(a) (5 points) Data generation. Please refer to the illustration of two regions and corresponding labels as follows:

```
1 import numpy as np
2 import matplotlib.pyplot as plt
3 import torch
4 import sys
```

```
1 points = np.random.rand(100000, 2) * 10 - 5
2 dists_to_coord = np.sqrt(np.sum(points ** 2, axis=1))
3 labels = np.sqrt(np.sum((points - np.array([[0, 0]])) ** 2, axis=1)) <= 2.5
```

```
1 plt.scatter(points[labels == 1, 0], points[labels == 1, 1], c='red', s=0.1)
2 plt.scatter(points[labels == 0, 0], points[labels == 0, 1], c='blue', s=0.1)
3 plt.axis('square')
4 plt.show()
```

(b) (5 points) Network design, implementation, training, and testing. Please report training loss, validation accuracy at every epoch by a line graph and report the testing accuracy of the final model of the MLP classifier.

- Network design: The MLP should contain at least one hidden layer, with at most 50

perceptrons (excluding perceptrons for input and output layers).

- Training method: The MLP must be trained using Cross Entropy Loss, Adam optimizer with a learning rate of 0.001, mini-batch size of 128 samples, and in at most 20 epochs.

```
points = np.random.rand(100000, 2) * 10 - 5
labels = np.sqrt(np.sum((points - np.array([[0, 0]])) ** 2, axis=1)) <= 2.5
train_dataset = torch.utils.data.TensorDataset(torch.as_tensor(points, dtype=torch.float
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=128, shuffle=True)

points = np.random.rand(20000, 2) * 10 - 5
labels = np.sqrt(np.sum((points - np.array([[0, 0]])) ** 2, axis=1)) <= 2.5
val_dataset = torch.utils.data.TensorDataset(torch.as_tensor(points, dtype=torch.float32
val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=128, shuffle=False)


points = np.random.rand(20000, 2) * 10 - 5
labels = np.sqrt(np.sum((points - np.array([[0, 0]])) ** 2, axis=1)) <= 2.5
test_dataset = torch.utils.data.TensorDataset(torch.as_tensor(points, dtype=torch.float3
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=128, shuffle=False)
```

```
model = torch.nn.Sequential(
    torch.nn.Linear(2, 20),
    torch.nn.LeakyReLU(),
    torch.nn.Linear(20, 1),
    torch.nn.Sigmoid(),
).cuda()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
criterion = torch.nn.BCELoss()
losses=[]
validation_accuracy = []
test_accuracy = []
for epoch in range(20):
    plt.clf()
    for points, labels in train_loader:
        preds = model(points.cuda())
        loss = criterion(preds, labels.cuda())
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
        sys.stdout.write(f'\rEpoch {epoch}. Training Loss: {loss.item()}')
        sys.stdout.flush()
    losses.append(loss.detach().cpu().numpy())

    with torch.no_grad():
        total_val_loss = 0.0
        acc_avg = 0
        for points, labels in val_loader:
            preds = model(points.cuda())
            total_val_loss += criterion(preds, labels.cuda())
```

```
            total_val_loss += criterion(preds, labels.cuda())
            preds = preds.squeeze(1).detach().cpu().numpy()
            points = points.detach().numpy()

            acc = 0
            for i in range(preds.shape[0]):
              if labels[i]==0 and preds[i]<0.5:
                 acc+=1
              elif labels[i]==1 and preds[i]>=0.5:
                acc+=1
            acc = acc/128
            acc_avg += acc
            plt.scatter(points[preds >= 0.5, 0], points[preds >= 0.5, 1], c='red', s=0.1
            plt.scatter(points[preds < 0.5, 0], points[preds < 0.5, 1], c='blue', s=0.1)
        validation_accuracy.append(acc_avg/len(val_loader))

        avg_val_loss = total_val_loss.item() / len(val_loader)
        print()
        sys.stdout.write(f'\rEpoch {epoch}. Validation Loss: {avg_val_loss}')
        sys.stdout.flush()
    avg_test_total = []
    with torch.no_grad():
        total_test_loss = 0.0
        acc_avg_test = 0
        for points, labels in test_loader:
            preds = model(points.cuda())
            total_test_loss += criterion(preds, labels.cuda())
            preds = preds.squeeze(1).detach().cpu().numpy()
            #if points[preds>=0.5, 0]
            #pred+=1

            acc_test = 0
            for i in range(preds.shape[0]):
              if labels[i]==0 and preds[i]<0.5:
                acc_test+=1
              elif labels[i]==1 and preds[i]>=0.5:
                acc_test+=1
            acc_test = acc_test/128
            acc_avg_test += acc_test

            points = points.detach().numpy()
            labels=labels.squeeze()
            plt.scatter(points[preds >= 0.5, 0], points[preds >= 0.5, 1], c='red', s=0.1
            plt.scatter(points[preds < 0.5, 0], points[preds < 0.5, 1], c='blue', s=0.1)
            avg_test_loss = total_test_loss.item() / len(test_loader)
            avg_test_total.append(avg_test_loss)
        test_accuracy.append(acc_avg_test/len(test_loader))

        #sys.stdout.write(f'\rEpoch {epoch}. Testing Loss: {avg_test_loss}')
        sys.stdout.flush()
    plt.axis('square')
```

```
80     plt.show()
```

```
import matplotlib.pyplot as plt

#plt.figure(figsize=(12, 4))
#plt.subplot(1, 2, 1)
plt.plot(range(len(losses)), losses)
plt.title('Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')

plt.plot(range(len(losses)), losses)
plt.title('Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')

```

```

plt.subplot(1, 2, 2)
plt.plot(range(len(validation_accuracy)), validation_accuracy)
plt.title('Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')

plt.tight_layout()
plt.show()
```

```
#the testing accuracy of the final model of the MLP classifier
print(test_accuracy[-1])
```

```
0.992734872611465
```

(a) (10 points) Data generation.

```
!wget https://user-images.githubusercontent.com/29124670/266203829-06705829-b2a2-472d-8
```

```
import cv2
from google.colab.patches import cv2_imshow

# bear_img = cv2.imread("bear_binary.jpg")
# cv2_imshow(bear_img)
```

```
--2023-10-03 22:13:53--  https://user-images.githubusercontent.com/29124670/266203829
Resolving user-images.githubusercontent.com (user-images.githubusercontent.com)... 18
Connecting to user-images.githubusercontent.com (user-images.githubusercontent.com)|1
HTTP request sent, awaiting response... 200 OK
Length: 38071 (37K) [image/jpeg]
Saving to: ‘bear_binary.jpg’

bear_binary.jpg     100%[===================>]  37.18K  --.-KB/s    in 0.001s

2023-10-03 22:13:53 (24.5 MB/s) - ‘bear_binary.jpg’ saved [38071/38071]
```

```
from PIL import Image
```

```
bear_img=Image.open("bear_binary.jpg")
```

```
def get_pixel_fromxy(bear_img,xy:tuple[float,float])->int:
  w=bear_img.width-1
  h=bear_img.height-1

  p_x=w*(1/2)*(xy[0]+1)
  p_y=h*(1/2)*(-xy[1]-1)

  p=bear_img.getpixel((p_x,p_y))

  return 1*(p>128)
```

```
points = np.random.rand(100000, 2) * 2 - 1
labels =np.array([get_pixel_fromxy(bear_img,xy=(i[0],i[1])) for i in points])
```

```
plt.scatter(points[labels == 1, 0], points[labels == 1, 1], c='white', s=0.1)
plt.scatter(points[labels == 0, 0], points[labels == 0, 1], c='black', s=0.1)
plt.axis('square')
plt.show()
```

```
plt.show()
```

(b) (10 points) Network design, implementation, training, and testing. Please report training loss, validation accuracy at every epoch by a line graph and report the testing accuracy of the final model of the MLP classifier.

- Network design: The MLP should contain at least one hidden layer, with at most 500 perceptrons (excluding perceptrons for input and output layers).
- Training method: The MLP must be trained using Cross Entropy Loss, Adam optimizer with a learning rate of 0.001, mini-batch size of 128 samples, and in at most 100 epochs.

```
points = np.random.rand(100000, 2) * 2 - 1
labels =np.array([get_pixel_fromxy(bear_img,xy=(i[0],i[1])) for i in points])
train_dataset = torch.utils.data.TensorDataset(torch.as_tensor(points, dtype=torch.float
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=128, shuffle=True)

points = np.random.rand(20000, 2) * 2 - 1
labels =np.array([get_pixel_fromxy(bear_img,xy=(i[0],i[1])) for i in points])
val_dataset = torch.utils.data.TensorDataset(torch.as_tensor(points, dtype=torch.float32
val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=128, shuffle=False)


points = np.random.rand(20000, 2) * 2 - 1
labels =np.array([get_pixel_fromxy(bear_img,xy=(i[0],i[1])) for i in points])
test_dataset = torch.utils.data.TensorDataset(torch.as_tensor(points, dtype=torch.float3
```

```
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=128, shuffle=False)
```

```
model = torch.nn.Sequential(
    torch.nn.Linear(2, 100),
    torch.nn.LeakyReLU(),
    torch.nn.Linear(100, 100),
    torch.nn.LeakyReLU(),
    torch.nn.Linear(100, 100),
    torch.nn.LeakyReLU(),
    torch.nn.Linear(100, 100),
    torch.nn.LeakyReLU(),
    torch.nn.Linear(100, 1),
    torch.nn.Sigmoid(),
).cuda()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
criterion = torch.nn.BCELoss()
losses=[]
validation_accuracy = []
test_accuracy = []
for epoch in range(100):
    plt.clf()
    for points, labels in train_loader:
        preds = model(points.cuda())
        loss = criterion(preds, labels.cuda())
        loss.backward()
        optimizer.step()
        optimizer.zero_grad()
        sys.stdout.write(f'\rEpoch {epoch}. Training Loss: {loss.item()}')
        sys.stdout.flush()
    losses.append(loss.detach().cpu().numpy())

    with torch.no_grad():
        total_val_loss = 0.0
        acc_avg = 0
        for points, labels in val_loader:
            preds = model(points.cuda())
            total_val_loss += criterion(preds, labels.cuda())
            preds = preds.squeeze(1).detach().cpu().numpy()
            points = points.detach().numpy()

            acc = 0
            for i in range(preds.shape[0]):
              if labels[i]==0 and preds[i]<0.5:
                 acc+=1
              elif labels[i]==1 and preds[i]>=0.5:
                acc+=1
            acc = acc/128
            acc_avg += acc
            plt.scatter(points[preds >= 0.5, 0], points[preds >= 0.5, 1], c='white', s=(
            plt.scatter(points[preds < 0.5, 0], points[preds < 0.5, 1], c='black', s=0.1
```

```
        validation_accuracy.append(acc_avg/len(val_loader))

        avg_val_loss = total_val_loss.item() / len(val_loader)
        print()
        sys.stdout.write(f'\rEpoch {epoch}. Validation Loss: {avg_val_loss}')
        sys.stdout.flush()
    avg_test_total = []
    with torch.no_grad():
        total_test_loss = 0.0
        acc_avg_test = 0
        for points, labels in test_loader:
            preds = model(points.cuda())
            total_test_loss += criterion(preds, labels.cuda())
            preds = preds.squeeze(1).detach().cpu().numpy()
            #if points[preds>=0.5, 0]
            #pred+=1

            acc_test = 0
            for i in range(preds.shape[0]):
              if labels[i]==0 and preds[i]<0.5:
                acc_test+=1
              elif labels[i]==1 and preds[i]>=0.5:
                acc_test+=1
            acc_test = acc_test/128
            acc_avg_test += acc_test

            points = points.detach().numpy()
            labels=labels.squeeze()
            plt.scatter(points[preds >= 0.5, 0], points[preds >= 0.5, 1], c='white', s=(
            plt.scatter(points[preds < 0.5, 0], points[preds < 0.5, 1], c='black', s=0.1
            avg_test_loss = total_test_loss.item() / len(test_loader)
            avg_test_total.append(avg_test_loss)
        test_accuracy.append(acc_avg_test/len(test_loader))

        #sys.stdout.write(f'\rEpoch {epoch}. Testing Loss: {avg_test_loss}')
        sys.stdout.flush()
    plt.axis('square')
    plt.show()
```

```
import matplotlib.pyplot as plt

#plt.figure(figsize=(12, 4))
#plt.subplot(1, 2, 1)
plt.plot(range(len(losses)), losses)
plt.title('Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')

```

```
plt.plot(range(len(losses)), losses)
plt.title('Training Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
```

```
plt.subplot(1, 2, 2)
plt.plot(range(len(validation_accuracy)), validation_accuracy)
plt.title('Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')

plt.tight_layout()
plt.show()
```

```
1 #the testing accuracy of the final model of the MLP classifier
2 print(test_accuracy[-1])
```

```
    0.9899482484076433
```