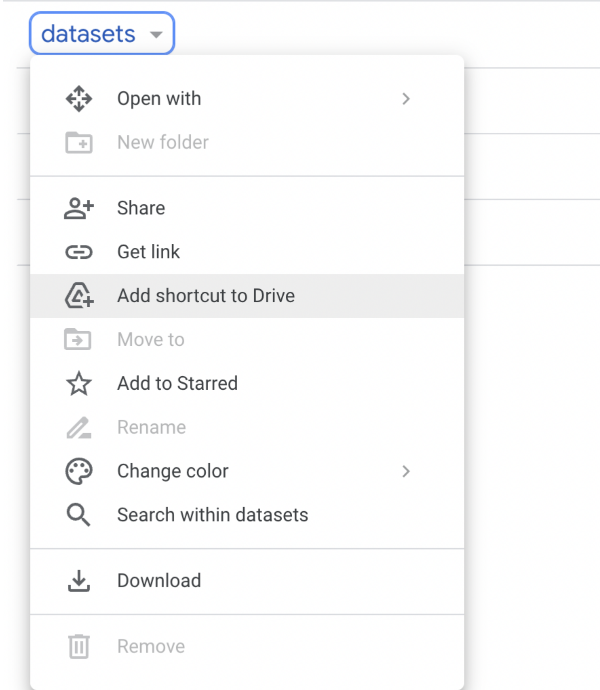
DATASET

The dataset is from Kaggle and has been modified for some questions. The shared ***'balanced_data'*** folder contains the balanced dataset and ***'imbalanced_data'*** contains its modified version for Part IV with class imbalance.

The dataset can be found here

Load the data on Google colab from Google drive:

1. Click on the Google Drive link of the **datasets** folder (make sure you login with your ualberta.ca email address)
2. Click on the drop down next to the name of the folder and select **Add Shortcut to Drive**

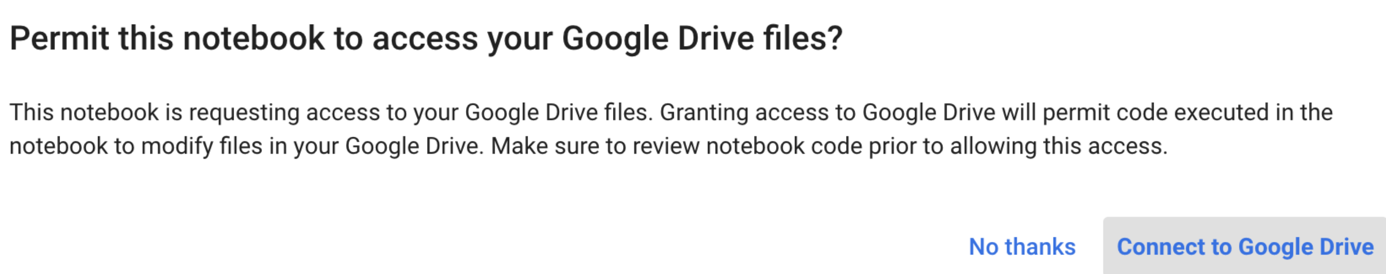


4. Come back to the **copy of this colab notebook** and mount the drive by running the cell below

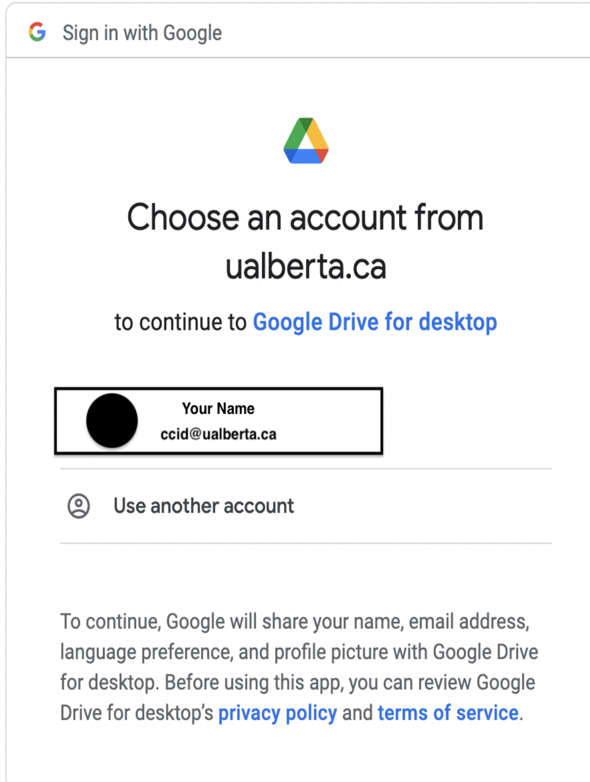
```
from google.colab import drive
#make sure you give the necessary authorization for colab to access your Google Drive
drive.mount('/content/drive')
```

```
    Mounted at /content/drive
```

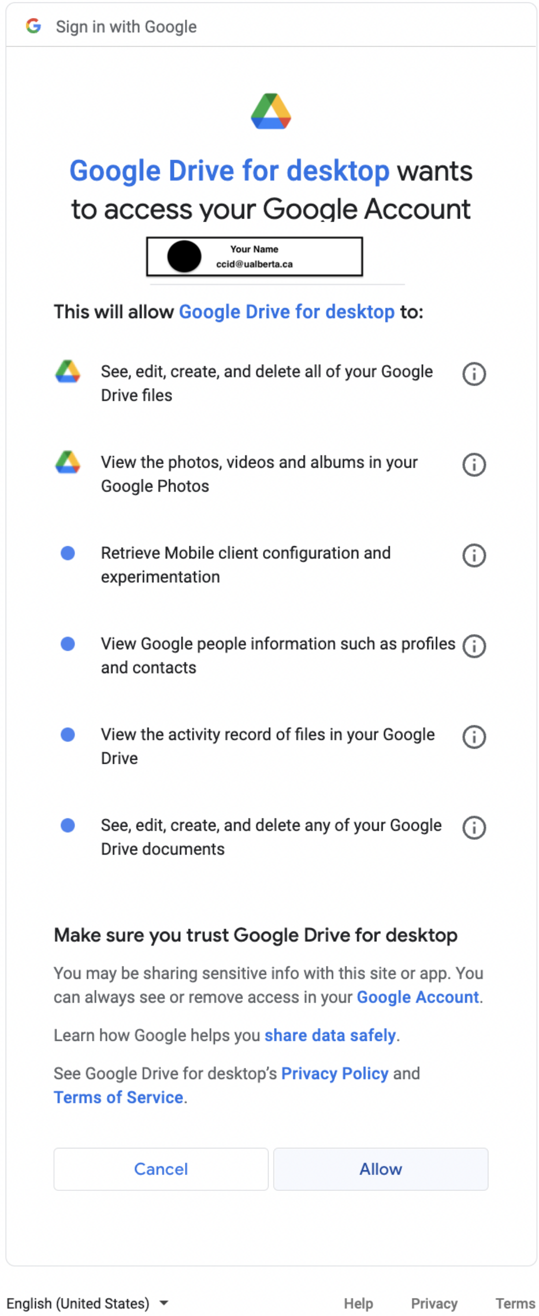
5. Click Connect to **Google Drive**



6. **Choose your ualberta.ca account**



7. **Grant permission**



8. If you want to access a folder called **'datasets'**, you can do this with:

```
dataset_dir = '/content/drive/MyDrive/datasets'
```

```
'''
Follow the above steps and include the paths for training and test datasets
'''
main_path = '/content/drive/MyDrive/datasets/balanced_data/train' #ENTER PATH HERE
test_path = '/content/drive/MyDrive/datasets/balanced_data/test' #ENTER PATH HERE
```

Part I: Activation functions for CNN [6 marks]

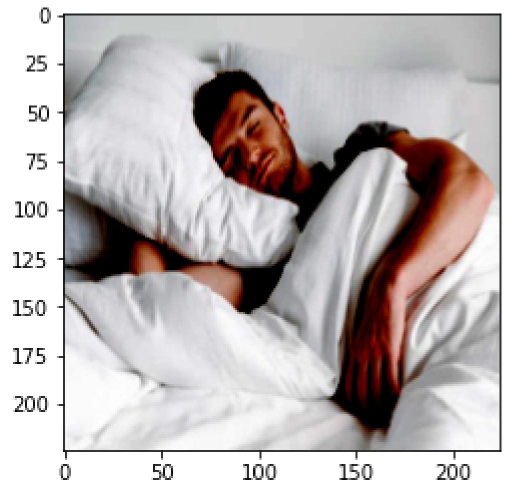
```
#Loading necessary libraries
import numpy as np
import pandas as pd
import skimage.io
from skimage import color
from skimage import io
import glob
import cv2
from scipy.ndimage.interpolation import map_coordinates
from scipy.ndimage.filters import gaussian_filter
import matplotlib.pyplot as plt
from torch.nn.modules.loss import BCEWithLogitsLoss
from torch.optim import lr_scheduler, Adam, SGD
import torch
import torchvision
from torch.utils.data import DataLoader
from torchvision import datasets, transforms
import torch.nn as nn
import torch.utils.data as data
import numpy as np
import os
import glob
import time
from sklearn.metrics import balanced_accuracy_score
from torch.autograd import Variable
from torch.nn import Linear, CrossEntropyLoss, Sequential, Conv2d, MaxPool2d, Module, Softmax
from torch.nn.modules.conv import ConvTranspose2d, Conv2d
from google.colab.patches import cv2_imshow
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from tqdm import tqdm
from torchvision import models
'''
Import any necessary libraries here to keep your code organized
'''
#import <some_library>
#from <something> import <something>
from torch.utils.data import WeightedRandomSampler


'''
DO NOT ALTER THE FOLLOWING CODE
'''
'''
NEW CHANGE on 16th Novemeber, 2022
Add the line below
'''
torch.cuda.empty_cache()
```

```
torch.cuda.empty_cache()
torch.manual_seed(0)
'''
Add the above line
'''
'''
DO NOT ALTER THE FOLLOWING CODE
'''
my_transforms = transforms.Compose([transforms.Resize((224,224)),   transforms.ToTensor(), tr
BATCH_SIZE = 16
IMAGE_SIZE = 32
NUM_CHANNELS = 3
n_epochs = 50 # the cnn will be trained for 50 epochs
dataset = datasets.ImageFolder(root=main_path, transform=my_transforms)
dataset_size = dataset.__len__() #compute the length of the training dataset
train_count = int(dataset_size * 0.8) #divide the training dataset to training and validation
val_count = dataset_size - train_count # keep the training proportion to 1 if no validation i
train_dataset, valid_dataset = data.random_split(dataset, [train_count, val_count]) #perform
y_train_indices = train_dataset.indices
y_train = [dataset.targets[i] for i in y_train_indices] #assign the labels or target variable
test_data = datasets.ImageFolder(test_path, transform=my_transforms)
'''
Following train, validation and test dataloaders will also be used in Part III: Resnets
'''
'''
NEW CHANGE on 16th Novemeber, 2022
'''
train_dataloader = DataLoader(train_dataset, batch_size=BATCH_SIZE, num_workers=2, shuffle=Tr
valid_dataloader = DataLoader(valid_dataset, batch_size=BATCH_SIZE, num_workers=2, )
test_dataloader = torch.utils.data.DataLoader(test_data, batch_size=BATCH_SIZE, )
'''
NEW CHANGE on 16th Novemeber, 2022
'''
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu') #check for gpu
print('Using ',device,'for model training') #print the device status
```

```
Using  cuda:0 for model training
```

```
#Visualize some training images
'''
DO NOT ALTER THE FOLLOWING CODE
'''
real_batch = next(iter(train_dataloader))
plt.figure(figsize=(8,8))
plt.axis("off")
plt.title("Training Images")
plt.imshow(np.transpose(torchvision.utils.make_grid(real_batch[0].to(device)[:64], padding=2,
plt.show()
plt.imshow(real_batch[0][5].permute(1,2,0), vmin=0, vmax=255)
plt.show()
print(f"Label: {real_batch[1][5]}")
```


```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with RGB data
```




```
Label: 0
```

```
'''
DO NOT ALTER THE FOLLOWING CODE
'''
def make_train_step(model, optimizer, loss_fn):
  '''
  INPUT: model, optimizer, loss function
  OUTPUT: train step
  '''
  def train_step(x,y):
    '''
    This function is used to train the model and update the model parameters. Do not change t
    '''
    #make prediction
    yhat = model(x)
    #enter train mode
    model.train()
    #compute loss
    loss = loss_fn(yhat,y)
    loss.backward()
    optimizer.step()
    optimizer.zero_grad()
    return loss
  return train_step


'''
DO NOT ALTER THE FOLLOWING CODE
```

```
'''
def train_model(model,n_epochs, loss_fn, train_step):
  '''
  This is the main function which is used to train the model, update weights, calculate loss
  '''
  train_losses = []
  val_losses = []
  epoch_train_losses = []
  epoch_val_losses = []
  for epoch in range(n_epochs):
    epoch_loss = 0
    for i ,data in tqdm(enumerate(train_dataloader), total = len(train_dataloader)): #iterate
      x_batch , y_batch = data
      x_batch = x_batch.to('cuda') #move to gpu
      y_batch = y_batch.unsqueeze(1).float() #convert target to same nn output shape
      y_batch = y_batch.to('cuda') #move to gpu
      loss = train_step(x_batch, y_batch)
      epoch_loss += loss/len(train_dataloader)
      train_losses.append(loss.cpu().detach().numpy())
    epoch_train_losses.append(epoch_loss)
    print('\nEpoch : {}, train loss : {}'.format(epoch+1,epoch_loss))
    #validation does not require gradient
    with torch.no_grad():
      cum_loss = 0
      for x_batch, y_batch in valid_dataloader:
        x_batch = x_batch.to('cuda')
        y_batch = y_batch.unsqueeze(1).float() #convert target to same nn output shape
        y_batch = y_batch.to('cuda')
        model.eval()#model to eval mode
        yhat = model(x_batch)
        val_loss = loss_fn(yhat,y_batch)
        cum_loss += loss/len(valid_dataloader)
        val_losses.append(val_loss.item())
      epoch_val_losses.append(cum_loss)
      print('Epoch : {}, val loss : {}'.format(epoch+1,cum_loss))
      best_loss = min(epoch_val_losses)
      #save best model
      if cum_loss <= best_loss:
        best_model_wts = model.state_dict()
  #load best model
  model.load_state_dict(best_model_wts)
  return model, train_losses,val_losses

def plot_losses(train_losses,val_losses):
  '''
  This function can be used to plot the training and validation losses. You can use this
  function to analyse the losses and judge if model was overfitting or if model shows some
  unusual behaviour.
  '''
  plt.plot(train_losses, label='Training loss')
  plt.plot(val_losses, label='Validation loss')
```

```
    plt.legend()
    plt.show()

def inference(model,test_data):
    '''
    As we are doing binary classification, this function uses sigmoid to change class probabili
    to either 0 or 1 class.
    '''
    y_pred = []
    y_true = []
    for idx in range(1, len(test_data)):
      y_true.append( test_data[idx][1])
      sample = torch.unsqueeze(test_data[idx][0], dim=0).to('cuda')
      if torch.sigmoid(model(sample)) < 0.5:
        y_pred.append(0)
      else:
        y_pred.append(1)
    return y_pred, y_true

def calc_loss(model, n_epochs):
    '''
    This function drives the training function, assigns the loss fuction and sets the optimiize
    '''
    loss_fn = BCEWithLogitsLoss()
    optimizer = torch.optim.Adam(model.parameters())
    train_step = make_train_step(model, optimizer, loss_fn)
    trained_model, train_losses, val_losses = train_model(model,n_epochs, loss_fn, train_step)
    return trained_model

def calc_accuracy(trained_model):
    '''
    This function is used for returning the calculated accuracies.
    '''
    y_pred, y_true = inference(trained_model,test_data)
    target_names = ['Adults', 'Kids']
    print('the accuracy is',accuracy_score(y_true, y_pred))
    print(classification_report(y_true, y_pred, target_names=target_names))
    print('the balanced accuracy is',balanced_accuracy_score(y_true, y_pred))
    return accuracy_score(y_true, y_pred)
```

Consider the following code snippet for a Neural Network

This Network is a very simple Network for your reference to implement a Neural Network of any given architecture.

```
class Net(Module):

    def __init__(self):
```

```
        super(Net, self).__init__()
        self.cnn_layers = Sequential(
            # Defining a 2D convolution layer
            Conv2d(NUM_CHANNELS, IMAGE_SIZE, kernel_size=3, stride=1, padding=1),
            BatchNorm2d(IMAGE_SIZE),
            MaxPool2d(kernel_size=2, stride=2),
            # Defining another 2D convolution layer
            Conv2d(32, 32, kernel_size=3, stride=1, padding=1),
            BatchNorm2d(32),
            MaxPool2d(kernel_size=2, stride=2),
        )
        self.linear_layers = Sequential(
            Linear(100352, 1)
        )
    # Defining the forward pass
    def forward(self, x):
        x = self.cnn_layers(x)
        x = x.view(x.size(0), -1)
        x = self.linear_layers(x)
        return x

# defining the model
model = Net()
# defining the optimizer
optimizer = Adam(model.parameters(), lr=0.07)
# defining the loss function
criterion = CrossEntropyLoss()
# checking if GPU is available
if torch.cuda.is_available():
    model = model.cuda()
    criterion = criterion.cuda()

print(model)

trained_model = calc_loss(model, n_epochs)
calc_accuracy(trained_model)
```

(1) Build a CNN for the following Model Architecture [3 marks]

```
Net(
  (cnn_1): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

```
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (cnn_2): Sequential(
    (0): Conv2d(32, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (cnn_3): Sequential(
    (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (cnn_4): Sequential(
    (0): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (cnn_5): Sequential(
    (0): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (cnn_6): Sequential(
    (0): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fully_1): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=50176, out_features=4096, bias=True)
  )
  (fully_2): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=4096, out_features=4096, bias=True)
  )
  (fully_3): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=4096, out_features=1000, bias=True)
  )
  (fully_4): Sequential(
    (0): Linear(in_features=1000, out_features=1, bias=True)
  )
)
```

The codebase is as follows:

```
from torch.nn.modules.conv import ConvTranspose2d, Conv2d
class Net(Module):
    def __init__(self):
        super(Net, self).__init__()
        '''
        Write the 6 CNNs and 4 fully connected CNNs here:
        '''
        self.cnn_layers = Sequential(
            # Defining a 2D convolution layer
            Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),
            # Defining another 2D convolution layer
            Conv2d(32, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),

            Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),

            Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),

            Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),

            Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

        )
        self.linear_layers = Sequential(
            Dropout(p=0.5, inplace=False),
            Linear(in_features=50176, out_features=4096, bias=True),

            Dropout(p=0.5, inplace=False),
            Linear(in_features=4096, out_features=4096, bias=True),

            Dropout(p=0.5, inplace=False),
            Linear(in_features=4096, out_features=1000, bias=True),

            Linear(in_features=1000, out_features=1, bias=True)
        )
    def forward(self, x):
        '''
        Define the forward pass
        '''
        x = self.cnn_layers(x)
        x = x.view(x.size(0), -1)
        x = self.linear_layers(x)
```

```
        return x
'''
DO NOT ALTER THE FOLLOWING CODE
'''
model = Net()
optimizer = Adam(model.parameters(), lr=0.07)
criterion = CrossEntropyLoss()
if torch.cuda.is_available():
    model = model.cuda()
    criterion = criterion.cuda()
print(model)
trained_model = calc_loss(model, n_epochs) #train the model
calc_accuracy(trained_model) # report the accuracy
```

```
Epoch : 39, val loss : 41.51344992392378
100%|██████████| 34/34 [00:09<00:00,  3.59it/s]
Epoch : 40, train loss : 80.09747314453125

Epoch : 40, val loss : 118.00894165039062
100%|██████████| 34/34 [00:09<00:00,  3.56it/s]
Epoch : 41, train loss : 59.75107192993164

Epoch : 41, val loss : 39.27111053466797
100%|██████████| 34/34 [00:09<00:00,  3.57it/s]
Epoch : 42, train loss : 24.554697036743164

Epoch : 42, val loss : 35.290855407714844
100%|██████████| 34/34 [00:09<00:00,  3.55it/s]
Epoch : 43, train loss : 20.00562858581543

Epoch : 43, val loss : 27.344970703125
100%|██████████| 34/34 [00:09<00:00,  3.58it/s]
Epoch : 44, train loss : 20.713529586791992

Epoch : 44, val loss : 12.072805404663086
100%|██████████| 34/34 [00:09<00:00,  3.58it/s]
Epoch : 45, train loss : 23.83782958984375

Epoch : 45, val loss : 26.377037048339844
100%|██████████| 34/34 [00:09<00:00,  3.60it/s]
Epoch : 46, train loss : 26.119722366333008

Epoch : 46, val loss : 18.94969367980957
100%|██████████| 34/34 [00:09<00:00,  3.59it/s]
Epoch : 47, train loss : 28.09427261352539

Epoch : 47, val loss : 21.78936767578125
100%|██████████| 34/34 [00:09<00:00,  3.54it/s]
Epoch : 48, train loss : 12.449796676635742

Epoch : 48, val loss : 18.318893432617188
100%|██████████| 34/34 [00:09<00:00,  3.57it/s]
Epoch : 49, train loss : 24.264476776123047

Epoch : 49, val loss : 30.468828201293945
100%|██████████| 34/34 [00:09<00:00,  3.58it/s]
```

```
Epoch : 50, train loss : 18.32427215576172

Epoch : 50, val loss : 6.611057281494141
the accuracy is 0.40336134453781514
              precision    recall  f1-score   support

      Adults       0.41      0.44      0.42        59
        Kids       0.40      0.37      0.38        60

    accuracy                           0.40       119
   macro avg       0.40      0.40      0.40       119
weighted avg       0.40      0.40      0.40       119

the balanced accuracy is 0.4036723163841808
0.40336134453781514
```

(2) Activation Functions [3 marks]

Plug in the following Activation Functions:

1. ReLU
2. SiLU
3. Sigmoid
4. Tanh
5. ELU

Your Network Architecture should be as follows:

```
Net(
  (cnn_1): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (cnn_2): Sequential(
    (0): Conv2d(32, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (cnn_3): Sequential(
    (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (cnn_4): Sequential(
    (0): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
```

```
      (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (cnn_5): Sequential(
      (0): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (cnn_6): Sequential(
      (0): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
      (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    )
    (fully_1): Sequential(
      (0): Dropout(p=0.5, inplace=False)
      (1): Linear(in_features=50176, out_features=4096, bias=True)
      (2): YOUR ACTIVATION FUNCTION COMES HERE
    )
    (fully_2): Sequential(
      (0): Dropout(p=0.5, inplace=False)
      (1): Linear(in_features=4096, out_features=4096, bias=True)
      (2): YOUR ACTIVATION FUNCTION COMES HERE
    )
    (fully_3): Sequential(
      (0): Dropout(p=0.5, inplace=False)
      (1): Linear(in_features=4096, out_features=1000, bias=True)
      (2): YOUR ACTIVATION FUNCTION COMES HERE
    )
    (fully_4): Sequential(
      (0): Linear(in_features=1000, out_features=1, bias=True)
    )
  )
```

```
'''
ReLU
'''
from torch.nn.modules.conv import ConvTranspose2d, Conv2d
class Net(Module):
    def __init__(self):
        super(Net, self).__init__()
        '''
        Write the 6 CNNs and 4 fully connected CNNs here
        '''
        self.cnn_layers = Sequential(
            # Defining a 2D convolution layer
            Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
```

```
            BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),
            # Defining another 2D convolution layer
            Conv2d(32, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),

            Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),

            Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),

            Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),

            Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

            )

        self.linear_layers = Sequential(
            Dropout(p=0.5, inplace=False),
            Linear(in_features=50176, out_features=4096, bias=True),
            Dropout(p=0.5, inplace=False),
            nn.ReLU(),

            Dropout(p=0.5, inplace=False),
            Linear(in_features=4096, out_features=4096, bias=True),
            nn.ReLU(),

            Dropout(p=0.5, inplace=False),
            Linear(in_features=4096, out_features=1000, bias=True),
            nn.ReLU(),

            Linear(in_features=1000, out_features=1, bias=True)
        )
    def forward(self, x):
        '''
        Define the forward pass :
        '''
        x = self.cnn_layers(x)
        x = x.view(x.size(0), -1)
        x = self.linear_layers(x)
        return x

'''
DO NOT ALTER THE FOLLOWING CODE
'''
model = Net()
```

```
optimizer = Adam(model.parameters(), lr=0.07)
criterion = CrossEntropyLoss()
if torch.cuda.is_available():
    model = model.cuda()
    criterion = criterion.cuda()
print(model)
trained_model = calc_loss(model, n_epochs) #train the model
calc_accuracy(trained_model) # report the accuracy
```

```
Epoch : 41, val loss : 0.6916751265525818
100%|██████████| 34/34 [00:08<00:00,  3.81it/s]
Epoch : 42, train loss : 0.6931805610656738

Epoch : 42, val loss : 0.6944507956504822
100%|██████████| 34/34 [00:08<00:00,  3.78it/s]
Epoch : 43, train loss : 0.6931415796279907

Epoch : 43, val loss : 0.6905621290206909
100%|██████████| 34/34 [00:09<00:00,  3.57it/s]
Epoch : 44, train loss : 0.6978998184204102

Epoch : 44, val loss : 0.6892213225364685
100%|██████████| 34/34 [00:09<00:00,  3.77it/s]
Epoch : 45, train loss : 0.6931379437446594

Epoch : 45, val loss : 0.6924229264259338
100%|██████████| 34/34 [00:08<00:00,  3.82it/s]
Epoch : 46, train loss : 0.693169891834259


Epoch : 46, val loss : 0.6924358606338501
100%|██████████| 34/34 [00:08<00:00,  3.80it/s]
Epoch : 47, train loss : 0.6931765675544739

Epoch : 47, val loss : 0.6965241432189941
100%|██████████| 34/34 [00:08<00:00,  3.81it/s]
Epoch : 48, train loss : 0.693169116973877

Epoch : 48, val loss : 0.6940121650695801
100%|██████████| 34/34 [00:09<00:00,  3.67it/s]
Epoch : 49, train loss : 0.6931530833244324

Epoch : 49, val loss : 0.6922976970672607
100%|██████████| 34/34 [00:08<00:00,  3.80it/s]
Epoch : 50, train loss : 0.6931747198104858

Epoch : 50, val loss : 0.6961726546287537
the accuracy is 0.5042016806722689
              precision    recall  f1-score   support

      Adults       0.00      0.00      0.00        59
        Kids       0.50      1.00      0.67        60

    accuracy                           0.50       119
   macro avg       0.25      0.50      0.34       119
```

```
weighted avg       0.25      0.50      0.34       119

the balanced accuracy is 0.5
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Undef
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Undef
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Undef
  _warn_prf(average, modifier, msg_start, len(result))
0.5042016806722689
```

```
'''
SiLU
'''
from torch.nn.modules.conv import ConvTranspose2d, Conv2d
class Net(Module):
    def __init__(self):
        super(Net, self).__init__()
        '''
        Write the 6 CNNs and 4 fully connected CNNs here
        '''
        self.cnn_layers = Sequential(
            # Defining a 2D convolution layer
            Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),
            # Defining another 2D convolution layer
            Conv2d(32, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),

            Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),

            Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),

            Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),

            Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

            )

        self.linear_layers = Sequential(
            Dropout(p=0.5, inplace=False),
            Linear(in_features=50176, out_features=4096, bias=True),
            Dropout(p=0.5, inplace=False),
```

```
            nn.SiLU(),

            Dropout(p=0.5, inplace=False),
            Linear(in_features=4096, out_features=4096, bias=True),
            nn.SiLU(),

            Dropout(p=0.5, inplace=False),
            Linear(in_features=4096, out_features=1000, bias=True),
            nn.SiLU(),

            Linear(in_features=1000, out_features=1, bias=True)
        )
    def forward(self, x):
        '''
        Define the forward pass :
        '''
        x = self.cnn_layers(x)
        x = x.view(x.size(0), -1)
        x = self.linear_layers(x)
        return x

'''
DO NOT ALTER THE FOLLOWING CODE
'''
model = Net()
optimizer = Adam(model.parameters(), lr=0.07)
criterion = CrossEntropyLoss()
if torch.cuda.is_available():
    model = model.cuda()
    criterion = criterion.cuda()
print(model)
trained_model = calc_loss(model, n_epochs) #train the model
calc_accuracy(trained_model) # report the accuracy
```

```
Epoch : 41, val loss : 0.6921964287757874
100%|██████████| 34/34 [00:08<00:00,  3.83it/s]
Epoch : 42, train loss : 0.6931336522102356

Epoch : 42, val loss : 0.688835859298706
100%|██████████| 34/34 [00:08<00:00,  3.87it/s]
Epoch : 43, train loss : 0.6931358575820923

Epoch : 43, val loss : 0.6968706846237183
100%|██████████| 34/34 [00:08<00:00,  3.93it/s]
Epoch : 44, train loss : 0.6931381821632385

Epoch : 44, val loss : 0.6941455602645874
100%|██████████| 34/34 [00:08<00:00,  3.87it/s]
Epoch : 45, train loss : 0.6931499242782593

Epoch : 45, val loss : 0.6922851800918579
100%|██████████| 34/34 [00:08<00:00,  3.82it/s]
```

```
Epoch : 46, train loss : 0.6931659579277039

Epoch : 46, val loss : 0.6973373889923096
100%|██████████| 34/34 [00:08<00:00,  3.92it/s]
Epoch : 47, train loss : 0.6931701898574829

Epoch : 47, val loss : 0.6931750774383545
100%|██████████| 34/34 [00:08<00:00,  3.89it/s]
Epoch : 48, train loss : 0.6931719779968262

Epoch : 48, val loss : 0.6941577792167664
100%|██████████| 34/34 [00:08<00:00,  3.85it/s]
Epoch : 49, train loss : 0.69314044713974

Epoch : 49, val loss : 0.6931750774383545
100%|██████████| 34/34 [00:08<00:00,  3.90it/s]
Epoch : 50, train loss : 0.6931585669517517

Epoch : 50, val loss : 0.6948122382164001
the accuracy is 0.5042016806722689
              precision    recall  f1-score   support

      Adults       0.00      0.00      0.00        59
        Kids       0.50      1.00      0.67        60

    accuracy                           0.50       119
   macro avg       0.25      0.50      0.34       119
weighted avg       0.25      0.50      0.34       119

the balanced accuracy is 0.5
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Undef
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Undef
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Undef
  _warn_prf(average, modifier, msg_start, len(result))
0.5042016806722689
```

```
'''
Sigmoid
'''
from torch.nn.modules.conv import ConvTranspose2d, Conv2d
class Net(Module):
    def __init__(self):
        super(Net, self).__init__()
        '''
        Write the 6 CNNs and 4 fully connected CNNs here
        '''
        self.cnn_layers = Sequential(
            # Defining a 2D convolution layer
            Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
```

```
            MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),
            # Defining another 2D convolution layer
            Conv2d(32, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),

            Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),

            Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),

            Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),

            Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

            )

        self.linear_layers = Sequential(
            Dropout(p=0.5, inplace=False),
            Linear(in_features=50176, out_features=4096, bias=True),
            Dropout(p=0.5, inplace=False),
            nn.Sigmoid(),

            Dropout(p=0.5, inplace=False),
            Linear(in_features=4096, out_features=4096, bias=True),
            nn.Sigmoid(),

            Dropout(p=0.5, inplace=False),
            Linear(in_features=4096, out_features=1000, bias=True),
            nn.Sigmoid(),

            Linear(in_features=1000, out_features=1, bias=True)
        )
    def forward(self, x):
        '''
        Define the forward pass :
        '''
        x = self.cnn_layers(x)
        x = x.view(x.size(0), -1)
        x = self.linear_layers(x)
        return x

'''
DO NOT ALTER THE FOLLOWING CODE
'''
model = Net()
optimizer = Adam(model.parameters(), lr=0.07)
```

```
criterion = CrossEntropyLoss()
if torch.cuda.is_available():
    model = model.cuda()
    criterion = criterion.cuda()
print(model)
trained_model = calc_loss(model, n_epochs) #train the model
calc_accuracy(trained_model) # report the accuracy
```

```
100%|██████████| 34/34 [00:09<00:00,  3.71it/s]
Epoch : 40, train loss : 0.6758832931518555

Epoch : 40, val loss : 0.6762614250183105
100%|██████████| 34/34 [00:09<00:00,  3.68it/s]
Epoch : 41, train loss : 0.6525259017944336

Epoch : 41, val loss : 0.5362340211868286
100%|██████████| 34/34 [00:09<00:00,  3.48it/s]
Epoch : 42, train loss : 0.6756006479263306

Epoch : 42, val loss : 0.6954056024551392
100%|██████████| 34/34 [00:09<00:00,  3.68it/s]
Epoch : 43, train loss : 0.6531364321708679

Epoch : 43, val loss : 0.6653584837913513
100%|██████████| 34/34 [00:09<00:00,  3.73it/s]
Epoch : 44, train loss : 0.6533393263816833

Epoch : 44, val loss : 0.753437876701355
100%|██████████| 34/34 [00:09<00:00,  3.69it/s]
Epoch : 45, train loss : 0.6532008051872253

Epoch : 45, val loss : 0.5966447591781616
100%|██████████| 34/34 [00:09<00:00,  3.56it/s]
Epoch : 46, train loss : 0.6555994749069214


Epoch : 46, val loss : 0.6797139644622803
100%|██████████| 34/34 [00:09<00:00,  3.56it/s]
Epoch : 47, train loss : 0.650086522102356

Epoch : 47, val loss : 0.5932134389877319
100%|██████████| 34/34 [00:09<00:00,  3.71it/s]
Epoch : 48, train loss : 0.649658203125

Epoch : 48, val loss : 0.6432350277900696
100%|██████████| 34/34 [00:09<00:00,  3.46it/s]
Epoch : 49, train loss : 0.664524495601654

Epoch : 49, val loss : 0.6587889790534973
100%|██████████| 34/34 [00:09<00:00,  3.70it/s]
Epoch : 50, train loss : 0.652068555355072

Epoch : 50, val loss : 0.5342998504638672
the accuracy is 0.5546218487394958
              precision    recall  f1-score   support
```

```
      Adults       0.54      0.68      0.60        59
        Kids       0.58      0.43      0.50        60

    accuracy                           0.55       119
   macro avg       0.56      0.56      0.55       119
weighted avg       0.56      0.55      0.55       119

the balanced accuracy is 0.5556497175141243
0.5546218487394958
```

```
'''
Tanh
'''
from torch.nn.modules.conv import ConvTranspose2d, Conv2d
class Net(Module):
    def __init__(self):
        super(Net, self).__init__()
        '''
        Write the 6 CNNs and 4 fully connected CNNs here
        '''
        self.cnn_layers = Sequential(
            # Defining a 2D convolution layer
            Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),
            # Defining another 2D convolution layer
            Conv2d(32, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),

            Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),

            Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),

            Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),

            Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

            )

        self.linear_layers = Sequential(
            Dropout(p=0.5, inplace=False),
            Linear(in_features=50176, out_features=4096, bias=True),
            Dropout(p=0.5, inplace=False),
            nn.Tanh(),
```

```
            Dropout(p=0.5, inplace=False),
            Linear(in_features=4096, out_features=4096, bias=True),
            nn.Tanh(),

            Dropout(p=0.5, inplace=False),
            Linear(in_features=4096, out_features=1000, bias=True),
            nn.Tanh(),

            Linear(in_features=1000, out_features=1, bias=True)
        )
    def forward(self, x):
        '''
        Define the forward pass :
        '''
        x = self.cnn_layers(x)
        x = x.view(x.size(0), -1)
        x = self.linear_layers(x)
        return x

'''
DO NOT ALTER THE FOLLOWING CODE
'''
model = Net()
optimizer = Adam(model.parameters(), lr=0.07)
criterion = CrossEntropyLoss()
if torch.cuda.is_available():
    model = model.cuda()
    criterion = criterion.cuda()
print(model)
trained_model = calc_loss(model, n_epochs) #train the model
calc_accuracy(trained_model) # report the accuracy
```

```
100%|██████████| 34/34 [00:09<00:00,  3.69it/s]
Epoch : 40, train loss : 0.7196840047836304

Epoch : 40, val loss : 0.5669674277305603
100%|██████████| 34/34 [00:08<00:00,  3.79it/s]
Epoch : 41, train loss : 0.6560000777244568

Epoch : 41, val loss : 0.6378151178359985
100%|██████████| 34/34 [00:08<00:00,  3.78it/s]
Epoch : 42, train loss : 0.6698829531669617

Epoch : 42, val loss : 0.7845102548599243
100%|██████████| 34/34 [00:08<00:00,  3.79it/s]
Epoch : 43, train loss : 0.6636332869529724

Epoch : 43, val loss : 0.5609439611434937
100%|██████████| 34/34 [00:09<00:00,  3.68it/s]
Epoch : 44, train loss : 0.6611030101776123

Epoch : 44, val loss : 0.7995293736457825
```

```
100%|██████████| 34/34 [00:09<00:00,  3.53it/s]
Epoch : 45, train loss : 0.6736108660697937

Epoch : 45, val loss : 0.542056679725647
100%|██████████| 34/34 [00:09<00:00,  3.66it/s]
Epoch : 46, train loss : 0.6757252216339111

Epoch : 46, val loss : 0.6344669461250305
100%|██████████| 34/34 [00:09<00:00,  3.59it/s]
Epoch : 47, train loss : 0.6527031064033508

Epoch : 47, val loss : 0.8013876676559448
100%|██████████| 34/34 [00:09<00:00,  3.75it/s]
Epoch : 48, train loss : 0.6377087235450745

Epoch : 48, val loss : 0.5914124250411987
100%|██████████| 34/34 [00:09<00:00,  3.43it/s]
Epoch : 49, train loss : 0.6720487475395203

Epoch : 49, val loss : 0.7692890763282776
100%|██████████| 34/34 [00:08<00:00,  3.79it/s]
Epoch : 50, train loss : 0.6797780394554138

Epoch : 50, val loss : 0.8123757243156433
the accuracy is 0.5714285714285714
              precision    recall  f1-score   support

      Adults       0.56      0.63      0.59        59
        Kids       0.58      0.52      0.55        60

    accuracy                           0.57       119
   macro avg       0.57      0.57      0.57       119
weighted avg       0.57      0.57      0.57       119

the balanced accuracy is 0.5718926553672317
0.5714285714285714
```

```
'''
ELU
'''
from torch.nn.modules.conv import ConvTranspose2d, Conv2d
class Net(Module):
    def __init__(self):
        super(Net, self).__init__()
        '''
        Write the 6 CNNs and 4 fully connected CNNs here
        '''
        self.cnn_layers = Sequential(
            # Defining a 2D convolution layer
            Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),
```

```
            # Defining another 2D convolution layer
            Conv2d(32, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),

            Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),

            Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),

            Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),

            Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

            )

        self.linear_layers = Sequential(
            Dropout(p=0.5, inplace=False),
            Linear(in_features=50176, out_features=4096, bias=True),
            Dropout(p=0.5, inplace=False),
            nn.ELU(),

            Dropout(p=0.5, inplace=False),
            Linear(in_features=4096, out_features=4096, bias=True),
            nn.ELU(),

            Dropout(p=0.5, inplace=False),
            Linear(in_features=4096, out_features=1000, bias=True),
            nn.ELU(),

            Linear(in_features=1000, out_features=1, bias=True)
        )
    def forward(self, x):
        '''
        Define the forward pass :
        '''
        x = self.cnn_layers(x)
        x = x.view(x.size(0), -1)
        x = self.linear_layers(x)
        return x

'''
DO NOT ALTER THE FOLLOWING CODE
'''
model = Net()
optimizer = Adam(model.parameters(), lr=0.07)
criterion = CrossEntropyLoss()
```

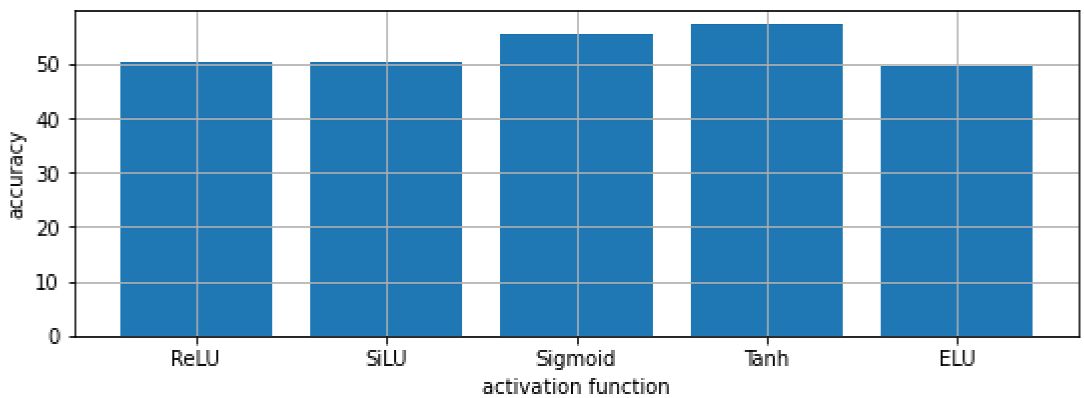
```
if torch.cuda.is_available():
    model = model.cuda()
    criterion = criterion.cuda()
print(model)
trained_model = calc_loss(model, n_epochs) #train the model
calc_accuracy(trained_model) # report the accuracy
```

```
Epoch : 41, val loss : 0.6981595158576965
100%|██████████| 34/34 [00:08<00:00,  3.78it/s]
Epoch : 42, train loss : 0.6907509565353394

Epoch : 42, val loss : 0.6977829337120056
100%|██████████| 34/34 [00:08<00:00,  3.78it/s]
Epoch : 43, train loss : 0.6999696493148804

Epoch : 43, val loss : 0.693679928779602
100%|██████████| 34/34 [00:09<00:00,  3.74it/s]
Epoch : 44, train loss : 0.7245017290115356

Epoch : 44, val loss : 0.7272980809211731
100%|██████████| 34/34 [00:10<00:00,  3.40it/s]
Epoch : 45, train loss : 0.7109299898147583

Epoch : 45, val loss : 0.6868575811386108
100%|██████████| 34/34 [00:09<00:00,  3.71it/s]
Epoch : 46, train loss : 0.7031164169311523

Epoch : 46, val loss : 0.707773745059967
100%|██████████| 34/34 [00:09<00:00,  3.67it/s]
Epoch : 47, train loss : 0.7046319842338562

Epoch : 47, val loss : 0.7085851430892944
100%|██████████| 34/34 [00:09<00:00,  3.65it/s]
Epoch : 48, train loss : 0.7314693927764893

Epoch : 48, val loss : 0.6932079792022705
100%|██████████| 34/34 [00:09<00:00,  3.63it/s]
Epoch : 49, train loss : 0.7551656365394592

Epoch : 49, val loss : 0.712471604347229
100%|██████████| 34/34 [00:10<00:00,  3.30it/s]
Epoch : 50, train loss : 0.7428637742996216

Epoch : 50, val loss : 0.9464645385742188
the accuracy is 0.4957983193277311
              precision    recall  f1-score   support

      Adults       0.50      1.00      0.66        59
        Kids       0.00      0.00      0.00        60

    accuracy                           0.50       119
   macro avg       0.25      0.50      0.33       119
weighted avg       0.25      0.50      0.33       119

the balanced accuracy is 0.5
```

```
the balanced accuracy is 0.5
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Undef
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Undef
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1318: Undef
  _warn_prf(average, modifier, msg_start, len(result))
0.4957983193277311
```

1. **Plot the accuracies for each activation function**

```
#ENTER CODE HERE
names = ['ReLU', 'SiLU', 'Sigmoid', 'Tanh','ELU']
values = [50.42, 50.42, 55.46, 57.14, 49.58]

plt.figure(figsize=(9, 3))
plt.bar(names, values)
plt.xlabel('activation function')
plt.ylabel('accuracy')
plt.grid(True)
plt.show()
```



2. **Which function performs better? Justify.**

ANSWER- Based on my codes and efforts, the Tanh function was the best in case of balanced accuracy. Since the dataset is balanced, we can use this accuracy measure.

We cannot justify the results unless we use a cross-validation method. However, Tanh performs good since it is zero centered, and with its derivative, both are monotonic. Although a downside of using Tanh is the vanishing gradient problem, but since we didn't use a long network, we didn't encounter this problem. I have to mention that the mean of tanh function would be closer to zero compared to the Sigmoid function, and outperforms it. It's a surprise that ReLU was not the best, but it really depends on the type of the network and requires a better assessment to justify.

Part II: Custom Activation Functions

(1) Implement any activation function of your OWN and DO NOT USE any predefined PyTorch Activation Functions [3 marks]

```
class custom_activation_function(nn.Module):
    '''
    Implementation of custom activation function
    '''
    def __init__(self, in_features, alpha = None):
        '''
        Initialization
        '''

    def forward(self, x):
        '''
        Forward pass of the function.
        '''

af = custom_activation_function(<some_parameters>)
x = torch.randn(256) # random tensor
x = af(x)
```

Your Network Architecture should be as follows:

```
Net(
  (cnn_1): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (cnn_2): Sequential(
    (0): Conv2d(32, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (cnn_3): Sequential(
    (0): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
```

```
  (cnn_4): Sequential(
    (0): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (cnn_5): Sequential(
    (0): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  )
  (cnn_6): Sequential(
    (0): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fully_1): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=50176, out_features=4096, bias=True)
  )
  (fully_2): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=4096, out_features=4096, bias=True)
  )
  (fully_3): Sequential(
    (0): Dropout(p=0.5, inplace=False)
    (1): Linear(in_features=4096, out_features=1000, bias=True)
  )
  (fully_4): Sequential(
    (0): Linear(in_features=1000, out_features=1, bias=True)
  )
  (a1): custom_activation_function()
  (a2): custom_activation_function()
  (a3): custom_activation_function()
)
```

Hint: Here, we are asking you to apply Custom activation functions to the Fully Connected Layers in the forward pass of the Network

```
def my_func(x):
  a = x>0
  b = torch.ones(a.shape)
  if torch.cuda.is_available():
    b = b.to("cuda")
  x_np = b * a

  return x_np
```

```
from torch.nn.modules.conv import ConvTranspose2d, Conv2d
class Net(Module):
    def __init__(self):
        super(Net, self).__init__()
        '''
        Write the 6 CNNs and 4 fully connected CNNs here
        '''
        self.cnn_layers = Sequential(
            # Defining a 2D convolution layer
            Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),
            # Defining another 2D convolution layer
            Conv2d(32, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),

            Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),

            Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),

            Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),

            Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

            )

        self.First_Dropout = Dropout(p=0.5, inplace=False)
        self.First_linear = Linear(in_features=50176, out_features=4096, bias=True)
        self.Second_Dropout = Dropout(p=0.5, inplace=False)
        self.Second_linear = Linear(in_features=4096, out_features=4096, bias=True)
        self.Third_Dropout = Dropout(p=0.5, inplace=False)
        self.Third_linear = Linear(in_features=4096, out_features=1000, bias=True)
        self.Fourth_linear = Linear(in_features=1000, out_features=1, bias=True)

        self.activation_function = my_func

    def forward(self, x):
        '''
        Define the forward pass :
        '''
        x = self.cnn_layers(x)
        x = x.view(x.size(0), -1)

        x = self.First_Dropout(x)
```

```
        x = self.First_linear(x)

        x = self.Second_Dropout(x)
        x = self.Second_linear(x)

        x = self.Third_Dropout(x)
        x = self.Third_linear(x)

        x = self.Fourth_linear(x)

        x = self.activation_function(x)
        x = self.activation_function(x)
        x = self.activation_function(x)

        return x

'''
DO NOT ALTER THE FOLLOWING CODE
'''
model = Net()
optimizer = Adam(model.parameters(), lr=0.07)
criterion = CrossEntropyLoss()
if torch.cuda.is_available():
    model = model.cuda()
    criterion = criterion.cuda()
print(model)
trained_model = calc_loss(model, n_epochs) #train the model
calc_accuracy(trained_model) # report the accuracy
```

```
Net(
  (cnn_layers): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tr
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(32, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
    (8): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
    (10): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
    (12): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
    (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (dropout1): Dropout(p=0.5, inplace=False)
  (linear1): Linear(in_features=50176, out_features=4096, bias=True)
  (dropout2): Dropout(p=0.5, inplace=False)
  (linear2): Linear(in_features=4096, out_features=4096, bias=True)
  (dropout3): Dropout(p=0.5, inplace=False)
  (linear3): Linear(in_features=4096, out_features=1000, bias=True)
```

```
  (linear4): Linear(in_features=1000, out_features=1, bias=True)
)
100%|██████████| 34/34 [04:35<00:00,  8.10s/it]
Epoch : 1, train loss : 0.7302287220954895

Epoch : 1, val loss : 0.7034124732017517
100%|██████████| 34/34 [00:03<00:00,  9.58it/s]
Epoch : 2, train loss : 0.7056767344474792

Epoch : 2, val loss : 0.6297773718833923
100%|██████████| 34/34 [00:03<00:00,  9.60it/s]
Epoch : 3, train loss : 0.7154108881950378

Epoch : 3, val loss : 0.7622132301330566
100%|██████████| 34/34 [00:03<00:00,  9.68it/s]
Epoch : 4, train loss : 0.7114231586456299

Epoch : 4, val loss : 0.7154419422149658
100%|██████████| 34/34 [00:03<00:00,  9.68it/s]
Epoch : 5, train loss : 0.7037389874458313

Epoch : 5, val loss : 0.7174227237701416
100%|██████████| 34/34 [00:03<00:00,  9.76it/s]
Epoch : 6, train loss : 0.7136957049369812

Epoch : 6, val loss : 0.6567777395248413
100%|██████████| 34/34 [00:03<00:00,  9.55it/s]
Epoch : 7, train loss : 0.7094559073448181

Epoch : 7, val loss : 0.6754783391952515
100%|██████████| 34/34 [00:04<00:00,  7.56it/s]
Epoch : 8, train loss : 0.7024308443069458
```

(2) Implement any COMPLEX activation function of your OWN and DO NOT USE any predefined PyTorch Activation Functions {CMPUT 566 only} [5 marks]

Implement any one of the following activation functions

1. Soft exponential
2. BReLU

```
def BReLU(x):

    x_copy = x.clone()
    evens = [i for i in range(0, x.shape[0], 2)]
    odds = [i for i in range(1, x.shape[0], 2)]

    x_copy[evens] = x_copy[evens]*(x_copy[evens]>0)
    x_copy[odds] = - x_copy[odds]
    x_copy[odds] = - (x_copy[odds]*(x_copy[odds]>0))
```

```
        return x_copy

from torch.nn.modules.conv import ConvTranspose2d, Conv2d
class Net(Module):
    def __init__(self):
        super(Net, self).__init__()
        '''
        Write the 6 CNNs and 4 fully connected CNNs here
        '''
        self.cnn_layers = Sequential(
            # Defining a 2D convolution layer
            Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),
            # Defining another 2D convolution layer
            Conv2d(32, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False),

            Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),

            Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),

            Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),

            Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1)),
            BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True),
            MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)

            )

        self.First_Dropout = Dropout(p=0.5, inplace=False)
        self.First_linear = Linear(in_features=50176, out_features=4096, bias=True)
        self.Second_Dropout = Dropout(p=0.5, inplace=False)
        self.Second_linear = Linear(in_features=4096, out_features=4096, bias=True)
        self.Third_Dropout = Dropout(p=0.5, inplace=False)
        self.Third_linear = Linear(in_features=4096, out_features=1000, bias=True)
        self.Fourth_linear = Linear(in_features=1000, out_features=1, bias=True)

        self.activation_function = BReLU

    def forward(self, x):
        '''
        Define the forward pass :
        '''
        x = self.cnn_layers(x)
        x = x.view(x.size(0), -1)
```

```
        x = self.First_Dropout(x)
        x = self.First_linear(x)

        x = self.Second_Dropout(x)
        x = self.Second_linear(x)

        x = self.Third_Dropout(x)
        x = self.Third_linear(x)

        x = self.Fourth_linear(x)

        x = self.activation_function(x)
        x = self.activation_function(x)
        x = self.activation_function(x)

        return x

'''
DO NOT ALTER THE FOLLOWING CODE
'''
model = Net()
optimizer = Adam(model.parameters(), lr=0.07)
criterion = CrossEntropyLoss()
if torch.cuda.is_available():
    model = model.cuda()
    criterion = criterion.cuda()
print(model)
trained_model = calc_loss(model, n_epochs) #train the model
calc_accuracy(trained_model) # report the accuracy
```

```
Net(
  (cnn_layers): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tr
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (3): Conv2d(32, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (4): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
    (5): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (6): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (7): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
    (8): Conv2d(512, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
    (10): Conv2d(256, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
    (12): Conv2d(128, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
    (14): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (dropout1): Dropout(p=0.5, inplace=False)
  (linear1): Linear(in_features=50176, out_features=4096, bias=True)
  (dropout2): Dropout(p=0.5, inplace=False)
  (linear2): Linear(in_features=4096, out_features=4096, bias=True)
  (dropout3): Dropout(p=0.5, inplace=False)
```

```
    (linear3): Linear(in_features=4096, out_features=1000, bias=True)
    (linear4): Linear(in_features=1000, out_features=1, bias=True)
  )
100%|██████████| 34/34 [00:08<00:00,  3.87it/s]
Epoch : 1, train loss : 85.93445587158203

Epoch : 1, val loss : 94.56427764892578
100%|██████████| 34/34 [00:09<00:00,  3.77it/s]
Epoch : 2, train loss : 15.797348976135254

Epoch : 2, val loss : 22.81747055053711
100%|██████████| 34/34 [00:08<00:00,  3.91it/s]
Epoch : 3, train loss : 16.885482788085938

Epoch : 3, val loss : 14.261885643005371
100%|██████████| 34/34 [00:08<00:00,  3.86it/s]
Epoch : 4, train loss : 4.673673629760742

Epoch : 4, val loss : 5.147424697875977
100%|██████████| 34/34 [00:08<00:00,  3.86it/s]
Epoch : 5, train loss : 2.657351493835449

Epoch : 5, val loss : 0.902996301651001
100%|██████████| 34/34 [00:08<00:00,  3.85it/s]
Epoch : 6, train loss : 1.8083627223968506

Epoch : 6, val loss : 0.43735912442207336
100%|██████████| 34/34 [00:09<00:00,  3.70it/s]
Epoch : 7, train loss : 1.251448154449463

Epoch : 7, val loss : 1.0923584699630737
100%|██████████| 34/34 [00:08<00:00,  3.81it/s]
Epoch : 8, train loss : 1.2339991331100464
```

Part III: ResNet [6 marks]

(1) Implement the following pretrained ResNet variants

1. ResNet18
2. ResNet50
3. ResNet152

You can refer the ResNet documentation in the RESOURCES tab.

```
model_resnet_variant = torch.hub.load('pytorch/vision:v0.10.0', 'resnet18', pretrained=True)
```

```
'''
DO NOT ALTER THE FOLLOWING CODE
```

```
'''
#freeze all the model parameters
for params in model_resnet_variant.parameters():
  params.requires_grad_ = False
'''
DO NOT ALTER THE ABOVE CODE
'''

nr_filters = 512 #ENTER CODE HERE
model_resnet_variant.fc = nn.Linear(nr_filters, 1)
model_resnet_variant = model_resnet_variant.to(device)

trained_model = calc_loss(model_resnet_variant, n_epochs)
calc_accuracy(trained_model)
```

```
Epoch : 39, val loss : 0.039809197187423706
100%|██████████| 34/34 [00:06<00:00,  5.62it/s]
Epoch : 40, train loss : 0.09467177838087082

Epoch : 40, val loss : 0.2055031806230545
100%|██████████| 34/34 [00:04<00:00,  8.10it/s]
Epoch : 41, train loss : 0.22025780379772186

Epoch : 41, val loss : 0.14168646931648254
100%|██████████| 34/34 [00:04<00:00,  8.27it/s]
Epoch : 42, train loss : 0.08100398629903793

Epoch : 42, val loss : 0.10083577036857605
100%|██████████| 34/34 [00:04<00:00,  8.44it/s]
Epoch : 43, train loss : 0.010623877868056297

Epoch : 43, val loss : 0.011014021001756191
100%|██████████| 34/34 [00:04<00:00,  8.28it/s]
Epoch : 44, train loss : 0.006972185336053371

Epoch : 44, val loss : 0.0010603333357721567
100%|██████████| 34/34 [00:04<00:00,  8.13it/s]
Epoch : 45, train loss : 0.007095981389284134

Epoch : 45, val loss : 0.01478629931807518
100%|██████████| 34/34 [00:04<00:00,  8.29it/s]
Epoch : 46, train loss : 0.0014366944087669253

Epoch : 46, val loss : 0.0023078268859535456
100%|██████████| 34/34 [00:04<00:00,  8.19it/s]
Epoch : 47, train loss : 0.0024302673991769552

Epoch : 47, val loss : 0.00036070041824132204
100%|██████████| 34/34 [00:04<00:00,  8.29it/s]
Epoch : 48, train loss : 0.0016170800663530827

Epoch : 48, val loss : 0.0002222525654360652
100%|██████████| 34/34 [00:05<00:00,  5.77it/s]
Epoch : 49, train loss : 0.019875988364219666
```

```
Epoch : 49, val loss : 0.19051824510097504
100%|██████████| 34/34 [00:03<00:00,  8.55it/s]
Epoch : 50, train loss : 0.009608306922018528

Epoch : 50, val loss : 0.0010098920902237296
the accuracy is 0.7394957983193278
              precision    recall  f1-score   support

      Adults       0.70      0.83      0.76        59
        Kids       0.80      0.65      0.72        60

    accuracy                           0.74       119
   macro avg       0.75      0.74      0.74       119
weighted avg       0.75      0.74      0.74       119

the balanced accuracy is 0.7402542372881356
0.7394957983193278
```

```
model_resnet_variant = torch.hub.load('pytorch/vision:v0.10.0', 'resnet50', pretrained=True)

'''
DO NOT ALTER THE FOLLOWING CODE
'''
#freeze all the model parameters
for params in model_resnet_variant.parameters():
  params.requires_grad_ = False
'''
DO NOT ALTER THE ABOVE CODE
'''

nr_filters = 2048 #ENTER CODE HERE
model_resnet_variant.fc = nn.Linear(nr_filters, 1)
model_resnet_variant = model_resnet_variant.to(device)

trained_model = calc_loss(model_resnet_variant, n_epochs)
calc_accuracy(trained_model)
```

```
Epoch : 39, val loss : 0.10347715020179749
100%|██████████| 34/34 [00:07<00:00,  4.73it/s]
Epoch : 40, train loss : 0.1567973792552948

Epoch : 40, val loss : 0.06479958444833755
100%|██████████| 34/34 [00:07<00:00,  4.77it/s]
Epoch : 41, train loss : 0.12819069623947144

Epoch : 41, val loss : 0.12836702167987823
100%|██████████| 34/34 [00:08<00:00,  4.20it/s]
Epoch : 42, train loss : 0.21564744412899017

Epoch : 42, val loss : 0.11000697314739227
100%|██████████| 34/34 [00:07<00:00,  4.62it/s]
Epoch : 43, train loss : 0.16612930595874786
```

```
Epoch : 43, val loss : 0.06829213351011276
100%|██████████| 34/34 [00:07<00:00,  4.71it/s]
Epoch : 44, train loss : 0.1925380975008011

Epoch : 44, val loss : 0.02443777024745941
100%|██████████| 34/34 [00:07<00:00,  4.63it/s]
Epoch : 45, train loss : 0.12095416337251663

Epoch : 45, val loss : 0.10721816122531891
100%|██████████| 34/34 [00:07<00:00,  4.78it/s]
Epoch : 46, train loss : 0.11778729408979416


Epoch : 46, val loss : 0.26210179924964905
100%|██████████| 34/34 [00:07<00:00,  4.81it/s]
Epoch : 47, train loss : 0.050890546292066574

Epoch : 47, val loss : 0.040519796311855316
100%|██████████| 34/34 [00:08<00:00,  3.89it/s]
Epoch : 48, train loss : 0.024928836151957512

Epoch : 48, val loss : 0.007248141802847385
100%|██████████| 34/34 [00:06<00:00,  4.86it/s]
Epoch : 49, train loss : 0.09164624661207199

Epoch : 49, val loss : 0.018315207213163376
100%|██████████| 34/34 [00:07<00:00,  4.73it/s]
Epoch : 50, train loss : 0.038136158138513565

Epoch : 50, val loss : 0.00500827981159091
the accuracy is 0.6722689075630253
              precision    recall  f1-score   support

      Adults       0.66      0.71      0.68        59
        Kids       0.69      0.63      0.66        60

    accuracy                           0.67       119
   macro avg       0.67      0.67      0.67       119
weighted avg       0.67      0.67      0.67       119

the balanced accuracy is 0.6725988700564971
0.6722689075630253
```

```
model_resnet_variant = torch.hub.load('pytorch/vision:v0.10.0', 'resnet152', pretrained=True)

'''
DO NOT ALTER THE FOLLOWING CODE
'''
#freeze all the model parameters
for params in model_resnet_variant.parameters():
  params.requires_grad_ = False
'''
DO NOT ALTER THE ABOVE CODE
'''
```

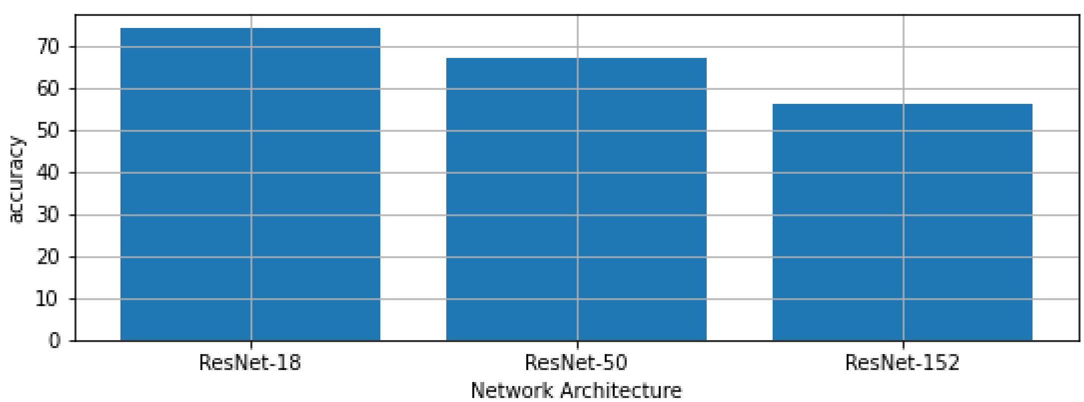
```
nr_filters = 2048 #ENTER CODE HERE
model_resnet_variant.fc = nn.Linear(nr_filters, 1)
model_resnet_variant = model_resnet_variant.to(device)

trained_model = calc_loss(model_resnet_variant, n_epochs)
calc_accuracy(trained_model)
```

```
Epoch : 39, val loss : 0.40233615040779114
100%|██████████| 34/34 [00:14<00:00,  2.40it/s]
Epoch : 40, train loss : 1.7383918762207031


Epoch : 40, val loss : 0.744766891002655
100%|██████████| 34/34 [00:14<00:00,  2.32it/s]
Epoch : 41, train loss : 1.3404788970947266

Epoch : 41, val loss : 0.8614333271980286
100%|██████████| 34/34 [00:14<00:00,  2.41it/s]
Epoch : 42, train loss : 0.7487322688102722

Epoch : 42, val loss : 0.72194504737854
100%|██████████| 34/34 [00:14<00:00,  2.40it/s]
Epoch : 43, train loss : 0.7156602740287781

Epoch : 43, val loss : 0.6688043475151062
100%|██████████| 34/34 [00:14<00:00,  2.35it/s]
Epoch : 44, train loss : 0.710350513458252

Epoch : 44, val loss : 0.735526442527771
100%|██████████| 34/34 [00:14<00:00,  2.41it/s]
Epoch : 45, train loss : 0.9298837780952454

Epoch : 45, val loss : 0.6714283227920532
100%|██████████| 34/34 [00:14<00:00,  2.35it/s]
Epoch : 46, train loss : 0.7130137085914612

Epoch : 46, val loss : 0.6446553468704224
100%|██████████| 34/34 [00:14<00:00,  2.35it/s]
Epoch : 47, train loss : 0.7108196020126343

Epoch : 47, val loss : 0.6738114953041077
100%|██████████| 34/34 [00:14<00:00,  2.41it/s]
Epoch : 48, train loss : 0.68343186378479

Epoch : 48, val loss : 0.6588366031646729
100%|██████████| 34/34 [00:14<00:00,  2.41it/s]
Epoch : 49, train loss : 0.6569996476173401

Epoch : 49, val loss : 0.626604437828064
100%|██████████| 34/34 [00:14<00:00,  2.37it/s]
Epoch : 50, train loss : 0.6405529379844666

Epoch : 50, val loss : 0.6039553880691528
the accuracy is 0.5630252100840336
              precision    recall  f1-score   support
```

```
         Adults       0.73      0.19      0.30        59
           Kids       0.54      0.93      0.68        60

       accuracy                           0.56       119
      macro avg       0.64      0.56      0.49       119
   weighted avg       0.64      0.56      0.49       119

the balanced accuracy is 0.5598870056497175
0.5630252100840336
```

(2) Which ResNet performs better? Justify.

```
#ENTER CODE HERE
names = ['ResNet-18', 'ResNet-50', 'ResNet-152']
values = [74.02, 67.26, 55.99]

plt.figure(figsize=(9, 3))
plt.bar(names, values)
plt.xlabel('Network Architecture')
plt.ylabel('accuracy')
plt.grid(True)
plt.show()
```



Your answer - Based on our results, ResNet-18 has outperformed the other two variants. Although it is true that the larger variants are more complicated and powerful, but freezing all weights and adding a linear function to fine-tune the model also needs a larger amount of data. Besides, the 50 and 152 variants have more weights compared to the 18 one, and this leads to overfitness on our small dataset.

Part IV: Class Imbalance and Sampling [5 marks]

```
main_path = '/content/drive/MyDrive/datasets/imbalanced_data/train' #ENTER PATH HERE
test_path = '/content/drive/MyDrive/datasets/imbalanced_data/test' #ENTER PATH HERE
```

```
'''
DO NOT ALTER THE FOLLOWING CODE
'''
'''
NEW CHANGE on 16th Novemeber, 2022
Add the line below
'''
torch.cuda.empty_cache()
torch.manual_seed(0)
'''
Add the above line
'''
'''
DO NOT ALTER THE FOLLOWING CODE
'''
my_transforms = transforms.Compose([transforms.Resize((224,224)),   transforms.ToTensor(), tr
BATCH_SIZE = 16
IMAGE_SIZE = 32
NUM_CHANNELS = 3
n_epochs = 50 # the cnn will be trained for 50 epochs
dataset = datasets.ImageFolder(root=main_path, transform=my_transforms)
dataset_size = dataset.__len__() #compute the length of the training dataset
train_count = int(dataset_size * 0.8) #divide the training dataset to training and validation
val_count = dataset_size - train_count # keep the training proportion to 1 if no validation i
train_dataset, valid_dataset = data.random_split(dataset, [train_count, val_count]) #perform
y_train_indices = train_dataset.indices
y_train = [dataset.targets[i] for i in y_train_indices] #assign the labels or target variable
test_data = datasets.ImageFolder(test_path, transform=my_transforms)
'''
Following train, validation and test dataloaders will also be used in Part III: Resnets
'''
'''
NEW CHANGE on 16th Novemeber, 2022
'''
train_dataloader = DataLoader(train_dataset, batch_size=BATCH_SIZE, num_workers=2, shuffle=Tr
valid_dataloader = DataLoader(valid_dataset, batch_size=BATCH_SIZE, num_workers=2, )
test_dataloader = torch.utils.data.DataLoader(test_data, batch_size=BATCH_SIZE, )
'''
NEW CHANGE on 16th Novemeber, 2022
'''
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu') #check for gpu
print('Using ',device,'for model training') #print the device status
```

```
Using  cuda:0 for model training
```

```
'''
DO NOT ALTER THE FOLLOWING CODE
'''
def train_model(model,n_epochs, loss_fn, train_step):
```

```
    '''
    This is the main function which is used to train the model, update weights, calculate loss
    '''
    train_losses = []
    val_losses = []
    epoch_train_losses = []
    epoch_val_losses = []
    for epoch in range(n_epochs):
      epoch_loss = 0
      for i ,data in tqdm(enumerate(train_dataloader), total = len(train_dataloader)): #iterate
        x_batch , y_batch = data
        x_batch = x_batch.to('cuda') #move to gpu
        y_batch = y_batch.unsqueeze(1).float() #convert target to same nn output shape
        y_batch = y_batch.to('cuda') #move to gpu
        loss = train_step(x_batch, y_batch)
        epoch_loss += loss/len(train_dataloader)
        train_losses.append(loss.cpu().detach().numpy())
      epoch_train_losses.append(epoch_loss)
      print('\nEpoch : {}, train loss : {}'.format(epoch+1,epoch_loss))
      #validation does not require gradient
      with torch.no_grad():
        cum_loss = 0
        for x_batch, y_batch in valid_dataloader:
          x_batch = x_batch.to('cuda')
          y_batch = y_batch.unsqueeze(1).float() #convert target to same nn output shape
          y_batch = y_batch.to('cuda')
          model.eval()#model to eval mode
          yhat = model(x_batch)
          val_loss = loss_fn(yhat,y_batch)
          cum_loss += loss/len(valid_dataloader)
          val_losses.append(val_loss.item())
        epoch_val_losses.append(cum_loss)
        print('Epoch : {}, val loss : {}'.format(epoch+1,cum_loss))
        best_loss = min(epoch_val_losses)
        #save best model
        if cum_loss <= best_loss:
          best_model_wts = model.state_dict()
    #load best model
    model.load_state_dict(best_model_wts)
    return model, train_losses,val_losses

  def plot_losses(train_losses,val_losses):
    '''
    This function can be used to plot the training and validation losses. You can use this
    function to analyse the losses and judge if model was overfitting or if model shows some
    unusual behaviour.
    '''
    plt.plot(train_losses, label='Training loss')
    plt.plot(val_losses, label='Validation loss')
    plt.legend()
    plt.show()
```

```
def inference(model,test_data):
  '''
  As we are doing binary classification, this function uses sigmoid to change class probabili
  to either 0 or 1 class.
  '''
  y_pred = []
  y_true = []
  for idx in range(1, len(test_data)):
    y_true.append( test_data[idx][1])
    sample = torch.unsqueeze(test_data[idx][0], dim=0).to('cuda')
    if torch.sigmoid(model(sample)) < 0.5:
      y_pred.append(0)
    else:
      y_pred.append(1)
  return y_pred, y_true

def calc_loss(model, n_epochs):
  '''
  This function drives the training function, assigns the loss fuction and sets the optimiize
  '''
  loss_fn = BCEWithLogitsLoss()
  optimizer = torch.optim.Adam(model.parameters())
  train_step = make_train_step(model, optimizer, loss_fn)
  trained_model, train_losses, val_losses = train_model(model,n_epochs, loss_fn, train_step)
  return trained_model

def calc_accuracy(trained_model):
  '''
  This function is used for returning the calculated accuracies.
  '''
  y_pred, y_true = inference(trained_model,test_data)
  target_names = ['Adults', 'Kids']
  print('the accuracy is',accuracy_score(y_true, y_pred))
  print(classification_report(y_true, y_pred, target_names=target_names))
  print('the balanced accuracy is',balanced_accuracy_score(y_true, y_pred))
  return accuracy_score(y_true, y_pred)
```

```
from torch.nn.modules.conv import ConvTranspose2d, Conv2d
class Net(Module):
    def __init__(self):
        super(Net, self).__init__()

        self.cnn_layers = Sequential(
            # Defining a 2D convolution layer
            Conv2d(NUM_CHANNELS, IMAGE_SIZE, kernel_size=3, stride=1, padding=1),
            BatchNorm2d(IMAGE_SIZE),
            nn.ReLU(inplace=True),
            MaxPool2d(kernel_size=2, stride=2),
            # Defining another 2D convolution layer
            Conv2d(32, 32, kernel_size=3, stride=1, padding=1),
```

```
            BatchNorm2d(32),
            nn.ReLU(inplace=True),
            MaxPool2d(kernel_size=2, stride=2)
        )

        self.linear_layers = Sequential(
            Linear(100352, 1)
        )

    def forward(self, x):

        x = self.cnn_layers(x)
        x = x.view(x.size(0), -1)
        x = self.linear_layers(x)
        return x
'''
DO NOT ALTER THE FOLLOWING CODE
'''
model = Net()
optimizer = Adam(model.parameters(), lr=0.07)
criterion = CrossEntropyLoss()
if torch.cuda.is_available():
    model = model.cuda()
    criterion = criterion.cuda()
print(model)
trained_model = calc_loss(model, n_epochs) #train the model
calc_accuracy(trained_model) # report the accuracy
```

```
Epoch : 39, val loss : 0.0
100%|██████████| 22/22 [00:01<00:00, 11.59it/s]
Epoch : 40, train loss : 0.005897128023207188

Epoch : 40, val loss : 0.0
100%|██████████| 22/22 [00:01<00:00, 11.44it/s]
Epoch : 41, train loss : 0.0018665837123990059

Epoch : 41, val loss : 0.0
100%|██████████| 22/22 [00:01<00:00, 11.61it/s]
Epoch : 42, train loss : 0.0008140868740156293

Epoch : 42, val loss : 0.0
100%|██████████| 22/22 [00:01<00:00, 11.67it/s]
Epoch : 43, train loss : 0.0002240426401840523

Epoch : 43, val loss : 0.0
100%|██████████| 22/22 [00:01<00:00, 11.32it/s]
Epoch : 44, train loss : 5.551737103814958e-06

Epoch : 44, val loss : 0.0
100%|██████████| 22/22 [00:01<00:00, 11.47it/s]
Epoch : 45, train loss : 1.2433804840839002e-05


Epoch : 45, val loss : 0.0
100%|██████████| 22/22 [00:01<00:00, 11.33it/s]
Epoch : 46, train loss : 0.0001019297633320093
```

```
Epoch : 46, train loss : 0.00010192976333200932

Epoch : 46, val loss : 0.0
100%|██████████| 22/22 [00:01<00:00, 11.25it/s]
Epoch : 47, train loss : 6.71055204293225e-06

Epoch : 47, val loss : 5.709799734177068e-05
100%|██████████| 22/22 [00:01<00:00, 11.64it/s]
Epoch : 48, train loss : 1.0027823009295389e-05

Epoch : 48, val loss : 0.0
100%|██████████| 22/22 [00:01<00:00, 11.61it/s]
Epoch : 49, train loss : 4.78048332297476e-06

Epoch : 49, val loss : 0.0
100%|██████████| 22/22 [00:01<00:00, 11.49it/s]
Epoch : 50, train loss : 4.020795131509658e-06

Epoch : 50, val loss : 0.0
the accuracy is 0.5546218487394958
              precision    recall  f1-score   support

      Adults       0.53      0.97      0.68        59
        Kids       0.82      0.15      0.25        60

    accuracy                           0.55       119
   macro avg       0.67      0.56      0.47       119
weighted avg       0.67      0.55      0.47       119

the balanced accuracy is 0.5580508474576271
0.5546218487394958
```

```
df = pd.DataFrame(y_train)
a = df.value_counts()


weights = []
for i in range(a.size):
  weights.append(a[i]/len(y_train))

weights.reverse()


weights_array = torch.from_numpy(np.array([weights[i] for i in y_train]))
sampler_W = WeightedRandomSampler(weights_array.type('torch.DoubleTensor'), len(y_train), rep


'''
DO NOT ALTER THE FOLLOWING CODE
'''
'''
NEW CHANGE on 16th Novemeber, 2022
Add the line below
'''
```

```
torch.cuda.empty_cache()
torch.manual_seed(0)
'''
Add the above line
'''
'''
DO NOT ALTER THE FOLLOWING CODE
'''
my_transforms = transforms.Compose([transforms.Resize((224,224)),   transforms.ToTensor(), tr
BATCH_SIZE = 16
IMAGE_SIZE = 32
NUM_CHANNELS = 3
n_epochs = 50 # the cnn will be trained for 50 epochs
dataset = datasets.ImageFolder(root=main_path, transform=my_transforms)
dataset_size = dataset.__len__() #compute the length of the training dataset
train_count = int(dataset_size * 0.8) #divide the training dataset to training and validation
val_count = dataset_size - train_count # keep the training proportion to 1 if no validation i
train_dataset, valid_dataset = data.random_split(dataset, [train_count, val_count]) #perform
y_train_indices = train_dataset.indices
y_train = [dataset.targets[i] for i in y_train_indices] #assign the labels or target variable
test_data = datasets.ImageFolder(test_path, transform=my_transforms)
'''
Following train, validation and test dataloaders will also be used in Part III: Resnets
'''
'''
NEW CHANGE on 16th Novemeber, 2022
'''
train_dataloader = DataLoader(train_dataset, batch_size=BATCH_SIZE, num_workers=2, sampler =
valid_dataloader = DataLoader(valid_dataset, batch_size=BATCH_SIZE, num_workers=2)
test_dataloader = torch.utils.data.DataLoader(test_data, batch_size=BATCH_SIZE, )
'''
NEW CHANGE on 16th Novemeber, 2022
'''
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu') #check for gpu
print('Using ',device,'for model training') #print the device status
```

```
Using  cuda:0 for model training
```

```
'''
DO NOT ALTER THE FOLLOWING CODE
'''
def train_model(model,n_epochs, loss_fn, train_step):
  '''
  This is the main function which is used to train the model, update weights, calculate loss
  '''
  train_losses = []
  val_losses = []
  epoch_train_losses = []
  epoch_val_losses = []
  for epoch in range(n_epochs):
```

```
    epoch_loss = 0
    for i ,data in tqdm(enumerate(train_dataloader), total = len(train_dataloader)): #iterate
      x_batch , y_batch = data
      x_batch = x_batch.to('cuda') #move to gpu
      y_batch = y_batch.unsqueeze(1).float() #convert target to same nn output shape
      y_batch = y_batch.to('cuda') #move to gpu
      loss = train_step(x_batch, y_batch)
      epoch_loss += loss/len(train_dataloader)
      train_losses.append(loss.cpu().detach().numpy())
    epoch_train_losses.append(epoch_loss)
    print('\nEpoch : {}, train loss : {}'.format(epoch+1,epoch_loss))
    #validation does not require gradient
    with torch.no_grad():
      cum_loss = 0
      for x_batch, y_batch in valid_dataloader:
        x_batch = x_batch.to('cuda')
        y_batch = y_batch.unsqueeze(1).float() #convert target to same nn output shape
        y_batch = y_batch.to('cuda')
        model.eval()#model to eval mode
        yhat = model(x_batch)
        val_loss = loss_fn(yhat,y_batch)
        cum_loss += loss/len(valid_dataloader)
        val_losses.append(val_loss.item())
      epoch_val_losses.append(cum_loss)
      print('Epoch : {}, val loss : {}'.format(epoch+1,cum_loss))
      best_loss = min(epoch_val_losses)
      #save best model
      if cum_loss <= best_loss:
        best_model_wts = model.state_dict()
  #load best model
  model.load_state_dict(best_model_wts)
  return model, train_losses,val_losses

def plot_losses(train_losses,val_losses):
  '''
  This function can be used to plot the training and validation losses. You can use this
  function to analyse the losses and judge if model was overfitting or if model shows some
  unusual behaviour.
  '''
  plt.plot(train_losses, label='Training loss')
  plt.plot(val_losses, label='Validation loss')
  plt.legend()
  plt.show()

def inference(model,test_data):
  '''
  As we are doing binary classification, this function uses sigmoid to change class probabili
  to either 0 or 1 class.
  '''
  y_pred = []
  y_true = []
```

```
    for idx in range(1, len(test_data)):
      y_true.append( test_data[idx][1])
      sample = torch.unsqueeze(test_data[idx][0], dim=0).to('cuda')
      if torch.sigmoid(model(sample)) < 0.5:
        y_pred.append(0)
      else:
        y_pred.append(1)
    return y_pred, y_true

  def calc_loss(model, n_epochs):
    '''
    This function drives the training function, assigns the loss fuction and sets the optimiize
    '''
    loss_fn = BCEWithLogitsLoss()
    optimizer = torch.optim.Adam(model.parameters())
    train_step = make_train_step(model, optimizer, loss_fn)
    trained_model, train_losses, val_losses = train_model(model,n_epochs, loss_fn, train_step)
    return trained_model

  def calc_accuracy(trained_model):
    '''
    This function is used for returning the calculated accuracies.
    '''
    y_pred, y_true = inference(trained_model,test_data)
    target_names = ['Adults', 'Kids']
    print('the accuracy is',accuracy_score(y_true, y_pred))
    print(classification_report(y_true, y_pred, target_names=target_names))
    print('the imbalanced accuracy is',balanced_accuracy_score(y_true, y_pred))
    return accuracy_score(y_true, y_pred)


  from torch.nn.modules.conv import ConvTranspose2d, Conv2d
  class Net(Module):
      def __init__(self):
          super(Net, self).__init__()

          self.cnn_layers = Sequential(
              # Defining a 2D convolution layer
              Conv2d(NUM_CHANNELS, IMAGE_SIZE, kernel_size=3, stride=1, padding=1),
              BatchNorm2d(IMAGE_SIZE),
              nn.ReLU(inplace=True),
              MaxPool2d(kernel_size=2, stride=2),
              # Defining another 2D convolution layer
              Conv2d(32, 32, kernel_size=3, stride=1, padding=1),
              BatchNorm2d(32),
              nn.ReLU(inplace=True),
              MaxPool2d(kernel_size=2, stride=2)
          )

          self.linear_layers = Sequential(
              Linear(100352, 1)
          )
```

```
    def forward(self, x):

        x = self.cnn_layers(x)
        x = x.view(x.size(0), -1)
        x = self.linear_layers(x)
        return x
'''
DO NOT ALTER THE FOLLOWING CODE
'''
model = Net()
optimizer = Adam(model.parameters(), lr=0.07)
criterion = CrossEntropyLoss()
if torch.cuda.is_available():
    model = model.cuda()
    criterion = criterion.cuda()
print(model)
trained_model = calc_loss(model, n_epochs) #train the model
calc_accuracy(trained_model) # report the accuracy
```

```
Epoch : 39, val loss : 0.0
100%|██████████| 22/22 [00:01<00:00, 11.61it/s]
Epoch : 40, train loss : 4.0303816604136955e-06

Epoch : 40, val loss : 0.0
100%|██████████| 22/22 [00:01<00:00, 12.07it/s]
Epoch : 41, train loss : 2.6082056137965992e-05

Epoch : 41, val loss : 0.0
100%|██████████| 22/22 [00:01<00:00, 11.17it/s]
Epoch : 42, train loss : 5.2316886467451695e-06

Epoch : 42, val loss : 0.0
100%|██████████| 22/22 [00:01<00:00, 11.50it/s]
Epoch : 43, train loss : 1.5208518107101554e-06

Epoch : 43, val loss : 0.0
100%|██████████| 22/22 [00:01<00:00, 11.45it/s]
Epoch : 44, train loss : 1.25706692415406e-05

Epoch : 44, val loss : 0.0
100%|██████████| 22/22 [00:01<00:00, 11.69it/s]
Epoch : 45, train loss : 6.637381375185214e-06

Epoch : 45, val loss : 0.0
100%|██████████| 22/22 [00:01<00:00, 11.56it/s]
Epoch : 46, train loss : 4.12832423535292e-06

Epoch : 46, val loss : 0.0
100%|██████████| 22/22 [00:01<00:00, 11.62it/s]
Epoch : 47, train loss : 4.861735305894399e-06


Epoch : 47, val loss : 0.0
100%|██████████| 22/22 [00:01<00:00, 11.53it/s]
Epoch : 48, train loss : 1.197456754198356e-06
```

```
Epoch : 48, val loss : 2.9802313861182483e-07
100%|██████████| 22/22 [00:01<00:00, 11.69it/s]
Epoch : 49, train loss : 1.090073783416301e-05

Epoch : 49, val loss : 0.0
100%|██████████| 22/22 [00:01<00:00, 11.79it/s]
Epoch : 50, train loss : 6.237190063984599e-06

Epoch : 50, val loss : 0.00010364174522692338
the accuracy is 0.5882352941176471
              precision    recall  f1-score   support

      Adults       0.55      0.98      0.70        59
        Kids       0.92      0.20      0.33        60

    accuracy                           0.59       119
   macro avg       0.74      0.59      0.52       119
weighted avg       0.74      0.59      0.51       119

the imbalanced accuracy is 0.5915254237288136
0.5882352941176471
```

Consider the imbalanced data and run the following CNN with and without Weighted Random Sampler

```
class Net(Module):
    def __init__(self):
        super(Net, self).__init__()

        self.cnn_layers = Sequential(
            # Defining a 2D convolution layer
            Conv2d(NUM_CHANNELS, IMAGE_SIZE, kernel_size=3, stride=1, padding=1),
            BatchNorm2d(IMAGE_SIZE),
            ReLU(inplace=True),
            MaxPool2d(kernel_size=2, stride=2),
            # Defining another 2D convolution layer
            Conv2d(32, 32, kernel_size=3, stride=1, padding=1),
            BatchNorm2d(32),
            ReLU(inplace=True),
            MaxPool2d(kernel_size=2, stride=2),
        )

        self.linear_layers = Sequential(
            Linear(100352, 1)
        )
```

```
    # Defining the forward pass
    def forward(self, x):
        x = self.cnn_layers(x)
        x = x.view(x.size(0), -1)
        x = self.linear_layers(x)
        #x = x.view(x.size(0), -1)

        return x
```

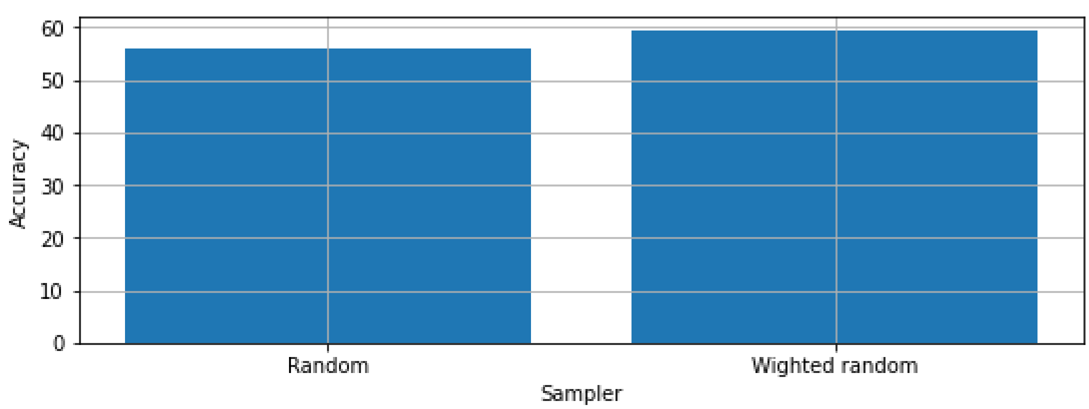
NOTE:

1. Change the main and test paths to the imbalanced dataset.
2. Sampler can be loaded to data loader as follows:

```
sampler_W = <ENTER CODE for weighted sampler>

train_dataloader = DataLoader(train_dataset, batch_size=BATCH_SIZE, num_workers=2, sampler = sampler_
valid_dataloader = DataLoader(valid_dataset, batch_size=BATCH_SIZE, num_workers=2)
```

Has the accuracy gone up or down? Why? Explain your answer.

```
#ENTER CODE HERE
names = ['Random', 'Wighted random']
values = [55.8, 59.15]

plt.figure(figsize=(9, 3))
plt.bar(names, values)
plt.xlabel('Sampler')
plt.ylabel('Accuracy')
plt.grid(True)
plt.show()
```



Answer - The accuracy has gone up. In cases where we have impbalanced datasets, random sampling leads to a network that has a bias towards the larger class. By using a wighted random sampler, we try to assign an equal probability to each data from any label. Hence, the network will be fed by the same amount of data from each class and won't be biased.

Part V: Data Augmentation{CMPUT 466 only}[5 marks]

1. **What is Data Augmentation? How does it help in combating the data imbalance issue?**

[ ] ↳ *10 cells hidden*