NEWS BROADCAST ANALYSIS

Christine K. C. Cheng

1 Shot detection

Sum of absolute differences

A score is assigned to each frame by calculating the sum of the absolute differences between consecutive frames for every pixel. This value is then normalized by the size of the frame. A threshold is selected through empirical experiments before score calculations. When the score of a frame is greater than the threshold, a shot change is declared. This method works quite well with simple videos but it is not robust against movements and changes in lighting.

Histogram differences

Each frame is converted into a gray-scale image. A histogram with 256 bins, representing all the possible values of a pixel, is created for each frame. Then, a score is assigned by calculating the sum of the absolute differences between histograms of consecutive frames.

Figure 1: Sum of absolute differences scores of clip 1

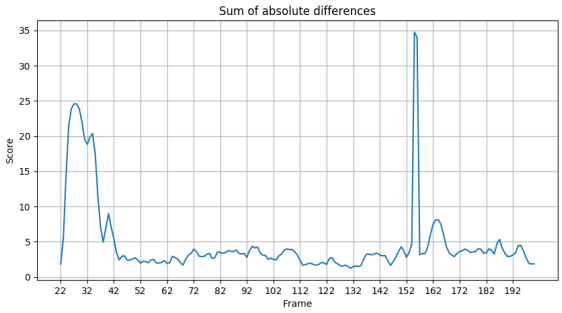


Figure 2: Histogram differences scores of clip 1

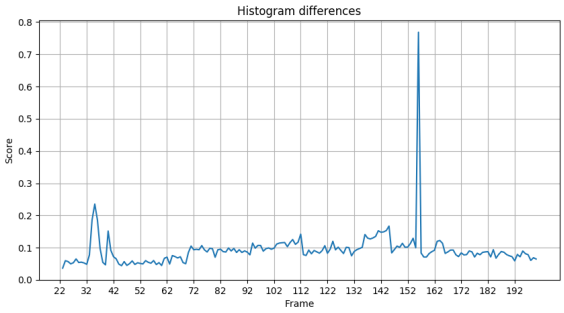


Figure 3: Sum of absolute differences scores of clip 2

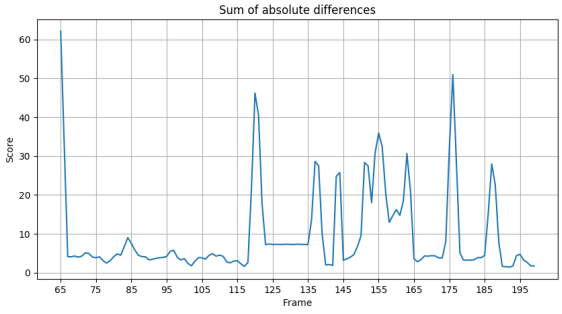


Figure 4: Histogram differences scores of clip 2

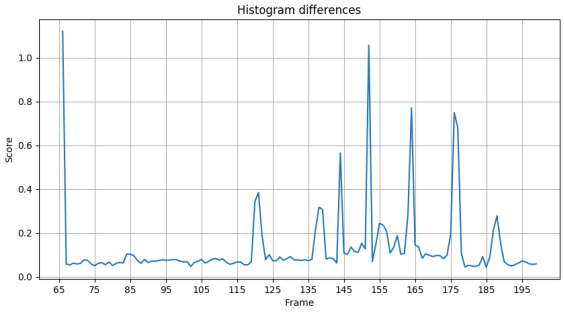


Figure 5: Sum of absolute differences scores of clip 3

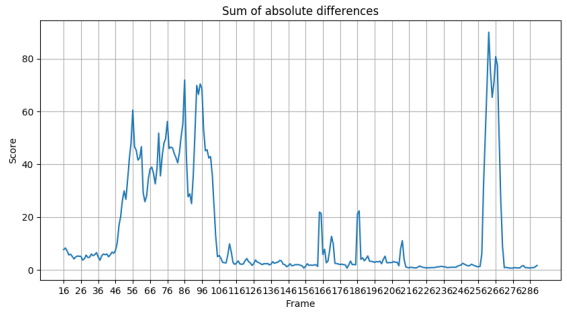
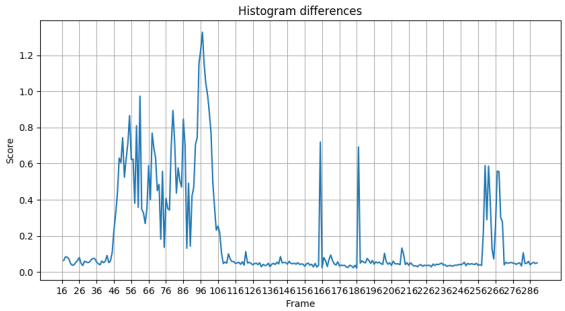


Figure 6: Histogram differences scores of clip 3



The relevant code is in `shot.py`. To get the graphs of shot detection, run the command below.

```
python3 run.py shot_detection -t <type> -i <path to frames>
```

2 Logo detection

Template matching is an object detection algorithm which is translation invariant but not scale or rotation invariant. As we are detecting a logo, we can assume that the target of detection will be in a known orientation. Due to the possibility that there may be multiple occurrences of the logo in a frame, we cannot simply match SIFT features between the logo template and a frame. Template matching is run on templates of different sizes because the size of the logo in a frame is unknown. Then, a score is calculated for all the matches by normalized cross-correlation. The normalized version is chosen because brighter patches will not have a higher score. Also, the score obtained will be in the range [0, 1] and this makes choosing a threshold more intuitive. As the logo of the template and the one in the frame might be of different, possibly due to different resolutions or styles, a looser threshold is first used to filter out the irrelevant matches.

For each match, if its score is greater than the loose threshold, it is kept. Then, the algorithm checks if that particular match is a slight translation of a match we have a already decided to keep. If it is, the match is discarded. This prevents having multiple boxes around one logo. Afterwards, SIFT descriptors are calculated for the remaining matches and a score is calculated using feature matching and Lowe's ratio test with the logo template. If a match's score is above a tighter threshold, it is declared as a match of the template and a box is put around the match.

The logo detection was the most difficult part of the project for me. I originally only did one pass with either normalized cross-correlation or SIFT feature matching. However, this led to fairly poor results, with the algorithm often unable to detect multiple logos and including many irrelevant matches. Therefore, the two passes approach, first with a looser threshold using normalized cross-correlation and then a tighter threshold with SIFT feature matching, is used. As can be seen in Figure 8, the algorithm can detect multiple logos.

Figure 7: Logo detection on frame 104 of clip 1

Figure 8: Logo detection on frame 52 of clip 1

The relevant code is in `logo.py`. To run logo detection, run the command below.

```
python3 run.py logo_detection -i <input directory> -o <output directory>
-d <logo path> -t <min threshold>
```

3 Face detection and tracking

There are 260 images in the female and male classes respectively. Each image is accompanied by a `.mat` file specifying the coordinates of the left eye, right eye, nose and mouth. The following rules are used to crop the images in order to obtain the faces.

$$\begin{aligned} start_x &= left\ eye_x - 0.5 \times (right\ eye_x - left\ eye_x) \\ end_x &= right\ eye_x + 0.5 \times (right\ eye_x - left\ eye_x) \\ start_y &= eyes_y - (mouth_y - eyes_y) \\ end_y &= mouth_y + (mouth_y - eyes_y) \end{aligned} \tag{1}$$

The relevant code for face cropping is in `crop_images()` in `face.py`.

The initial attempt to detect faces is to use skin detection - trying to filter out skin in images. Figure 9 and Figure 10 show the color distributions of faces in RGB and HSV color spaces. The HSV color space has narrower distributions, especially with hue. Although this method works sometimes as seen in Figure 11, it is not successful in general. It fails to detect a large area of the face of the man on the right in Figure 12 and includs a lot of the background.This model is especially poor when other things in the frame are very similar to human skin tone.

Figure 9: RGB distribution of fe male training images

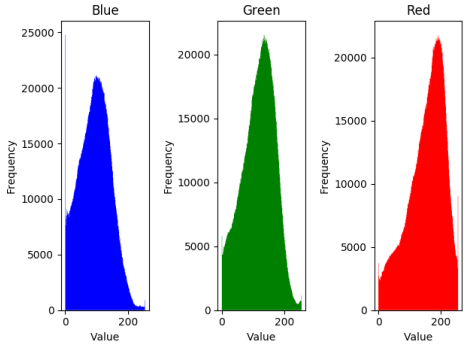


Figure 10: HSV distribution of female training images

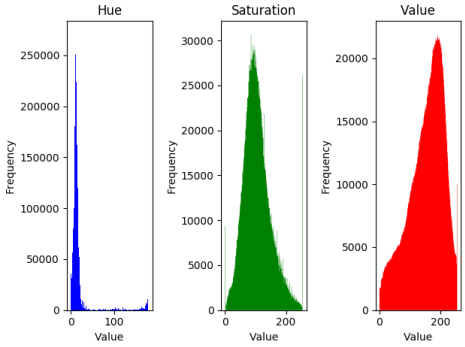


Figure 11: HSV colour detection on frame 160 of clip 1

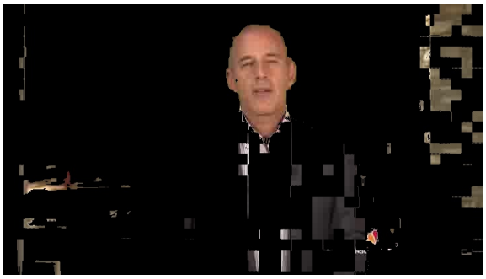


Figure 12: HSV colour detection on frame 50 of clip 1

The relevant code for HSV face detection is in `visualize_distributions()` and `face_detection_hsv()` in `face.py`.

I ended up using `cv2.CascadeClassifier` for detecting faces. The full name of this classifier is Haar feature-based cascade classifier for object detection. It is a OpenCV pre-trained classifier for face stored in an XML file. It works fairly well. It has no trouble detecting multiple people in a frame or people of colour, like in Figure 13. However, it is sometimes unable to detect faces in a certain position. The detector was unable to detect the person on the left in any of the frames he appeared in that particular position in Figure 14.

After obtaining the faces in a frame, for each face in the current frame, the SIFT descriptors are found. Then, they are matched to the SIFT descriptors of each faces in the previous frame using feature matching and Lowe's ratio test. Then, a score is obtained from the number of matches. If the score is above a set threshold, the algorithm declares that the face we are looking at is found in the previous frame and we will display the index assigned to that particular face in the previous frame. If the face is not found, we assign a new index to the face.

Figure 13: Face detection on frame 69 of clip 2

Figure 14: Face detection on frame 110 of clip 1

4 Gender classification

90% of the images (234 images from each class) are used for training, whereas the other 10% (26 images from each class) are used for testing the accuracy of the model.

SVM

The SIFT descriptors of the training images are passed into the SVM model for training. For each detected face, the SIFT descriptors are extracted and fed to the trained SVM model. Then, a prediction for each descriptor is obtained. If more descriptors are predicted as female than male, the image is classified as female. If more descriptors are predicted as male than female, the image is classified as male. If there are equal number of descriptors being predicted as both female and male, the image is then classified as unknown.

Neural network

The same as SVM, except with a neural network model instead. The model uses a binary crossentropy loss, adam for optimization, and accuracy as the metric.

CNN

All the training and testing images are padded with black borders to obtain a square shape and then resized to be 72 pixels by 72 pixels. The training images are then passed to the CNN model shown in Figure 16 for training. The model uses a binary crossentropy loss, stochastic gradient descent for optimization, and accuracy as the metric. Each detected face is padded to obtain a square shape. Then, the image is resized to be 72 pixels by 72 pixels. The resized image is then passed to the trained cnn model and a category prediction is obtained.

Figure 15: Neural network model

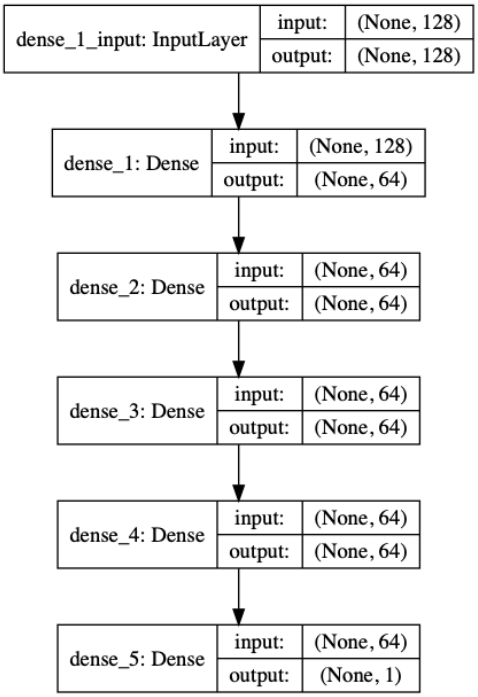
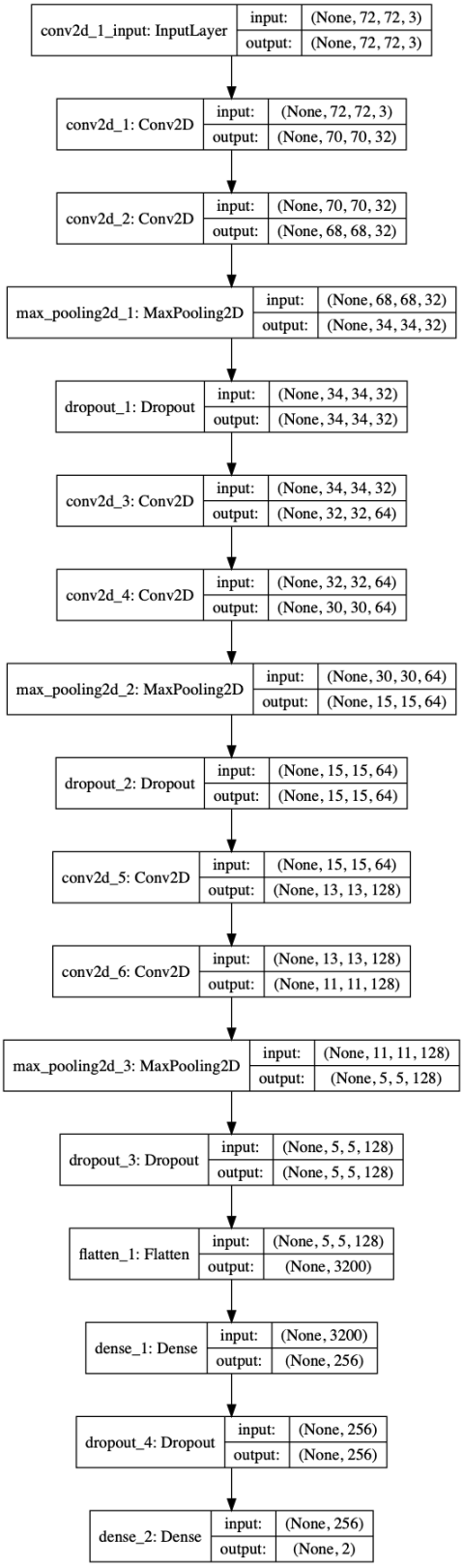


Figure 16: CNN model



The relevant code for training any of the three models is in `train_model()` in `face.py`. To train a gender classification model, run the command below.

```
python3 run.py train -m <model path after training>
-c <classification model (SVM, NN_SIFT, or CNN)>
```

The relevant code for face detection (including gender classification and face tracking) is in `face_detection()` in `face.py`. Run the command below.

```
python3 run.py face_detection -i <input directory> -o <output directory>
-c <classification model (SVM, NN_SIFT, or CNN)> -m <trained model path>
```

Performance

Table 1 shows the test accuracies of the three models. As expected, CNN performed poorly, achieving an accuracy that is equivalent to random guesses, due to the very small training data size of 468.

Table 1: Gender classification performance

Model	Description	Accuracy on test set
SVM	Using SIFT descriptors of faces	100.00%
Neural network	Using SIFT descriptors of faces	92.30%
CNN	Using cropped and resized faces	50.00%

5 References

Video shot boundary detection based on color histogram

Wikipedia - Shot transition detection

6 Code

Listing 1: shot.py

```
import cv2
import matplotlib.pyplot as plt
import numpy as np
import os

def shot_detection(input_dir, type):
    scores = []
    img_names = [img for img in os.listdir(input_dir) if img.endswith(".jpg")]
    # Sort images by name in ascending order
    img_names.sort(key=lambda img: int("".join(filter(str.isdigit, img))))
    start_idx = int(img_names[0][:img_names[0].find(".jpg")])
    end_idx = int(img_names[-1][:img_names[-1].find(".jpg")])

    # Sum of absolute differences
    if type == "SAD2":
        prev_img = None
        for i in range(len(img_names)):
            # Default type is numpy.uint64
            curr_img = cv2.imread(os.path.join(input_dir, img_names[i])).astype(np.int64)
            if i == 0:
                r, c, d = curr_img.shape
                next_img = cv2.imread(os.path.join(input_dir, img_names[i + 1])).astype(np.
                    ↪ int64)
                score = np.sum(np.abs(curr_img - next_img))
            elif i == len(img_names) - 1:
                score = np.sum(np.abs(curr_img - prev_img))
            else:
                next_img = cv2.imread(os.path.join(input_dir, img_names[i + 1])).astype(np.
                    ↪ int64)
                score = 0.5 * np.sum(np.abs(curr_img - prev_img)) + 0.5 * np.sum(np.abs(
                    ↪ curr_img - next_img))
            scores.append(score)
            prev_img = curr_img
        x = np.arange(start_idx, end_idx + 1)
        scores = np.array(scores) / (r * c * d)
        title = "Sum of absolute differences"
        new_filename = "output/" + input_dir.name + "_score_sad2.png"

    elif type == "SAD":
        prev_img = None
        for i in range(len(img_names)):
            # Default type is numpy.uint64
            curr_img = cv2.imread(os.path.join(input_dir, img_names[i])).astype(np.int64)
            if i == 0:
                r, c, d = curr_img.shape
                prev_img = curr_img
                continue
            score = np.sum(np.abs(curr_img - prev_img))
            scores.append(score)
            prev_img = curr_img
        x = np.arange(start_idx + 1, end_idx + 1)
        scores = np.array(scores) / (r * c * d)
        title = "Sum of absolute differences"
        new_filename = "output/" + input_dir.name + "_score_sad2.png"

    # Histogram differences
```

```
    elif type == "HD":
        prev_histogram = None
        for i in range(len(img_names)):
            # Default type is numpy.uint8
            curr_img = cv2.imread(os.path.join(input_dir, img_names[i]))#.astype(np.int64)
            # Default type is numpy.uint8
            curr_img_g = cv2.cvtColor(curr_img, cv2.COLOR_BGR2GRAY).astype(np.int16)
            histogram = np.histogram(np.ravel(curr_img_g), bins=np.arange(-1, 256))
            if i == 0:
                r, c = curr_img_g.shape
                prev_histogram = histogram[0]
                continue
            score = np.sum(np.abs(histogram[0] - prev_histogram))
            scores.append(score)
            prev_histogram = histogram[0]

        x = np.arange(start_idx + 1, end_idx + 1)
        scores = np.array(scores) / (r * c)
        title = "Histogram differences"
        new_filename = "output/" + input_dir.name + "_score_hd.png"

    else:
        raise ValueError("Illegal type value")

    print(len(x), len(scores))
    f = plt.figure(figsize=(10, 5))
    ax = f.gca()
    ax.set_xticks(np.arange(start_idx, end_idx, 10))
    plt.title(title)
    plt.xlabel("Frame")
    plt.ylabel("Score")
    plt.plot(x, scores)
    plt.grid()
    f.savefig(new_filename)
    print("Output saved to " + new_filename)
```

Listing 2: logo.py

```
import cv2
import imutils
import numpy as np
import os

import copy


# def logo_detection(input_dir, output_dir,logo_path):
# print("detect_logo")

# if not os.path.isdir(output_dir):
# os.mkdir(output_dir)

# template = cv2.imread(str(logo_path))
# template_g = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)

# w, h = template_g.shape[::-1]

# # Compute template of different sizes
# scales = np.linspace(0.1, 1.0, 25)[::-1]
# templates = []
```

```
# ratios = []
# for scale in scales:
# resized = imutils.resize(template_g, width=int(template_g.shape[1] * scale))
# template_canny = cv2.Canny(resized, 50, 200)
# templates.append(template_canny)
# ratios.append(resized.shape[1] / float(template_g.shape[1]))

# for img_name in os.listdir(input_dir):
# if not img_name.endswith(".jpg"):
# continue
# img = cv2.imread(os.path.join(input_dir, img_name))
# img_g = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# img_canny = cv2.Canny(img_g, 50, 200)
# # cv2.imshow("img_canny", img_canny)
# # cv2.waitKey(0)

# found = None
# # Loop through the templates from small to big
# for i in range(len(templates) - 1, -1, -1):
# # Stop when template is bigger than image
# if img_g.shape[0] < templates[i].shape[0] or img_g.shape[1] < templates[i].shape[1]:
# break

# result = cv2.matchTemplate(img_canny, templates[i], cv2.TM_CCORR_NORMED) # img_g.shape - template_g.
    ↪ shape + 1
# _, max_val, _, max_loc = cv2.minMaxLoc(result)
# # print(max_val)
# # temp_img = img
# # r = ratios[i]
# # start_x, start_y = max_loc[0], max_loc[1]
# # end_x, end_y = int((max_loc[0] + w * r)), int((max_loc[1] + h * r))
# # cv2.rectangle(temp_img, (start_x, start_y), (end_x, end_y), (0, 255, 0), thickness=2)
# # cv2.imshow("temp_img", temp_img)
# # cv2.waitKey(0)

# if found is None or max_val > found[0]:
# found = (max_val, max_loc, ratios[i])

# max_val, max_loc, r = found
# start_x, start_y = max_loc[0], max_loc[1]
# end_x, end_y = int((max_loc[0] + w * r)), int((max_loc[1] + h * r))

# cv2.rectangle(img, (start_x, start_y), (end_x, end_y), (0, 255, 0), thickness=2)
# cv2.imwrite(os.path.join(output_dir, img_name), img)

# cv2.imshow("img", img)
# cv2.waitKey(0)

def get_score(img1, img2):
    sift = cv2.xfeatures2d.SIFT_create()
    kp1, des1 = sift.detectAndCompute(img1, None)
    kp2, des2 = sift.detectAndCompute(img2, None)
    if len(kp1) < 2 or len(kp2) < 2:
        return 0
    index_params = dict(algorithm=0, trees=5)
    flann = cv2.FlannBasedMatcher(index_params, None)
    matches = flann.knnMatch(des1, des2, k=2)
    good_matches = []
    for m, n in matches:
```

```
81          if m.distance < 0.6*n.distance:
82              good_matches.append(m)
83      num_kps = 0
84      if len(kp1) <= len(kp2):
85          num_kps = len(kp1)
86      else:
87          num_kps = len(kp2)
88      score = len(good_matches) / num_kps * 100
89      return score
90
91  def logo_detection(input_dir, output_dir, logo_path, min_threshold):
92      print("detect_logo")
93
94      if not os.path.isdir(output_dir):
95          os.mkdir(output_dir)
96
97      template = cv2.imread(str(logo_path))
98      template_g = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)
99
100     w, h = template_g.shape[::-1]
101
102     # Compute template of different sizes
103     if template.shape[0] < 50:
104         scales = np.linspace(0.8, 1.0, 6)[::-1]
105     else:
106         # scales = np.linspace(0.1, 1.0, 25)[::-1]
107         scales = np.linspace(0.5, 1.0, 10)[::-1]
108     templates = []
109     ratios = []
110     for scale in scales:
111         resized = imutils.resize(template_g, width=int(template_g.shape[1] * scale))
112         template_canny = cv2.Canny(resized, 50, 200)
113         # templates.append(template_canny)
114         templates.append(resized)
115         ratios.append(resized.shape[1] / float(template_g.shape[1]))
116
117     for img_name in os.listdir(input_dir):
118         if not img_name.endswith(".jpg"):
119             continue
120         img = cv2.imread(os.path.join(input_dir, img_name))
121         img_g = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
122         img_canny = cv2.Canny(img_g, 100, 200)
123
124         p = -1
125         q = -1
126         matches = []
127         # Loop through the templates from small to big
128         for i in range(len(templates) - 1, -1, -1):
129             # Stop when template is bigger than image
130             if img_g.shape[0] < templates[i].shape[0] or img_g.shape[1] < templates[i].
                    ↪ shape[1]:
131                 break
132
133             # match = cv2.matchTemplate(img_canny, templates[i], cv2.TM_CCORR_NORMED) # img_g.shape -
                    ↪  template_g.shape + 1
134             match = cv2.matchTemplate(img_g, templates[i], cv2.TM_CCORR_NORMED) # img_g.shape
                    ↪  - template_g.shape + 1
135             if p == -1 and q == -1:
136                 p, q = match.shape
```

```
            m, n = match.shape
            matches.append(np.pad(match, ((0, p - m), (0, q - n)), mode="constant",
                ↪ constant_values=0))

        boxes = []

        matches = np.array(matches)
        r, max_y, max_x = np.unravel_index(np.argmax(matches), matches.shape)
        r = ratios[len(ratios) - 1 - r]
        max_val = np.max(matches)
        max_thresh = max(max_val * 0.95, min_threshold)
        # print("max_val:", max_val, min_threshold, max_val < min_threshold)
        if max_val < min_threshold:
            cv2.imwrite(os.path.join(output_dir, img_name), img)
            continue
        start_x, start_y = max_x, max_y
        end_x, end_y = int((max_x + w * r)), int((max_y + h * r))
        match_score = get_score(template, img[start_y:end_y, start_x:end_x])
        # if match_score > 0:
            # print("How good it's the match: ", match_score, img_name)
        if match_score < 5:
            cv2.imwrite(os.path.join(output_dir, img_name), img)
            continue
        boxes.append((r, max_y, max_x, 1))

        match_locations = np.where(matches >= max_thresh)
        for i in range(len(match_locations[0])):
            r1, y1, x1 = ratios[len(ratios) - 1 - match_locations[0][i]], match_locations
                ↪ [1][i], match_locations[2][i]
            found = False
            for j in range(len(boxes)):
                r2, y2, x2, count = boxes[j]
                # Check if two boxes of the same size overlap or
                # if a smaller one is contained in the bigger one
                if (r1 == r2 and np.abs(x1 - x2) < w * r1 and np.abs(y1 - y2) < h * r1) or
                    ↪ \
                    ((r1 < r2) and (x1 <= (x2 + w * r2) <= (x1 + w * r1)) and (y1 <= (y2 + h
                        ↪  * r2) <= (y1 + h * r1))) or\
                    (np.abs(x1 - x2) < 0.5 * w * r1 and np.abs((x1 + w * r1) - (x2 + w * r2
                        ↪ )) < 0.5 * w * r1 and\
                    np.abs(y1 - y2) < 0.5 * h * r1 and np.abs((y1 + h * r1) - (y2 + h * r2)
                        ↪ ) < 0.5 * h * r1):
                    boxes[j] = (r2, y2, x2, count + 1)
                    found = True
                    break
            if not found:
                start_x, start_y = x1, y1
                end_x, end_y = int((x1 + w * r1)), int((y1 + h * r1))
                # cv2.imshow("", img[start_y:end_y, start_x:end_x])
                # cv2.waitKey(0)
                match_score = get_score(template, img[start_y:end_y, start_x:end_x])
                # if match_score > 0:
                    # print("How good it's the match: ", match_score, img_name)
                if match_score > 5:
                    boxes.append((r1, y1, x1, 1))

        # print("boxes:", boxes)

        for r, y, x, count in boxes:
```

```
            start_x, start_y = x, y
            end_x, end_y = int((x + w * r)), int((y + h * r))
            # print(start_x, start_y, end_x, end_y)
            cv2.rectangle(img, (start_x, start_y), (end_x, end_y), (0, 255, 0), thickness
                ↪ =2)

        cv2.imwrite(os.path.join(output_dir, img_name), img)
        # cv2.imshow("img", img)
        # cv2.waitKey(0)
```

Listing 3: face.py

```
import copy
import cv2
import matplotlib.pyplot as plt
import numpy as np
import os
import pickle
import scipy.io, scipy.misc
from skimage import io
from skimage import transform as tf
from skimage.color import rgb2hsv
from sklearn import svm
from utils import shuffle

def visualize_distributions(imgs_dir):
    """
    Visualize the distributions of values in HSV and BGR of pictures in imgs_dir.
    """
    H, S, V = None, None, None
    B, G, R = None, None, None
    for filename in os.listdir(imgs_dir):
            if not filename.endswith(".png"):
                continue
            img = cv2.imread(imgs_dir + filename)
            hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
            if H is None:
                H = hsv_img[..., 0].flatten()
                S = hsv_img[..., 1].flatten()
                V = hsv_img[..., 2].flatten()
                B = img[..., 0].flatten()
                G = img[..., 1].flatten()
                R = img[..., 2].flatten()
            else:
                H = np.concatenate([H, hsv_img[..., 0].flatten()])
                S = np.concatenate([S, hsv_img[..., 1].flatten()])
                V = np.concatenate([V, hsv_img[..., 2].flatten()])
                B = np.concatenate([B, img[..., 0].flatten()])
                G = np.concatenate([G, img[..., 1].flatten()])
                R = np.concatenate([R, img[..., 2].flatten()])

    # Plot
    f = plt.figure()
    ax1 = f.add_subplot(1, 3, 1)
    ax1.hist(H, bins=180,
        range=(0.0, 180.0), histtype="stepfilled", color="b", label="Hue")
    plt.title("Hue")
    plt.xlabel("Value")
    plt.ylabel("Frequency")
    ax2 = f.add_subplot(1, 3, 2)
```

```
49      ax2.hist(S, bins=256,
50          range=(0.0, 255.0),histtype="stepfilled", color="g", label="Saturation")
51      plt.title("Saturation")
52      plt.xlabel("Value")
53      plt.ylabel("Frequency")
54      ax3 = f.add_subplot(1, 3, 3)
55      ax3.hist(V, bins=256,
56          range=(0.0, 255.0), histtype="stepfilled", color="r", label="Value")
57      plt.title("Value")
58      plt.xlabel("Value")
59      plt.ylabel("Frequency")
60      f.tight_layout()
61      f.savefig(imgs_dir + "_hsv_distributions.png")
62      plt.show()
63
64      f = plt.figure()
65      ax1 = f.add_subplot(1, 3, 1)
66      ax1.hist(B, bins=256,
67          range=(0.0, 255.0), histtype="stepfilled", color="b", label="Blue")
68      plt.title("Blue")
69      plt.xlabel("Value")
70      plt.ylabel("Frequency")
71      ax2 = f.add_subplot(1, 3, 2)
72      ax2.hist(G, bins=256,
73          range=(0.0, 255.0),histtype="stepfilled", color="g", label="Saturation")
74      plt.title("Green")
75      plt.xlabel("Value")
76      plt.ylabel("Frequency")
77      ax3 = f.add_subplot(1, 3, 3)
78      ax3.hist(R, bins=256,
79          range=(0.0, 255.0), histtype="stepfilled", color="r", label="Red")
80      plt.title("Red")
81      plt.xlabel("Value")
82      plt.ylabel("Frequency")
83      f.tight_layout()
84      f.savefig(imgs_dir + "_rgb_distributions.png")
85      plt.show()
86
87  def crop_images(old_dir, new_dir):
88      """
89      Crop .jpg images in old_dir given coordinates of
90      left eye, right eye, nose and mouth in .mat files.
91      Save cropped images in new_dir.
92      """
93      print("crop_images")
94      os.mkdir(new_dir)
95
96      H, S, V = None, None, None
97      for filename in os.listdir(old_dir):
98          if not filename.endswith(".jpg"):
99              continue
100
101         index = filename.find(".jpg")
102         name = filename[:index]
103
104         # Approximate coordinates of face
105         coords = scipy.io.loadmat(old_dir + name + ".mat")
106         start_x = int(coords["x"][0][0] - 0.5*(coords["x"][1][0] - coords["x"][0][0]))
107         end_x = int(coords["x"][1][0] + 0.5*(coords["x"][1][0] - coords["x"][0][0]))
```

```
108         start_y = int(coords["y"][0][0] - (coords["y"][3][0] - coords["y"][0][0]))
109         end_y = int(coords["y"][3][0] + (coords["y"][3][0] - coords["y"][2][0]))
110         img = io.imread(old_dir + filename)
111         face = img[start_y:end_y, start_x:end_x]
112         # Save cropped image
113         scipy.misc.imsave(new_dir + name + ".png", face)
114
115 def resize_images(input_dir, output_dir, size=72):
116     """
117     Resize images in input_dir to size x size.
118     """
119     print("resize_images")
120     if not os.path.isdir(output_dir):
121         os.mkdir(output_dir)
122
123     for filename in os.listdir(input_dir):
124         if not filename.endswith(".png"):
125             continue
126         img = cv2.imread(os.path.join(input_dir, filename))
127         x, y, d = img.shape
128         # Pad image to square
129         if x > y:
130             padded_img = np.pad(img, ((0, 0), (0, x - y), (0, 0)),
131                 mode="constant", constant_values=0)
132         else:
133             padded_img = np.pad(img, ((0, y - x), (0, 0), (0, 0)),
134                 mode="constant", constant_values=0)
135         resized_img = cv2.resize(padded_img, (size, size))
136         scipy.misc.imsave(output_dir + filename, cv2.cvtColor(resized_img, cv2.
            ↪ COLOR_BGR2RGB))
137
138 def get_features(input_dir):
139     """
140     Get keypoints and descriptors of the images in input_dir with SIFT.
141     """
142     kps, des = None, None
143     sift = cv2.xfeatures2d.SIFT_create()
144
145     for filename in os.listdir(input_dir):
146         if not filename.endswith(".png"):
147             continue
148         img = io.imread(input_dir + filename)
149         kp, d = sift.detectAndCompute(img, None)
150
151         if des is None:
152             kps = kp
153             des = d
154         else:
155             kps = np.concatenate([kps, kp], axis=0)
156             des = np.concatenate([des, d], axis=0)
157     return kps, des
158
159 def get_data(f_dir, m_dir):
160     """
161     Get the images of the F and M classes from their respective directories and shuffle.
162     """
163     import keras
164     x = []
165     y = [0] * len(os.listdir(f_dir)) + [1] * len(os.listdir(m_dir))
```

```
166     imgs = os.listdir(f_dir) + os.listdir(m_dir)
167     for filename in os.listdir(f_dir):
168         if not filename.endswith(".png"):
169             continue
170         x.append(cv2.imread(os.path.join(f_dir, filename)))
171     for filename in os.listdir(m_dir):
172         if not filename.endswith(".png"):
173             continue
174         x.append(cv2.imread(os.path.join(m_dir, filename)))
175 
176     x, y = shuffle(x, y)
177     y = keras.utils.to_categorical(y, num_classes=2)
178     return x, y
179 
180 def train_model(model_path, classification):
181     OLD_F_DIR = "original_data/female/"
182     OLD_M_DIR = "original_data/male/"
183     F_TRAIN_DIR = "data/female_train/"
184     M_TRAIN_DIR = "data/male_train/"
185     F_TEST_DIR = "data/female_test/"
186     M_TEST_DIR = "data/male_test/"
187     F_TRAIN_CNN_DIR = "data/female_cnn_train/"
188     M_TRAIN_CNN_DIR = "data/male_cnn_train/"
189     F_TEST_CNN_DIR = "data/female_cnn_test/"
190     M_TEST_CNN_DIR = "data/male_cnn_test/"
191 
192     if not os.path.isdir("data/"):
193         os.mkdir("data/")
194         crop_images(OLD_F_DIR, F_TRAIN_DIR)
195         crop_images(OLD_M_DIR, M_TRAIN_DIR)
196 
197     if classification == "SVM":
198         f_train_kps, f_train_des = get_features(F_TRAIN_DIR)
199         m_train_kps, m_train_des = get_features(M_TRAIN_DIR)
200         x_train = np.concatenate([f_train_des, m_train_des], axis=0)
201         y_train = [-1] * len(f_train_des) + [1] * len(m_train_des)
202         model = svm.SVC(kernel="rbf", gamma="scale", C=10.0)
203         print("Start training")
204         model.fit(x_train, y_train)
205         # Save model
206         pickle.dump(model, open(model_path, "wb"))
207         print("Model saved as " + str(model_path))
208 
209     elif classification == "NN_SIFT":
210         import keras
211         from keras.layers import Conv2D, Dense, Dropout, Flatten, MaxPooling2D
212         from keras.models import Sequential
213         from keras.optimizers import Adam, SGD
214         from keras.utils import plot_model
215 
216         f_train_kps, f_train_des = get_features(F_TRAIN_DIR)
217         m_train_kps, m_train_des = get_features(M_TRAIN_DIR)
218         x_train = np.concatenate([f_train_des, m_train_des], axis=0)
219         y_train = [0] * len(f_train_des) + [1] * len(m_train_des)
220         x_train, y_train = shuffle(x_train, y_train)
221 
222         f_test_kps, f_test_des = get_features(F_TEST_DIR)
223         m_test_kps, m_test_des = get_features(M_TEST_DIR)
224         x_test = np.concatenate([f_test_des, m_test_des], axis=0)
```

```
225         y_test = [0] * len(f_test_des) + [1] * len(m_test_des)
226         x_test, y_test = shuffle(x_test, y_test)
227 
228         model = Sequential()
229 
230         model.add(Dense(64, activation='relu', kernel_initializer='random_normal', input_dim
                =128))
231         model.add(Dense(64, activation='relu', kernel_initializer='random_normal'))
232         model.add(Dense(64, activation='relu', kernel_initializer='random_normal'))
233         model.add(Dense(64, activation='relu', kernel_initializer='random_normal'))
234         model.add(Dense(1, activation='sigmoid', kernel_initializer='random_normal'))
235 
236         model.compile(optimizer ='adam',loss='binary_crossentropy', metrics =['accuracy'])
237 
238         model.fit(x_train, y_train, batch_size=10, epochs=20)
239         score = model.evaluate(x_test, y_test)
240         print("score:", score)
241 
242         model.save(model_path)
243         print("Model saved as " + str(model_path))
244         plot_model(model, to_file="output/nn_model.png", show_shapes=True)
245 
246     elif classification == "CNN":
247         import keras
248         from keras.layers import Conv2D, Dense, Dropout, Flatten, MaxPooling2D
249         from keras.models import Sequential
250         from keras.optimizers import Adam, SGD
251         from keras.utils import plot_model
252 
253         np.random.seed(123)
254         size = 72
255 
256         # resize_images(F_TRAIN_DIR, F_TRAIN_CNN_DIR, size)
257         # resize_images(M_TRAIN_DIR, M_TRAIN_CNN_DIR, size)
258         # resize_images(F_TEST_DIR, F_TEST_CNN_DIR, size)
259         # resize_images(M_TEST_DIR, M_TEST_CNN_DIR, size)
260 
261         x_train, y_train = get_data(F_TRAIN_CNN_DIR, M_TRAIN_CNN_DIR)
262         x_test, y_test = get_data(F_TEST_CNN_DIR, M_TEST_CNN_DIR)
263 
264         model = Sequential()
265         model.add(Conv2D(32, (3, 3), activation="relu", input_shape=(size, size, 3)))
266         model.add(Conv2D(32, (3, 3), activation="relu"))
267         model.add(MaxPooling2D(pool_size=(2, 2)))
268         model.add(Dropout(0.25))
269 
270         model.add(Conv2D(64, (3, 3), activation="relu"))
271         model.add(Conv2D(64, (3, 3), activation="relu"))
272         model.add(MaxPooling2D(pool_size=(2, 2)))
273         model.add(Dropout(0.25))
274 
275         model.add(Conv2D(128, (3, 3), activation="relu"))
276         model.add(Conv2D(128, (3, 3), activation="relu"))
277         model.add(MaxPooling2D(pool_size=(2, 2)))
278         model.add(Dropout(0.25))
279 
280         model.add(Flatten())
281         model.add(Dense(256, activation="relu"))
282         model.add(Dropout(0.5))
```

```
283         model.add(Dense(2, activation="softmax"))
284 
285         adam = Adam(lr=0.001)
286         model.compile(loss="binary_crossentropy", optimizer=adam, metrics=["accuracy"])
287 
288         model.fit(x_train, y_train, epochs=10)
289         score = model.evaluate(x_test, y_test)
290         print("score:", score)
291 
292         model.save(model_path)
293         print("Model saved as " + str(model_path))
294         plot_model(model, to_file="output/cnn_model.png", show_shapes=True)
295 
296     else:
297         raise ValueError("Illegal classification value")
298 
299 def predict_model(model_path):
300     print("predict_model")
301     F_TEST_DIR = "data/female_test/"
302     M_TEST_DIR = "data/male_test/"
303 
304     ext = os.path.splitext(model_path)[1]
305     if ext == ".sav":
306         model = pickle.load(open(model_path, "rb"))
307     elif ext == ".h5":
308         from keras.models import load_model
309         model = load_model(model_path)
310     else:
311         raise ValueError("Not valid model extension")
312 
313     sift = cv2.xfeatures2d.SIFT_create()
314 
315     correct = 0
316     for filename in os.listdir(F_TEST_DIR):
317         if not filename.endswith(".png"):
318             continue
319         img = cv2.imread(os.path.join(F_TEST_DIR, filename))
320         kps, des = sift.detectAndCompute(img, None)
321         result = model.predict(des)
322         if ext == ".sav":
323             f_count = np.count_nonzero(result == -1)
324             m_count = np.count_nonzero(result == 1)
325         elif ext == ".h5":
326             f_count = np.sum(result < 0.5)
327             m_count = np.sum(result > 0.5)
328         if f_count > m_count:
329             correct += 1
330 
331     for filename in os.listdir(M_TEST_DIR):
332         if not filename.endswith(".png"):
333             continue
334         img = cv2.imread(os.path.join(M_TEST_DIR, filename))
335         kps, des = sift.detectAndCompute(img, None)
336         result = model.predict(des)
337         if ext == ".sav":
338             f_count = np.count_nonzero(result == -1)
339             m_count = np.count_nonzero(result == 1)
340         elif ext == ".h5":
341             f_count = np.sum(result < 0.5)
```

```
342             m_count = np.sum(result > 0.5)
343         if m_count > f_count:
344             correct += 1
345     score = correct / (len(os.listdir(F_TEST_DIR)) + len(os.listdir(M_TEST_DIR))) * 100
346     print(str(score) + "% of images are categorized correctly")
347
348 def face_detection_hsv(input_dir, output_dir):
349     print("face_detection_hsv")
350
351     if not os.path.isdir(output_dir):
352         os.mkdir(output_dir)
353     for filename in os.listdir(input_dir):
354         if not filename.endswith(".jpg"):
355             continue
356         img = cv2.imread(os.path.join(input_dir, filename))
357
358         # HSV color detection
359         hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
360
361         lower1 = np.array([0, 48, 80])
362         upper1 = np.array([20, 255, 255])
363         mask1 = cv2.inRange(hsv_img, lower1, upper1) # img_hsv.shape
364         lower2 = np.array([170, 0, 0])
365         upper2 = np.array([180, 255, 255])
366         mask2 = cv2.inRange(hsv_img, lower2, upper2) # img_hsv.shape
367         mask = cv2.bitwise_or(mask1, mask2)
368
369         kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (10, 10))
370         new_img = cv2.bitwise_and(img, img, mask=mask)
371         cv2.imwrite(os.path.join(output_dir, filename), new_img)
372
373 def face_detection_cascade(input_dir, output_dir, model_path, classification):
374     print("face_detection_cascade")
375     XML_FILENAME = "haarcascade_frontalface_default.xml"
376     F_TEXT = "Female"
377     M_TEXT = "Male"
378     O_TEXT = "Not sure"
379     F_COLOR = (0, 0, 255)
380     M_COLOR = (255, 0, 0)
381     O_COLOR = (0, 255, 0)
382
383     if classification == "SVM":
384         model = pickle.load(open(model_path, "rb"))
385     elif classification == "NN_SIFT":
386         from keras.models import load_model
387         model = load_model(model_path)
388     elif classification == "CNN":
389         from keras.models import load_model
390         model = load_model(model_path)
391         size = 72
392     else:
393         raise ValueError("Illegal classification value")
394
395     if not os.path.isdir(output_dir):
396         os.mkdir(output_dir)
397
398     exts = [".jpg", ".png"]
399     img_names = [img for img in os.listdir(input_dir) if img.endswith(tuple(exts))]
400     # Sort images by name in ascending order
```

```
401     img_names.sort(key=lambda img: int("".join(filter(str.isdigit, img))))
402     sift = cv2.xfeatures2d.SIFT_create()
403     index_params = dict(algorithm=0, trees=5)
404     flann = cv2.FlannBasedMatcher(index_params, None)
405     face_count = 0
406     prev_faces = {}
407 
408     for filename in img_names:
409         img = cv2.imread(os.path.join(input_dir, filename))
410         img_g = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
411 
412         ##############################
413         ### Cascade face detection ###
414         ##############################
415         face_cascade = cv2.CascadeClassifier(XML_FILENAME)
416         faces = face_cascade.detectMultiScale(img_g, scaleFactor=1.3,
417             minNeighbors=3, minSize=(70, 70))
418         curr_faces = {}
419 
420         for x, y, w, h in faces:
421             ######################
422             ### Face tracking ###
423             ######################
424             face = img[y:y+h, x:x+w]
425             kp1, des1 = sift.detectAndCompute(face, None)
426             found_face = False
427             for (x2, y2, w2, h2), index in prev_faces.items():
428                 face2 = img[y2:y2+h, x2:x2+w]
429                 kp2, des2 = sift.detectAndCompute(face2, None)
430                 if len(kp1) < 2 or len(kp2) < 2:
431                     continue
432                 matches = flann.knnMatch(des1, des2, k=2)
433                 good_matches = []
434                 for m, n in matches:
435                     if m.distance < 0.6*n.distance:
436                         good_matches.append(m)
437                 num_kps = 0
438                 if len(kp1) <= len(kp2):
439                     num_kps = len(kp1)
440                 else:
441                     num_kps = len(kp2)
442                 score = len(good_matches) / num_kps * 100
443                 if score > 45:
444                     found_face = True
445                     face_number = index
446                     curr_faces[(x, y, w, h)] = index
447             if not found_face:
448                 face_number = face_count
449                 curr_faces[(x, y, w, h)] = face_count
450                 face_count += 1
451 
452             ##############################
453             ### Gender classification ###
454             ##############################
455             if classification == "SVM":
456                 # Get SIFT descriptors
457                 kps, des = sift.detectAndCompute(face, None)
458                 # Predict female or male
459                 result = model.predict(des)
```

```
                f_count = np.count_nonzero(result == -1)
                m_count = np.count_nonzero(result == 1)
                # Plot color box
                if f_count > m_count:
                    text = str(face_number) + " " + F_TEXT + ": " +\
                        format(f_count/(f_count + m_count)*100, ".2f") + "%"
                    color = F_COLOR
                elif m_count > f_count:
                    text = str(face_number) + " " + M_TEXT + ": " +\
                        format(m_count/(f_count + m_count)*100, ".2f") + "%"
                    color = M_COLOR
                else:
                    text = str(face_number) + " " + O_TEXT
                    color = O_COLOR

            elif classification == "NN_SIFT":
                assert w == h
                kps, des = sift.detectAndCompute(face, None)
                prediction = model.predict(des)
                f_count = np.sum(prediction < 0.5)
                m_count = np.sum(prediction > 0.5)
                if f_count > m_count:
                    text = str(face_number) + " " + F_TEXT + ": " +\
                        format(f_count/(f_count + m_count)*100, ".2f") + "%"
                    color = F_COLOR
                elif m_count > f_count:
                    text = str(face_number) + " " + M_TEXT + ": " +\
                        format(m_count/(f_count + m_count)*100, ".2f") + "%"
                    color = M_COLOR
                else:
                    text = str(face_number) + " " + O_TEXT
                    color = O_COLOR

            elif classification == "CNN":
                assert w == h
                face = cv2.resize(face, (size, size))
                prediction = model.predict(np.array([face]), verbose=0)
                if prediction == -1:
                    text = str(face_number) + " " + F_TEXT + ": " +\
                        format(prediction[0]*100, ".2f") + "%"
                    color = F_COLOR
                elif prediction == 1:
                    text = str(face_number) + " " + M_TEXT + ": " +\
                        format(prediction[0]*100, ".2f") + "%"
                    color = M_COLOR
                else:
                    text = str(face_number) + " " + O_TEXT
                    color = O_COLOR
            cv2.rectangle(img, (x, y), (x+w, y+h), color, thickness=2)
            cv2.putText(img, text, (x, y-10), color=color,
                fontFace=cv2.FONT_HERSHEY_PLAIN, fontScale=1)

        prev_faces = curr_faces
        cv2.imwrite(os.path.join(output_dir, filename), img)
        cv2.imshow("img", img)
        cv2.waitKey(0)
```

Listing 4: utils.py

```
import cv2
```

```
import os
import numpy as np

def make_video(imgs_dir, vid_name, fps):
    """
    Make vid_name.mp4 using images in imgs_dir.
    """
    OUTPUT_DIR = "output/"
    if not os.path.isdir(OUTPUT_DIR):
        os.mkdir(OUTPUT_DIR)
    exts = [".jpg", ".png"]
    imgs = [img for img in os.listdir(imgs_dir) if img.endswith(tuple(exts))]
    # Sort images by name in ascending order
    imgs.sort(key=lambda img: int("".join(filter(str.isdigit, img))))
    frame = cv2.imread(os.path.join(imgs_dir, imgs[0]))
    h, w, _ = frame.shape

    # Make sure vid_name does not include a file extension
    index = vid_name.find(".")
    if index != -1:
        vid_name = vid_name[:index]
    file_path = OUTPUT_DIR + vid_name + ".mp4"
    vid = cv2.VideoWriter(file_path, cv2.VideoWriter_fourcc(*"MP4V"), fps, (w, h))
    for img in imgs:
        vid.write(cv2.imread(os.path.join(imgs_dir, img)))
    vid.release()
    print("Video is now in ", file_path)

def shuffle(x, y):
    """
    Shuffle data x and their labels y.
    """
    assert len(x) == len(y)
    idx = np.random.permutation(len(x))
    x, y = np.array(x)[idx], np.array(y)[idx]
    return x, y
```