In [ ]:
```python
import numpy as np
import torch
import torch.nn as nn
import matplotlib.pyplot as plt
import torchvision
import torchvision.transforms as transforms
from imutils import paths
import shutil
import os
```

In [ ]:
```python
# use gpu or cpu
if torch.cuda.is_available():
    device = torch.device("cuda")
else:
    device = torch.device("cpu")
```

Take ResNet18 network architecture. See https://pytorch.org/vision/stable/models.html and Load in the pre-trained weights. See again https://pytorch.org/vision/stable/models.html.

In [ ]:
```python
from torchvision.models import resnet18, ResNet18_Weights
# Using pretrained weights:
# Best available weights (currently alias for IMAGENET1K_V2)
# Note that these weights may change across versions
weights = ResNet18_Weights.DEFAULT
model = resnet18(weights = weights)  # deprecated
model = resnet18(True)  # deprecated
model.eval()
```

Out[ ]:

```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=False)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=False)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
```

```
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=1000, bias=True)
)
```

In [ ]:

```python
def replace_last_layer(model):
    # get the number of input features of the last layer
    num_ftrs = model.fc.in_features
    # replace the last layer with a new one followed by the sigmoid layer
    model.fc = nn.Sequential(nn.Linear(num_ftrs, 1, bias=True), nn.Sigmoid())
    return model

# Replace the last layer with a fully connected layer followed by a sigmoid activation function
model = replace_last_layer(model)
```

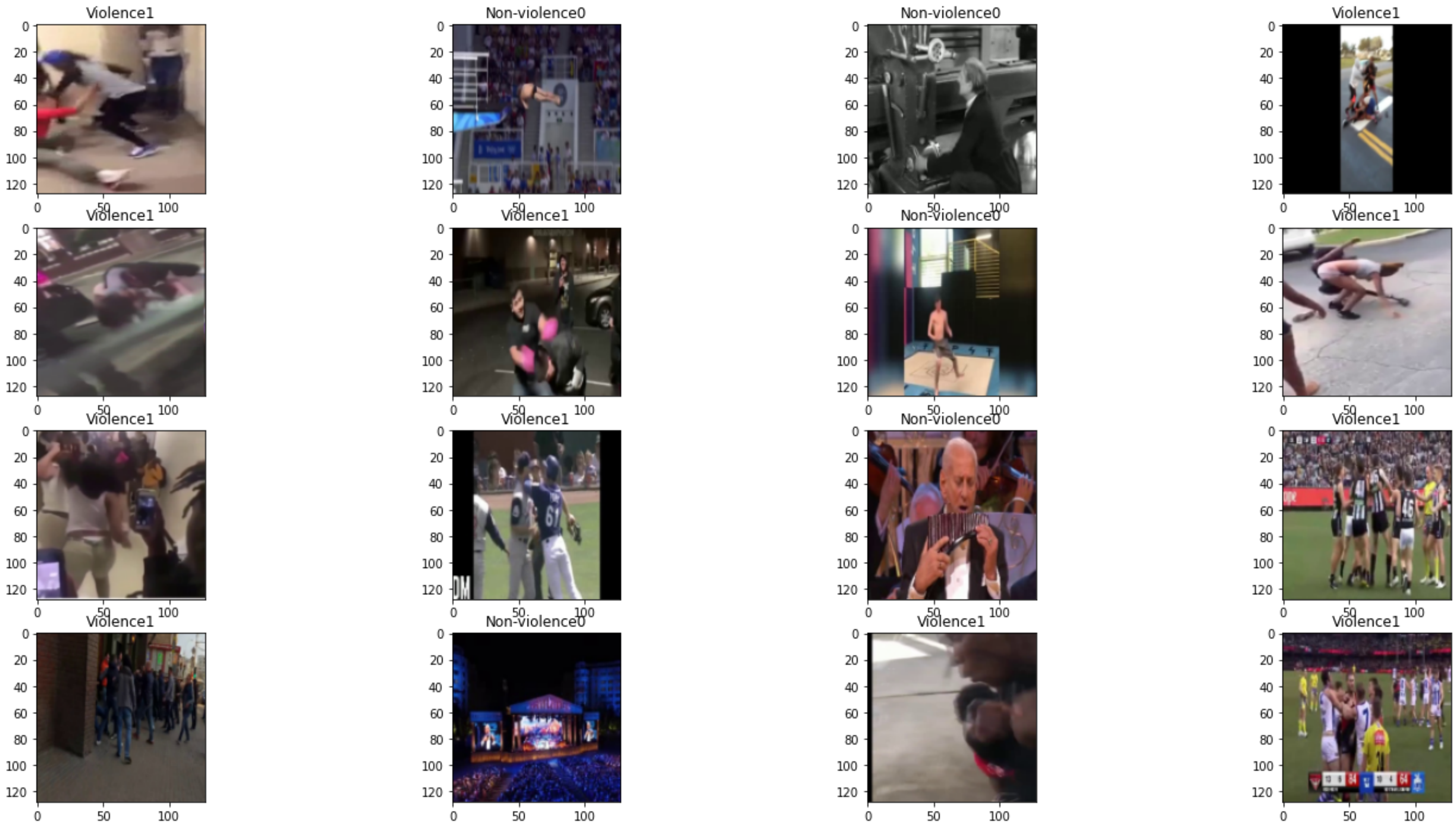
In [ ]:
```
for param in model.parameters():
    # Freeze all the layers
    param.requires_grad = False
# Unfreeze the last layer
for param in model.fc.parameters():
    param.requires_grad = True
```

In [ ]:
```
# To resize images, convert to tensor images and normalize them
img_transforms = transforms.Compose([transforms.Resize((128, 128)),
                                     transforms.ToTensor(),
                                     transforms.Normalize((0, 0, 0),(1, 1, 1))])
# Load the dataset
train_imgs = torchvision.datasets.ImageFolder(root='C:/Users/ginny/projects/final_project_holder/Real_Life_Violence_Dataset/tra
val_imgs = torchvision.datasets.ImageFolder(root='C:/Users/ginny/projects/final_project_holder/Real_Life_Violence_Dataset/val_
test_imgs = torchvision.datasets.ImageFolder(root='C:/Users/ginny/projects/final_project_holder/Real_Life_Violence_Dataset/test
# Create the dataloaders
train_data_loader = torch.utils.data.DataLoader(train_imgs, batch_size=16, shuffle=True, num_workers=4)
val_data_loader = torch.utils.data.DataLoader(val_imgs, batch_size=16, shuffle=False, num_workers=4)
test_data_loader = torch.utils.data.DataLoader(test_imgs, batch_size=16, shuffle=False, num_workers=4)

# Print the number of images in a batch
fig, ax = plt.subplots(4, 4, figsize=(24,12))
for images, labels in train_data_loader:
    for i in range(len(images)):
        ax[i//4, i%4].imshow(images[i].permute(1, 2, 0))
        if labels[i] == 0:
            ax[i//4, i%4].set_title('Non-violence' + str(labels[i].item()))
        else:
            ax[i//4, i%4].set_title('Violence' + str(labels[i].item()))
    break

# To calculate the accuracy
def accuracy(preds, trues):
    total_accuracy = 0
    # Convert preds to 0 or 1
    preds = [1 if preds[i] >= 0.5 else 0 for i in range(len(preds))]
    for i in range(len(preds)):
        if preds[i] == trues[i]:
            total_accuracy += 1
    acc = (total_accuracy / len(preds)) * 100
    return acc
def train_one_epoch(train_data_loader, computeLoss, optimizer, train_loss, train_accuracy, model):
    epoch_loss = []
    epoch_acc = []
    for images, labels in train_data_loader:
        #Load images and labels to device
```

```python
        images = images.to(device)
        labels = labels.to(device)
        # Reshape to match with predictions shape
        labels = labels.reshape((labels.shape[0], 1))
        #Reset Gradients
        optimizer.zero_grad()
        #Forward
        predictions = model(images)
        #Recasting  to float
        predictions = predictions.to(torch.float32)
        labels = labels.to(torch.float32)
        #Compute Loss and add to epoch loss
        loss = computeLoss(predictions, labels)
        epoch_loss.append(loss.item())
        #Calculating Accuracy and adding to epoch accuracy
        acc = accuracy(predictions, labels)
        epoch_acc.append(acc)
        #Backward
        loss.backward()
        # Update Weights
        optimizer.step()
    # Average Loss and Accuracy
    epoch_loss = np.mean(epoch_loss)
    epoch_acc = np.mean(epoch_acc)
    # Add to tracking train logs
    train_loss.append(epoch_loss)
    train_accuracy.append(epoch_acc)
    return epoch_loss, epoch_acc

def val_one_epoch(val_data_loader, best_val_acc, computeLoss, val_loss, val_accuracy, model):
    epoch_loss = []
    epoch_acc = []
    for images, labels in val_data_loader:
        #Load images and labels to device
        images = images.to(device)
        labels = labels.to(device)
        # Reshape to match with predictions shape
        labels = labels.reshape((labels.shape[0], 1))
        #Forward
        predictions = model(images)
        #Recasting  to float
        predictions = predictions.to(torch.float32)
        labels = labels.to(torch.float32)
        #Calculating Loss
        loss = computeLoss(predictions, labels)
        epoch_loss.append(loss.item())
        #Calculating Accuracy
```

```python
        acc = accuracy(predictions, labels)
        epoch_acc.append(acc)
    # Average Loss and Accuracy
    epoch_loss = np.mean(epoch_loss)
    epoch_acc = np.mean(epoch_acc)
    # Add to validation logs
    val_loss.append(epoch_loss)
    val_accuracy.append(epoch_acc)
    # Check and save if best model
    if epoch_acc > best_val_acc:
        best_val_acc = epoch_acc
        torch.save(model.state_dict(),"resnet18_best.pth")
        print("Model saved, best accuracy: %f" %(best_val_acc))
    return epoch_loss, epoch_acc, best_val_acc

# Training and Evaluation
def train_and_evaluate(train_data_loader, val_data_loader, computeLoss, optimizer, epochs, model):
    best_val_acc = 0
    # Logs - Helpful for plotting after training finishes
    train_loss = []
    train_accuracy = []
    val_loss = []
    val_accuracy = []
    for epoch in range(epochs):
        #Training
        loss, acc = train_one_epoch(train_data_loader, computeLoss, optimizer, train_loss, train_accuracy, model)
        #Print Epoch Details
        print("\nTraining, Epoch: %d, Loss: %f, Accuracy: %f" % (epoch, loss, acc))
        #Validation
        loss, acc, best_val_acc = val_one_epoch(val_data_loader, best_val_acc, computeLoss, val_loss, val_accuracy, model)
        #Print Epoch Details
        print("Validating, Epoch: %d, Loss: %f, Accuracy: %f" % (epoch, loss, acc))

    # Plot Results
    #Loss
    plt.title("Loss")
    plt.plot(train_loss, label="Train Loss")
    plt.plot(val_loss, label="Validation Loss")
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.show()

    #Accuracy
    plt.title("Accuracy")
    plt.plot(train_accuracy, label="Train Accuracy")
    plt.plot(val_accuracy, label="Validation Accuracy")
    plt.xlabel("Epochs")
```

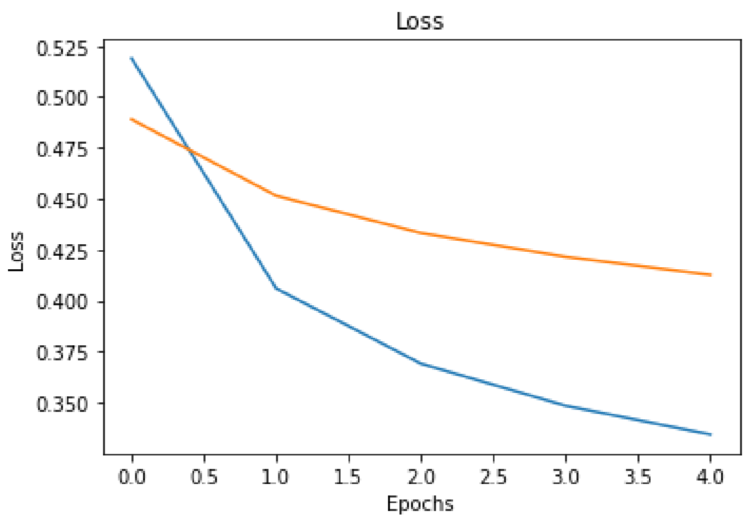
```
    plt.ylabel("Accuracy")
    plt.show()
```

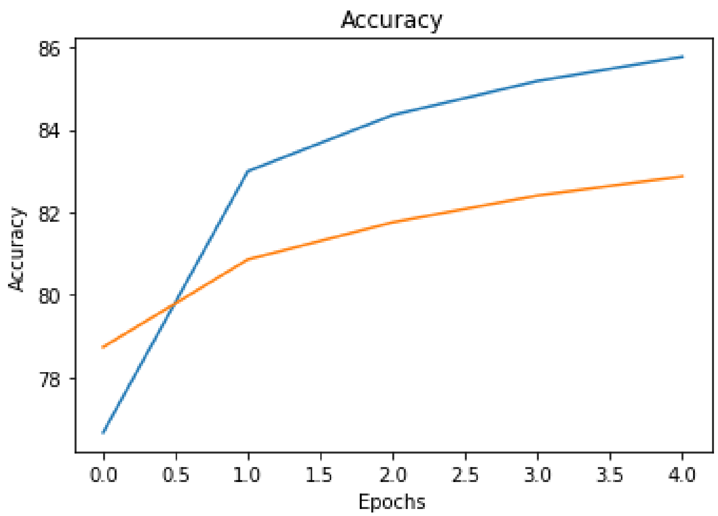


In [ ]:

```
# Loading model to device
model.to(device)
# Loss and optimizer
computeLoss = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr = 1e-5)
epochs = 5
train_and_evaluate(train_data_loader, val_data_loader, computeLoss, optimizer, epochs, model)
```

```
Training, Epoch: 0, Loss: 0.518903, Accuracy: 76.660537
Model saved, best accuracy: 78.731118
Validating, Epoch: 0, Loss: 0.489006, Accuracy: 78.731118

Training, Epoch: 1, Loss: 0.406029, Accuracy: 82.992237
Model saved, best accuracy: 80.853474
Validating, Epoch: 1, Loss: 0.451485, Accuracy: 80.853474

Training, Epoch: 2, Loss: 0.369138, Accuracy: 84.351573
Model saved, best accuracy: 81.752266
Validating, Epoch: 2, Loss: 0.433212, Accuracy: 81.752266

Training, Epoch: 3, Loss: 0.348484, Accuracy: 85.174814
Model saved, best accuracy: 82.401813
Validating, Epoch: 3, Loss: 0.421542, Accuracy: 82.401813

Training, Epoch: 4, Loss: 0.334399, Accuracy: 85.761988
Model saved, best accuracy: 82.866314
Validating, Epoch: 4, Loss: 0.412710, Accuracy: 82.866314
```



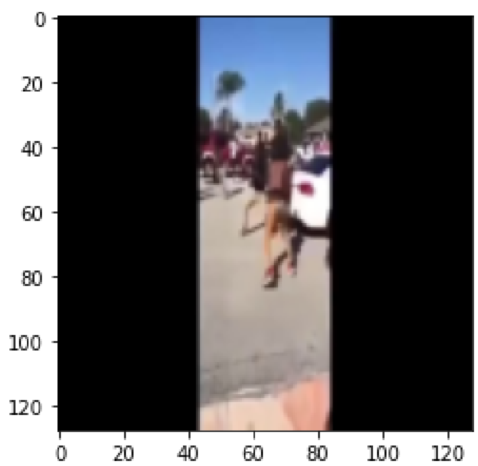


In [ ]:
```
def printRandomMistakePredictions(dataset, model):
    count = 0
    for images, labels in dataset:
        for i in range(len(images)):
            if count < 1:
                #Load images and labels to device
                image = images[i].to(device)
                label = labels[i].to(device)
                label_text = label.item()
                #Forward
                prediction = model(image.unsqueeze(0))
                _, pred = torch.max(prediction, 1)
                #Recasting  to float
                label = label.to(torch.float32)
                pred = pred.to(torch.float32)
                if pred != label:
                    print("Prediction: %d, Label: %d" % (pred, label_text))
                    plt.imshow(image.cpu().permute(1, 2, 0))
                    plt.show()
                    count += 1
```

In [ ]:
```
def test_model(test_data_loader, model, computeLoss):
    total_loss = []
    total_acc = []
    for images, labels in test_data_loader:
        #Load images and labels to device
        images = images.to(device)
        labels = labels.to(device)
        # Reshape to match with predictions shape
```

```
        labels = labels.reshape((labels.shape[0], 1))
        #Forward
        predictions = model(images)
        #Recasting  to float
        predictions = predictions.to(torch.float32)
        labels = labels.to(torch.float32)
        #Compute Loss and add to epoch loss
        loss = computeLoss(predictions, labels)
        total_loss.append(loss.item())
        #Calculating Accuracy and adding to epoch accuracy
        acc = accuracy(predictions, labels)
        total_acc.append(acc)
    # Average Loss and Accuracy
    total_loss = np.mean(total_loss)
    total_acc = np.mean(total_acc)
    return total_loss, total_acc

#Testing
loss, acc = test_model(test_data_loader, model, computeLoss)
#Print Epoch Details
print("\nLoss: %f, Accuracy: %f" % (loss, acc))
printRandomMistakePredictions(test_data_loader, model)
```

```
Loss: 0.416905, Accuracy: 79.135321
Prediction: 0, Label: 1
```



In [ ]:

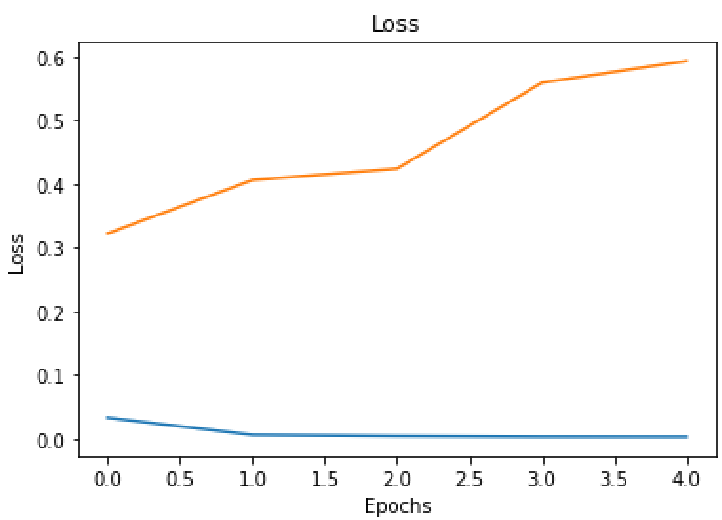
```
for param in model.parameters():
    # Unfreeze all the layers
    param.requires_grad = True
# Print to check if the all layers are unfreezed
"""count = 0
for param in model.parameters():
```

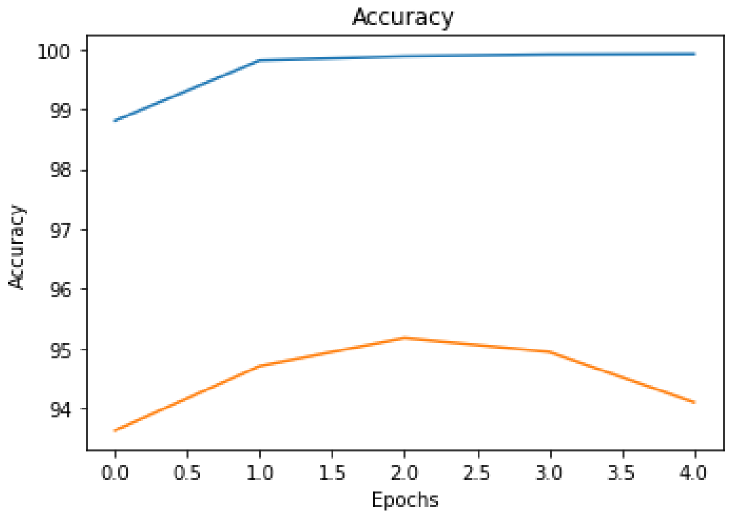
```
    print(count, param.requires_grad)
    count+=1"""
```

Out[ ]: 'count = 0\nfor param in model.parameters():\n   print(count, param.requires_grad)\n   count+=1'

In [ ]:
```
# Loading model to device
model.to(device)
# Loss and optimizer
computeLoss = nn.BCELoss()
optimizer = torch.optim.Adam(model.parameters(), lr = 1e-5)
epochs = 5
train_and_evaluate(train_data_loader, val_data_loader, computeLoss, optimizer, epochs, model)
```

```
Training, Epoch: 0, Loss: 0.032477, Accuracy: 98.802183
Model saved, best accuracy: 93.625378
Validating, Epoch: 0, Loss: 0.322208, Accuracy: 93.625378

Training, Epoch: 1, Loss: 0.005692, Accuracy: 99.809491
Model saved, best accuracy: 94.701662
Validating, Epoch: 1, Loss: 0.405901, Accuracy: 94.701662

Training, Epoch: 2, Loss: 0.003925, Accuracy: 99.879896
Model saved, best accuracy: 95.169940
Validating, Epoch: 2, Loss: 0.424052, Accuracy: 95.169940

Training, Epoch: 3, Loss: 0.002836, Accuracy: 99.910267
Validating, Epoch: 3, Loss: 0.558964, Accuracy: 94.935801

Training, Epoch: 4, Loss: 0.002627, Accuracy: 99.919010
Validating, Epoch: 4, Loss: 0.593000, Accuracy: 94.097432
```





In [ ]:
```
#Testing
loss, acc = test_model(test_data_loader, model, computeLoss)
#Print Epoch Details
print("\nLoss: %f, Accuracy: %f" % (loss, acc))
printRandomMistakePredictions(test_data_loader, model)
```

```
Loss: 0.254996, Accuracy: 96.282110
Prediction: 0, Label: 1
```