NEWS BROADCAST ANALYSIS

Christine K. C. Cheng

1 Shot detection

A score is calculated for each frame using either the sum of absolute differences method or the histogram differences method. Given the mean μ and standard deviation σ of all the scores, a threshold is chosen as $\mu + k \times \sigma$, where k is an integer. A shot change is declared when the value of the score becomes greater than the threshold and then becomes smaller than the threshold (similar to a peak above the threshold line).

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. 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.

Performance

Let C be the number of correctly identified cuts, M be the number of cuts that are not identified, and F be the number of falsely identified cuts. To evaluate how well I am detecting the shots, I calculate the recall (V), precision (P) and F_1 scores for each clip. They are defined as below.

$$\begin{aligned} V &= \frac{C}{C+M} \\ P &= \frac{C}{C+F} \\ F_1 &= \frac{2 \times P \times V}{P+V} \end{aligned} \tag{1}$$

Table 1 and Table 2 shows the respective scores for both methods. On average, histogram differences achieve a higher score across all three categories than the sum of absolute differences method.

Table 1: Shot detection performance with sum of absolute differences

Clip	Correct (C)	Missed (M)	Falsely detected (F)	Recall (V)	Precision (P)	F_1
1	1	0	1	1	0.5	0.6667
2	7	0	2	1	0.7778	0.8750
3	5	16	0	0.2381	1	0.3846

Table 2: Shot detection performance with histogram differences

Clip	Correct (C)	Missed (M)	Falsely detected (F)	Recall (V)	Precision (P)	F_1
1	1	0	0	1	1	1
2	6	1	1	0.8571	0.8571	0.8571
3	15	6	1	0.7142	0.9375	0.8118

Figure 1: Sum of absolute differences scores of clip 1

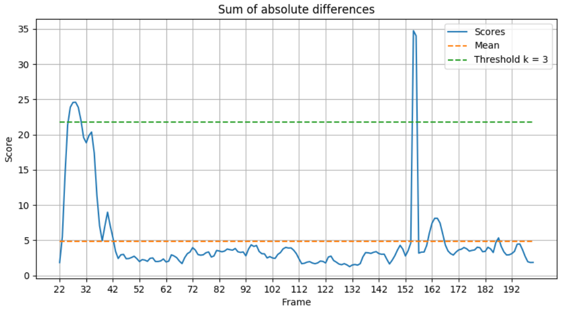


Figure 2: Histogram differences scores of clip 1

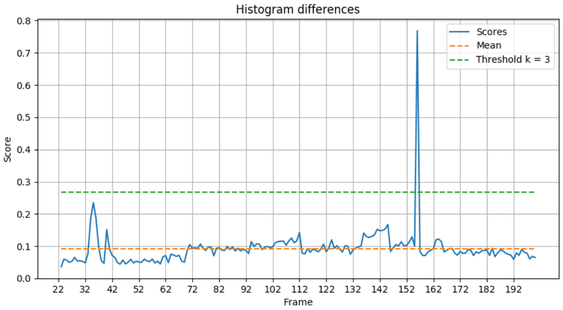


Figure 3: Sum of absolute differences scores of clip 2

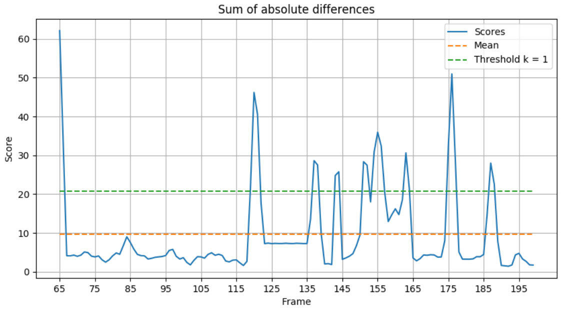


Figure 4: Histogram differences scores of clip 2

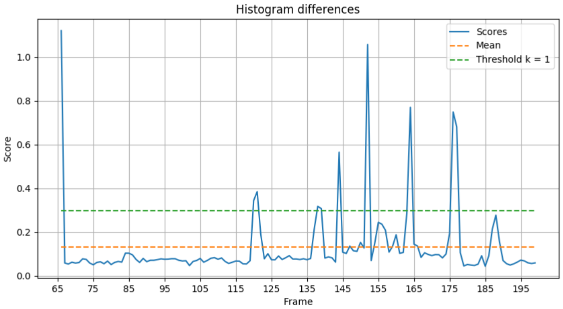


Figure 5: Sum of absolute differences scores of clip 3

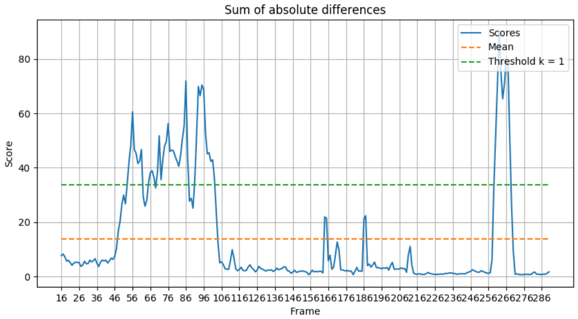
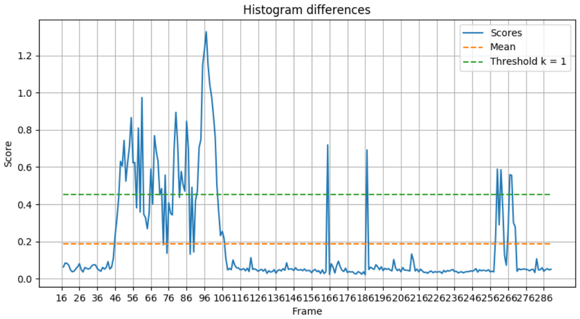


Figure 6: Histogram differences scores of clip 3



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>
```

To add shot numbers into frames, run the command below. The shot number will appear on the bottom left corner of each frame.

```
python3 run.py add_shot_numbe -i <input directory> -o <output directory>
-k <k for thresholding>
```

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

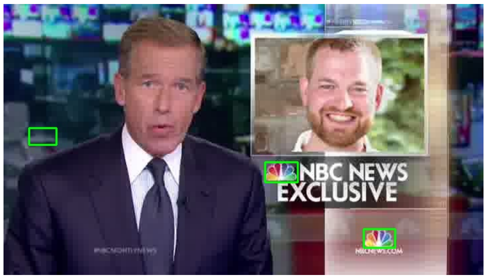
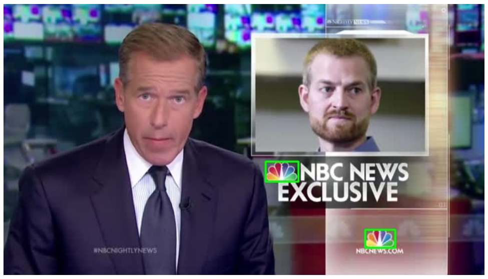


Figure 8: Logo detection on frame 52 of clip 1



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 for NCC>
```

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.

$$\begin{aligned}
start_x &= left\ eye_x - 0.5 \times (right\ eye_x - left\ eye_x) \\
end_x &= right\ eye_x + 0.5 \times (right\ eye_x - left\ eye_x) \\
start_y &= eyes_y - (mouth_y - eyes_y) \\
end_y &= mouth_y + (mouth_y - eyes_y)
\end{aligned} \tag{2}$$

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

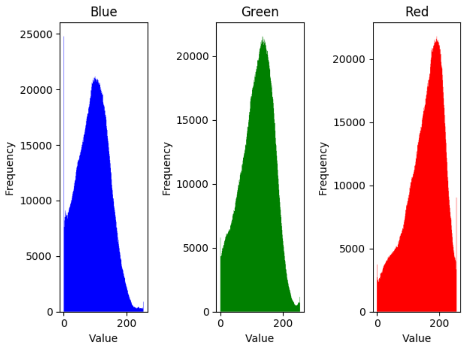


Figure 10: HSV distribution of female training images

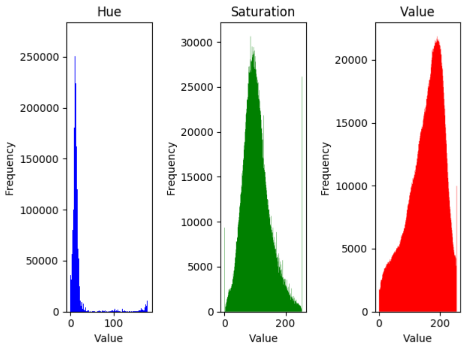


Figure 11: HSV colour detection on frame 160 of clip 1

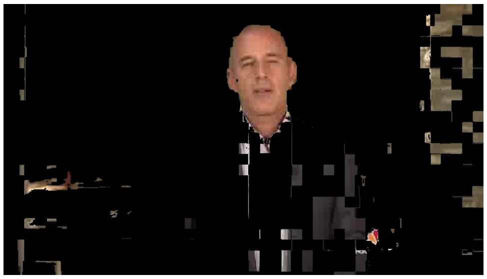
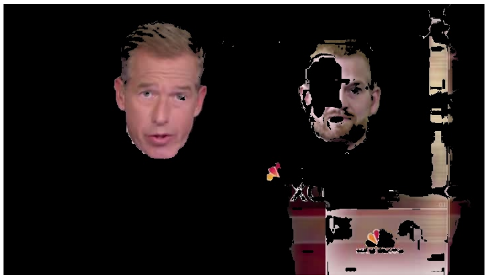


Figure 12: HSV colour detection on frame 50 of clip 1



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

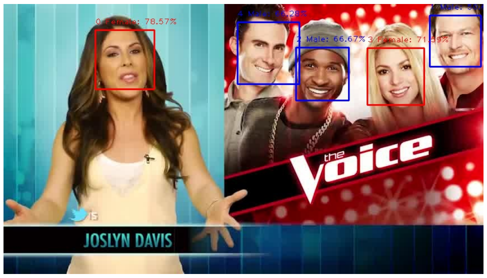
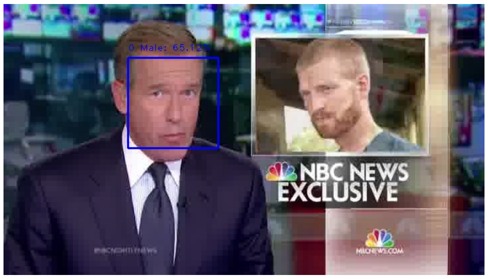


Figure 14: Face detection on frame 110 of clip 1



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

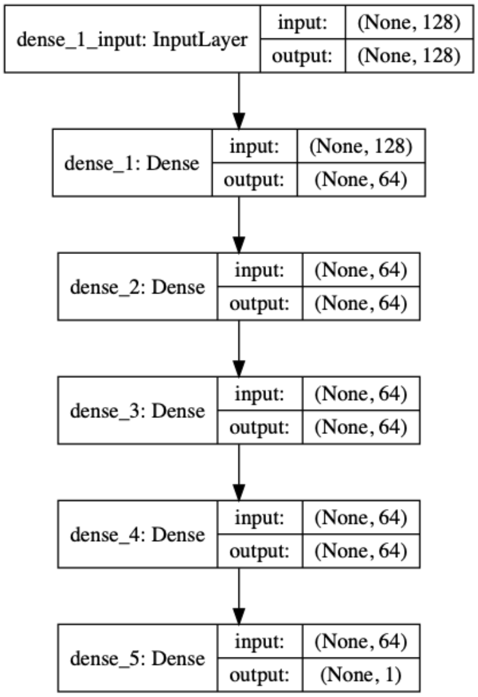
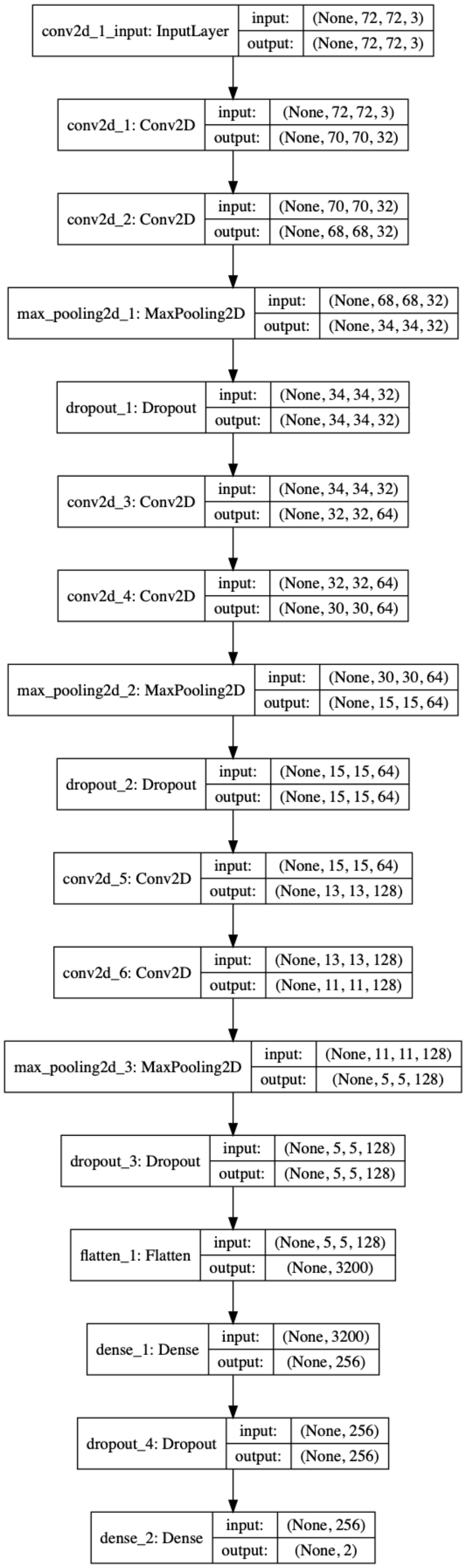

dense_1_input: InputLayer | input: (None, 128) | output: (None, 128)
dense_1: Dense | input: (None, 128) | output: (None, 64)
dense_2: Dense | input: (None, 64) | output: (None, 64)
dense_3: Dense | input: (None, 64) | output: (None, 64)
dense_4: Dense | input: (None, 64) | output: (None, 64)
dense_5: Dense | input: (None, 64) | output: (None, 1)

Figure 16: CNN model


conv2d_1_input: InputLayer | input: (None, 72, 72, 3) | output: (None, 72, 72, 3)
conv2d_1: Conv2D | input: (None, 72, 72, 3) | output: (None, 70, 70, 32)
conv2d_2: Conv2D | input: (None, 70, 70, 32) | output: (None, 68, 68, 32)
max_pooling2d_1: MaxPooling2D | input: (None, 68, 68, 32) | output: (None, 34, 34, 32)
dropout_1: Dropout | input: (None, 34, 34, 32) | output: (None, 34, 34, 32)
conv2d_3: Conv2D | input: (None, 34, 34, 32) | output: (None, 32, 32, 64)
conv2d_4: Conv2D | input: (None, 32, 32, 64) | output: (None, 30, 30, 64)
max_pooling2d_2: MaxPooling2D | input: (None, 30, 30, 64) | output: (None, 15, 15, 64)
dropout_2: Dropout | input: (None, 15, 15, 64) | output: (None, 15, 15, 64)
conv2d_5: Conv2D | input: (None, 15, 15, 64) | output: (None, 13, 13, 128)
conv2d_6: Conv2D | input: (None, 13, 13, 128) | output: (None, 11, 11, 128)
max_pooling2d_3: MaxPooling2D | input: (None, 11, 11, 128) | output: (None, 5, 5, 128)
dropout_3: Dropout | input: (None, 5, 5, 128) | output: (None, 5, 5, 128)
flatten_1: Flatten | input: (None, 5, 5, 128) | output: (None, 3200)
dense_1: Dense | input: (None, 3200) | output: (None, 256)
dropout_4: Dropout | input: (None, 256) | output: (None, 256)
dense_2: Dense | input: (None, 256) | output: (None, 2)

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 3 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 3: 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 Make video

The relevant code for combining all the frames to a video is in `make_video()` in `utils.py`. To do logo detection, face detection, face tracking and gender classification and make a video, run the command below.

```
python3 run.py run_all -i <input directory> -o <output directory>
-d <logo path> -t <min threshold for NCC> -m <trained model path>
-c <classification model (SVM, NN_SIFT, or CNN)> -v <name of output video>
-f <frame per second>
```

6 References

Video shot boundary detection based on color histogram

Wikipedia - Shot transition detection

Shot detection using pixel wise difference with adaptive threshold and color histogram method in compressed and uncompressed video

7 Code

Listing 1: shot.py

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

def add_shot_number(input_dir, output_dir, shots):
    """
    """
    exts = [".jpg", ".png"]
    img_names = [img for img in os.listdir(input_dir) if img.endswith(tuple(exts))]
    # Sort images by name in ascending order
    img_names.sort(key=lambda img: int("".join(filter(str.isdigit, img))))

    i = 0
    shot = 0
    for filename in img_names:
        image_num = int("".join(filter(str.isdigit, filename)))
        img = cv2.imread(os.path.join(input_dir, filename))
        if i < len(shots) and image_num == shots[i]:
            shot += 1
            i += 1
        # Add shot number to frame
        text = str(shot)
        # Get width and height of the text box
        text_width, text_height = cv2.getTextSize(text, cv2.FONT_HERSHEY_PLAIN, fontScale
            ↪ =1.5, thickness=2)[0]
        # Set the text start position
        text_x = 10
        text_y = img.shape[0] - 10
        box_coords = ((text_x, text_y + 2), (text_x + text_width - 2, text_y -
            ↪ text_height - 4))
        cv2.rectangle(img, box_coords[0], box_coords[1], (255, 255, 255), cv2.FILLED)
        cv2.putText(img, text, (text_x, text_y), cv2.FONT_HERSHEY_PLAIN, fontScale=1.5,
            ↪ color=(0, 0, 0), thickness=2)
        # Save frame
        cv2.imwrite(os.path.join(output_dir, filename), img)
        cv2.imshow("A box!", img)
        cv2.waitKey(0)

def shot_detection(input_dir, method, k):
    """
    Get graphs of scores of shot changes using the method specify in method.
    Method should be either SAD2 or HD.
    """
    scores = []
    exts = [".jpg", ".png"]
    img_names = [img for img in os.listdir(input_dir) if img.endswith(tuple(exts))]
    # 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 method == "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 method == "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 method == "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 method value")

    y_mean = [np.mean(scores)] * len(x)
    threshold = [np.mean(scores) + k * np.std(scores)] * len(x)
    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, label="Scores")
    # Plot the average line
    plt.plot(x, y_mean, label="Mean", linestyle="--")
    plt.plot(x, threshold, label="Threshold k = " + str(k), linestyle="--")
    plt.legend(loc="upper right")
    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

# 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):
    """
    Get similarity score between img1 and img2 using SIFT features matching and
    Lowe's ratio testing.
    """
    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:
        if m.distance < 0.6*n.distance:
            good_matches.append(m)
    num_kps = 0
    if len(kp1) <= len(kp2):
        num_kps = len(kp1)
    else:
        num_kps = len(kp2)
    score = len(good_matches) / num_kps * 100
    return score

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

    if not os.path.isdir(output_dir):
```

```python
        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
    if template.shape[0] < 50:
        scales = np.linspace(0.8, 1.0, 6)[::-1]
    else:
        # scales = np.linspace(0.1, 1.0, 25)[::-1]
        scales = np.linspace(0.5, 1.0, 10)[::-1]
    templates = []
    ratios = []
    for scale in scales:
        resized = imutils.resize(template_g, width=int(template_g.shape[1] * scale))
        templates.append(resized)
        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, 100, 200)

        p = -1
        q = -1
        matches = []
        # 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

            # First pass - normalized cross correlation
            match = cv2.matchTemplate(img_g, templates[i], cv2.TM_CCORR_NORMED) # img_g.shape - template_g.shape + 1
            if p == -1 and q == -1:
                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)
        # If the match with the highest score is smaller than the min threshold,
        # there is no match in this image and just saves the input image.
        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 < 5:
            cv2.imwrite(os.path.join(output_dir, img_name), img)
            continue
        boxes.append((r, max_y, max_x, 1))

        # Matches obtained from first pass
        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))
                # Second pass - SIFT features matching
                match_score = get_score(template, img[start_y:end_y, start_x:end_x])
                if match_score > 5:
                    boxes.append((r1, y1, x1, 1))

        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

```python
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)
    ax2.hist(S, bins=256,
        range=(0.0, 255.0),histtype="stepfilled", color="g", label="Saturation")
    plt.title("Saturation")
    plt.xlabel("Value")
    plt.ylabel("Frequency")
    ax3 = f.add_subplot(1, 3, 3)
    ax3.hist(V, bins=256,
        range=(0.0, 255.0), histtype="stepfilled", color="r", label="Value")
    plt.title("Value")
    plt.xlabel("Value")
    plt.ylabel("Frequency")
    f.tight_layout()
    f.savefig(imgs_dir + "_hsv_distributions.png")
    plt.show()

    f = plt.figure()
    ax1 = f.add_subplot(1, 3, 1)
    ax1.hist(B, bins=256,
        range=(0.0, 255.0), histtype="stepfilled", color="b", label="Blue")
    plt.title("Blue")
    plt.xlabel("Value")
    plt.ylabel("Frequency")
    ax2 = f.add_subplot(1, 3, 2)
    ax2.hist(G, bins=256,
        range=(0.0, 255.0),histtype="stepfilled", color="g", label="Saturation")
    plt.title("Green")
    plt.xlabel("Value")
    plt.ylabel("Frequency")
    ax3 = f.add_subplot(1, 3, 3)
    ax3.hist(R, bins=256,
        range=(0.0, 255.0), histtype="stepfilled", color="r", label="Red")
    plt.title("Red")
    plt.xlabel("Value")
    plt.ylabel("Frequency")
    f.tight_layout()
    f.savefig(imgs_dir + "_rgb_distributions.png")
    plt.show()

def crop_images(old_dir, new_dir):
    """
    Crop .jpg images in old_dir given coordinates of left eye, right eye,
    nose and mouth in .mat files. Save cropped images in new_dir.
    """
    print("crop_images")
    os.mkdir(new_dir)

    H, S, V = None, None, None
    for filename in os.listdir(old_dir):
        if not filename.endswith(".jpg"):
            continue

        index = filename.find(".jpg")
        name = filename[:index]

        # Approximate coordinates of face
        coords = scipy.io.loadmat(old_dir + name + ".mat")
        start_x = int(coords["x"][0][0] - 0.5*(coords["x"][1][0] - coords["x"][0][0]))
        end_x = int(coords["x"][1][0] + 0.5*(coords["x"][1][0] - coords["x"][0][0]))
        start_y = int(coords["y"][0][0] - (coords["y"][3][0] - coords["y"][0][0]))
        end_y = int(coords["y"][3][0] + (coords["y"][3][0] - coords["y"][2][0]))
        img = io.imread(old_dir + filename)
        face = img[start_y:end_y, start_x:end_x]
        # Save cropped image
        scipy.misc.imsave(new_dir + name + ".png", face)

def resize_images(input_dir, output_dir, size=72):
    """
    Resize images in input_dir to size x size.
    """
    print("resize_images")
    if not os.path.isdir(output_dir):
        os.mkdir(output_dir)

    for filename in os.listdir(input_dir):
        if not filename.endswith(".png"):
            continue
        img = cv2.imread(os.path.join(input_dir, filename))
        x, y, d = img.shape
        # Pad image to square
        if x > y:
            padded_img = np.pad(img, ((0, 0), (0, x - y), (0, 0)),
                mode="constant", constant_values=0)
        else:
            padded_img = np.pad(img, ((0, y - x), (0, 0), (0, 0)),
                mode="constant", constant_values=0)
        resized_img = cv2.resize(padded_img, (size, size))
        scipy.misc.imsave(output_dir + filename, cv2.cvtColor(resized_img, cv2.COLOR_BGR2RGB))

def get_features(input_dir):
    """
    Get keypoints and descriptors of the images in input_dir with SIFT.
    """
    kps, des = None, None
    sift = cv2.xfeatures2d.SIFT_create()

    for filename in os.listdir(input_dir):
        if not filename.endswith(".png"):
            continue
        img = io.imread(input_dir + filename)
        kp, d = sift.detectAndCompute(img, None)

        if des is None:
            kps = kp
            des = d
        else:
            kps = np.concatenate([kps, kp], axis=0)
            des = np.concatenate([des, d], axis=0)
    return kps, des

def get_data(f_dir, m_dir):
    """
    Get the images of the F and M classes from their respective directories and shuffle.
    """
    import keras
    x = []
    y = [0] * len(os.listdir(f_dir)) + [1] * len(os.listdir(m_dir))
    imgs = os.listdir(f_dir) + os.listdir(m_dir)
    for filename in os.listdir(f_dir):
        if not filename.endswith(".png"):
            continue
        x.append(cv2.imread(os.path.join(f_dir, filename)))
    for filename in os.listdir(m_dir):
        if not filename.endswith(".png"):
            continue
        x.append(cv2.imread(os.path.join(m_dir, filename)))

    x, y = shuffle(x, y)
    y = keras.utils.to_categorical(y, num_classes=2)
    return x, y

def train_model(model_path, classification):
    """
    Train either a SVM or neural network model using SIFT features or
    a CNN using face images.
    """
    OLD_F_DIR = "original_data/female/"
    OLD_M_DIR = "original_data/male/"
    F_TRAIN_DIR = "data/female_train/"
    M_TRAIN_DIR = "data/male_train/"
    F_TEST_DIR = "data/female_test/"
    M_TEST_DIR = "data/male_test/"
    F_TRAIN_CNN_DIR = "data/female_cnn_train/"
    M_TRAIN_CNN_DIR = "data/male_cnn_train/"
    F_TEST_CNN_DIR = "data/female_cnn_test/"
    M_TEST_CNN_DIR = "data/male_cnn_test/"

    if not os.path.isdir("data/"):
        os.mkdir("data/")
        crop_images(OLD_F_DIR, F_TRAIN_DIR)
        crop_images(OLD_M_DIR, M_TRAIN_DIR)

    if classification == "SVM":
        f_train_kps, f_train_des = get_features(F_TRAIN_DIR)
        m_train_kps, m_train_des = get_features(M_TRAIN_DIR)
        x_train = np.concatenate([f_train_des, m_train_des], axis=0)
        y_train = [-1] * len(f_train_des) + [1] * len(m_train_des)
        model = svm.SVC(kernel="rbf", gamma="scale", C=10.0)

        model.fit(x_train, y_train)
        # Save model
        pickle.dump(model, open(model_path, "wb"))
        print("Model saved as " + str(model_path))

    elif classification == "NN_SIFT":
        import keras
        from keras.layers import Conv2D, Dense, Dropout, Flatten, MaxPooling2D
        from keras.models import Sequential
        from keras.optimizers import Adam, SGD
        from keras.utils import plot_model

        f_train_kps, f_train_des = get_features(F_TRAIN_DIR)
        m_train_kps, m_train_des = get_features(M_TRAIN_DIR)
        x_train = np.concatenate([f_train_des, m_train_des], axis=0)
        y_train = [0] * len(f_train_des) + [1] * len(m_train_des)
        x_train, y_train = shuffle(x_train, y_train)

        f_test_kps, f_test_des = get_features(F_TEST_DIR)
        m_test_kps, m_test_des = get_features(M_TEST_DIR)
        x_test = np.concatenate([f_test_des, m_test_des], axis=0)
        y_test = [0] * len(f_test_des) + [1] * len(m_test_des)
        x_test, y_test = shuffle(x_test, y_test)

        model = Sequential()
        model.add(Dense(64, activation='relu', kernel_initializer='random_normal', input_dim=128))
        model.add(Dense(64, activation='relu', kernel_initializer='random_normal'))
        model.add(Dense(64, activation='relu', kernel_initializer='random_normal'))
        model.add(Dense(64, activation='relu', kernel_initializer='random_normal'))
        model.add(Dense(1, activation='sigmoid', kernel_initializer='random_normal'))
        model.compile(optimizer ='adam',loss='binary_crossentropy', metrics =['accuracy'])

        model.fit(x_train, y_train, batch_size=10, epochs=20)
        score = model.evaluate(x_test, y_test)
        print("score:", score)

        model.save(model_path)
        print("Model saved as " + str(model_path))
        plot_model(model, to_file="output/nn_model.png", show_shapes=True)
    elif classification == "CNN":
        import keras
        from keras.layers import Conv2D, Dense, Dropout, Flatten, MaxPooling2D
        from keras.models import Sequential
        from keras.optimizers import Adam, SGD
        from keras.utils import plot_model

        np.random.seed(123)
        size = 72

        # Pad and resize images to size by size
        # resize_images(F_TRAIN_DIR, F_TRAIN_CNN_DIR, size)
        # resize_images(M_TRAIN_DIR, M_TRAIN_CNN_DIR, size)
        # resize_images(F_TEST_DIR, F_TEST_CNN_DIR, size)
        # resize_images(M_TEST_DIR, M_TEST_CNN_DIR, size)

        x_train, y_train = get_data(F_TRAIN_CNN_DIR, M_TRAIN_CNN_DIR)
        x_test, y_test = get_data(F_TEST_CNN_DIR, M_TEST_CNN_DIR)

        model = Sequential()
        model.add(Conv2D(32, (3, 3), activation="relu", input_shape=(size, size, 3)))
        model.add(Conv2D(32, (3, 3), activation="relu"))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))

        model.add(Conv2D(64, (3, 3), activation="relu"))
        model.add(Conv2D(64, (3, 3), activation="relu"))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))

        model.add(Conv2D(128, (3, 3), activation="relu"))
        model.add(Conv2D(128, (3, 3), activation="relu"))
        model.add(MaxPooling2D(pool_size=(2, 2)))
        model.add(Dropout(0.25))

        model.add(Flatten())
        model.add(Dense(256, activation="relu"))
        model.add(Dropout(0.5))
        model.add(Dense(2, activation="softmax"))

        adam = Adam(lr=0.001)
        model.compile(loss="binary_crossentropy", optimizer=adam, metrics=["accuracy"])

        model.fit(x_train, y_train, epochs=10)
        score = model.evaluate(x_test, y_test)
        print("score:", score)

        model.save(model_path)
        print("Model saved as " + str(model_path))
        plot_model(model, to_file="output/cnn_model.png", show_shapes=True)

    else:
        raise ValueError("Illegal classification value")

def predict_model(model_path):
    """
    Get accuacy of test set for either the SVM or neural network model.
    """
```

```python
    print("predict_model")
    F_TEST_DIR = "data/female_test/"
    M_TEST_DIR = "data/male_test/"

    ext = os.path.splitext(model_path)[1]
    if ext == ".sav":
        model = pickle.load(open(model_path, "rb"))
    elif ext == ".h5":
        from keras.models import load_model
        model = load_model(model_path)
    else:
        raise ValueError("Not valid model extension")

    sift = cv2.xfeatures2d.SIFT_create()

    correct = 0
    for filename in os.listdir(F_TEST_DIR):
        if not filename.endswith(".png"):
            continue
        img = cv2.imread(os.path.join(F_TEST_DIR, filename))
        kps, des = sift.detectAndCompute(img, None)
        result = model.predict(des)
        if ext == ".sav":
            f_count = np.count_nonzero(result == -1)
            m_count = np.count_nonzero(result == 1)
        elif ext == ".h5":
            f_count = np.sum(result < 0.5)
            m_count = np.sum(result > 0.5)
        if f_count > m_count:
            correct += 1

    for filename in os.listdir(M_TEST_DIR):
        if not filename.endswith(".png"):
            continue
        img = cv2.imread(os.path.join(M_TEST_DIR, filename))
        kps, des = sift.detectAndCompute(img, None)
        result = model.predict(des)
        if ext == ".sav":
            f_count = np.count_nonzero(result == -1)
            m_count = np.count_nonzero(result == 1)
        elif ext == ".h5":
            f_count = np.sum(result < 0.5)
            m_count = np.sum(result > 0.5)
        if m_count > f_count:
            correct += 1
    score = correct / (len(os.listdir(F_TEST_DIR)) + len(os.listdir(M_TEST_DIR))) * 100
    print(str(score) + "% of images are categorized correctly")

def face_detection_hsv(input_dir, output_dir):
    """
    Face detection with HSV color detection (does not work well).
    """
    print("face_detection_hsv")

    if not os.path.isdir(output_dir):
        os.mkdir(output_dir)
    for filename in os.listdir(input_dir):
        if not filename.endswith(".jpg"):
            continue
        img = cv2.imread(os.path.join(input_dir, filename))

        # HSV color detection
        hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

        lower1 = np.array([0, 48, 80])
        upper1 = np.array([20, 255, 255])
        mask1 = cv2.inRange(hsv_img, lower1, upper1) # img_hsv.shape
        lower2 = np.array([170, 0, 0])
        upper2 = np.array([180, 255, 255])
        mask2 = cv2.inRange(hsv_img, lower2, upper2) # img_hsv.shape
        mask = cv2.bitwise_or(mask1, mask2)

        kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (10, 10))
        new_img = cv2.bitwise_and(img, img, mask=mask)
        cv2.imwrite(os.path.join(output_dir, filename), new_img)

def face_detection_cascade(input_dir, output_dir, model_path, classification):
    """
    Face detection with classifier, face tracking, and gender classification
    using the method specified in classification variable using model_path.
    """
    print("face_detection_cascade")
    XML_FILENAME = "models/haarcascade_frontalface_default.xml"
    F_TEXT = "Female"
    M_TEXT = "Male"
    O_TEXT = "Not sure"
    F_COLOR = (0, 0, 255)
    M_COLOR = (255, 0, 0)
    O_COLOR = (0, 255, 0)

    if classification == "SVM":
        model = pickle.load(open(model_path, "rb"))
    elif classification == "NN_SIFT":
        from keras.models import load_model
        model = load_model(model_path)
    elif classification == "CNN":
        from keras.models import load_model
        model = load_model(model_path)
        size = 72
    else:
        raise ValueError("Illegal classification value")

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

    exts = [".jpg", ".png"]
    img_names = [img for img in os.listdir(input_dir) if img.endswith(tuple(exts))]
    # Sort images by name in ascending order
    img_names.sort(key=lambda img: int("".join(filter(str.isdigit, img))))
    sift = cv2.xfeatures2d.SIFT_create()
    index_params = dict(algorithm=0, trees=5)
    flann = cv2.FlannBasedMatcher(index_params, None)
    face_count = 0
    prev_faces = {}

    for filename in img_names:
        img = cv2.imread(os.path.join(input_dir, filename))
        img_g = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

        ##############################
        ### Cascade face detection ###
        ##############################
        face_cascade = cv2.CascadeClassifier(XML_FILENAME)
        faces = face_cascade.detectMultiScale(img_g, scaleFactor=1.3,
            minNeighbors=3, minSize=(70, 70))
        curr_faces = {}

        for x, y, w, h in faces:
            #####################
            ### Face tracking ###
            #####################
            face = img[y:y+h, x:x+w]
            kp1, des1 = sift.detectAndCompute(face, None)
            found_face = False
            for (x2, y2, w2, h2), index in prev_faces.items():
                face2 = img[y2:y2+h, x2:x2+w]
                kp2, des2 = sift.detectAndCompute(face2, None)
                if len(kp1) < 2 or len(kp2) < 2:
                    continue
                matches = flann.knnMatch(des1, des2, k=2)
                good_matches = []
                for m, n in matches:
                    if m.distance < 0.6*n.distance:
                        good_matches.append(m)
                num_kps = 0
                if len(kp1) <= len(kp2):
                    num_kps = len(kp1)
                else:
                    num_kps = len(kp2)
                score = len(good_matches) / num_kps * 100
                if score > 45:
                    found_face = True
                    face_number = index
                    curr_faces[(x, y, w, h)] = index
            if not found_face:
                face_number = face_count
                curr_faces[(x, y, w, h)] = face_count
                face_count += 1

            #############################
            ### Gender classification ###
            #############################
            if classification == "SVM":
                # Get SIFT descriptors
                kps, des = sift.detectAndCompute(face, None)
                # Predict female or male
                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
```

Listing 5: run.py

```
import click
from pathlib import Path

@click.group()
def main():
    pass

@main.command()
@click.option("−−model_path", "−m", type=Path, default="models/cnn_model.h5")
# @click.option("−−model_path", "−m", type=Path, default="models/svm_model.sav")
@click.option("−−classification", "−c", default="CNN", help="SVM or NN_SIFT or CNN")
def train(∗∗kwargs):
    from face import train_model
    train_model(∗∗kwargs)

@main.command()
@click.option("−−model_path", "−m", type=Path, default="models/nn_model.h5")
# @click.option("−−model_path", "−m", type=Path, default="models/svm_model.sav")
def predict(∗∗kwargs):
    from face import predict_model
    predict_model(∗∗kwargs)

@main.command()
@click.option("−−model_path", "−m", type=Path, default="models/svm_model.sav")
# @click.option("−−model_path", "−m", type=Path, default="models/nn_model.h5")
@click.option("−−classification", "−c", default="SVM", help="SVM or NN_SIFT or CNN")
# Clip 1
@click.option("−−input_dir", "−i", type=Path, default="original_data/clip_1/")
@click.option("−−output_dir", "−o", type=Path, default="output/clip_1_face/")
# Clip 2
# @click.option("−−input_dir", "−i", type=Path, default="original_data/clip_2/")
# @click.option("--output_dir", "-o", type=Path, default="output/clip_2/")
def face_detection(**kwargs):
    from face import face_detection_hsv, face_detection_cascade
    # face_detection_hsv(**kwargs)
    face_detection_cascade(**kwargs)

@main.command()
# Clip 1
# @click.option("--input_dir", "-i", type=Path, default="original_data/clip_1/")
# @click.option("--output_dir", "-o", type=Path, default="output/clip_1_logo/")
# @click.option("--logo_path", "-d", type=Path, default="data/clip_1_logo2.png")
# @click.option("--min_threshold", "-t", type=float, default=0.87)
# Clip 2
# @click.option("--input_dir", "-i", type=Path, default="original_data/clip_2/")
# @click.option("--output_dir", "-o", type=Path, default="output/clip_2/")
# @click.option("--logo_path", "-d", type=Path, default="data/clip_2_logo.png")
# @click.option("--min_threshold", "-t", type=float, default=0.82)
# Clip 3
@click.option("--input_dir", "-i", type=Path, default="original_data/clip_3/")
@click.option("--output_dir", "-o", type=Path, default="output/clip_3/")
@click.option("--logo_path", "-d", type=Path, default="data/clip_3_logo.png")
@click.option("--min_threshold", "-t", type=float, default=0.82)
def logo_detection(**kwargs):
    from logo import logo_detection
    logo_detection(**kwargs)

@main.command()
@click.option("--input_dir", "-i", type=Path, default="original_data/clip_3/")
@click.option("--method", "-m", default="SAD2", help="SAD2 or HD")
@click.option("--k", "-k", default=1, help="k used in threshold")
def shot_detection(**kwargs):
    from shot import shot_detection
    shot_detection(**kwargs)

@main.command()
@click.option("--imgs_dir", "-i", default="original_data/clip_1/", type=Path)
@click.option("--vid_name", "-v", default="clip_1")
@click.option("--fps", "-fps", default=6, type=int, help="Frame per second")
def make_video(**kwargs):
    from utils import make_video
    make_video(**kwargs)

@main.command()
# Clip 1
@click.option("--input_dir", "-i", type=Path, default="original_data/clip_1/")
@click.option("--output_dir", "-o", type=Path, default="output/clip_1_shots/")
@click.option("--shots", "-s", type=list, default=[22, 156])
# Clip 2
@click.option("--input_dir", "-i", type=Path, default="original_data/clip_2/")
@click.option("--output_dir", "-o", type=Path, default="output/clip_2_shots/")
@click.option("--shots", "-s", type=list, default=[22, 156])
def add_shot_number(**kwargs):
    from shot import add_shot_number
    add_shot_number(i**kwargs)

@main.command()
# Clip 1
# @click.option("--input_dir", "-i", type=Path, default="original_data/clip_1/")
# @click.option("--output_dir", "-o", type=Path, default="output/clip_1/")
# @click.option("--logo_path", "-d", type=Path, default="data/clip_1_logo.png")
# @click.option("--min_threshold", "-t", type=float, default=0.87)
# @click.option("--model_path", "-m", type=Path, default="models/svm_model.sav")
# @click.option("--classification", "-c", default="SVM", help="SVM or NN_SIFT or CNN")
# @click.option("--vid_name", "-v", default="clip_1")
# @click.option("--fps", "-fps", default=6, type=int, help="Frame per second")
# Clip 2
# @click.option("--input_dir", "-i", type=Path, default="original_data/clip_2/")
# @click.option("--output_dir", "-o", type=Path, default="output/clip_2/")
# @click.option("--logo_path", "-d", type=Path, default="data/clip_2_logo.png")
# @click.option("--min_threshold", "-t", type=float, default=0.82)
# @click.option("--model_path", "-m", type=Path, default="models/svm_model.sav")
# @click.option("--classification", "-c", default="SVM", help="SVM or NN_SIFT or CNN")
# @click.option("--vid_name", "-v", default="clip_2")
# @click.option("--fps", "-fps", default=6, type=int, help="Frame per second")
# Clip 3
@click.option("--input_dir", "-i", type=Path, default="original_data/clip_3/")
@click.option("--output_dir", "-o", type=Path, default="output/clip_3/")
@click.option("--logo_path", "-d", type=Path, default="data/clip_3_logo.png")
@click.option("--min_threshold", "-t", type=float, default=0.82)
@click.option("--model_path", "-m", type=Path, default="models/svm_model.sav")
@click.option("--classification", "-c", default="SVM", help="SVM or NN_SIFT or CNN")
@click.option("--vid_name", "-v", default="clip_3")
@click.option("--fps", "-fps", default=6, type=int, help="Frame per second")
def run_all(input_dir, output_dir, logo_path, min_threshold, model_path, \
    classification, vid_name, fps):
    from face import face_detection_cascade
    from logo import logo_detection
    from utils import make_video
    logo_detection(input_dir=input_dir, output_dir=output_dir,\
        logo_path=logo_path, min_threshold=min_threshold)
    face_detection_cascade(input_dir=output_dir, output_dir=output_dir,\
        model_path=model_path, classification=classification)
    make_video(imgs_dir=output_dir, vid_name=vid_name, fps=fps)

if __name__ == "__main__":
    main()
```