CS216 HW2

Name : Sihat Afnan

Human Face dataset used for training : Link

Test Dataset : Link

Used Architectures

- Linear Classifier
  - Logistic Regression
  - SVM
- Matched Filter
- Neural Network
  - Fast-RCNN

Linear_Classifier(Logistic_regression_and_SVM)_to_Detect_Faces

February 13, 2025

[ ]:
```
import pandas as pd
import numpy as np
import cv2
import os
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
import matplotlib.patches as patches
from PIL import Image
from sklearn.metrics import accuracy_score
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

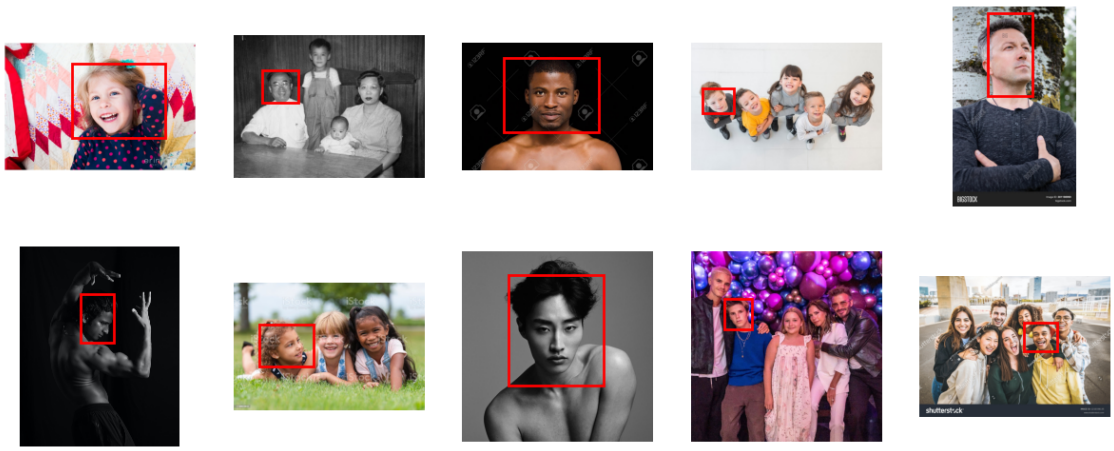
1 Load dataset

[ ]:
```
faces = pd.read_csv('/content/drive/MyDrive/CS216 Files/HW2/Face Data/faces.
↪csv')
image_dir = '/content/drive/MyDrive/CS216 Files/HW2/Face Data/images'
```

2 Visualize data

[ ]:
```
#@title
sample_images = faces.sample(n=10)

fig, axes = plt.subplots(2, 5, figsize=(15, 6))

for ax, (_, row) in zip(axes.flatten(), sample_images.iterrows()):
    image_path = os.path.join(image_dir, row['image_name'])
    image = Image.open(image_path)
    ax.imshow(image)
    ax.axis('off')
```

```
    x0, y0, x1, y1 = row['x0'], row['y0'], row['x1'], row['y1']
    bbox = patches.Rectangle((x0, y0), x1 - x0, y1 - y0, linewidth=2,␣
↪edgecolor='red', facecolor='none')
    ax.add_patch(bbox)

plt.show()
```



3 Generate negative samples (random patches from the image that are not faces) and Train the models

[ ]:
```
def generate_negative_samples(data, img_dir, num_samples=500):
    features = []
    labels = []
    for _, row in data.iterrows():
        img_path = os.path.join(img_dir, row['image_name'])
        img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
        if img is not None:
            # Extract positive face sample
            face = img[int(row['y0']):int(row['y1']), int(row['x0']):
↪int(row['x1'])]
            face_resized = cv2.resize(face, (64, 64)).flatten()
            features.append(face_resized)
            labels.append(1)  # Face

            # Generate a random background patch as a negative sample
            for _ in range(2):  # Add multiple background samples per face
                h, w = img.shape
                x0, y0 = np.random.randint(0, w - 64), np.random.randint(0, h -␣
↪64)
```

```
                if x0 > row['x1'] or y0 > row['y1']:  # Ensure it's not␣
 ↪overlapping the face
                    background = img[y0:y0+64, x0:x0+64]
                    background_resized = cv2.resize(background, (64, 64)).
 ↪flatten()
                    features.append(background_resized)
                    labels.append(0)  # Background

    features = np.array(features)
    labels = np.array(labels)
    return features, labels

# Prepare features and labels
X, y = generate_negative_samples(faces, image_dir)

# Split into training and testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42, stratify=y)

# Train Logistic Regression
logistic_model = LogisticRegression(max_iter=1000)
logistic_model.fit(X_train, y_train)

# Train SVM
svm_model = SVC(kernel='linear', probability=True)
svm_model.fit(X_train, y_train)
```

```
/usr/local/lib/python3.11/dist-packages/sklearn/linear_model/_logistic.py:465:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[ ]:
```
SVC(kernel='linear', probability=True)
```

4 Function to apply Non-Maximum Suppression (NMS) based on IoU

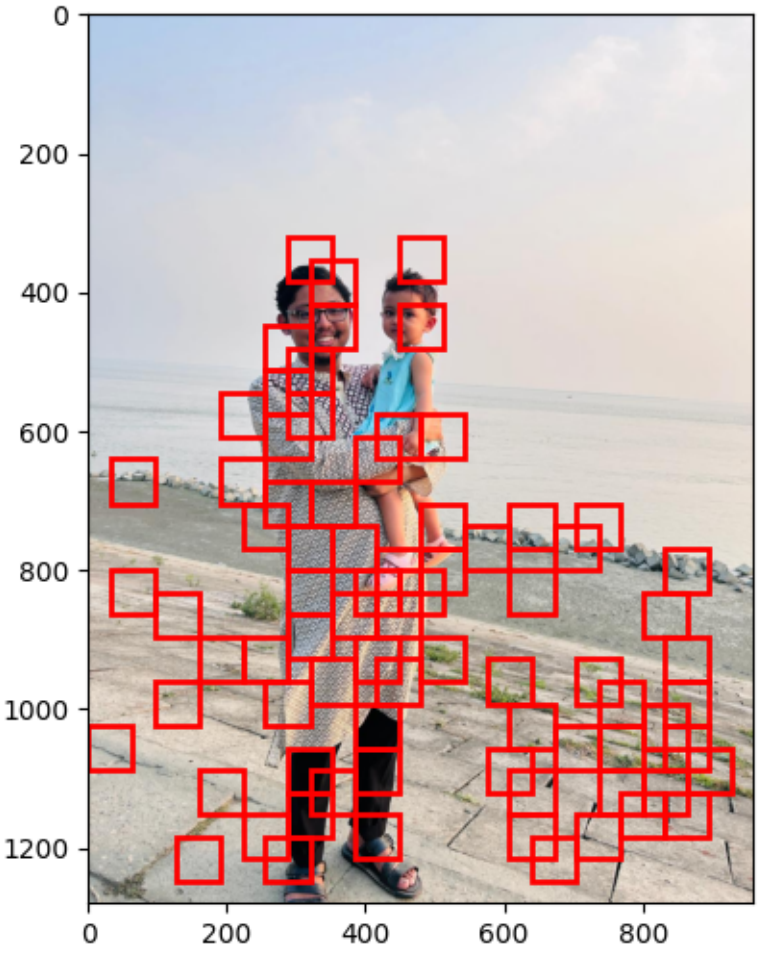
[ ]:
```
def non_max_suppression(boxes, scores, threshold=0.5):
    if len(boxes) == 0:
        return []
```

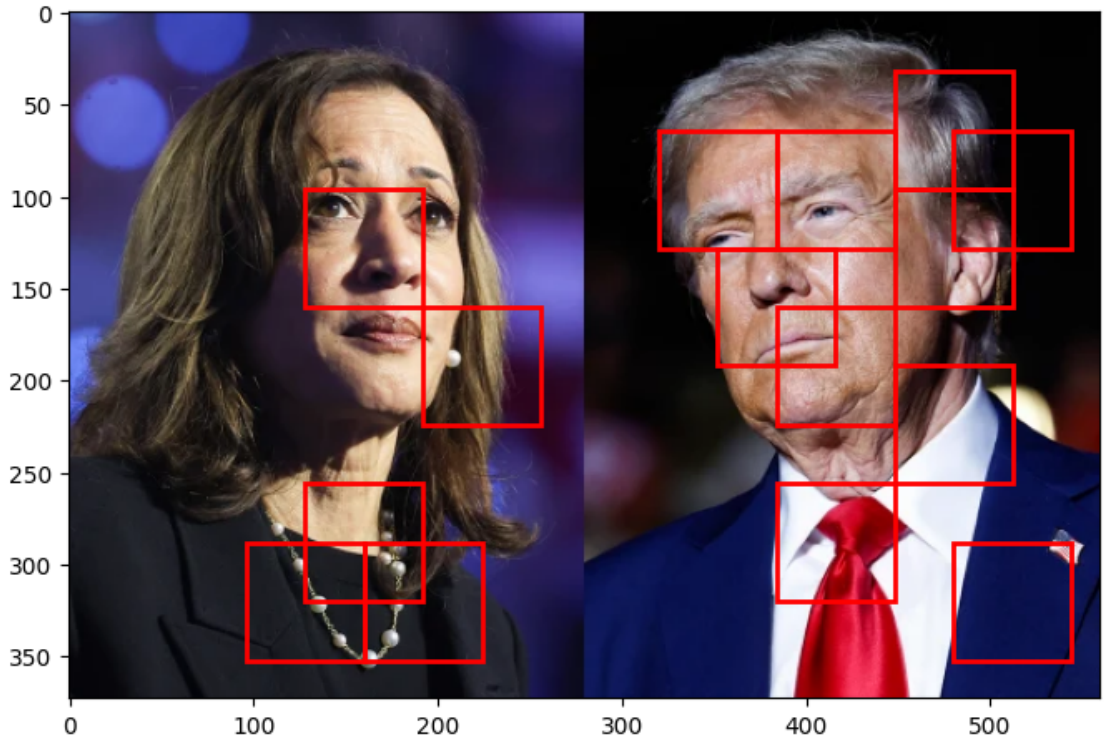
```
    boxes = np.array(boxes)
    scores = np.array(scores)

    x1, y1, x2, y2 = boxes[:, 0], boxes[:, 1], boxes[:, 2], boxes[:, 3]
    areas = (x2 - x1 + 1) * (y2 - y1 + 1)
    order = scores.argsort()[::-1]  # Sort scores in descending order

    keep = []
    while order.size > 0:
        i = order[0]  # Pick the highest score box
        keep.append(i)

        # Compute IoU with other boxes
        xx1 = np.maximum(x1[i], x1[order[1:]])
        yy1 = np.maximum(y1[i], y1[order[1:]])
        xx2 = np.minimum(x2[i], x2[order[1:]])
        yy2 = np.minimum(y2[i], y2[order[1:]])

        w = np.maximum(0, xx2 - xx1 + 1)
        h = np.maximum(0, yy2 - yy1 + 1)
        inter = w * h

        iou = inter / (areas[i] + areas[order[1:]] - inter)
        order = order[np.where(iou <= threshold)[0] + 1]  # Keep boxes with IoU␣
 ↪below threshold

    return keep

# Function to detect faces in an image
def detect_faces(img_path, model, threshold=0.5):
    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)
    h, w = img.shape

    step_size = 32  # Sliding window step size
    window_size = 64  # Detection window size
    detected_boxes = []
    scores = []

    for y in range(0, h - window_size, step_size):
        for x in range(0, w - window_size, step_size):
            patch = img[y:y + window_size, x:x + window_size]
            patch_resized = cv2.resize(patch, (64, 64)).flatten().reshape(1, -1)
            score = model.predict_proba(patch_resized)[0][1]  # Face probability

            if score > threshold:
                detected_boxes.append([x, y, x + window_size, y + window_size])
```

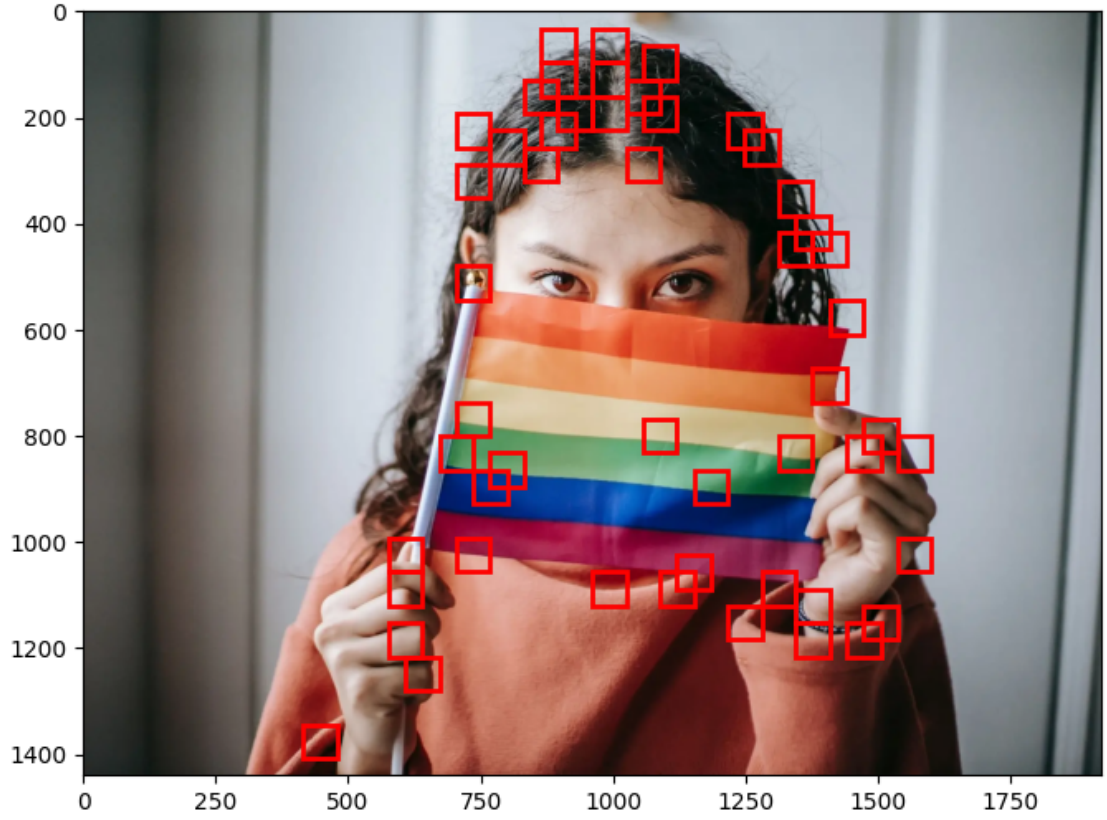
```
                scores.append(score)

    # Apply NMS to remove overlapping detections
    keep_indices = non_max_suppression(detected_boxes, scores, threshold=0.3)
    final_boxes = [detected_boxes[i] for i in keep_indices]

    return final_boxes
```

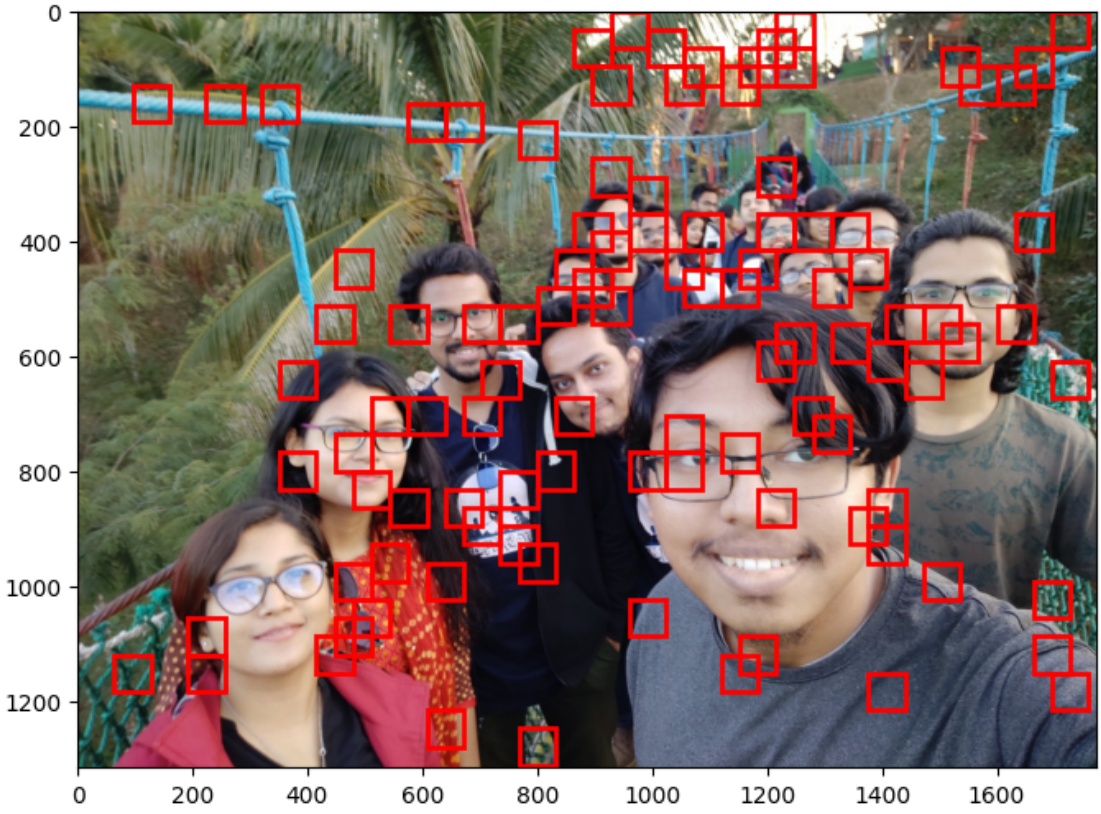
5 Testing SVM model

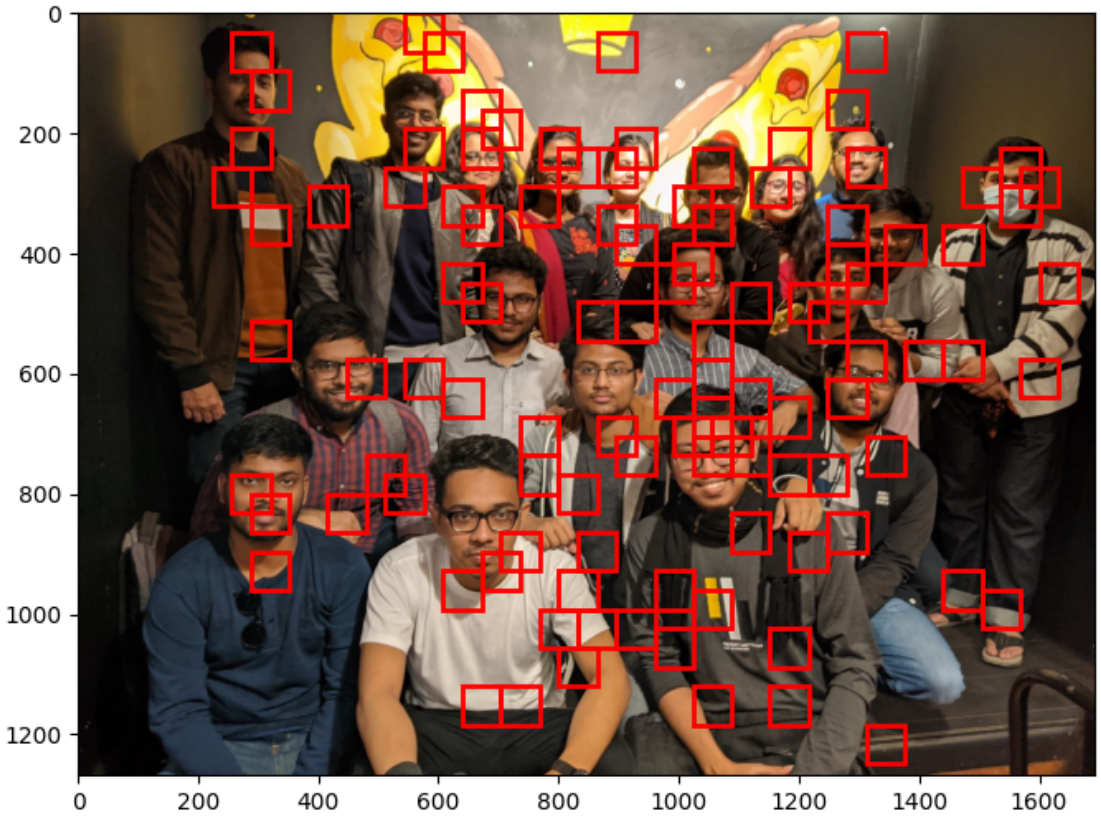
[ ]:
```
#@title
test_dir = "/content/drive/MyDrive/CS216 Files/HW2/Test Images"

for filename in os.listdir(test_dir):
    test_image_path = os.path.join(test_dir, filename)
    detected_faces = detect_faces(test_image_path, svm_model)

    # Plot detection results
    img = cv2.imread(test_image_path, cv2.IMREAD_COLOR)
    fig, ax = plt.subplots(1, figsize=(8, 6))
    ax.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))

    for box in detected_faces:
        x0, y0, x1, y1 = box
        rect = patches.Rectangle((x0, y0), x1 - x0, y1 - y0, linewidth=2,␣
 ↪edgecolor='red', facecolor='none')
        ax.add_patch(rect)

    plt.show()
```

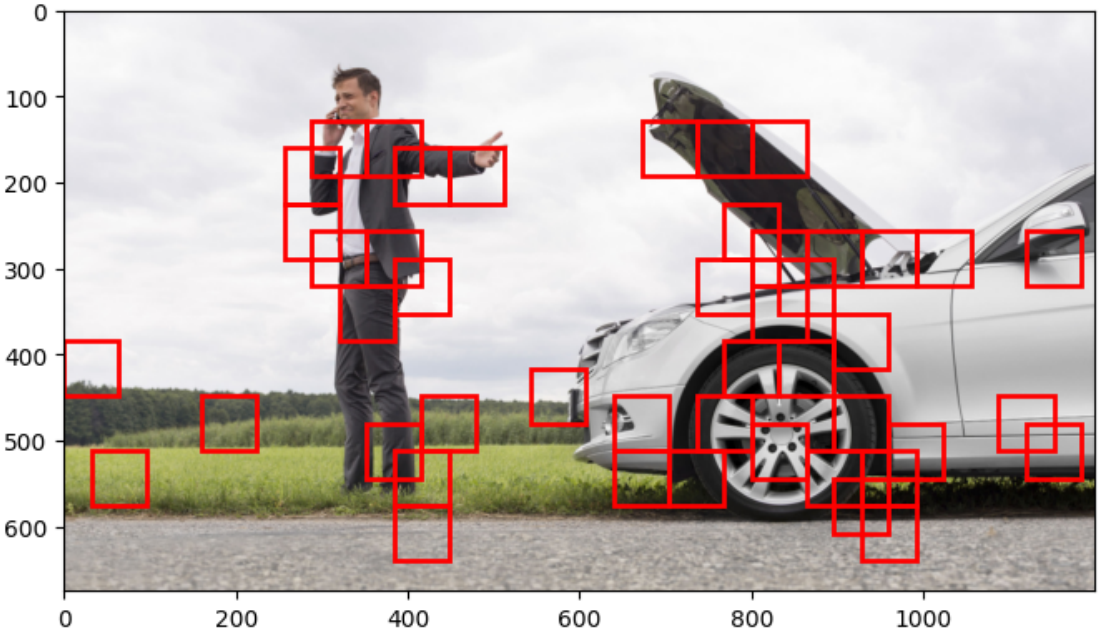

0
200
400
600
800
1000
1200
0
200
400
600
800


0
50
100
150
200
250
300
350
0
100
200
300
400
500


0
200
400
600
800
1000
1200
1400
0
250
500
750
1000
1250
1500
1750

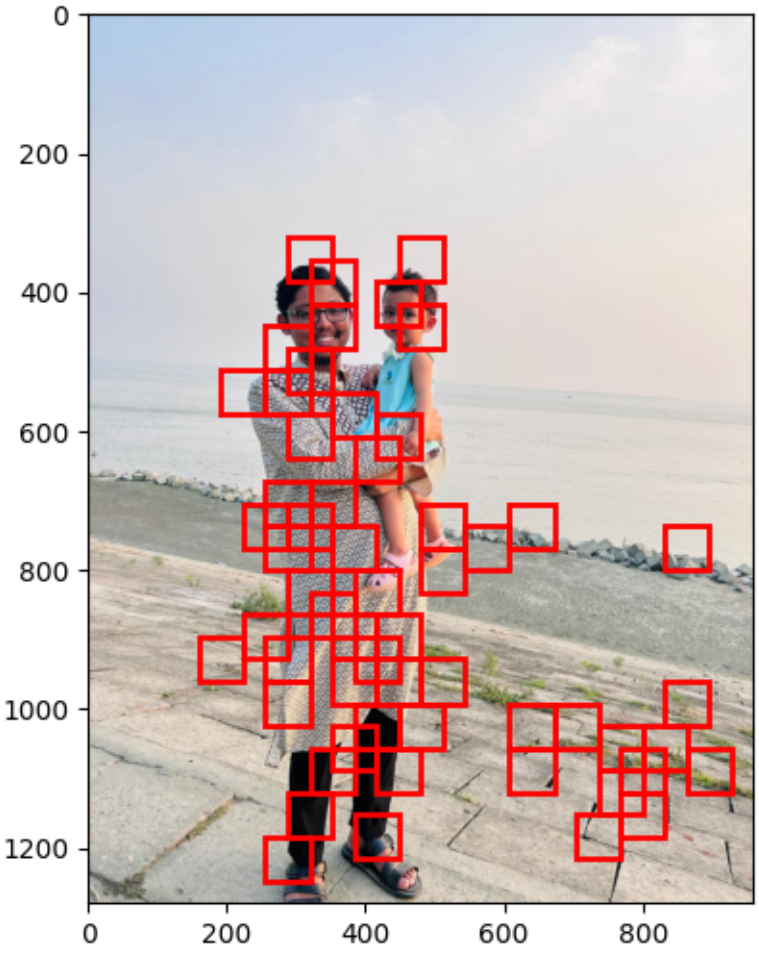

0
200
400
600
800
1000
1200
0
200
400
600
800
1000
1200
1400
1600

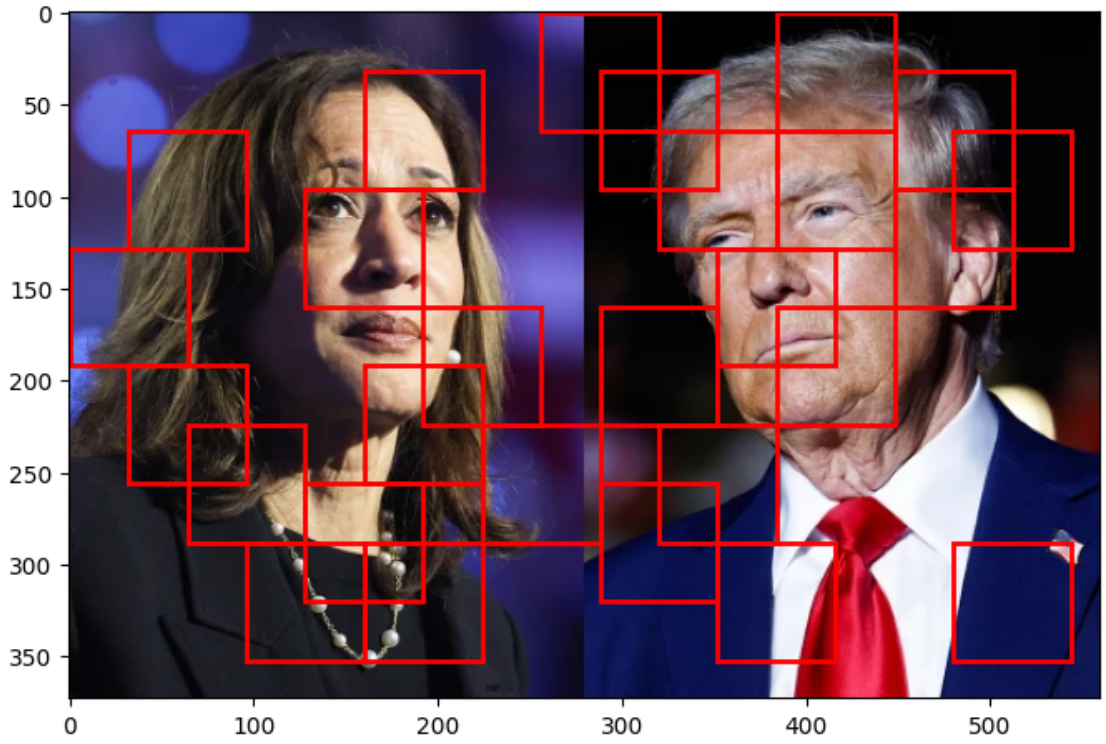

0
200
400
600
800
1000
1200
0
200
400
600
800
1000
1200
1400
1600


0
100
200
300
400
500
600
0
200
400
600
800
1000

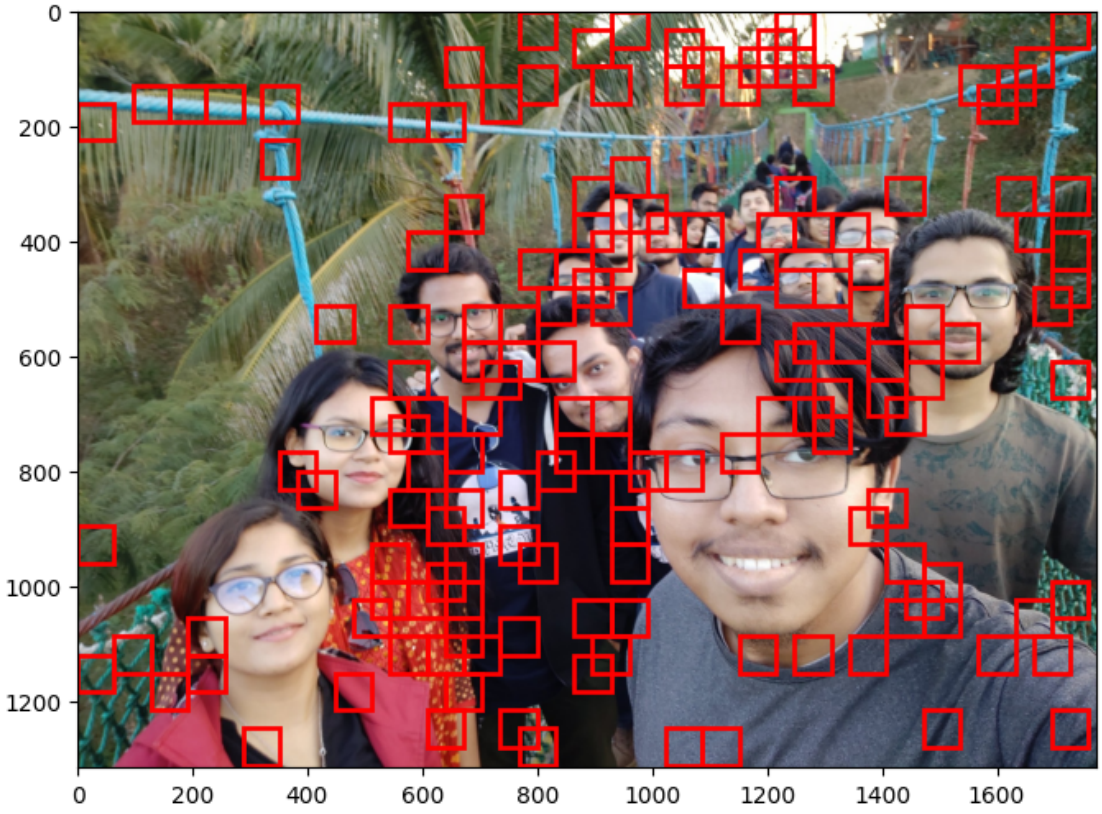
6 Testing Logistic Regression

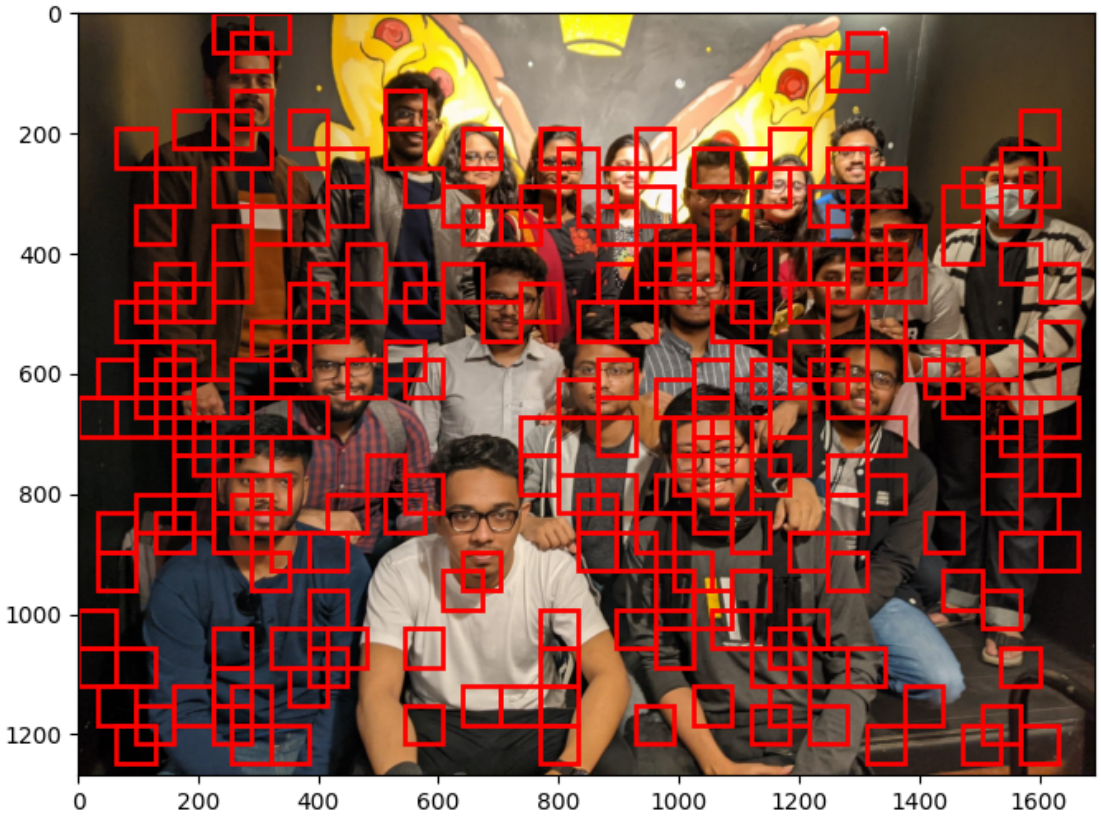
[ ]:
```
#@title
test_dir = "/content/drive/MyDrive/CS216 Files/HW2/Test Images"

for filename in os.listdir(test_dir):
    test_image_path = os.path.join(test_dir, filename)
    detected_faces = detect_faces(test_image_path, logistic_model)

    # Plot detection results
    img = cv2.imread(test_image_path, cv2.IMREAD_COLOR)
    fig, ax = plt.subplots(1, figsize=(8, 6))
    ax.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))

    for box in detected_faces:
        x0, y0, x1, y1 = box
        rect = patches.Rectangle((x0, y0), x1 - x0, y1 - y0, linewidth=2,␣
↪edgecolor='red', facecolor='none')
        ax.add_patch(rect)

    plt.show()
```


0
200
400
600
800
1000
1200
0
200
400
600
800


0
50
100
150
200
250
300
350
0
100
200
300
400
500


0
200
400
600
800
1000
1200
1400
0
250
500
750
1000
1250
1500
1750


0
200
400
600
800
1000
1200
0
200
400
600
800
1000
1200
1400
1600


0
200
400
600
800
1000
1200
0
200
400
600
800
1000
1200
1400
1600


0
100
200
300
400
500
600
0
200
400
600
800
1000

It seems that linear classifier performs very poor in face detection or any object detection task from an image

Matched_Filter_to_Detect_Faces

February 13, 2025

1 Applying Matched Filter to Detect Human Faces

[ ]:
```
import pandas as pd
import numpy as np
import cv2
import matplotlib.pyplot as plt
import os
from PIL import Image
import matplotlib.patches as patches
from scipy.signal import convolve2d
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

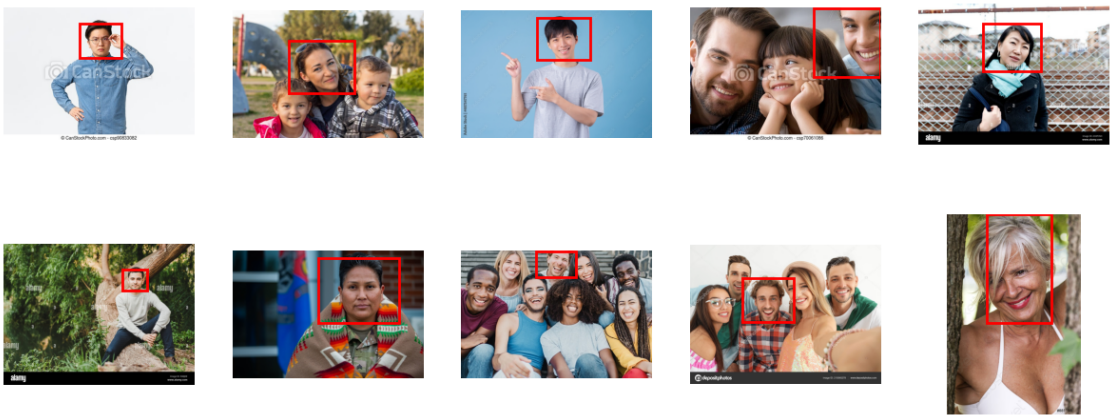
2 Load dataset

[ ]:
```
faces = pd.read_csv('/content/drive/MyDrive/CS216 Files/HW2/Face Data/faces.
 ↪csv')

# Randomly sample 500 images
faces = faces.sample(n=500, random_state=42)

# Directory for images
image_dir = '/content/drive/MyDrive/CS216 Files/HW2/Face Data/images'
```

3 Visualize data

[ ]:
```
sample_images = faces.sample(n=10)

fig, axes = plt.subplots(2, 5, figsize=(15, 6))

for ax, (_, row) in zip(axes.flatten(), sample_images.iterrows()):
    image_path = os.path.join(image_dir, row['image_name'])
    image = Image.open(image_path)
    ax.imshow(image)
```

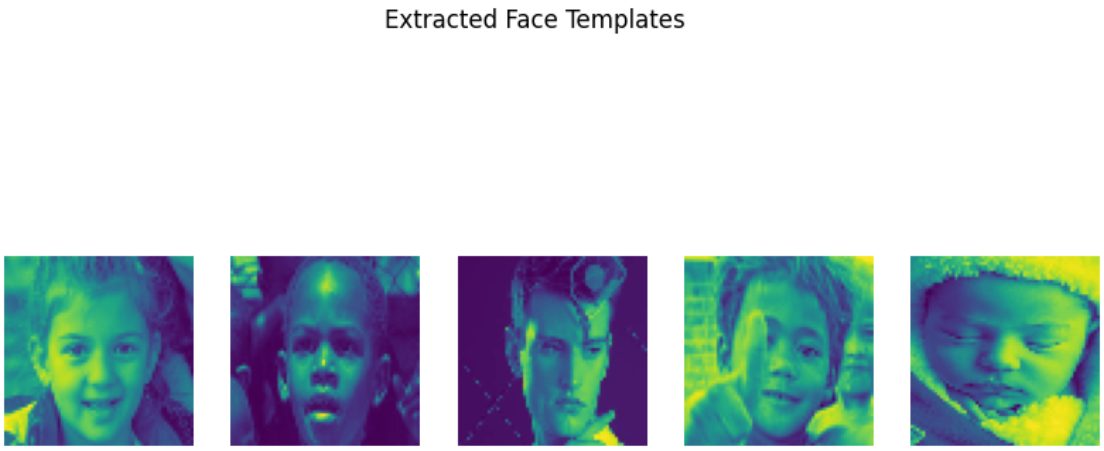
```
    ax.axis('off')

    x0, y0, x1, y1 = row['x0'], row['y0'], row['x1'], row['y1']
    bbox = patches.Rectangle((x0, y0), x1 - x0, y1 - y0, linewidth=2,␣
 ↪edgecolor='red', facecolor='none')
    ax.add_patch(bbox)

plt.show()
```



4 Extract face templates

[ ]:
```
def extract_face_templates(data, img_dir):
    templates = []
    for _, row in data.iterrows():
        img_path = os.path.join(img_dir, row['image_name'])
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        if img is not None:
            face = img[int(row['y0']):int(row['y1']), int(row['x0']):
 ↪int(row['x1'])]
            face_resized = cv2.resize(face, (64, 64))  # Resize to fixed size
            templates.append(cv2.cvtColor(face_resized, cv2.COLOR_BGR2GRAY))  #␣
 ↪Convert to grayscale for filtering
    return templates

# Extract templates
face_templates = extract_face_templates(faces, image_dir)

# Plot a few templates to verify correctness
def plot_templates(templates, num=5):
    plt.figure(figsize=(10, 5))
    for i in range(min(num, len(templates))):
```

```
        plt.subplot(1, num, i + 1)
        plt.imshow(templates[i])
        plt.axis('off')
    plt.suptitle('Extracted Face Templates')
    plt.show()

# Plot first few templates
plot_templates(face_templates)
```



5 Matched Filter Method

[ ]:
```
# Create the matched filter
def create_matched_filter(templates):
    avg_template = np.mean(templates, axis=0)
    normalized_template = (avg_template - np.mean(avg_template)) / np.
 ↪std(avg_template)
    return normalized_template

matched_filter = create_matched_filter(face_templates)

# Apply the matched filter on test images
def apply_matched_filter(img, filter_kernel, threshold=0.5):
    normalized_img = (img - np.mean(img)) / np.std(img)
    response = convolve2d(normalized_img, filter_kernel, mode='same')
    detection = response > threshold * np.max(response)
    return detection, response
```

6 Non-Maximum Suppression with IoU

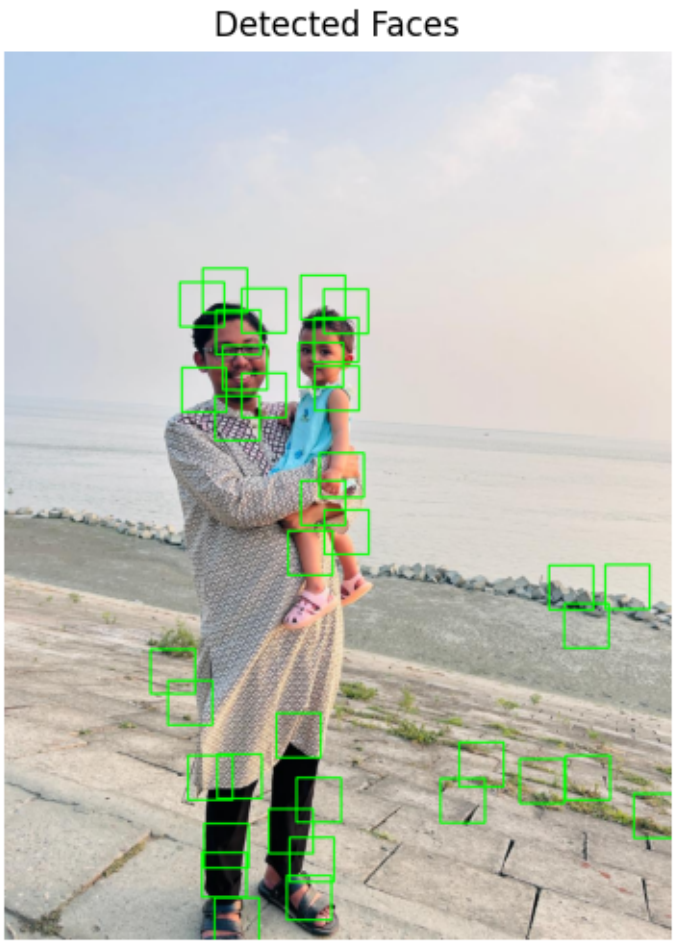
[ ]:
```
def non_max_suppression(detections, response, threshold=0.2, iou_threshold=0.2):
    boxes = [(x, y, response[y, x]) for y in range(detections.shape[0]) for x␣
 ↪in range(detections.shape[1]) if detections[y, x]]
    boxes = sorted(boxes, key=lambda b: b[2], reverse=True)  # Sort by␣
 ↪confidence
    selected_boxes = []

    def iou(box1, box2, size=64):
        x1, y1 = box1[:2]
        x2, y2 = box2[:2]
        xi1, yi1 = max(x1, x2), max(y1, y2)
        xi2, yi2 = min(x1 + size, x2 + size), min(y1 + size, y2 + size)
        inter_area = max(0, xi2 - xi1) * max(0, yi2 - yi1)
        union_area = 2 * (size * size) - inter_area
        return inter_area / union_area

    while boxes:
        best_box = boxes.pop(0)
        selected_boxes.append(best_box)
        boxes = [box for box in boxes if iou(best_box, box) < iou_threshold]

    return selected_boxes
```

7 Test on new dataset

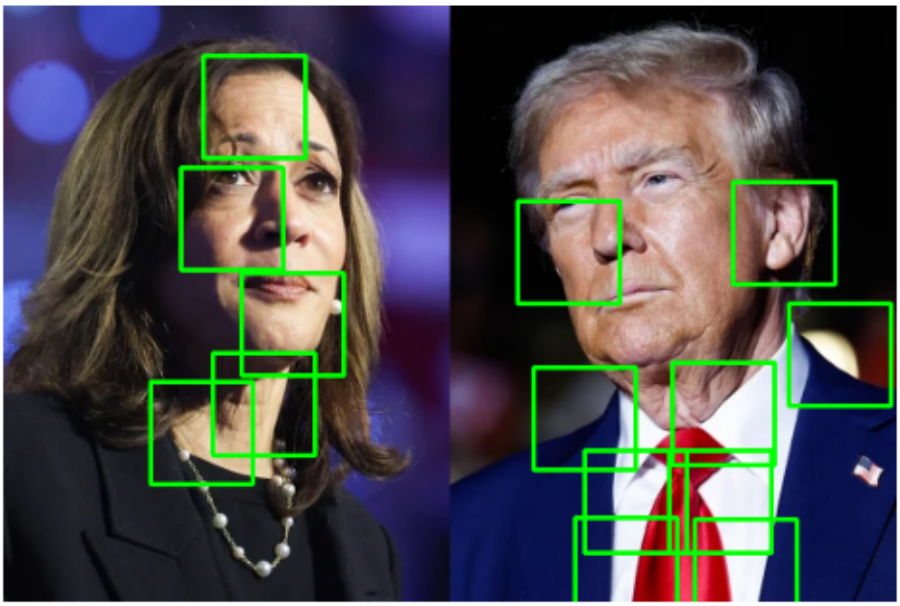
[ ]:
```
def test_matched_filter(test_dir, filter_kernel, threshold=0.5):
    test_images = [f for f in os.listdir(test_dir)]
    for image_name in test_images:
        img_path = os.path.join(test_dir, image_name)
        img = cv2.imread(img_path, cv2.IMREAD_COLOR)
        if img is None:
            continue

        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        detections, response = apply_matched_filter(gray, filter_kernel,␣
 ↪threshold)

        nms_detections = non_max_suppression(detections, response, 0.2)

        for x, y, _ in nms_detections:
            cv2.rectangle(img, (x - 32, y - 32), (x + 32, y + 32), (0, 255, 0),␣
 ↪2)

        plt.figure(figsize=(6, 6))
```

```
        plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
        plt.title("Detected Faces")
        plt.axis('off')
        plt.show()

# Test directory
test_dir = "/content/drive/MyDrive/CS216 Files/HW2/Test Images"

# Test the matched filter
print("Testing the matched filter on test images:")
test_matched_filter(test_dir, matched_filter)
```

```
Testing the matched filter on test images:
```



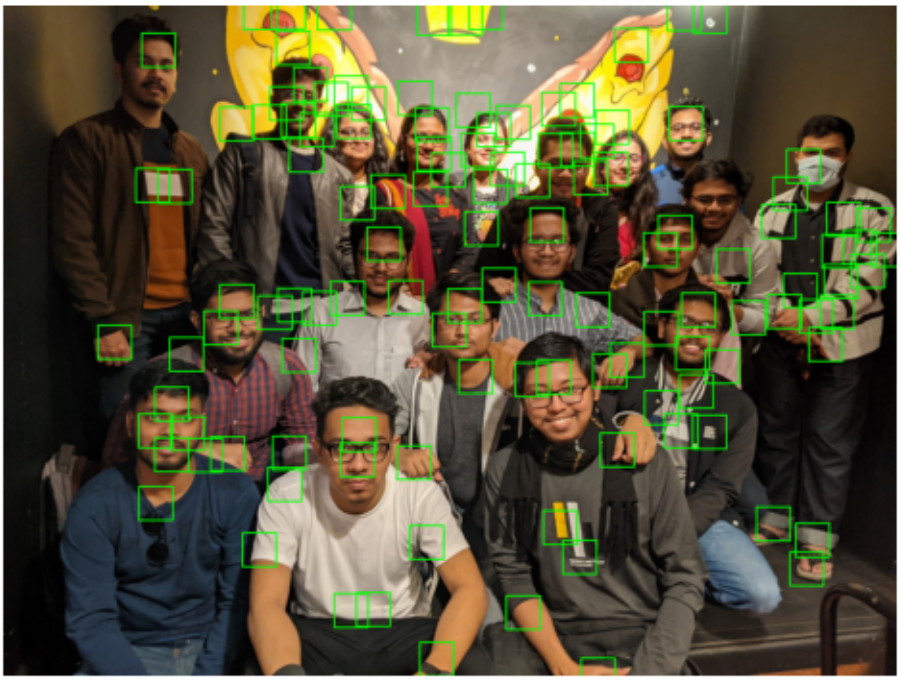
Detected Faces

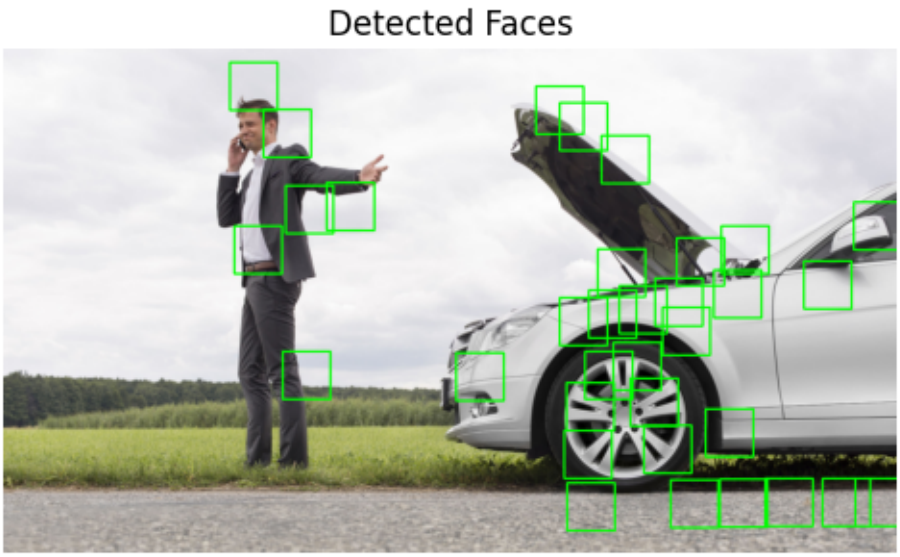


Detected Faces

Detected Faces

Detected Faces





As we can see, similar to linear classifier, it doesn't perform well on face detection as well.

Neural_Network(Fast_RCNN)_to_Detect_Faces

February 13, 2025

1 Applying Neural Network to Detect Human Faces

[ ]:
```
import torch
import pandas as pd
import shutil
import os
import cv2
import numpy as np
import glob as glob
from torch.utils.data import Dataset, DataLoader
import matplotlib.pyplot as plt
import random
```

[ ]:
```
BATCH_SIZE = 4 # increase / decrease according to GPU memeory
RESIZE_TO = 512 # resize the image for training and transforms

DEVICE = torch.device('cuda') if torch.cuda.is_available() else torch.
 ↪device('cpu')

CLASSES = ['background', 'Face']
NUM_CLASSES = 2
# check if cuda (GPU) is available
torch.cuda.is_available()
```

[ ]:
```
True
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

2 Helper methods to load data

[ ]:
```
#@title
def train_valid_test_split(faces_csv=None, split=0.15):
    all_df = pd.read_csv(faces_csv)
```

```
    all_df = all_df.sample(n=500, random_state=7)

    all_df.sample(frac=1)
    len_df = len(all_df)

    trainTest_split = int((1-split)*len_df)

    trainVal_df = all_df[:trainTest_split]
    test_df = all_df[trainTest_split:]

    lenTV_df = len(trainVal_df)

    trainVal_split = int((1-split)*lenTV_df)

    train_df = trainVal_df[:trainVal_split]
    valid_df = trainVal_df[trainVal_split:]

    return train_df, valid_df, test_df

class Averager:
    def __init__(self):
        self.current_total = 0.0
        self.iterations = 0.0

    def send(self, value):
        self.current_total += value
        self.iterations += 1

    @property
    def value(self):
        if self.iterations == 0:
            return 0
        else:
            return 1.0 * self.current_total / self.iterations

    def reset(self):
        self.current_total = 0.0
        self.iterations = 0.0


def collate_fn(batch):
    """
    To handle the data loading as different images may have different number
    of objects and to handle varying size tensors as well.
    """
    return tuple(zip(*batch))
```

```
# Creating the dataset class
class Faces(Dataset):
    def __init__(self, dataset, width, height, dir_path="/content/drive/MyDrive/
 ↪CS216 Files/HW2/Face Data/images"):
        self.dir_path = dir_path
        self.height = height
        self.width = width
        self.dataset = dataset

        # copy image names to list
        self.set_image_names = self.dataset['image_name'].tolist()

        # get all the image names in sorted order
        self.image_paths = glob.glob(f"{self.dir_path}/*.jpg")
        self.all_images = [image_path.split('/')[-1] for image_path in self.
 ↪image_paths]
        self.all_images = sorted(self.all_images)

        # cut down to only images present in dataset
        self.images = []

        for i in self.set_image_names:
            for j in self.all_images:
                if i == j:
                    self.images.append(i)

    def __getitem__(self, idx):

        # capture the image name and the full image path
        # print(idx)
        # print(len(self.images))
        image_name = self.images[idx]
        image_path = os.path.join(self.dir_path, image_name)

        # read the image
        image = cv2.imread(image_path)

        # convert BGR to RGB color format and resize
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB).astype(np.float32)
        image_resized = cv2.resize(image, (self.width, self.height))
        image_resized /= 255.0

        # channel first transposing
        image_resized = np.transpose(image_resized, (2, 0, 1))

        boxes = []
        labels = []
```

```
        # Copy bounding box coordinates and image dimensions
        filtered_df = self.dataset.loc[self.dataset['image_name'] == image_name]

        for i in range(len(filtered_df)):

            # xmax = left corner x-coordinates
            xmin = int(filtered_df['x0'].iloc[i])
            # xmax = right corner x-coordinates
            xmax = int(filtered_df['x1'].iloc[i])
            # ymin = left corner y-coordinates
            ymin = int(filtered_df['y0'].iloc[i])
            # ymax = right corner y-coordinates
            ymax = int(filtered_df['y1'].iloc[i])

            image_width = int(filtered_df['width'].iloc[i])
            image_height = int(filtered_df['height'].iloc[i])

            # resize the bounding boxes according to the...
            # ... desired `width`, `height`
            xmin_final = (xmin/image_width)*self.width
            xmax_final = (xmax/image_width)*self.width
            ymin_final = (ymin/image_height)*self.height
            yamx_final = (ymax/image_height)*self.height

            boxes.append([xmin_final, ymin_final, xmax_final, yamx_final])
            labels.append(1) # 1 because there is only one class

        # bounding box to tensor
        boxes = torch.as_tensor(boxes, dtype=torch.float32)

        # area of the bounding boxes
        area = (boxes[:, 3] - boxes[:, 1]) * (boxes[:, 2] - boxes[:, 0])

        # crowd instances
        if boxes.shape[0] > 1:
            iscrowd = torch.ones((boxes.shape[0],), dtype=torch.int64)
        else:
            iscrowd = torch.zeros((boxes.shape[0],), dtype=torch.int64)

        # label to tensor
        labels = torch.as_tensor(labels, dtype=torch.int64)

        # prepare the final `target` dictionary
        target = {}
        target["boxes"] = boxes
        target["labels"] = labels
```

```
        target["area"] = area
        target["iscrowd"] = iscrowd
        image_id = torch.tensor([idx])
        target["image_id"] = image_id

        return image_resized, target

    def __len__(self):
        return len(self.set_image_names)
```

3 load data

[ ]:
```
#@title
train_df, valid_df, test_df = train_valid_test_split(faces_csv='/content/drive/
 ↪MyDrive/CS216 Files/HW2/Face Data/faces.csv')
train_dataset = Faces(train_df, RESIZE_TO, RESIZE_TO)
valid_dataset = Faces(valid_df, RESIZE_TO, RESIZE_TO)

# defining train and validation sets data loaders

train_loader = DataLoader(
    train_dataset,
    batch_size=BATCH_SIZE,
    shuffle=True,
    num_workers=0,
    collate_fn=collate_fn
)
valid_loader = DataLoader(
    valid_dataset,
    batch_size=BATCH_SIZE,
    shuffle=False,
    num_workers=0,
    collate_fn=collate_fn
)
```

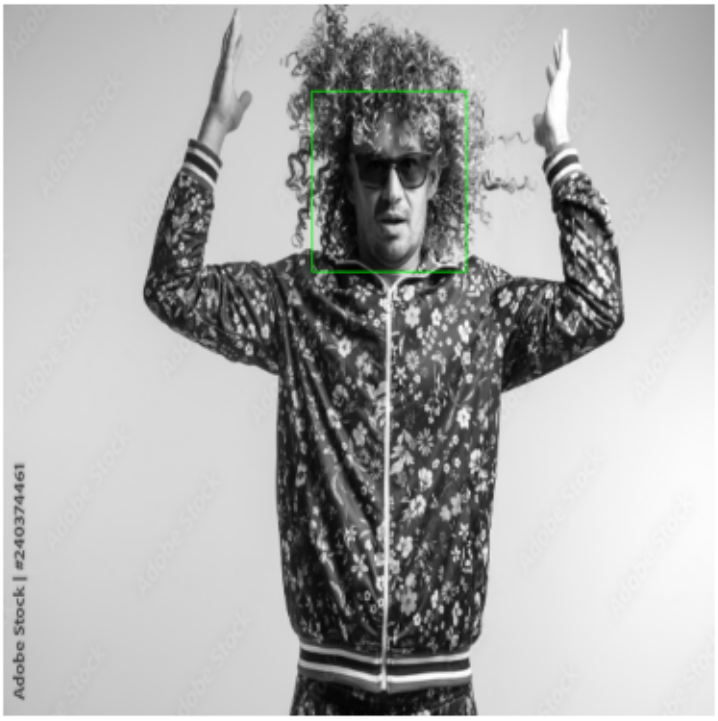
4 Visualize data

[ ]:
```
#@title
def visualize_sample(image, target):
    for box in target['boxes']:
        cv2.rectangle(
            image,
            (int(box[0]), int(box[1])), (int(box[2]), int(box[3])),
            (0, 255, 0), 1
        )
```

```
    plt.imshow(image)
    plt.axis('off')
    plt.show()


for i in range(3):
    image, target = random.choice(train_dataset)
    image = np.transpose(image, (1, 2, 0))
    visualize_sample(image, target)
```

```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with
RGB data ([0..1] for floats or [0..255] for integers). Got range [0.0..255.0].
```


```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with
RGB data ([0..1] for floats or [0..255] for integers). Got range [0.0..255.0].
```




```
WARNING:matplotlib.image:Clipping input data to the valid range for imshow with
RGB data ([0..1] for floats or [0..255] for integers). Got range [0.0..255.0].
```

[ ]:
```
#@title Defining the model(FastRCNN)
import torchvision
from torchvision.models.detection.faster_rcnn import FastRCNNPredictor
import torch

def create_model(num_classes, model_path=None):
    # Load Faster RCNN pre-trained model
    model = torchvision.models.detection.
 ↪fasterrcnn_resnet50_fpn(weights='DEFAULT')

    # Get the number of input features
    in_features = model.roi_heads.box_predictor.cls_score.in_features
    # Define a new head for the detector with the required number of classes
    model.roi_heads.box_predictor = FastRCNNPredictor(in_features, num_classes)

    # If a model path is provided, load the model weights
    if model_path:
        model.load_state_dict(torch.load(model_path, map_location=torch.
 ↪device('cpu')))

    return model
```

5 Fast RCNN

6 function for running training iterations

[ ]:
```
#@title
def train(train_data_loader, model):
    print('Training')
    global train_itr
    global train_loss_list

     # initialize tqdm progress bar
    prog_bar = tqdm(train_data_loader, total=len(train_data_loader))

    for i, data in enumerate(prog_bar):
        optimizer.zero_grad()
        images, targets = data

        images = list(torch.from_numpy(image).to(DEVICE) for image in images)
        targets = [{k: v.to(DEVICE) for k, v in t.items()} for t in targets]
        loss_dict = model(images, targets)
        losses = sum(loss for loss in loss_dict.values())
```

```
        loss_value = losses.item()
        train_loss_list.append(loss_value)
        train_loss_hist.send(loss_value)
        losses.backward()
        optimizer.step()
        train_itr += 1

        # update the loss value beside the progress bar for each iteration
        prog_bar.set_description(desc=f"Loss: {loss_value:.4f}")
    return train_loss_list

def validate(valid_data_loader, model):
    print('Validating')
    global val_itr
    global val_loss_list

    # initialize tqdm progress bar
    prog_bar = tqdm(valid_data_loader, total=len(valid_data_loader))

    for i, data in enumerate(prog_bar):
        images, targets = data

        images = list(torch.from_numpy(image).to(DEVICE) for image in images)
        targets = [{k: v.to(DEVICE) for k, v in t.items()} for t in targets]

        with torch.no_grad():
            loss_dict = model(images, targets)
        losses = sum(loss for loss in loss_dict.values())
        loss_value = losses.item()
        val_loss_list.append(loss_value)
        val_loss_hist.send(loss_value)
        val_itr += 1
        # update the loss value beside the progress bar for each iteration
        prog_bar.set_description(desc=f"Loss: {loss_value:.4f}")
    return val_loss_list
```

7 training the model

[ ]:
```
#@title

from tqdm.auto import tqdm
import time

plt.style.use('ggplot')
```

```
model_path = '/content/drive/MyDrive/CS216 Files/HW2/Face Data/model10.pth'
model = create_model(num_classes=NUM_CLASSES, model_path=model_path)

NUM_EPOCHS = 10 # number of epochs to train for

# initialize the model and move to the computation device
model = create_model(num_classes=NUM_CLASSES)
model = model.to(DEVICE)
# get the model parameters
params = [p for p in model.parameters() if p.requires_grad]
# define the optimizer
optimizer = torch.optim.SGD(params, lr=0.001, momentum=0.9, weight_decay=0.0005)
# initialize the Averager class
train_loss_hist = Averager()
val_loss_hist = Averager()
train_itr = 1
val_itr = 1
# train and validation loss lists to store loss values of all...
# ... iterations till ena and plot graphs for all iterations
train_loss_list = []
val_loss_list = []
# name to save the trained model with
MODEL_NAME = 'model'

# start the training epochs
for epoch in range(NUM_EPOCHS):
    print(f"\nEPOCH {epoch+1} of {NUM_EPOCHS}")

    # reset the training and validation loss histories for the current epoch
    train_loss_hist.reset()
    val_loss_hist.reset()

    # start timer and carry out training and validation
    start = time.time()
    train_loss = train(train_loader, model)
    val_loss = validate(valid_loader, model)

    print(f"Epoch #{epoch+1} train loss: {train_loss_hist.value:.3f}")
    print(f"Epoch #{epoch+1} validation loss: {val_loss_hist.value:.3f}")
    end = time.time()
    print(f"Took {((end - start) / 60):.3f} minutes for epoch {epoch+1}")

    if (epoch+1) == NUM_EPOCHS: # save loss plots and model once at the end
        # create two subplots, one for each, training and validation
        figure_1, train_ax = plt.subplots()
        figure_2, valid_ax = plt.subplots()
        train_ax.plot(train_loss, color='blue')
```

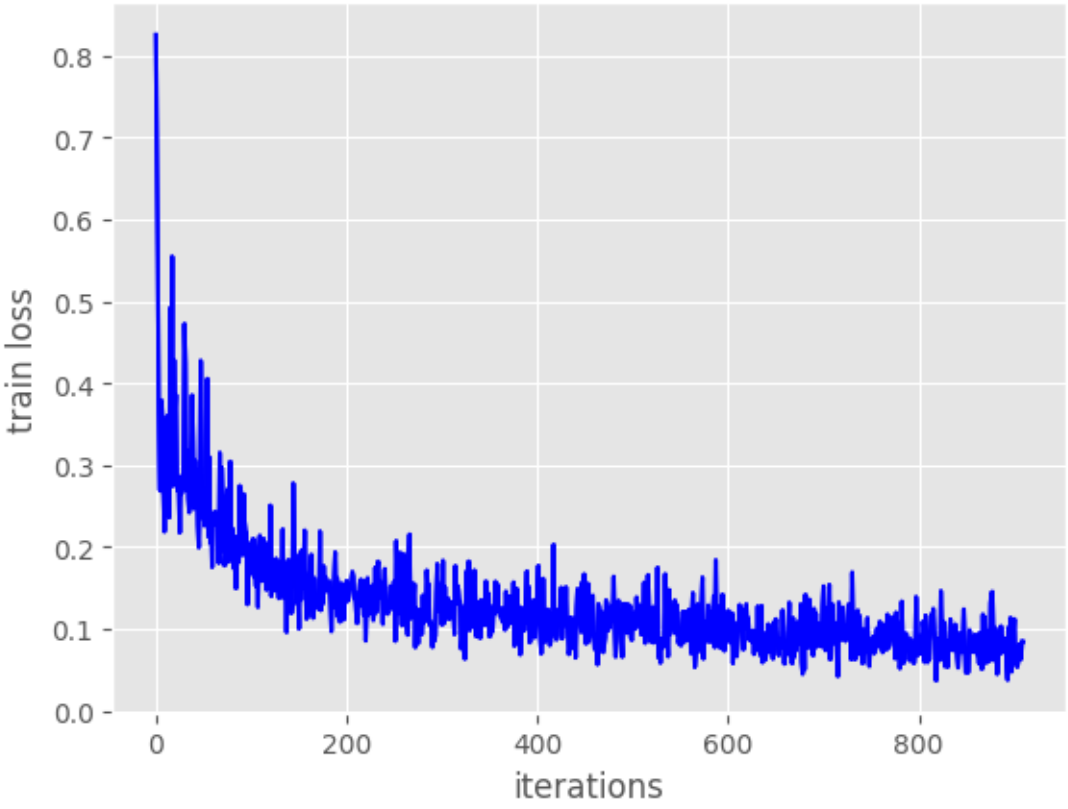
```
        train_ax.set_xlabel('iterations')
        train_ax.set_ylabel('train loss')
        valid_ax.plot(val_loss, color='red')
        valid_ax.set_xlabel('iterations')
        valid_ax.set_ylabel('validation loss')
        figure_1.savefig(f"/content/drive/MyDrive/CS216 Files/HW2/Face Data/
 ↪train_loss_{epoch+1}.png")
        figure_2.savefig(f"/content/drive/MyDrive/CS216 Files/HW2/Face Data/
 ↪valid_loss_{epoch+1}.png")
        torch.save(model.state_dict(), f"/content/drive/MyDrive/CS216 Files/HW2/
 ↪Face Data/model{epoch+1}.pth")

train_ax.plot(train_loss, color='blue')
train_ax.set_xlabel('iterations')
train_ax.set_ylabel('train loss')

valid_ax.plot(val_loss, color='red')
valid_ax.set_xlabel('iterations')
valid_ax.set_ylabel('validation loss')
plt.show()
```

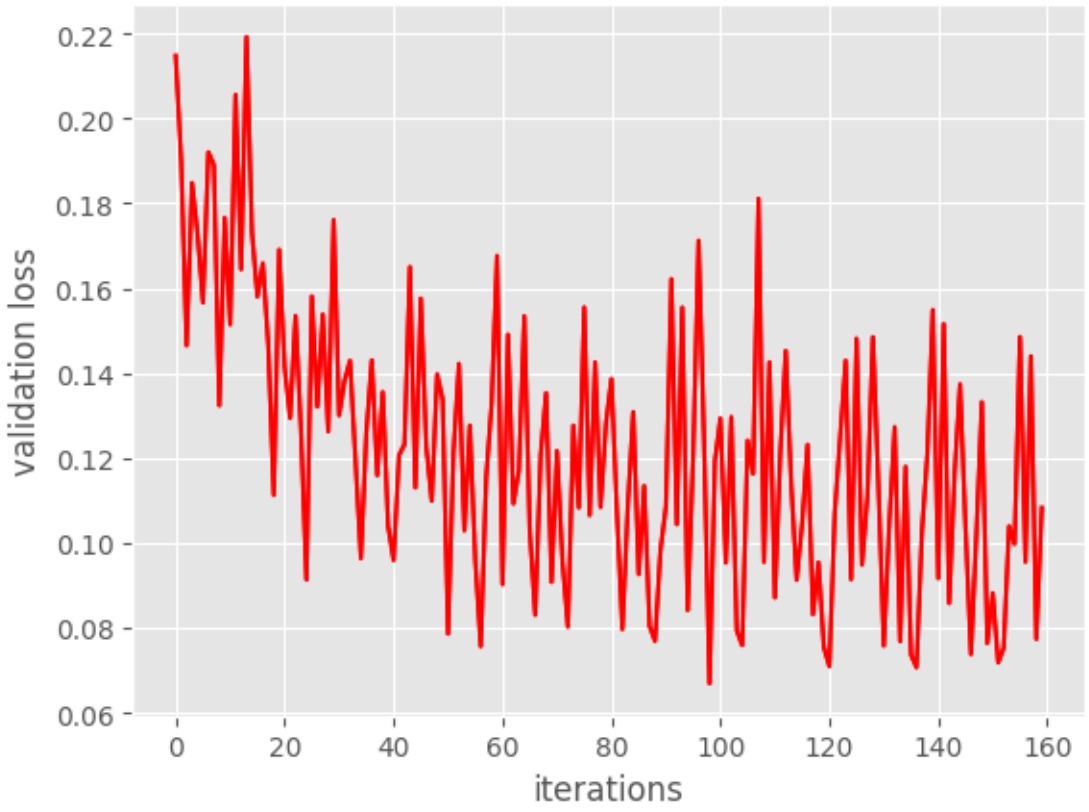
```
<ipython-input-7-7064b65388ef>:17: FutureWarning: You are using `torch.load`
with `weights_only=False` (the current default value), which uses the default
pickle module implicitly. It is possible to construct malicious pickle data
which will execute arbitrary code during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-models for
more details). In a future release, the default value for `weights_only` will be
flipped to `True`. This limits the functions that could be executed during
unpickling. Arbitrary objects will no longer be allowed to be loaded via this
mode unless they are explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control of the
loaded file. Please open an issue on GitHub for any issues related to this
experimental feature.
  model.load_state_dict(torch.load(model_path,
map_location=torch.device('cpu')))


EPOCH 1 of 10
Training
  0%|          | 0/91 [00:00<?, ?it/s]
Validating
  0%|          | 0/16 [00:00<?, ?it/s]
Epoch #1 train loss: 0.289
Epoch #1 validation loss: 0.177
Took 1.317 minutes for epoch 1
```

```
EPOCH 2 of 10
Training
  0%|          | 0/91 [00:00<?, ?it/s]
Validating
  0%|          | 0/16 [00:00<?, ?it/s]
Epoch #2 train loss: 0.168
Epoch #2 validation loss: 0.141
Took 1.304 minutes for epoch 2

EPOCH 3 of 10
Training
  0%|          | 0/91 [00:00<?, ?it/s]
Validating
  0%|          | 0/16 [00:00<?, ?it/s]
Epoch #3 train loss: 0.141
Epoch #3 validation loss: 0.124
Took 1.299 minutes for epoch 3

EPOCH 4 of 10
Training
  0%|          | 0/91 [00:00<?, ?it/s]
Validating
  0%|          | 0/16 [00:00<?, ?it/s]
Epoch #4 train loss: 0.124
Epoch #4 validation loss: 0.119
Took 1.304 minutes for epoch 4

EPOCH 5 of 10
Training
  0%|          | 0/91 [00:00<?, ?it/s]
Validating
  0%|          | 0/16 [00:00<?, ?it/s]
Epoch #5 train loss: 0.114
Epoch #5 validation loss: 0.116
Took 1.302 minutes for epoch 5

EPOCH 6 of 10
Training
```

```
  0%|          | 0/91 [00:00<?, ?it/s]
Validating
  0%|          | 0/16 [00:00<?, ?it/s]
Epoch #6 train loss: 0.107
Epoch #6 validation loss: 0.110
Took 1.306 minutes for epoch 6

EPOCH 7 of 10
Training
  0%|          | 0/91 [00:00<?, ?it/s]
Validating
  0%|          | 0/16 [00:00<?, ?it/s]
Epoch #7 train loss: 0.100
Epoch #7 validation loss: 0.116
Took 1.301 minutes for epoch 7

EPOCH 8 of 10
Training
  0%|          | 0/91 [00:00<?, ?it/s]
Validating
  0%|          | 0/16 [00:00<?, ?it/s]
Epoch #8 train loss: 0.093
Epoch #8 validation loss: 0.108
Took 1.300 minutes for epoch 8

EPOCH 9 of 10
Training
  0%|          | 0/91 [00:00<?, ?it/s]
Validating
  0%|          | 0/16 [00:00<?, ?it/s]
Epoch #9 train loss: 0.087
Epoch #9 validation loss: 0.108
Took 1.304 minutes for epoch 9

EPOCH 10 of 10
Training
  0%|          | 0/91 [00:00<?, ?it/s]
Validating
  0%|          | 0/16 [00:00<?, ?it/s]
```

```
Epoch #10 train loss: 0.080
Epoch #10 validation loss: 0.102
Took 1.300 minutes for epoch 10
```

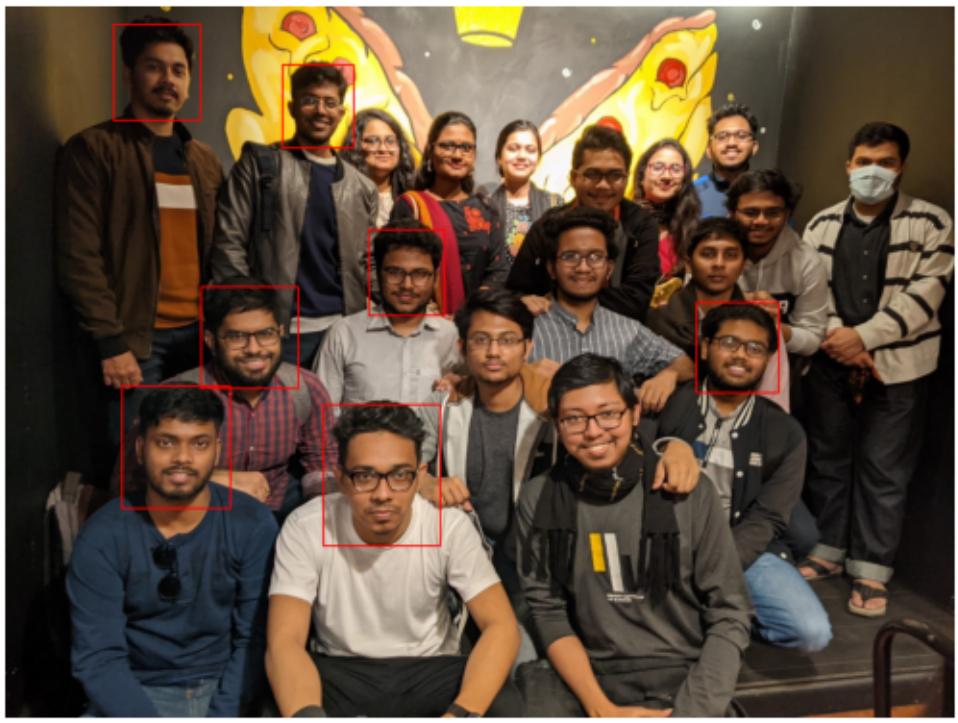




8 Testing on New Dataset

[ ]:
```
#@title
test_images = []
test_dir = "/content/drive/MyDrive/CS216 Files/HW2/Test Images"
test_images = [f for f in os.listdir(test_dir)]

model.eval()
model = model.to(DEVICE)
detection_threshold = 0.8

for image_name in test_images:
    image = cv2.imread(os.path.join(test_dir,image_name))
    orig_image = image.copy()
    # BGR to RGB
    image = cv2.cvtColor(orig_image, cv2.COLOR_BGR2RGB).astype(np.float32)
    # make the pixel range between 0 and 1
    image /= 255.0
    # bring color channels to front
    image = np.transpose(image, (2, 0, 1)).astype(float)
```

```
    # convert to tensor
    image = torch.tensor(image, dtype=torch.float).cuda()
    # add batch dimension
    image = torch.unsqueeze(image, 0)
    with torch.no_grad():
        outputs = model(image)

    # load all detection to CPU for further operations
    outputs = [{k: v.to('cpu') for k, v in t.items()} for t in outputs]
    # carry further only if there are detected boxes
    if len(outputs[0]['boxes']) != 0:
        boxes = outputs[0]['boxes'].data.numpy()
        scores = outputs[0]['scores'].data.numpy()
        # filter out boxes according to `detection_threshold`
        boxes = boxes[scores >= detection_threshold].astype(np.int32)
        draw_boxes = boxes.copy()

        # draw the bounding boxes
        for j, box in enumerate(draw_boxes):
            cv2.rectangle(orig_image,
                          (int(box[0]), int(box[1])),
                          (int(box[2]), int(box[3])),
                          (0, 0, 255), 2)
        image_rgb = cv2.cvtColor(orig_image, cv2.COLOR_BGR2RGB)
        plt.imshow(image_rgb)
        plt.axis('off')
        plt.show()

print('TEST PREDICTIONS COMPLETE')
```



```
TEST PREDICTIONS COMPLETE
```

As we can see, even though we trained with only 500 samples and for only 10 epochs, the neural network does brilliant on new unseen images of people than the linear classifiers and matched filters