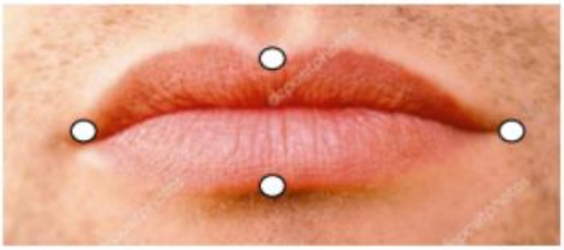
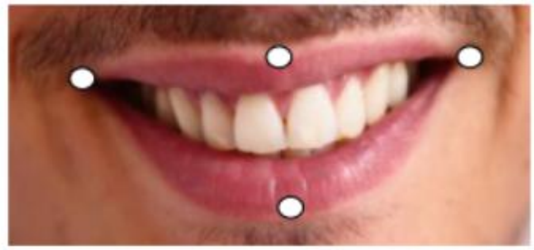
EE655

ASSIGNMENT 1

Q1) Given the four key points of the mouth shown below, design at least three features that can help us monitor whether a person is smiling. **[2 Marks]**





```python
import numpy as np

# WIDTH TO HEIGHT RATIO
def width_to_height_ratio(left_corner, right_corner, top_lip, bottom_lip):
    width = np.linalg.norm(np.array(left_corner) - np.array(right_corner))
    height = np.linalg.norm(np.array(top_lip) - np.array(bottom_lip))
    return width / max(height, 1)  # Avoid division by zero

# LIP DISTANCE
def lip_distance(top_lip, bottom_lip):
    return np.linalg.norm(np.array(top_lip) - np.array(bottom_lip))

# CORNER DISTANCE RATIO
def corner_distance_ratio(left_corner, right_corner):
    width = np.linalg.norm(np.array(left_corner) - np.array(right_corner))
    height = abs(left_corner[1] - right_corner[1])
    return height / max(width, 1)
```

Q2) Implement a modified LeNet architecture from scratch and train it on the MNIST dataset. Your LeNet architecture must incorporate the following changes: **[3 marks]**

- Include a softmax layer at the end.
- Use *x*sigmoid(x)* as the activation function.
- Replace average pooling with max pooling.
- Use only 3×3 filters in convolutional layers.

```
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

# Data transformations
transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

# Load MNIST dataset
train_dataset = torchvision.datasets.MNIST(root='./data', train=True, download=True, transform=transform)
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=64, shuffle=True)

test_dataset = torchvision.datasets.MNIST(root='./data', train=False, download=True, transform=transform)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=64, shuffle=False)

# Swish-like activation function
def sigmoid_activation(x):
    return x * torch.sigmoid(x)

# Modified LeNet Model
class ModifiedLeNet(nn.Module):
    def __init__(self):
        super(ModifiedLeNet, self).__init__()
        self.conv1 = nn.Conv2d(1, 6, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(6, 16, kernel_size=3)
```

```
        self.conv3 = nn.Conv2d(16, 120, kernel_size=3)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.fc1 = nn.Linear(120 * 4 * 4, 84)
        self.fc2 = nn.Linear(84, 10)

    def forward(self, x):
        x = self.pool(sigmoid_activation(self.conv1(x)))
        x = self.pool(sigmoid_activation(self.conv2(x)))
        x = sigmoid_activation(self.conv3(x))
        x = torch.flatten(x, start_dim=1)
        x = sigmoid_activation(self.fc1(x))
        x = self.fc2(x)  # No Softmax (CrossEntropyLoss applies it internally)
        return x

# Device configuration
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = ModifiedLeNet().to(device)
```

```python
# Loss function & Optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training Loop
epochs = 10
for epoch in range(1, epochs + 1):
    model.train()
    running_loss = 0.0

    for images, labels in train_loader:
        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()

    avg_loss = running_loss / len(train_loader)
    print(f"Epoch {epoch}/{epochs}, Loss: {avg_loss:.4f}")

# Evaluation
model.eval()
correct = 0
total = 0
```

```python
with torch.no_grad():
    for images, labels in test_loader:
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        predicted = outputs.argmax(dim=1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

accuracy = 100 * correct / total
print(f'Accuracy: {accuracy:.2f}%')
```

OUTPUT:

```
Epoch 1/10, Loss: 0.1944
Epoch 2/10, Loss: 0.0502
Epoch 3/10, Loss: 0.0359
Epoch 4/10, Loss: 0.0271
Epoch 5/10, Loss: 0.0214
Epoch 6/10, Loss: 0.0167
Epoch 7/10, Loss: 0.0141
Epoch 8/10, Loss: 0.0129
Epoch 9/10, Loss: 0.0099
Epoch 10/10, Loss: 0.0079
Accuracy: 99.24%
```

Q3) Implement a modified Histogram of Oriented Gradients (HoG) feature extraction algorithm from scratch, using the "Robert cross edge detector" for computing derivatives. Extract features from images in the Cat and Dog dataset and train a Random Forest classifier. **[2 marks]**

```python
import cv2
import os
import numpy as np
import matplotlib.pyplot as plt
from skimage.feature import hog
from sklearn.metrics import classification_report, confusion_matrix
from tqdm import tqdm

CATEGORIES = ["cats", "dogs"]

def load_data(folder, img_size):
    data, labels = [], []
    for category in CATEGORIES:
        path = os.path.join(folder, category)
        label = CATEGORIES.index(category)
        for img_name in tqdm(os.listdir(path), desc=f"Loading {category}"):
            img = cv2.imread(os.path.join(path, img_name))
            if img is not None:
                img = cv2.resize(img, (img_size, img_size))
                img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
                data.append(img.astype(np.float32) / 255.0)
                labels.append(label)
    return np.array(data).reshape(-1, img_size, img_size, 1), np.array(labels)

train_path = r"C:\Users\Ayush\Desktop\training_set"
test_path = r"C:\Users\Ayush\Desktop\test_set"
X_train, y_train = load_data(train_path, 128)
X_test, y_test = load_data(test_path, 128)
```

```python
from scipy.ndimage import convolve

roberts_x = np.array([[1, 0], [0, -1]], dtype=np.float32)
roberts_y = np.array([[0, 1], [-1, 0]], dtype=np.float32)

def roberts_gradient(image):
    Gx, Gy = convolve(image, roberts_x), convolve(image, roberts_y)
    mag = np.sqrt(Gx**2 + Gy**2)
    orient = np.arctan2(Gy, Gx) * (180 / np.pi)
    orient[orient < 0] += 180
    return mag, orient

def batch_gradients(images):
    mags, orients = [], []
    for img in images:
        mag, orient = roberts_gradient(img.squeeze())
        mags.append(mag)
        orients.append(orient)
    return np.array(mags), np.array(orients)

gradient_magnitudes, gradient_orientations = batch_gradients(X_train)
```

```python
from RCedge import roberts_gradient

def hog_features(image, cell_size=8, block_size=2, bins=9):
    h, w = image.shape
    mag, orient = roberts_gradient(image)

    cells_x, cells_y = w // cell_size, h // cell_size
    hog_desc = np.zeros((cells_y, cells_x, bins))

    bin_width = 180 / bins
    for i in range(cells_y):
        for j in range(cells_x):
            cell_mag = mag[i * cell_size:(i + 1) * cell_size, j * cell_size:(j + 1) * cell_size]
            cell_orient = orient[i * cell_size:(i + 1) * cell_size, j * cell_size:(j + 1) * cell_size]
            hist = np.zeros(bins)
            for x in range(cell_size):
                for y in range(cell_size):
                    hist[int(cell_orient[x, y] // bin_width) % bins] += cell_mag[x, y]
            hog_desc[i, j, :] = hist

    blocks_x, blocks_y = cells_x - block_size + 1, cells_y - block_size + 1
    norm_hog = np.zeros((blocks_y, blocks_x, bins * (block_size ** 2)))

    for i in range(blocks_y):
        for j in range(blocks_x):
            block = hog_desc[i:i + block_size, j:j + block_size, :].flatten()
            norm_hog[i, j, :] = block / (np.linalg.norm(block) + 1e-6)

    return norm_hog.flatten()
```

```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
import numpy as np

X_train_hog = np.array([hog_features(img.squeeze()) for img in X_train])
X_test_hog = np.array([hog_features(img.squeeze()) for img in X_test])

rf_clf = RandomForestClassifier(n_estimators=100, random_state=42)
rf_clf.fit(X_train_hog, y_train)

y_pred_rf = rf_clf.predict(X_test_hog)

print("\n", classification_report(y_test, y_pred_rf))
print("\n", confusion_matrix(y_test, y_pred_rf))
```

OUTPUT:

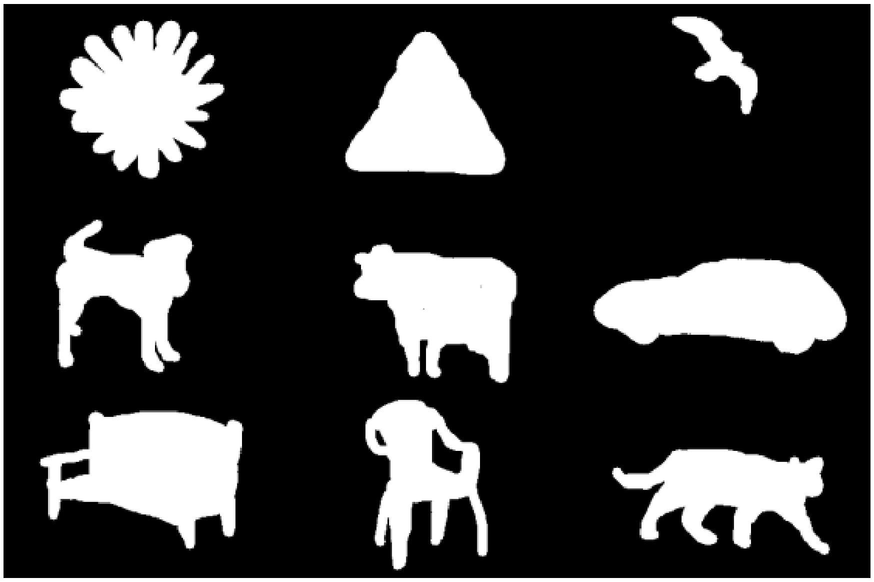
```
              precision    recall  f1-score   support

           0       0.68      0.76      0.72      1011
           1       0.73      0.63      0.68      1012

    accuracy                           0.70      2023
   macro avg       0.70      0.70      0.70      2023
weighted avg       0.70      0.70      0.70      2023


 [[772 239]
 [370 642]]
```

Q4) Develop an algorithm from scratch to programmatically count the number of objects present in the binary image shown below. **[3 marks]**



```python
import cv2
import numpy as np

image_path = r"C:\Users\Ayush\OneDrive\Desktop\Objects.png"
image = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)

if image is None:
    raise FileNotFoundError(f"Error: Could not load image at {image_path}")

def count_objects(image):
    height, width = image.shape
    visited = np.zeros((height, width), dtype=bool)
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1), (-1, -1), (-1, 1), (1, -1), (1, 1)]

    def dfs(x, y):
        stack = [(x, y)]
        while stack:
            cx, cy = stack.pop()
            if 0 <= cx < height and 0 <= cy < width and image[cx, cy] == 255 and not visited[cx, cy]:
                visited[cx, cy] = True
                stack.extend((cx + dx, cy + dy) for dx, dy in directions)

    _, binary_image = cv2.threshold(image, 128, 255, cv2.THRESH_BINARY)
    object_count = sum(dfs(i, j) or 1 for i in range(height) for j in range(width) if binary_image[i, j] == 255 and not visited[i, j])

    return object_count

num_objects = count_objects(image)
print(f"Number of objects detected: {num_objects}")
```

OUTPUT:

```
Number of objects detected: 9
```