In [2]:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import math
# Input data files are available in the read-only "../input/" directory
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    pass
```

Question 1

In [5]:

```
##Points point 1:(x1,y1):upper point
 ##       point2: (x2,y2): left point
 ##        point3: (x3,y3):right point
 ##         point4: (x4,y4):lower point

def euclidean_distance(point1, point2):
        return math.sqrt((point1[0] - point2[0])**2 + (point1[1] - point2[1])**2)

def calculate_angle(point1, point2):
        delta_x = point2[0] - point1[0]
        delta_y = point2[1] - point1[1]
        return math.atan2(delta_y, delta_x)
```

In [6]:

```
## ratio of distance between left point and upper point and lower point and left point.
## if this is less than 1, then this means that person is smiling else they are not
def is_smiling_based_on_distance_ratio(upper_point, left_point, right_point, lower_point):

    # Parameters:
    #     upper_point (tuple): (x1, y1) coordinates of the upper point
    #     left_point (tuple): (x2, y2) coordinates of the left point
    #     right_point (tuple): (x3, y3) coordinates of the right point (not used here)
    #     lower_point (tuple): (x4, y4) coordinates of the lower point

    # Calculate distances
    dist_left_upper = euclidean_distance(left_point, upper_point)
    dist_left_lower = euclidean_distance(left_point, lower_point)

    # Calculate ratio
    ratio = dist_left_upper / dist_left_lower

    #if ratio <1 then smiling else not
    #ratio can be set something other than 1 also
    return ratio < 1
```

In [7]:

```
## we will find the ratio of angle made by line joining upper point and horizontal line(line joining the left
## point and right point) to the angle made by line joining the lower point and horizontal line.
##if this angle is less than some threshold(epsilon) then we will say that person is smiling else we will
## say that s/he is not
## the value of epsilon can be set by user
def is_smiling_based_on_angle_ratio(upper_point, left_point, right_point, lower_point, epsilon=0.5):


    # Midpoint of horizontal line (left and right points)
    mid_horizontal = ((left_point[0] + right_point[0]) / 2, (left_point[1] + right_point[1]) / 2)

    # Calculate angles in radians
    angle_upper = calculate_angle(mid_horizontal, upper_point)
    angle_lower = calculate_angle(mid_horizontal, lower_point)

    # Calculate ratio of absolute angles
    angle_ratio = abs(angle_upper) / abs(angle_lower)

    # Check if ratio is less than threshold epsilon
    return angle_ratio < epsilon
```

In [8]:

```
##Now, we will find the ratio of radius of circle passing through (top point, left point, right point) and
## radius of circle passing through (lowe point, left point, right point). If it is less than a particular
## threshold then person is smiling else s/he is not

def calculate_radius(point1, point2, point3):

    # Calculate  lengths of sides of the triangle
    a = euclidean_distance(point1,point2)
    b = euclidean_distance(point2,point3)
    c = euclidean_distance(point1,point3)
    # Calculating the semi-perimeter of a triangle
    s = (a + b + c) / 2

    # Calculating the area of the triangle using Heron's formula
    area = math.sqrt(s * (s - a) * (s - b) * (s - c))

    # Circumcircle radius formula: R = (a * b * c) / (4 * Area)
    if area == 0:  # Avoiding division by zero
        return float('inf')

    radius = (a * b * c) / (4 * area)
    return radius


def is_smiling_based_on_circumcircle_ratio(upper_point, left_point, right_point, lower_point, threshold=1.5):

    # Calculating radii of circumcircles
    radius_upper = calculate_radius(upper_point, left_point, right_point)
    radius_lower = calculate_radius(lower_point, left_point, right_point)

    # Calculating ratio of radii
    ratio = radius_upper / radius_lower

    # Determine if smiling based on threshold
    return ratio < threshold
```

Question 2

In [9]:

```
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import torchvision
import torchvision.transforms as transforms
from torch.utils.data import DataLoader
```

In [10]:

```
class modifiedActivationFunction(nn.Module): ##Modified activation function
    def forward(self, x):
        return x * torch.sigmoid(x)
```

In [11]:

```
class ModifiedLeNet(nn.Module):
    def __init__(self):
        super(ModifiedLeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=3, stride=1, padding=1)  # Using 3x3 filter
        self.conv2 = nn.Conv2d(6, 16, kernel_size=3, stride=1, padding=1) # Using 3x3 filter
        self.fc1 = nn.Linear(16 * 7 * 7, 120)
        self.fc2 = nn.Linear(120, 84)
        self.fc3 = nn.Linear(84, 10)
        self.activation = modifiedActivationFunction()   #Using modified activation function
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2) #Using Max pooling instead of Average pooling

    def forward(self, x):
        x = self.pool(self.activation(self.conv1(x)))
        x = self.pool(self.activation(self.conv2(x)))
        x = x.view(x.size(0), -1)  # Flattening the output
        x = self.activation(self.fc1(x))
        x = self.activation(self.fc2(x))
        x = self.fc3(x)
        return F.softmax(x, dim=1)  # Using the Softmax at the end
```

In [12]:

```
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.1307,), (0.3081,)) ##here we are normalizing it Normalize(mean, std)
])

trainset = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=transform)
testset = torchvision.datasets.MNIST(root='./data', train=False, download=True, transform=transform)

trainloader = DataLoader(trainset, batch_size=64, shuffle=True)
testloader = DataLoader(testset, batch_size=64, shuffle=False)
```

```
Downloading http://yann.lecun.com/exdb/mnist/train-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-images-idx3-ubyte.gz to ./data/MNIST/raw/train-images-idx3-ubyt
e.gz
```

```
100%|██████████| 9.91M/9.91M [00:00<00:00, 11.5MB/s]
Extracting ./data/MNIST/raw/train-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/train-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/train-labels-idx1-ubyte.gz to ./data/MNIST/raw/train-labels-idx1-ubyt
e.gz
100%|██████████| 28.9k/28.9k [00:00<00:00, 343kB/s]
Extracting ./data/MNIST/raw/train-labels-idx1-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-images-idx3-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw/t10k-images-idx3-ubyte.
gz
100%|██████████| 1.65M/1.65M [00:00<00:00, 3.18MB/s]
Extracting ./data/MNIST/raw/t10k-images-idx3-ubyte.gz to ./data/MNIST/raw

Downloading http://yann.lecun.com/exdb/mnist/t10k-labels-idx1-ubyte.gz
Failed to download (trying next):
HTTP Error 404: Not Found

Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz
Downloading https://ossci-datasets.s3.amazonaws.com/mnist/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw/t10k-labels-idx1-ubyte.
gz
100%|██████████| 4.54k/4.54k [00:00<00:00, 3.48MB/s]
Extracting ./data/MNIST/raw/t10k-labels-idx1-ubyte.gz to ./data/MNIST/raw
```

In [13]:

```python
# Initializing the model, loss function, and optimizer
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = ModifiedLeNet().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

In [14]:

```
#Training the model
num_epochs = 10
for epoch in range(num_epochs):
    model.train()
    running_loss = 0.0
    correct = 0
    total = 0
    for images, labels in trainloader:
        images, labels = images.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(images)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
    accuracy = 100 * correct / total
    print(f"Epoch {epoch+1}, Loss: {running_loss/len(trainloader):.4f}, Accuracy: {accuracy:.2f}%")

# Evaluating the model
model.eval()
correct = 0
total = 0
with torch.no_grad():
    for images, labels in testloader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

print(f"Test Accuracy: {100 * correct / total:.2f}%")
```

```
Epoch 1, Loss: 1.5528, Accuracy: 91.62%
Epoch 2, Loss: 1.4919, Accuracy: 97.04%
Epoch 3, Loss: 1.4846, Accuracy: 97.71%
Epoch 4, Loss: 1.4798, Accuracy: 98.18%
Epoch 5, Loss: 1.4771, Accuracy: 98.43%
Epoch 6, Loss: 1.4755, Accuracy: 98.59%
Epoch 7, Loss: 1.4745, Accuracy: 98.67%
Epoch 8, Loss: 1.4733, Accuracy: 98.79%
Epoch 9, Loss: 1.4719, Accuracy: 98.92%
Epoch 10, Loss: 1.4723, Accuracy: 98.90%
Test Accuracy: 98.86%
```

Question 3

In [15]:

```
import cv2
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
from skimage.feature import hog
```

In [20]:

```
##defining the roberts cross edge filter
def roberts_cross_edge(image):
    kernel1 = np.array([[1, 0], [0, -1]], dtype=np.float32) ##defining the
    kernel2 = np.array([[0, 1], [-1, 0]], dtype=np.float32)

    gx = cv2.filter2D(image, -1, kernel1)
    gy = cv2.filter2D(image, -1, kernel2)

    gradient_magnitude = np.sqrt(gx ** 2 + gy ** 2) ## we can also use the magnitude of abs(gx) + abs(gy)
    return gradient_magnitude
```

In [21]:

```
def extract_hog_features(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    image = cv2.resize(image, (128, 128))  # Resizing the images to 128x128 for the consistency
    edge_image = roberts_cross_edge(image) #Applying the robert cross edge detector
    #extracting the hog features
    hog_features, _ = hog(edge_image, pixels_per_cell=(8, 8), cells_per_block=(2, 2), visualize=True)
    return hog_features
```

In [22]:

```
def load_dataset(folder):
    X, y = [], []   ##for storing the dataset and labels
    categories = ['cats', 'dogs']
    cnt=0 ##just count variable
    for label, category in enumerate(categories):
        category_path = os.path.join(folder, category)
        for file in os.listdir(category_path):
            img_path = os.path.join(category_path, file)
            image = cv2.imread(img_path)
            cnt+=1
            if image is not None:
                features = extract_hog_features(image) ##extracting the hog features
                X.append(features)             ##storing the features in X
                y.append(label)
            if(cnt%1000==0):
                print(f"{cnt} images processed")
    return np.array(X), np.array(y)
```

In [23]:

```
test_dataset_path="/kaggle/input/cat-and-dog/test_set/test_set"    #test path
train_dataset_path="/kaggle/input/cat-and-dog/training_set/training_set" #train path
X_test,y_test=load_dataset(test_dataset_path)
X_train, y_train=load_dataset(train_dataset_path)
```

```
1000 images processed
2000 images processed
1000 images processed
2000 images processed
3000 images processed
4000 images processed
5000 images processed
6000 images processed
7000 images processed
8000 images processed
```

In [24]:

```
classifier_model = RandomForestClassifier(n_estimators=100, random_state=42) #initialising the model
classifier_model.fit(X_train, y_train)       #training the model
```

Out[24]:

```
▾ RandomForestClassifier
RandomForestClassifier(random_state=42)
```

In [25]:

```
y_pred = classifier_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print(f"The Accuracy of the model is: {accuracy:.2f}")
```

```
The Accuracy of the model is: 0.64
```

Question 4

In [26]:

```
from PIL import Image
```

In [32]:

```
##A function to load the image and convert it into the matrix
def load_image_to_matrix(image_path):
     ##loading the png image and converting it to the matrix
    img = Image.open(image_path).convert('L')
    img_matrix = np.array(img)  # Convert image to NumPy array
    img_matrix = (img_matrix > 127).astype(int)
    return img_matrix

def countObjects(img_matrix):     ##function to calculate no. of objects in a matrix

    ##it is a stack based depth first search algorithm
    def iterative_f(start_x, start_y):
        stack = [(start_x, start_y)]
        while stack:
            x, y = stack.pop()
            if 0 <= x < img_matrix.shape[0] and 0 <= y < img_matrix.shape[1] and img_matrix[x, y] == 1:
                img_matrix[x, y] = 0  # Mark visited
                stack.append((x+1, y))
                stack.append((x-1, y))
                stack.append((x, y+1))
                stack.append((x, y-1))

    noOfObjects = 0
    for i in range(img_matrix.shape[0]):
        for j in range(img_matrix.shape[1]):
            if img_matrix[i, j] == 1:
                noOfObjects += 1
                iterative_f(i, j)

    return noOfObjects
```

In [33]:
```
def countNoOfObjects(image_path):

   image_matrix=load_image_to_matrix(image_path) ##loading and converting the image into matrix
   return countObjects(image_matrix) ##counting the no. of objects in the matrix
```

In [34]:
```
image_path="/kaggle/input/assig1-png/Assig1.png" ##path to the image
print(f"The no. of objects present in the image is {countNoOfObjects(image_path)}")
```

```
The no. of objects present in the image is 9
```