Introduction: Glasses detection plays an important role in face recognition and soft biometrices for person identification. However, automatic glasses detection is still a challenging problem under real application scenarios, because face variations, light conditions, and self-occlusion, have significant influence on its performance.

Objective: Create a deep learning algorithm that can diffrentiate the people with eye glasses and without eye glasses on facial analysis

Outcomes of the Project

- There are two diffrent classes in the dataset, Try to display the images of each class.
- Use CNN for model building.
- Explain the parameters in CNN and how it will alter the model building.
- Draw inference on Image augmentation.
- Draw inference on epochs and batch sizes.
- CNN model should be able to classify the people with and without glasses.

In [3]:
```
conda install pytorch torchvision torchaudio -c pytorch
```

```
Retrieving notices: ...working... done
Collecting package metadata (current_repodata.json): ...working... done
Solving environment: ...working... done

# All requested packages already installed.


Note: you may need to restart the kernel to use updated packages.
```

In [4]:
```
import torch
import torch.nn as nn
import torchvision
import torch.nn.functional as F
import numpy as np
import pandas as pd
from torchvision import datasets, transforms, models, utils
from torch.utils.data import Dataset, DataLoader
```

```
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import os
device = "cuda" if torch.cuda.is_available() else "cpu"
print(device)
```

```
cpu
```

In [16]:
```
from zipfile import ZipFile
with ZipFile('glasses.zip', 'r') as zf:
    zf.extractall('data')
```

In [17]:
```
data_dir = 'data'
```

In [18]:
```
train_transform = torchvision.transforms.Compose([transforms.ToTensor(),
                                                  transforms.Resize((32,32)),
                                                  transforms.RandomCrop(32),
                                                 ])

train_dataset = torchvision.datasets.ImageFolder(data_dir, transform = train_transfo

trainloader = torch.utils.data.DataLoader(
    train_dataset, batch_size=64, shuffle = True
)

print((train_dataset[0][0].shape))

K = len(set(train_dataset.targets))
print(K)
```

```
torch.Size([3, 32, 32])
2
```

In [19]:
```
class CNN(nn.Module):
    def __init__(self, K):
        super(CNN, self).__init__()

        self.conv1 = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size = 3, padding = 1),
            nn.ReLU(),
            nn.BatchNorm2d(32),
            nn.MaxPool2d(2),
        )

        self.conv2 = nn.Sequential(
            nn.Conv2d(32, 64, kernel_size = 3, padding = 1),
            nn.ReLU(),
            nn.BatchNorm2d(64),
            nn.MaxPool2d(2),
        )

        self.conv3 = nn.Sequential(
            nn.Conv2d(64, 128, kernel_size = 3, padding = 1),
            nn.ReLU(),
            nn.BatchNorm2d(128),
            nn.MaxPool2d(2),
        )

        self.fc1 = nn.Linear(128 * 4 * 4, 1024)
        self.fc2 = nn.Linear(1024, 512)
        self.fc3 = nn.Linear(512, K)

    def forward(self, x):
        x = self.conv1(x)
```

```
        x = self.conv2(x)
        x = self.conv3(x)
        x = x.view(x.size(0), -1)
        x = F.dropout(x, p=0.12)
        x = F.relu(self.fc1(x))
        x = F.dropout(x, p=0.8)
        x = F.relu(self.fc2(x))
        x = F.dropout(x, p=0.6)
        x = self.fc3(x)
        return x
```

In [20]:
```
model = CNN(K)
```

In [21]:
```
model.to(device)
```

Out[21]:
```
CNN(
  (conv1): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv3): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc1): Linear(in_features=2048, out_features=1024, bias=True)
  (fc2): Linear(in_features=1024, out_features=512, bias=True)
  (fc3): Linear(in_features=512, out_features=2, bias=True)
)
```

In [22]:
```
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters())
```

In [23]:
```
def Glasses(model, criterion, optimizer, trainloader, epochs):
  train_losses = np.zeros(epochs)

  for i in range(epochs):
    train_loss = []
    for inputs, targets in trainloader:
      inputs, targets = inputs.to(device), targets.to(device)

      optimizer.zero_grad()

      outputs = model(inputs)
      loss = criterion(outputs, targets)

      loss.backward()
      optimizer.step()

      train_loss.append(loss.item())
```

```
    train_loss = np.mean(train_loss)

    train_losses[i] = train_loss

    print(f'Epoch:{i+1}/{epochs}, Train Loss: {train_loss:.4f}')

  return train_losses
```

In [36]:
```
train_losses = Glasses(model, criterion, optimizer, trainloader, epochs=40)
```

```
Epoch:1/40, Train Loss: 0.2904
Epoch:2/40, Train Loss: 0.3738
Epoch:3/40, Train Loss: 0.1444
Epoch:4/40, Train Loss: 0.2376
Epoch:5/40, Train Loss: 0.3745
Epoch:6/40, Train Loss: 0.3491
Epoch:7/40, Train Loss: 0.4123
Epoch:8/40, Train Loss: 0.2791
Epoch:9/40, Train Loss: 0.2895
Epoch:10/40, Train Loss: 0.3048
Epoch:11/40, Train Loss: 0.2535
Epoch:12/40, Train Loss: 0.3468
Epoch:13/40, Train Loss: 0.3194
Epoch:14/40, Train Loss: 0.2206
Epoch:15/40, Train Loss: 0.1679
Epoch:16/40, Train Loss: 0.4586
Epoch:17/40, Train Loss: 0.4838
Epoch:18/40, Train Loss: 0.0837
Epoch:19/40, Train Loss: 0.2209
Epoch:20/40, Train Loss: 0.2837
Epoch:21/40, Train Loss: 0.3755
Epoch:22/40, Train Loss: 0.2773
Epoch:23/40, Train Loss: 0.2852
Epoch:24/40, Train Loss: 0.2236
Epoch:25/40, Train Loss: 0.1788
Epoch:26/40, Train Loss: 0.2809
Epoch:27/40, Train Loss: 0.1307
Epoch:28/40, Train Loss: 0.2982
Epoch:29/40, Train Loss: 0.1910
Epoch:30/40, Train Loss: 0.2103
Epoch:31/40, Train Loss: 0.2967
Epoch:32/40, Train Loss: 0.2807
Epoch:33/40, Train Loss: 0.1209
Epoch:34/40, Train Loss: 0.4163
Epoch:35/40, Train Loss: 0.2700
Epoch:36/40, Train Loss: 0.2955
Epoch:37/40, Train Loss: 0.1669
Epoch:38/40, Train Loss: 0.3135
Epoch:39/40, Train Loss: 0.2756
Epoch:40/40, Train Loss: 0.2842
```

- We have increased the epoch size from 10 to 30 to improve the accuracy of the model. Now, let's see if this helps us or not in upcoming steps.

In [37]:
```
PATH = "Glasses.pt"

# save

torch.save(model, PATH)
```

In [38]:
```
# load
```

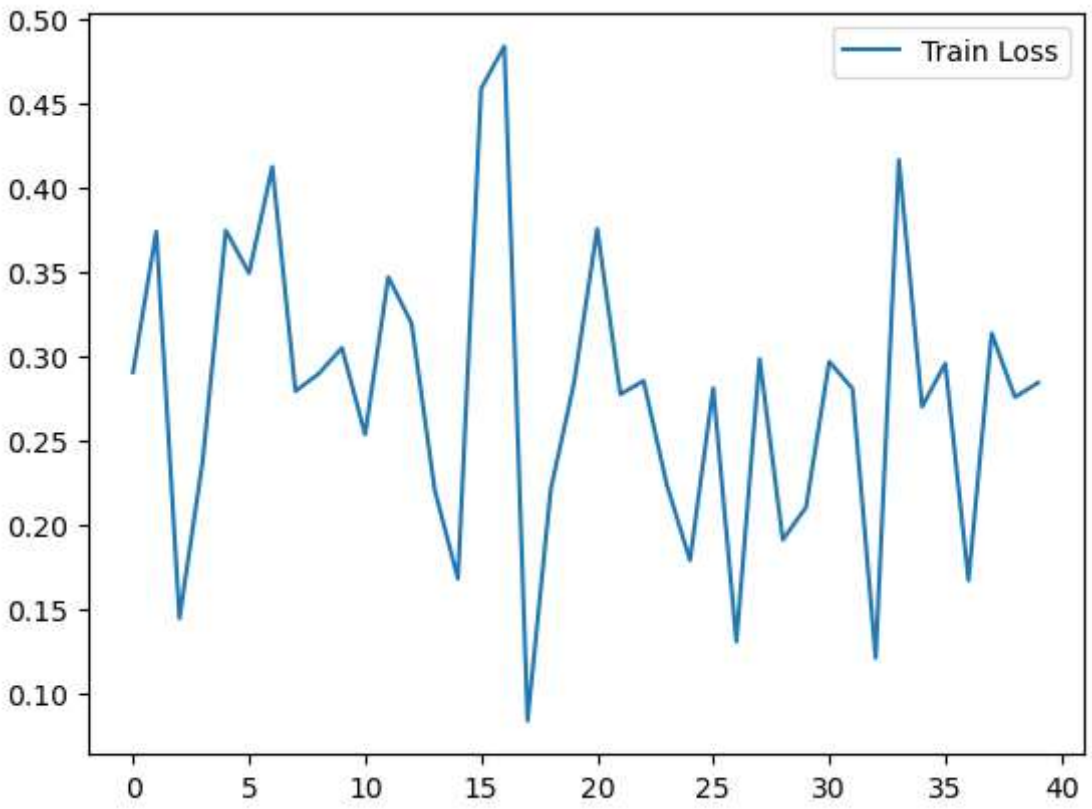
```
model = torch.load(PATH)
model.eval()
```

Out[38]:

```
CNN(
  (conv1): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv2): Sequential(
    (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=T
rue)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (conv3): Sequential(
    (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=
True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (fc1): Linear(in_features=2048, out_features=1024, bias=True)
  (fc2): Linear(in_features=1024, out_features=512, bias=True)
  (fc3): Linear(in_features=512, out_features=2, bias=True)
)
```

In [39]:

```
plt.plot(train_losses, label = "Train Loss")
plt.legend()
plt.show()
```



In [40]:

```
correct = 0
total = 0

for inputs, targets in trainloader:
```

```
    inputs, targets = inputs.to(device), targets.to(device)

    output = model.forward(inputs)

    total += criterion(output, targets).item()

    ps = torch.exp(output).data
    equality = (targets.data == ps.max(1)[1])

    correct += equality.type_as(torch.FloatTensor()).mean()

print("Train Accuracy: {:.3f}%".format(100 * correct/len(trainloader)))
```

```
Train Accuracy: 95.384%
```

- It did help us, as we can see our accuracy score getting improved by increasing the number of epoch. Our accuracy score is 95.384.

In [29]:
```
def imshow(image, ax=None, title=None):
    """Imshow for Tensor."""
    if ax is None:
        fig, ax = plt.subplots()
    image = image.numpy().transpose((1, 2, 0))

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

    return ax
```

In [30]:
```
model.to('cpu')
model.eval()

images, targets = next(iter(trainloader))
print(images.shape, targets.shape)
fig, axes = plt.subplots(figsize=(10, 10), ncols=5)

for i in range(5):
    ax = axes[i]
    imshow(images[i], ax=ax)
```

```
torch.Size([64, 3, 32, 32]) torch.Size([64])
```

In [31]:
```
with torch.no_grad():
    output = model.forward(images)

ps = F.softmax(output,dim=1)
```

In [32]:
```
rand_img = np.random.randint(64, size=1)[0]
rand_img
```

Out[32]:
```
60
```

In [33]:
```
probability = ps[rand_img].data.numpy().squeeze()
probability
```

Out[33]:
```
array([5.737357e-06, 9.999943e-01], dtype=float32)
```

In [34]:
```
imshow(images[rand_img])
```

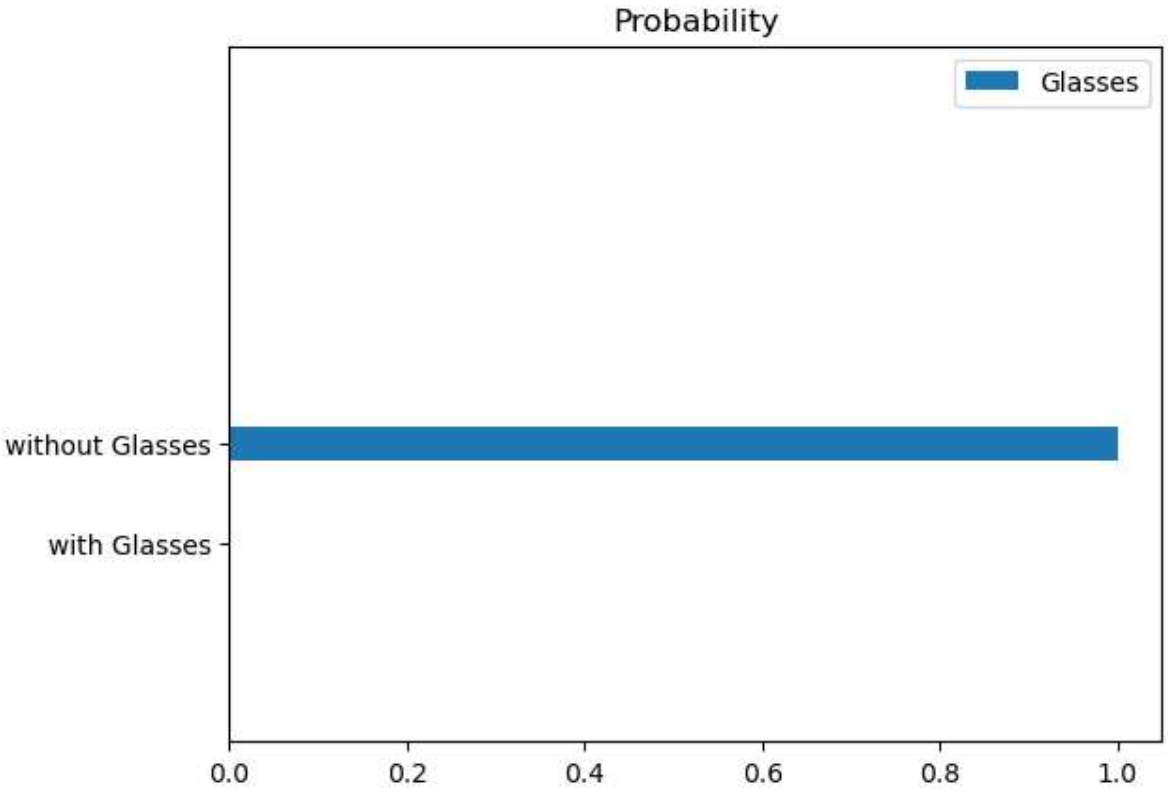
Out[34]:
```
<AxesSubplot:>
```

In [35]:
```
ind = np.arange(2)
targets = ['with Glasses','without Glasses']
width = 0.35
locations = ind

class_probability = plt.barh(ind, probability, width, alpha=1, label = 'Glasses')

plt.yticks(np.arange(2))
plt.title('Probability')
plt.yticks(locations,targets)

#Legend
plt.legend()
plt.ylim(top=5)
plt.ylim(bottom=-2)
plt.show();
```


Probability
Glasses
without Glasses
with Glasses
0.0
0.2
0.4
0.6
0.8
1.0

Thank You