In [ ]:
```
import pandas as pd #For reading csv files.
import numpy as np
import matplotlib.pyplot as plt #For plotting.

import PIL.Image as Image #For working with image files.

#Importing torch
import torch
import torch.nn as nn
import torch.nn.functional as F

from torch.utils.data import Dataset,DataLoader #For working with data.

from torchvision import models,transforms
import os
```

In [ ]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu') #Use GPU if i
print(device) #
```

```
cuda
```

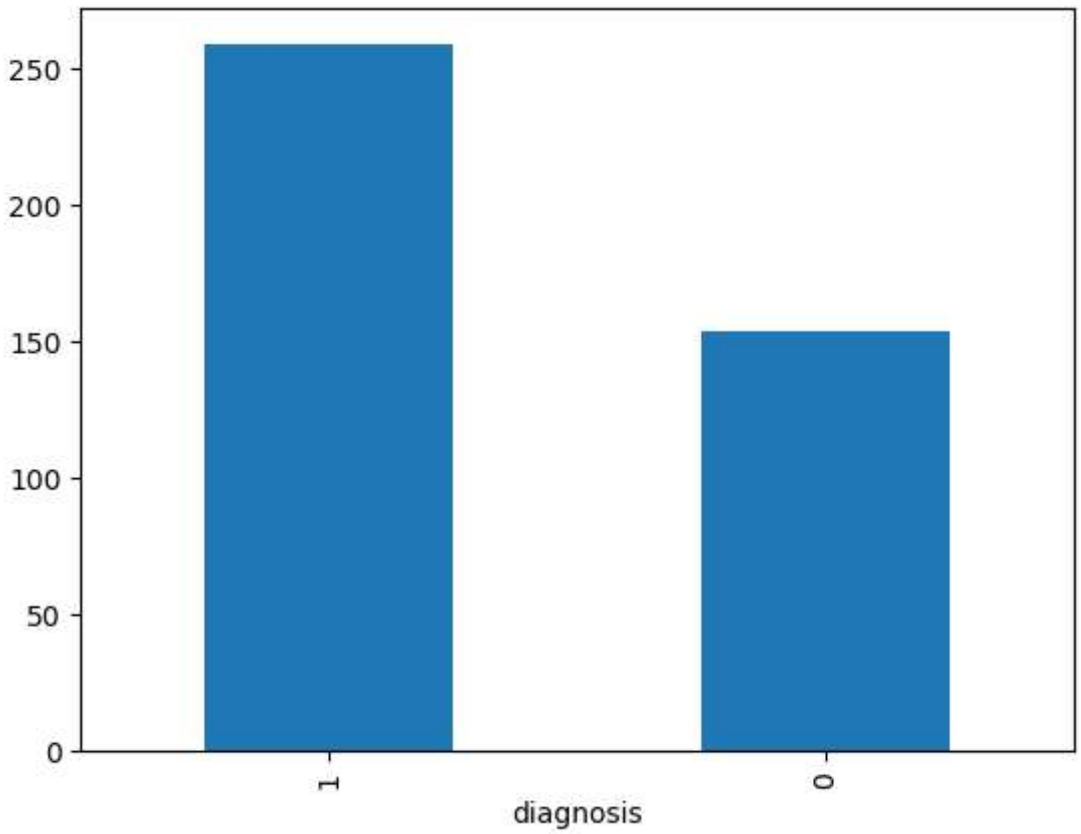
In [ ]:
```
base_path = "C:\\Users\\hsr30\\Documents\\minor-project\\datasets\\disease-grading-
```

In [ ]:
```
train_df = pd.read_csv(os.path.join(base_path,"groundtruths", "idrid-train.csv"))
print(f'No.of.training_samples: {len(train_df)}')

test_df = pd.read_csv(os.path.join(base_path,"groundtruths", "idrid-test.csv"))
print(f'No.of.testing_samples: {len(test_df)}')
```

```
No.of.training_samples: 413
No.of.testing_samples: 102
```

In [ ]:
```
train_df['diagnosis'].value_counts().plot(kind='bar')
```

Out[ ]:
```
<Axes: xlabel='diagnosis'>
```



Balancing the dataset

In [ ]:
```
from sklearn.utils import class_weight #For calculating weights for each class.
class_weights = class_weight.compute_class_weight(class_weight='balanced',classes=n
class_weights = torch.tensor(class_weights,dtype=torch.float).to(device)
print(class_weights)
```

```
tensor([1.3409, 0.7973], device='cuda:0')
```

Dataset

In [ ]:
```
class CustomDataset(Dataset): # Inherits from the Dataset class.
    '''
    dataset class overloads the __init__, __len__, __getitem__ methods of the Datas

    Attributes :
        df:  DataFrame object for the csv file.
        data_path: Location of the dataset.
        image_transform: Transformations to apply to the image.
        train: A boolean indicating whether it is a training_set or not.
    '''

    def __init__(self,df,data_path,image_transform=None,train=True): # Constructor.
        super(Dataset,self).__init__() #Calls the constructor of the Dataset class.
        self.df = df
        self.data_path = data_path
        self.image_transform = image_transform
        self.train = train
```

```
    def __len__(self):
        return len(self.df) #Returns the number of samples in the dataset.

    def __getitem__(self,index):
        image_id = self.df['Image name'][index]
        image = Image.open(f'{self.data_path}/{image_id}.png') #Image.
        if self.image_transform :
            image = self.image_transform(image) #Applies transformation to the imag

        if self.train :
            label = self.df['diagnosis'][index] #Label.
            return image,label #If train == True, return image & label.

        else:
            return image #If train != True, return image.
```

In [ ]:
```
image_transform = transforms.Compose([transforms.Resize([512,512]),
                                      transforms.ToTensor(),
                                      transforms.Normalize((0.485, 0.456, 0.406), (
```

In [ ]:
```
#Transformations to apply to the image.
data_set = CustomDataset(train_df,os.path.join(base_path, 'final-training'),image_t
print(f'No.of.samples in the dataset: {len(data_set)}')
#Split the data_set so that valid_set contains 0.1 samples of the data_set.
split_ratio = 0.2
train_num = int(len(data_set)*(1-split_ratio))
valid_num = len(data_set) - train_num
print(f'No.of.samples in the training_set: {train_num}')

train_set,valid_set = torch.utils.data.random_split(data_set,[train_num,valid_num])
```

```
No.of.samples in the dataset: 413
No.of.samples in the training_set: 330
```

In [ ]:
```
train_dataloader = DataLoader(train_set,batch_size=32,shuffle=True) #DataLoader for
valid_dataloader = DataLoader(valid_set,batch_size=8,shuffle=False)
```

In [ ]:
```
#Since we've less data, we'll use Transfer learning.
model = models.resnet34(weights=models.ResNet34_Weights.IMAGENET1K_V1) #Downloads t

#Replace the Final layer of pretrained resnet34 with 4 new layers.
model.fc = nn.Sequential(nn.Linear(512,256),
                         nn.ReLU(inplace=True),
                         nn.Linear(256,128),
                         nn.ReLU(inplace=True),
                         nn.Linear(128,64),
                         nn.ReLU(inplace=True),
                         nn.Linear(64,2),
                     )
```

In [ ]:
```
model = model.to(device)
```

Training and Validations

Helper functions for training and validation

In [ ]:
```
def train(dataloader,model,loss_fn,optimizer):
    '''
    train function updates the weights of the model based on the
    loss using the optimizer in order to get a lower loss.

    Args :
         dataloader: Iterator for the batches in the data_set.
         model: Given an input produces an output by multiplying the input with the
         loss_fn: Calculates the discrepancy between the label & the model's predic
         optimizer: Updates the model weights.

    Returns :
         Average loss per batch which is calculated by dividing the losses for all
         with the number of batches.
    '''

    model.train() #Sets the model for training.

    total = 0
    correct = 0
    running_loss = 0

    for batch,(x,y) in enumerate(dataloader): #Iterates through the batches.

        output = model(x.to(device)) #model's predictions.
        loss   = loss_fn(output,y.to(device)) #loss calculation.

        running_loss += loss.item()

        total        += y.size(0)
        predictions   = output.argmax(dim=1).cpu().detach() #Index for the highest
        correct      += (predictions == y.cpu().detach()).sum().item() #No.of.cases

        optimizer.zero_grad() #Gradient values are set to zero.
        loss.backward() #Calculates the gradients.
        optimizer.step() #Updates the model weights.


    avg_loss = running_loss/len(dataloader) # Average loss for a single batch

    print(f'\nTraining Loss = {avg_loss:.6f}',end='\t')
    print(f'Accuracy on Training set = {100*(correct/total):.6f}% [{correct}/{total

    return avg_loss
```

In [ ]:
```
def validate(dataloader,model,loss_fn):
    '''
    validate function calculates the average loss per batch and the accuracy of the

    Args :
         dataloader: Iterator for the batches in the data_set.
         model: Given an input produces an output by multiplying the input with the
         loss_fn: Calculates the discrepancy between the label & the model's predic
```

```
    Returns :
         Average loss per batch which is calculated by dividing the losses for all
         with the number of batches.
    '''

    model.eval() #Sets the model for evaluation.

    total = 0
    correct = 0
    running_loss = 0

    with torch.no_grad(): #No need to calculate the gradients.

        for x,y in dataloader:

            output        = model(x.to(device)) #model's output.
            loss          = loss_fn(output,y.to(device)).item() #loss calculation.
            running_loss += loss

            total        += y.size(0)
            predictions   = output.argmax(dim=1).cpu().detach()
            correct      += (predictions == y.cpu().detach()).sum().item()

    avg_loss = running_loss/len(dataloader) #Average loss per batch.

    print(f'\nValidation Loss = {avg_loss:.6f}',end='\t')
    print(f'Accuracy on Validation set = {100*(correct/total):.6f}% [{correct}/{tot

    return avg_loss
```

Optimizing the model

In [ ]:
```
def optimize(train_dataloader,valid_dataloader,model,loss_fn,optimizer,nb_epochs):
    '''
    optimize function calls the train & validate functions for (nb_epochs) times.

    Args :
        train_dataloader: DataLoader for the train_set.
        valid_dataloader: DataLoader for the valid_set.
        model: Given an input produces an output by multiplying the input with the
        loss_fn: Calculates the discrepancy between the label & the model's predict
        optimizer: Updates the model weights.
        nb_epochs: Number of epochs.

    Returns :
        Tuple of lists containing losses for all the epochs.
    '''
    #Lists to store losses for all the epochs.
    train_losses = []
    valid_losses = []

    for epoch in range(nb_epochs):
        print(f'\nEpoch {epoch+1}/{nb_epochs}')
        print('------------------------------')
```

```
        train_loss = train(train_dataloader,model,loss_fn,optimizer) #Calls the tra
        train_losses.append(train_loss)
        valid_loss = validate(valid_dataloader,model,loss_fn) #Calls the validate f
        valid_losses.append(valid_loss)

    print('\nTraining has completed!')

    return train_losses,valid_losses
```

Train the model

In [ ]:
```
loss_fn   = nn.CrossEntropyLoss(weight=class_weights) #CrossEntropyLoss with class_
optimizer = torch.optim.SGD(model.parameters(),lr=0.001)
nb_epochs = 60
#Call the optimize function.
train_losses, valid_losses = optimize(train_dataloader,valid_dataloader,model,loss_
```

```
Epoch 1/60
-------------------------------

Training Loss = 0.692976        Accuracy on Training set = 38.181818% [126/330]

Validation Loss = 0.693450      Accuracy on Validation set = 36.144578% [30/83]

Epoch 2/60
-------------------------------

Training Loss = 0.692990        Accuracy on Training set = 38.181818% [126/330]

Validation Loss = 0.693504      Accuracy on Validation set = 34.939759% [29/83]

Epoch 3/60
-------------------------------

Training Loss = 0.694222        Accuracy on Training set = 39.090909% [129/330]

Validation Loss = 0.693797      Accuracy on Validation set = 37.349398% [31/83]

Epoch 4/60
-------------------------------

Training Loss = 0.692965        Accuracy on Training set = 38.484848% [127/330]

Validation Loss = 0.693939      Accuracy on Validation set = 37.349398% [31/83]

Epoch 5/60
-------------------------------

Training Loss = 0.691739        Accuracy on Training set = 38.787879% [128/330]

Validation Loss = 0.693139      Accuracy on Validation set = 37.349398% [31/83]

Epoch 6/60
-------------------------------

Training Loss = 0.691575        Accuracy on Training set = 40.909091% [135/330]

Validation Loss = 0.692573      Accuracy on Validation set = 38.554217% [32/83]

Epoch 7/60
-------------------------------

Training Loss = 0.691314        Accuracy on Training set = 41.212121% [136/330]

Validation Loss = 0.691925      Accuracy on Validation set = 40.963855% [34/83]

Epoch 8/60
-------------------------------

Training Loss = 0.691055        Accuracy on Training set = 42.727273% [141/330]

Validation Loss = 0.691459      Accuracy on Validation set = 43.373494% [36/83]
```

```
Epoch 9/60
-------------------------------

Training Loss = 0.690641        Accuracy on Training set = 48.484848% [160/330]

Validation Loss = 0.690916      Accuracy on Validation set = 45.783133% [38/83]

Epoch 10/60
-------------------------------

Training Loss = 0.690887        Accuracy on Training set = 46.969697% [155/330]

Validation Loss = 0.690465      Accuracy on Validation set = 50.602410% [42/83]

Epoch 11/60
-------------------------------

Training Loss = 0.689169        Accuracy on Training set = 54.242424% [179/330]

Validation Loss = 0.690154      Accuracy on Validation set = 51.807229% [43/83]

Epoch 12/60
-------------------------------

Training Loss = 0.689242        Accuracy on Training set = 56.969697% [188/330]

Validation Loss = 0.689753      Accuracy on Validation set = 55.421687% [46/83]

Epoch 13/60
-------------------------------

Training Loss = 0.689226        Accuracy on Training set = 60.000000% [198/330]

Validation Loss = 0.689447      Accuracy on Validation set = 61.445783% [51/83]

Epoch 14/60
-------------------------------

Training Loss = 0.688351        Accuracy on Training set = 61.515152% [203/330]

Validation Loss = 0.688960      Accuracy on Validation set = 60.240964% [50/83]

Epoch 15/60
-------------------------------

Training Loss = 0.688101        Accuracy on Training set = 63.030303% [208/330]

Validation Loss = 0.688580      Accuracy on Validation set = 67.469880% [56/83]

Epoch 16/60
-------------------------------

Training Loss = 0.687834        Accuracy on Training set = 64.545455% [213/330]

Validation Loss = 0.688525      Accuracy on Validation set = 66.265060% [55/83]
```

```
Epoch 17/60
-------------------------------

Training Loss = 0.687143        Accuracy on Training set = 64.848485% [214/330]

Validation Loss = 0.688068      Accuracy on Validation set = 66.265060% [55/83]

Epoch 18/60
-------------------------------

Training Loss = 0.687554        Accuracy on Training set = 66.363636% [219/330]

Validation Loss = 0.687639      Accuracy on Validation set = 67.469880% [56/83]

Epoch 19/60
-------------------------------

Training Loss = 0.686803        Accuracy on Training set = 68.484848% [226/330]

Validation Loss = 0.687192      Accuracy on Validation set = 67.469880% [56/83]

Epoch 20/60
-------------------------------

Training Loss = 0.686130        Accuracy on Training set = 70.303030% [232/330]

Validation Loss = 0.686863      Accuracy on Validation set = 67.469880% [56/83]

Epoch 21/60
-------------------------------

Training Loss = 0.685951        Accuracy on Training set = 71.818182% [237/330]

Validation Loss = 0.686062      Accuracy on Validation set = 67.469880% [56/83]

Epoch 22/60
-------------------------------

Training Loss = 0.686186        Accuracy on Training set = 69.696970% [230/330]

Validation Loss = 0.685975      Accuracy on Validation set = 68.674699% [57/83]

Epoch 23/60
-------------------------------

Training Loss = 0.685163        Accuracy on Training set = 72.424242% [239/330]

Validation Loss = 0.685845      Accuracy on Validation set = 69.879518% [58/83]

Epoch 24/60
-------------------------------

Training Loss = 0.684790        Accuracy on Training set = 70.909091% [234/330]

Validation Loss = 0.685380      Accuracy on Validation set = 69.879518% [58/83]
```

```
Epoch 25/60
-------------------------------

Training Loss = 0.684314        Accuracy on Training set = 71.515152% [236/330]

Validation Loss = 0.685110      Accuracy on Validation set = 69.879518% [58/83]

Epoch 26/60
-------------------------------

Training Loss = 0.683902        Accuracy on Training set = 73.030303% [241/330]

Validation Loss = 0.684695      Accuracy on Validation set = 68.674699% [57/83]

Epoch 27/60
-------------------------------

Training Loss = 0.683402        Accuracy on Training set = 73.939394% [244/330]

Validation Loss = 0.684142      Accuracy on Validation set = 72.289157% [60/83]

Epoch 28/60
-------------------------------

Training Loss = 0.683594        Accuracy on Training set = 72.424242% [239/330]

Validation Loss = 0.683733      Accuracy on Validation set = 72.289157% [60/83]

Epoch 29/60
-------------------------------

Training Loss = 0.682751        Accuracy on Training set = 73.636364% [243/330]

Validation Loss = 0.683236      Accuracy on Validation set = 73.493976% [61/83]

Epoch 30/60
-------------------------------

Training Loss = 0.682648        Accuracy on Training set = 74.545455% [246/330]

Validation Loss = 0.682972      Accuracy on Validation set = 73.493976% [61/83]

Epoch 31/60
-------------------------------

Training Loss = 0.682491        Accuracy on Training set = 76.666667% [253/330]

Validation Loss = 0.682704      Accuracy on Validation set = 74.698795% [62/83]

Epoch 32/60
-------------------------------

Training Loss = 0.681406        Accuracy on Training set = 74.242424% [245/330]

Validation Loss = 0.682191      Accuracy on Validation set = 74.698795% [62/83]
```

```
Epoch 33/60
-------------------------------

Training Loss = 0.681511        Accuracy on Training set = 74.848485% [247/330]

Validation Loss = 0.681543      Accuracy on Validation set = 74.698795% [62/83]

Epoch 34/60
-------------------------------

Training Loss = 0.680020        Accuracy on Training set = 73.636364% [243/330]

Validation Loss = 0.681099      Accuracy on Validation set = 74.698795% [62/83]

Epoch 35/60
-------------------------------

Training Loss = 0.679562        Accuracy on Training set = 73.636364% [243/330]

Validation Loss = 0.681203      Accuracy on Validation set = 73.493976% [61/83]

Epoch 36/60
-------------------------------

Training Loss = 0.679441        Accuracy on Training set = 75.151515% [248/330]

Validation Loss = 0.680538      Accuracy on Validation set = 73.493976% [61/83]

Epoch 37/60
-------------------------------

Training Loss = 0.679322        Accuracy on Training set = 75.454545% [249/330]

Validation Loss = 0.680153      Accuracy on Validation set = 73.493976% [61/83]

Epoch 38/60
-------------------------------

Training Loss = 0.678833        Accuracy on Training set = 75.151515% [248/330]

Validation Loss = 0.679370      Accuracy on Validation set = 74.698795% [62/83]

Epoch 39/60
-------------------------------

Training Loss = 0.680094        Accuracy on Training set = 74.848485% [247/330]

Validation Loss = 0.678707      Accuracy on Validation set = 73.493976% [61/83]

Epoch 40/60
-------------------------------

Training Loss = 0.677981        Accuracy on Training set = 76.060606% [251/330]

Validation Loss = 0.678642      Accuracy on Validation set = 74.698795% [62/83]
```

```
Epoch 41/60
-------------------------------

Training Loss = 0.677695        Accuracy on Training set = 75.757576% [250/330]

Validation Loss = 0.678288      Accuracy on Validation set = 74.698795% [62/83]

Epoch 42/60
-------------------------------

Training Loss = 0.676449        Accuracy on Training set = 76.060606% [251/330]

Validation Loss = 0.677460      Accuracy on Validation set = 74.698795% [62/83]

Epoch 43/60
-------------------------------

Training Loss = 0.675856        Accuracy on Training set = 76.363636% [252/330]

Validation Loss = 0.676933      Accuracy on Validation set = 74.698795% [62/83]

Epoch 44/60
-------------------------------

Training Loss = 0.675459        Accuracy on Training set = 76.363636% [252/330]

Validation Loss = 0.676264      Accuracy on Validation set = 74.698795% [62/83]

Epoch 45/60
-------------------------------

Training Loss = 0.674597        Accuracy on Training set = 74.242424% [245/330]

Validation Loss = 0.675960      Accuracy on Validation set = 74.698795% [62/83]

Epoch 46/60
-------------------------------

Training Loss = 0.673893        Accuracy on Training set = 75.151515% [248/330]

Validation Loss = 0.675249      Accuracy on Validation set = 74.698795% [62/83]

Epoch 47/60
-------------------------------

Training Loss = 0.673080        Accuracy on Training set = 73.939394% [244/330]

Validation Loss = 0.674557      Accuracy on Validation set = 74.698795% [62/83]

Epoch 48/60
-------------------------------

Training Loss = 0.672584        Accuracy on Training set = 74.848485% [247/330]

Validation Loss = 0.673949      Accuracy on Validation set = 74.698795% [62/83]
```

```
Epoch 49/60
-------------------------------

Training Loss = 0.672024        Accuracy on Training set = 74.848485% [247/330]

Validation Loss = 0.674085      Accuracy on Validation set = 74.698795% [62/83]

Epoch 50/60
-------------------------------

Training Loss = 0.671915        Accuracy on Training set = 75.151515% [248/330]

Validation Loss = 0.672969      Accuracy on Validation set = 75.903614% [63/83]

Epoch 51/60
-------------------------------

Training Loss = 0.670175        Accuracy on Training set = 75.454545% [249/330]

Validation Loss = 0.672477      Accuracy on Validation set = 75.903614% [63/83]

Epoch 52/60
-------------------------------

Training Loss = 0.669890        Accuracy on Training set = 76.666667% [253/330]

Validation Loss = 0.672149      Accuracy on Validation set = 75.903614% [63/83]

Epoch 53/60
-------------------------------

Training Loss = 0.669836        Accuracy on Training set = 74.545455% [246/330]

Validation Loss = 0.671213      Accuracy on Validation set = 75.903614% [63/83]

Epoch 54/60
-------------------------------

Training Loss = 0.669144        Accuracy on Training set = 73.636364% [243/330]

Validation Loss = 0.670880      Accuracy on Validation set = 74.698795% [62/83]

Epoch 55/60
-------------------------------

Training Loss = 0.668022        Accuracy on Training set = 75.151515% [248/330]

Validation Loss = 0.670617      Accuracy on Validation set = 74.698795% [62/83]

Epoch 56/60
-------------------------------

Training Loss = 0.667739        Accuracy on Training set = 73.939394% [244/330]

Validation Loss = 0.669946      Accuracy on Validation set = 74.698795% [62/83]
```

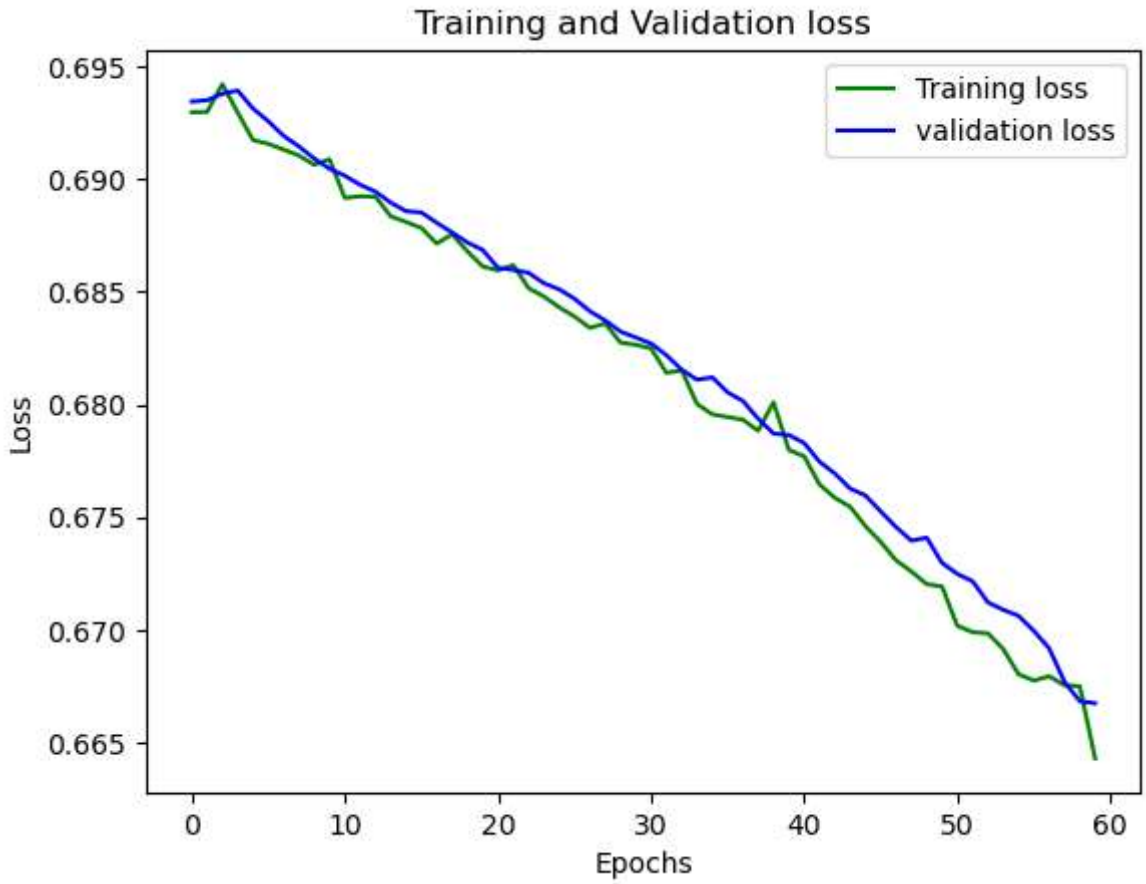
```
Epoch 57/60
-------------------------------

Training Loss = 0.667933        Accuracy on Training set = 76.363636% [252/330]

Validation Loss = 0.669173      Accuracy on Validation set = 74.698795% [62/83]

Epoch 58/60
-------------------------------

Training Loss = 0.667531        Accuracy on Training set = 73.939394% [244/330]

Validation Loss = 0.667685      Accuracy on Validation set = 74.698795% [62/83]

Epoch 59/60
-------------------------------

Training Loss = 0.667489        Accuracy on Training set = 75.454545% [249/330]

Validation Loss = 0.666824      Accuracy on Validation set = 77.108434% [64/83]

Epoch 60/60
-------------------------------

Training Loss = 0.664286        Accuracy on Training set = 75.151515% [248/330]

Validation Loss = 0.666732      Accuracy on Validation set = 75.903614% [63/83]

Training has completed!
```

Save the model

In [ ]:
```
# save , load model
base_path_model_save = "C:\\Users\\hsr30\\Documents\\minor-project\\disease-grading
num_model = len(os.listdir(base_path_model_save))
torch.save(model.state_dict(), os.path.join(base_path_model_save, f"{num_model}-res
```

Post training analysis

In [ ]:
```
#Plot the graph of train_losses & valid_losses against nb_epochs.
epochs = range(nb_epochs)
plt.plot(epochs, train_losses, 'g', label='Training loss')
plt.plot(epochs, valid_losses, 'b', label='validation loss')
plt.title('Training and Validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()
```



Testing

In [ ]:
```
def test(dataloader,model):
    '''
    test function predicts the labels given an image batches.

    Args :
         dataloader: DataLoader for the test_set.
         model: Given an input produces an output by multiplying the input with the

    Returns :
         List of predicted labels.
    '''

    model.eval() #Sets the model for evaluation.

    labels = [] #List to store the predicted labels.

    with torch.no_grad():

        for batch,x in enumerate(dataloader):

            output = model(x.to(device))

            predictions = output.argmax(dim=1).cpu().detach().tolist() #Predicted l
            labels.extend(predictions)
```

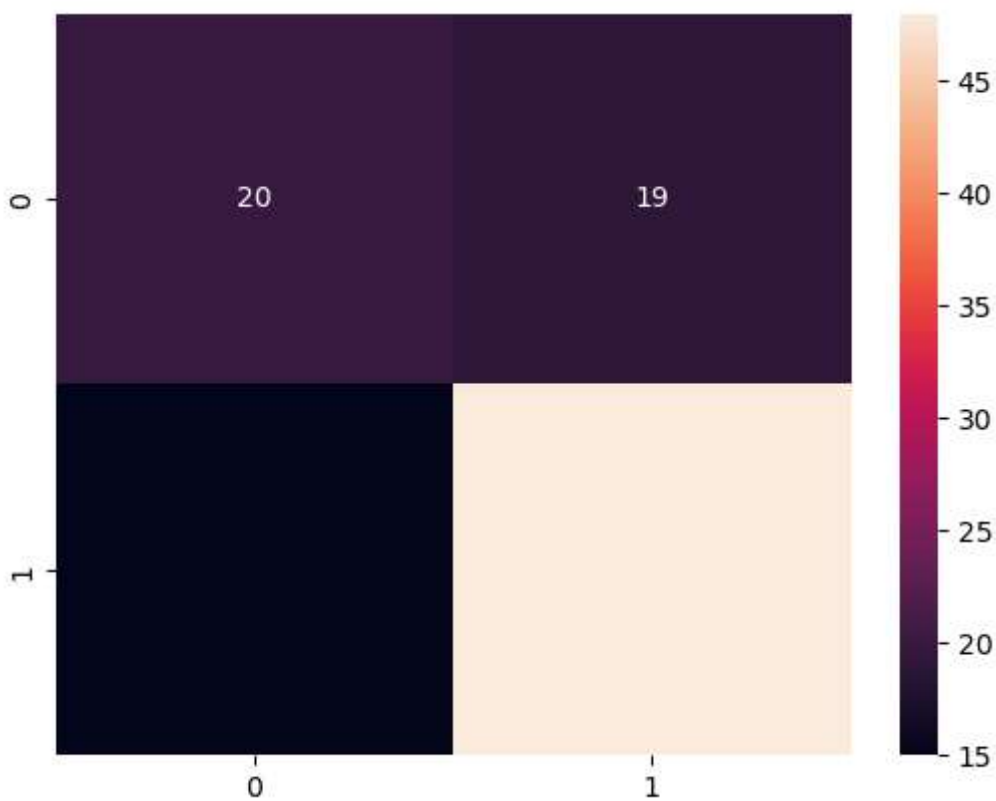
```
    print('Testing has completed')

    return labels
```

In [ ]:
```
test_set = CustomDataset(test_df,os.path.join(base_path, 'final-testing'),image_tra

test_dataloader = DataLoader(test_set, batch_size=16, shuffle=False) #DataLoader fo
```

In [ ]:
```
labels = test(test_dataloader,model)
# accuracy
correct = 0
total = 0
for i in range(len(labels)):
    if labels[i] == test_df['diagnosis'][i]:
        correct += 1
    total += 1
print(f'Accuracy on test set = {100*(correct/total):.6f}% [{correct}/{total}]') #Pr
```

```
Testing has completed
Accuracy on test set = 66.666667% [68/102]
```

In [ ]:
```
# confusion matrix
from sklearn.metrics import confusion_matrix
import seaborn as sns
cm = confusion_matrix(test_df['diagnosis'], labels)
print(cm)
sns.heatmap(cm, annot=True, fmt="d")
```

```
[[20 19]
 [15 48]]
```

Out[ ]: <Axes: >