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

$$\text{threshold} = \mu + k \times \sigma$$

where k is an integer. A shot is a series of frames taken by continuously by one camera. 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. This method is more robust against small changes in a frame.

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 the algorithm is detecting the shots, the recall (V), precision (P) and F_1 scores for each clip are calculated. They are defined as below.

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

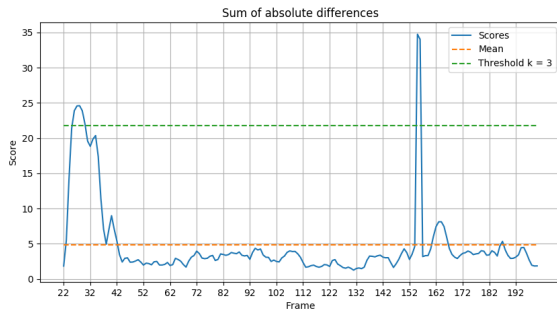Figure 1: Sum of absolute differences scores of clip 1

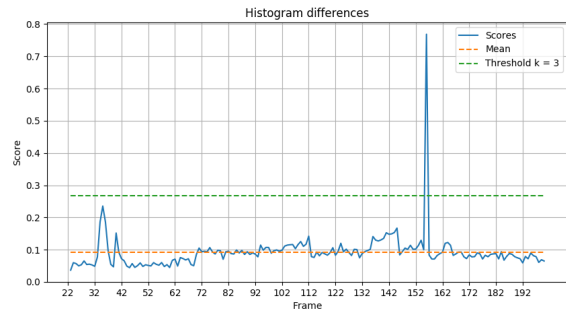Figure 2: Histogram differences scores of clip 1

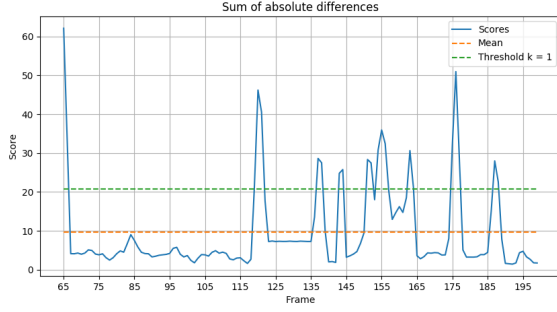Figure 3: Sum of absolute differences scores of clip 2

Figure 4: Histogram differences scores of clip 2
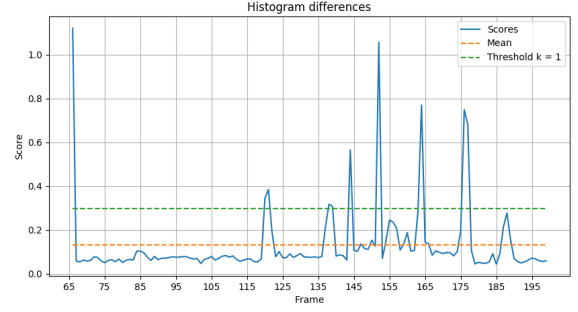

Figure 5: Sum of absolute differences scores of clip 3
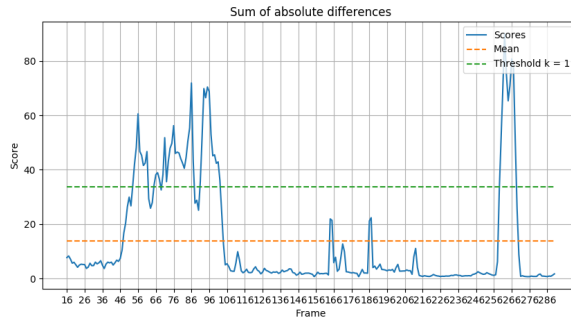
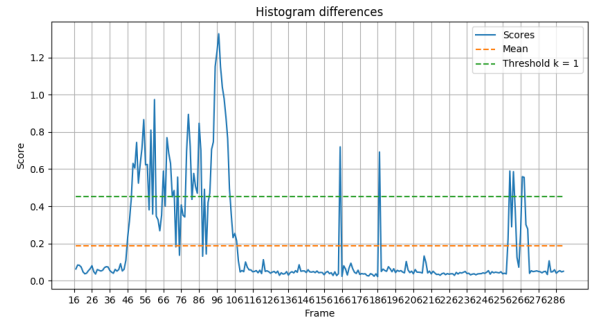Figure 6: Histogram differences scores of clip 3

Table 1 and Table 2 shows the respective scores for both methods. On average, histogram differences achieve a higher score across all three metrics comparing to sum of absolute differences method. For example, the transition from frame 164 (Figure 7) to frame 165 (Figure 8) is not detected by the latter but it is by the former. It performs especially better in the montage section in clip 3, where there are frequent shot changes.

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

2

Figure 7: Frame 164 of clip 3 　　　　　　 Figure 8: Frame 165 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 the news company's 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 in the template and the one in the frame might be slightly different, possibly due to the differences in 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 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 different threshold, it is declared as a match of the template and a box is put around the match.

Initially, I 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 with another threshold with SIFT feature matching, is used. As can be seen in Figure 10, the algorithm can detect multiple logos. In hindsight, it would have been more efficient to scale down the images to multiple scales instead of scaling

down the templates. This is because sliding the template across the image takes much longer than resizing every frame to different scales.

Figure 9: Logo detection on frame 104 of clip 1

Figure 10: 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 11 and Figure 12 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 13, it is not successful in general. It fails to detect a large area of the face of the man on the right in Figure 14 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.

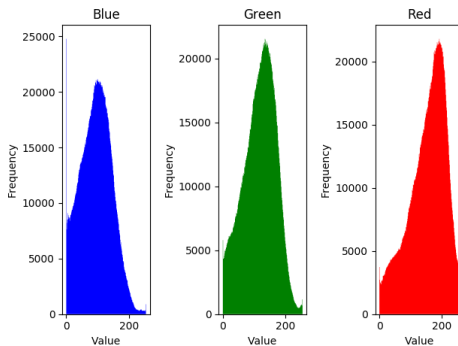Figure 11: RGB distribution of fe male training images

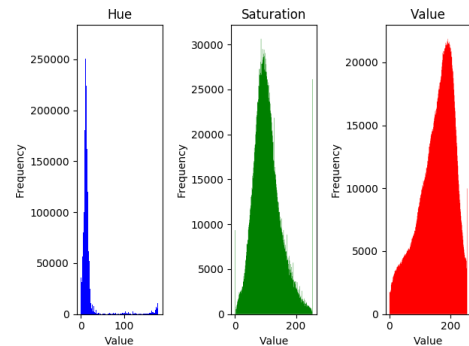Figure 12: HSV distribution of female training images

Figure 13: HSV colour detection on frame 160 of clip 1

Figure 14: 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 15. However, it is usually unable to detect faces turned to the side. The detector was unable to detect the person on the left in any of the frames he appeared in that particular position in Figure 16.

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 all the 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. The index of each face is labelled on top of the box.

Figure 15: Face detection on frame 69 of clip 2

Figure 16: 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. The accuracy is defined as the percent of correctly identified faces, not SIFT descriptors, in the cases of support-vector machine and neural network.

SVM

The SIFT descriptors of the training images are passed into the SVM model for training. A radial basis function kernel is used. Then, for each detected face in a frame, the SIFT descriptors are extracted and fed to the trained SVM model. 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. The neural network used is shown in Figure 17.

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 18 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 17: 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 18: 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, 64)

conv2d_2: Conv2D	input:	(None, 70, 70, 64)
	output:	(None, 68, 68, 64)

max_pooling2d_1: MaxPooling2D	input:	(None, 68, 68, 64)
	output:	(None, 34, 34, 64)

dropout_1: Dropout	input:	(None, 34, 34, 64)
	output:	(None, 34, 34, 64)

conv2d_3: Conv2D	input:	(None, 34, 34, 64)
	output:	(None, 32, 32, 128)

conv2d_4: Conv2D	input:	(None, 32, 32, 128)
	output:	(None, 30, 30, 128)

max_pooling2d_2: MaxPooling2D	input:	(None, 30, 30, 128)
	output:	(None, 15, 15, 128)

dropout_2: Dropout	input:	(None, 15, 15, 128)
	output:	(None, 15, 15, 128)

conv2d_5: Conv2D	input:	(None, 15, 15, 128)
	output:	(None, 13, 13, 256)

conv2d_6: Conv2D	input:	(None, 13, 13, 256)
	output:	(None, 11, 11, 256)

max_pooling2d_3: MaxPooling2D	input:	(None, 11, 11, 256)
	output:	(None, 5, 5, 256)

dropout_3: Dropout	input:	(None, 5, 5, 256)
	output:	(None, 5, 5, 256)

flatten_1: Flatten	input:	(None, 5, 5, 256)
	output:	(None, 6400)

dense_1: Dense	input:	(None, 6400)
	output:	(None, 512)

dropout_4: Dropout	input:	(None, 512)
	output:	(None, 512)

dense_2: Dense	input:	(None, 512)
	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>
-s <min size for face detection>
```

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. It consistently classify all the faces as either female and male.

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> -s <min size for face detection>
```

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):
    """
    Add shot number using the shot changes specified in variable shots.
    """
    print("add_shot_number")
    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))))

    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")])
```

9

```python
54
55    # Sum of absolute differences
56    if method == "SAD2":
57        prev_img = None
58        for i in range(len(img_names)):
59            # Default type is numpy.uint64
60            curr_img = cv2.imread(os.path.join(input_dir, img_names[i])).astype(np.int64)
61            if i == 0:
62                r, c, d = curr_img.shape
63                next_img = cv2.imread(os.path.join(input_dir, img_names[i + 1])).astype(np.
                    ↪ int64)
64                score = np.sum(np.abs(curr_img − next_img))
65            elif i == len(img_names) − 1:
66                score = np.sum(np.abs(curr_img − prev_img))
67            else:
68                next_img = cv2.imread(os.path.join(input_dir, img_names[i + 1])).astype(np.
                    ↪ int64)
69                score = 0.5 * np.sum(np.abs(curr_img − prev_img)) + 0.5 * np.sum(np.abs(
                    ↪ curr_img − next_img))
70            scores.append(score)
71            prev_img = curr_img
72        x = np.arange(start_idx, end_idx + 1)
73        scores = np.array(scores) / (r * c * d)
74        title = "Sum of absolute differences"
75        new_filename = "output/" + input_dir.name + "_score_sad2.png"
76
77    elif method == "SAD":
78        prev_img = None
79        for i in range(len(img_names)):
80            # Default type is numpy.uint64
81            curr_img = cv2.imread(os.path.join(input_dir, img_names[i])).astype(np.int64)
82            if i == 0:
83                r, c, d = curr_img.shape
84                prev_img = curr_img
85                continue
86            score = np.sum(np.abs(curr_img − prev_img))
87            scores.append(score)
88            prev_img = curr_img
89        x = np.arange(start_idx + 1, end_idx + 1)
90        scores = np.array(scores) / (r * c * d)
91        title = "Sum of absolute differences"
92        new_filename = "output/" + input_dir.name + "_score_sad2.png"
93
94    # Histogram differences
95    elif method == "HD":
96        prev_histogram = None
97        for i in range(len(img_names)):
98            # Default type is numpy.uint8
99            curr_img = cv2.imread(os.path.join(input_dir, img_names[i]))
100           # Default type is numpy.uint8
101           curr_img_g = cv2.cvtColor(curr_img, cv2.COLOR_BGR2GRAY).astype(np.int16)
102           histogram = np.histogram(np.ravel(curr_img_g), bins=np.arange(−1, 256))
103           if i == 0:
104               r, c = curr_img_g.shape
105               prev_histogram = histogram[0]
106               continue
107           score = np.sum(np.abs(histogram[0] − prev_histogram))
108           scores.append(score)
109           prev_histogram = histogram[0]
```

```
110
111          x = np.arange(start_idx + 1, end_idx + 1)
112          scores = np.array(scores) / (r * c)
113          title = "Histogram differences"
114          new_filename = "output/" + input_dir.name + "_score_hd.png"
115
116      else:
117          raise ValueError("Illegal method value")
118
119      y_mean = [np.mean(scores)] * len(x)
120      threshold = [np.mean(scores) + k * np.std(scores)] * len(x)
121      f = plt.figure(figsize=(10, 5))
122      ax = f.gca()
123      ax.set_xticks(np.arange(start_idx, end_idx, 10))
124      plt.title(title)
125      plt.xlabel("Frame")
126      plt.ylabel("Score")
127      plt.plot(x, scores, label="Scores")
128      # Plot the average line
129      plt.plot(x, y_mean, label="Mean", linestyle="--")
130      plt.plot(x, threshold, label="Threshold k = " + str(k), linestyle="--")
131      plt.legend(loc="upper right")
132      plt.grid()
133      f.savefig(new_filename)
134      print("Output saved to " + new_filename)
```

Listing 2: logo.py

```
1  import cv2
2  import imutils
3  import numpy as np
4  import os
5
6  def get_score(img1, img2):
7      """
8      Get similarity score between img1 and img2 using SIFT features matching and
9      Lowe's ratio testing.
10     """
11     sift = cv2.xfeatures2d.SIFT_create()
12     kp1, des1 = sift.detectAndCompute(img1, None)
13     kp2, des2 = sift.detectAndCompute(img2, None)
14     if len(kp1) < 2 or len(kp2) < 2:
15         return 0
16     index_params = dict(algorithm=0, trees=5)
17     flann = cv2.FlannBasedMatcher(index_params, None)
18     matches = flann.knnMatch(des1, des2, k=2)
19     good_matches = []
20     for m, n in matches:
21         if m.distance < 0.6*n.distance:
22             good_matches.append(m)
23     num_kps = 0
24     if len(kp1) <= len(kp2):
25         num_kps = len(kp1)
26     else:
27         num_kps = len(kp2)
28     score = len(good_matches) / num_kps * 100
29     return score
30
31 def logo_detection(input_dir, output_dir, logo_path, min_threshold):
32     print("detect_logo")
```

```python
33
34        if not os.path.isdir(output_dir):
35            os.mkdir(output_dir)
36
37        template = cv2.imread(str(logo_path))
38        template_g = cv2.cvtColor(template, cv2.COLOR_BGR2GRAY)
39
40        w, h = template_g.shape[::-1]
41
42        # Compute template of different sizes
43        if template.shape[0] < 50:
44            # scales = np.linspace(0.8, 1.0, 6)[::-1]
45            scales = np.linspace(0.9, 1.0, 2)[::-1]
46        else:
47            # scales = np.linspace(0.1, 1.0, 25)[::-1]
48            scales = np.linspace(0.5, 1.0, 10)[::-1]
49        templates = []
50        ratios = []
51        for scale in scales:
52            resized = imutils.resize(template_g, width=int(template_g.shape[1] * scale))
53            templates.append(resized)
54            ratios.append(resized.shape[1] / float(template_g.shape[1]))
55
56        for img_name in os.listdir(input_dir):
57            if not img_name.endswith(".jpg"):
58                continue
59            img = cv2.imread(os.path.join(input_dir, img_name))
60            img_g = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
61            img_canny = cv2.Canny(img_g, 100, 200)
62
63            p = -1
64            q = -1
65            matches = []
66            # Loop through the templates
67            for i in range(len(templates) - 1, -1, -1):
68                # Stop when template is bigger than image
69                if img_g.shape[0] < templates[i].shape[0] or img_g.shape[1] < templates[i].
                    ↪ shape[1]:
70                    break
71
72                # First pass - normalized cross correlation
73                match = cv2.matchTemplate(img_g, templates[i], cv2.TM_CCORR_NORMED) # img_g.shape
                    ↪ - template_g.shape + 1
74                if p == -1 and q == -1:
75                    p, q = match.shape
76                m, n = match.shape
77                matches.append(np.pad(match, ((0, p - m), (0, q - n)), mode="constant",
                    ↪ constant_values=0))
78
79            boxes = []
80
81            matches = np.array(matches)
82            r, max_y, max_x = np.unravel_index(np.argmax(matches), matches.shape)
83            r = ratios[len(ratios) - 1 - r]
84            max_val = np.max(matches)
85            max_thresh = max(max_val * 0.95, min_threshold)
86            # If the match with the highest score is smaller than the min threshold,
87            # there is no match in this image and just saves the input image.
88            if max_val < min_threshold:
```

```
 89                cv2.imwrite(os.path.join(output_dir, img_name), img)
 90                continue
 91            start_x, start_y = max_x, max_y
 92            end_x, end_y = int((max_x + w * r)), int((max_y + h * r))
 93            match_score = get_score(template, img[start_y:end_y, start_x:end_x])
 94            if match_score < 5:
 95                cv2.imwrite(os.path.join(output_dir, img_name), img)
 96                continue
 97            boxes.append((r, max_y, max_x, 1))
 98
 99        # Matches obtained from first pass
100        match_locations = np.where(matches >= max_thresh)
101        for i in range(len(match_locations[0])):
102            r1, y1, x1 = ratios[len(ratios) - 1 - match_locations[0][i]], match_locations
                    ↪ [1][i], match_locations[2][i]
103            found = False
104            for j in range(len(boxes)):
105                r2, y2, x2, count = boxes[j]
106                # Check if two boxes of the same size overlap or
107                # if a smaller one is contained in the bigger one
108                if (r1 == r2 and np.abs(x1 - x2) < w * r1 and np.abs(y1 - y2) < h * r1) or
                        ↪ \
109                    ((r1 < r2) and (x1 <= (x2 + w * r2) <= (x1 + w * r1)) and (y1 <= (y2 + h
                            ↪ * r2) <= (y1 + h * r1))) or\
110                    (np.abs(x1 - x2) < 0.5 * w * r1 and np.abs((x1 + w * r1) - (x2 + w * r2
                            ↪ )) < 0.5 * w * r1 and\
111                    np.abs(y1 - y2) < 0.5 * h * r1 and np.abs((y1 + h * r1) - (y2 + h * r2)
                            ↪ ) < 0.5 * h * r1):
112                    boxes[j] = (r2, y2, x2, count + 1)
113                    found = True
114                    break
115            if not found:
116                start_x, start_y = x1, y1
117                end_x, end_y = int((x1 + w * r1)), int((y1 + h * r1))
118                # Second pass - SIFT features matching
119                match_score = get_score(template, img[start_y:end_y, start_x:end_x])
120                if match_score > 5:
121                    boxes.append((r1, y1, x1, 1))
122
123        for r, y, x, count in boxes:
124            start_x, start_y = x, y
125            end_x, end_y = int((x + w * r)), int((y + h * r))
126            cv2.rectangle(img, (start_x, start_y), (end_x, end_y), (0, 255, 0), thickness
                    ↪ =2)
127
128        cv2.imwrite(os.path.join(output_dir, img_name), img)
129        # cv2.imshow("img", img)
130        # cv2.waitKey(0)
```

Listing 3: face.py

```
1  import copy
2  import cv2
3  import matplotlib.pyplot as plt
4  import numpy as np
5  import os
6  import pickle
7  import scipy.io, scipy.misc
8  from skimage import io
9  from skimage import transform as tf
```

```python
10    from skimage.color import rgb2hsv
11    from sklearn import svm
12    from utils import shuffle
13
14    def visualize_distributions(imgs_dir):
15        """
16        Visualize the distributions of values in HSV and BGR of pictures in imgs_dir.
17        """
18        H, S, V = None, None, None
19        B, G, R = None, None, None
20        for filename in os.listdir(imgs_dir):
21                if not filename.endswith(".png"):
22                    continue
23                img = cv2.imread(imgs_dir + filename)
24                hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
25                if H is None:
26                    H = hsv_img[..., 0].flatten()
27                    S = hsv_img[..., 1].flatten()
28                    V = hsv_img[..., 2].flatten()
29                    B = img[..., 0].flatten()
30                    G = img[..., 1].flatten()
31                    R = img[..., 2].flatten()
32                else:
33                    H = np.concatenate([H, hsv_img[..., 0].flatten()])
34                    S = np.concatenate([S, hsv_img[..., 1].flatten()])
35                    V = np.concatenate([V, hsv_img[..., 2].flatten()])
36                    B = np.concatenate([B, img[..., 0].flatten()])
37                    G = np.concatenate([G, img[..., 1].flatten()])
38                    R = np.concatenate([R, img[..., 2].flatten()])
39
40        # Plot
41        f = plt.figure()
42        ax1 = f.add_subplot(1, 3, 1)
43        ax1.hist(H, bins=180,
44            range=(0.0, 180.0), histtype="stepfilled", color="b", label="Hue")
45        plt.title("Hue")
46        plt.xlabel("Value")
47        plt.ylabel("Frequency")
48        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 left eye, right eye,
90      nose and mouth in .mat files. Save cropped images in new_dir.
91      """
92      print("crop_images")
93      os.mkdir(new_dir)
94
95      H, S, V = None, None, None
96      for filename in os.listdir(old_dir):
97          if not filename.endswith(".jpg"):
98              continue
99
100         index = filename.find(".jpg")
101         name = filename[:index]
102
103         # Approximate coordinates of face
104         coords = scipy.io.loadmat(old_dir + name + ".mat")
105         start_x = int(coords["x"][0][0] − 0.5*(coords["x"][1][0] − coords["x"][0][0]))
106         end_x = int(coords["x"][1][0] + 0.5*(coords["x"][1][0] − coords["x"][0][0]))
107         start_y = int(coords["y"][0][0] − (coords["y"][3][0] − coords["y"][0][0]))
108         end_y = int(coords["y"][3][0] + (coords["y"][3][0] − coords["y"][2][0]))
109         img = io.imread(old_dir + filename)
110         face = img[start_y:end_y, start_x:end_x]
111         # Save cropped image
112         scipy.misc.imsave(new_dir + name + ".png", face)
113
114  def resize_images(input_dir, output_dir, size=72):
115      """
116      Resize images in input_dir to size x size.
117      """
118      print("resize_images")
119      if not os.path.isdir(output_dir):
120          os.mkdir(output_dir)
121
122      for filename in os.listdir(input_dir):
123          if not filename.endswith(".png"):
124              continue
125          img = cv2.imread(os.path.join(input_dir, filename))
126          x, y, d = img.shape
127          # Pad image to square
```

```python
128          if x > y:
129              padded_img = np.pad(img, ((0, 0), (0, x − y), (0, 0)),
130                  mode="constant", constant_values=0)
131          else:
132              padded_img = np.pad(img, ((0, y − x), (0, 0), (0, 0)),
133                  mode="constant", constant_values=0)
134          resized_img = cv2.resize(padded_img, (size, size))
135          scipy.misc.imsave(output_dir + filename, cv2.cvtColor(resized_img, cv2.
                  ↪ COLOR_BGR2RGB))
136
137  def get_features(input_dir):
138      """
139      Get keypoints and descriptors of the images in input_dir with SIFT.
140      """
141      kps, des = None, None
142      sift = cv2.xfeatures2d.SIFT_create()
143
144      for filename in os.listdir(input_dir):
145          if not filename.endswith(".png"):
146              continue
147          img = io.imread(input_dir + filename)
148          kp, d = sift.detectAndCompute(img, None)
149
150          if des is None:
151              kps = kp
152              des = d
153          else:
154              kps = np.concatenate([kps, kp], axis=0)
155              des = np.concatenate([des, d], axis=0)
156      return kps, des
157
158  def get_data(f_dir, m_dir):
159      """
160      Get the images of the F and M classes from their respective directories and shuffle.
161      """
162      import keras
163      x = []
164      y = [0] * len(os.listdir(f_dir)) + [1] * len(os.listdir(m_dir))
165      imgs = os.listdir(f_dir) + os.listdir(m_dir)
166      for filename in os.listdir(f_dir):
167          if not filename.endswith(".png"):
168              continue
169          x.append(cv2.imread(os.path.join(f_dir, filename)))
170      for filename in os.listdir(m_dir):
171          if not filename.endswith(".png"):
172              continue
173          x.append(cv2.imread(os.path.join(m_dir, filename)))
174
175      x, y = shuffle(x, y)
176      y = keras.utils.to_categorical(y, num_classes=2)
177      return x, y
178
179  def train_model(model_path, classification):
180      """
181      Train either a SVM or neural network model using SIFT features or
182      a CNN using face images.
183      """
184      OLD_F_DIR = "original_data/female/"
185      OLD_M_DIR = "original_data/male/"
```

```python
186        F_TRAIN_DIR = "data/female_train/"
187        M_TRAIN_DIR = "data/male_train/"
188        F_TEST_DIR = "data/female_test/"
189        M_TEST_DIR = "data/male_test/"
190        F_TRAIN_CNN_DIR = "data/female_cnn_train/"
191        M_TRAIN_CNN_DIR = "data/male_cnn_train/"
192        F_TEST_CNN_DIR = "data/female_cnn_test/"
193        M_TEST_CNN_DIR = "data/male_cnn_test/"
194
195    if not os.path.isdir("data/"):
196        os.mkdir("data/")
197        crop_images(OLD_F_DIR, F_TRAIN_DIR)
198        crop_images(OLD_M_DIR, M_TRAIN_DIR)
199
200    if classification == "SVM":
201        f_train_kps, f_train_des = get_features(F_TRAIN_DIR)
202        m_train_kps, m_train_des = get_features(M_TRAIN_DIR)
203        x_train = np.concatenate([f_train_des, m_train_des], axis=0)
204        y_train = [-1] * len(f_train_des) + [1] * len(m_train_des)
205        model = svm.SVC(kernel="rbf", gamma="scale", C=10.0)
206
207        model.fit(x_train, y_train)
208        # Save model
209        pickle.dump(model, open(model_path, "wb"))
210        print("Model saved as " + str(model_path))
211
212    elif classification == "NN_SIFT":
213        import keras
214        from keras.layers import Conv2D, Dense, Dropout, Flatten, MaxPooling2D
215        from keras.models import Sequential
216        from keras.optimizers import Adam, SGD
217        from keras.utils import plot_model
218
219        f_train_kps, f_train_des = get_features(F_TRAIN_DIR)
220        m_train_kps, m_train_des = get_features(M_TRAIN_DIR)
221        x_train = np.concatenate([f_train_des, m_train_des], axis=0)
222        y_train = [0] * len(f_train_des) + [1] * len(m_train_des)
223        x_train, y_train = shuffle(x_train, y_train)
224
225        f_test_kps, f_test_des = get_features(F_TEST_DIR)
226        m_test_kps, m_test_des = get_features(M_TEST_DIR)
227        x_test = np.concatenate([f_test_des, m_test_des], axis=0)
228        y_test = [0] * len(f_test_des) + [1] * len(m_test_des)
229        x_test, y_test = shuffle(x_test, y_test)
230
231        model = Sequential()
232        model.add(Dense(64, activation='relu', kernel_initializer='random_normal', input_dim
               ↪ =128))
233        model.add(Dense(64, activation='relu', kernel_initializer='random_normal'))
234        model.add(Dense(64, activation='relu', kernel_initializer='random_normal'))
235        model.add(Dense(64, activation='relu', kernel_initializer='random_normal'))
236        model.add(Dense(1, activation='sigmoid', kernel_initializer='random_normal'))
237        model.compile(optimizer ='adam',loss='binary_crossentropy', metrics =['accuracy'])
238
239        model.fit(x_train, y_train, batch_size=10, epochs=20)
240        score = model.evaluate(x_test, y_test)
241        print("score:", score)
242
243        model.save(model_path)
```

```python
244          print("Model saved as " + str(model_path))
245          plot_model(model, to_file="output/nn_model.png", show_shapes=True)
246
247      elif classification == "CNN":
248          import keras
249          from keras.layers import Conv2D, Dense, Dropout, Flatten, MaxPooling2D
250          from keras.models import Sequential
251          from keras.optimizers import Adam, SGD
252          from keras.utils import plot_model
253
254          np.random.seed(123)
255          size = 72
256
257          # Pad and resize images to size by size
258          # resize_images(F_TRAIN_DIR, F_TRAIN_CNN_DIR, size)
259          # resize_images(M_TRAIN_DIR, M_TRAIN_CNN_DIR, size)
260          # resize_images(F_TEST_DIR, F_TEST_CNN_DIR, size)
261          # resize_images(M_TEST_DIR, M_TEST_CNN_DIR, size)
262
263          x_train, y_train = get_data(F_TRAIN_CNN_DIR, M_TRAIN_CNN_DIR)
264          x_test, y_test = get_data(F_TEST_CNN_DIR, M_TEST_CNN_DIR)
265
266          model = Sequential()
267          model.add(Conv2D(32, (3, 3), activation="relu", input_shape=(size, size, 3)))
268          model.add(Conv2D(32, (3, 3), activation="relu"))
269          model.add(MaxPooling2D(pool_size=(2, 2)))
270          model.add(Dropout(0.25))
271
272          model.add(Conv2D(64, (3, 3), activation="relu"))
273          model.add(Conv2D(64, (3, 3), activation="relu"))
274          model.add(MaxPooling2D(pool_size=(2, 2)))
275          model.add(Dropout(0.25))
276
277          model.add(Conv2D(128, (3, 3), activation="relu"))
278          model.add(Conv2D(128, (3, 3), activation="relu"))
279          model.add(MaxPooling2D(pool_size=(2, 2)))
280          model.add(Dropout(0.25))
281
282          model.add(Flatten())
283          model.add(Dense(256, activation="relu"))
284          model.add(Dropout(0.5))
285          model.add(Dense(2, activation="softmax"))
286
287          adam = Adam(lr=0.0005)
288          model.compile(loss="binary_crossentropy", optimizer=adam, metrics=["accuracy"])
289
290          model.fit(x_train, y_train, epochs=10)
291          score = model.evaluate(x_test, y_test)
292          print("score:", score)
293
294          model.save(model_path)
295          print("Model saved as " + str(model_path))
296          plot_model(model, to_file="output/cnn_model.png", show_shapes=True)
297
298      else:
299          raise ValueError("Illegal classification value")
300
301  def predict_model(model_path):
302      """
```

```
303        Get accuacy of test set for either the SVM or neural network model.
304        """
305        print("predict_model")
306        F_TEST_DIR = "data/female_test/"
307        M_TEST_DIR = "data/male_test/"
308
309        ext = os.path.splitext(model_path)[1]
310        if ext == ".sav":
311            model = pickle.load(open(model_path, "rb"))
312        elif ext == ".h5":
313            from keras.models import load_model
314            model = load_model(model_path)
315        else:
316            raise ValueError("Not valid model extension")
317
318        sift = cv2.xfeatures2d.SIFT_create()
319
320        correct = 0
321        for filename in os.listdir(F_TEST_DIR):
322            if not filename.endswith(".png"):
323                continue
324            img = cv2.imread(os.path.join(F_TEST_DIR, filename))
325            kps, des = sift.detectAndCompute(img, None)
326            result = model.predict(des)
327            if ext == ".sav":
328                f_count = np.count_nonzero(result == -1)
329                m_count = np.count_nonzero(result == 1)
330            elif ext == ".h5":
331                f_count = np.sum(result < 0.5)
332                m_count = np.sum(result > 0.5)
333            if f_count > m_count:
334                correct += 1
335
336        for filename in os.listdir(M_TEST_DIR):
337            if not filename.endswith(".png"):
338                continue
339            img = cv2.imread(os.path.join(M_TEST_DIR, filename))
340            kps, des = sift.detectAndCompute(img, None)
341            result = model.predict(des)
342            if ext == ".sav":
343                f_count = np.count_nonzero(result == -1)
344                m_count = np.count_nonzero(result == 1)
345            elif ext == ".h5":
346                f_count = np.sum(result < 0.5)
347                m_count = np.sum(result > 0.5)
348            if m_count > f_count:
349                correct += 1
350        score = correct / (len(os.listdir(F_TEST_DIR)) + len(os.listdir(M_TEST_DIR))) * 100
351        print(str(score) + "% of images are categorized correctly")
352
353    def face_detection_hsv(input_dir, output_dir):
354        """
355        Face detection with HSV color detection (does not work well).
356        """
357        print("face_detection_hsv")
358
359        if not os.path.isdir(output_dir):
360            os.mkdir(output_dir)
361        for filename in os.listdir(input_dir):
```

```python
362             if not filename.endswith(".jpg"):
363                 continue
364             img = cv2.imread(os.path.join(input_dir, filename))
365
366             # HSV color detection
367             hsv_img = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
368
369             lower1 = np.array([0, 48, 80])
370             upper1 = np.array([20, 255, 255])
371             mask1 = cv2.inRange(hsv_img, lower1, upper1) # img_hsv.shape
372             lower2 = np.array([170, 0, 0])
373             upper2 = np.array([180, 255, 255])
374             mask2 = cv2.inRange(hsv_img, lower2, upper2) # img_hsv.shape
375             mask = cv2.bitwise_or(mask1, mask2)
376
377             kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (10, 10))
378             new_img = cv2.bitwise_and(img, img, mask=mask)
379             cv2.imwrite(os.path.join(output_dir, filename), new_img)
380
381 def face_detection_cascade(input_dir, output_dir, model_path, classification, min_size):
382     """
383     Face detection with classifier, face tracking, and gender classification
384     using the method specified in classification variable using model_path.
385     """
386     print("face_detection_cascade")
387     XML_FILENAME = "models/haarcascade_frontalface_default.xml"
388     F_TEXT = "Female"
389     M_TEXT = "Male"
390     O_TEXT = "Not sure"
391     F_COLOR = (0, 0, 255)
392     M_COLOR = (255, 0, 0)
393     O_COLOR = (0, 255, 0)
394
395     if classification == "SVM":
396         model = pickle.load(open(model_path, "rb"))
397     elif classification == "NN_SIFT":
398         from keras.models import load_model
399         model = load_model(model_path)
400     elif classification == "CNN":
401         from keras.models import load_model
402         model = load_model(model_path)
403         size = 72
404     else:
405         raise ValueError("Illegal classification value")
406
407     if not os.path.isdir(output_dir):
408         os.mkdir(output_dir)
409
410     exts = [".jpg", ".png"]
411     img_names = [img for img in os.listdir(input_dir) if img.endswith(tuple(exts))]
412     # Sort images by name in ascending order
413     img_names.sort(key=lambda img: int("".join(filter(str.isdigit, img))))
414     sift = cv2.xfeatures2d.SIFT_create()
415     index_params = dict(algorithm=0, trees=5)
416     flann = cv2.FlannBasedMatcher(index_params, None)
417     face_count = 0
418     prev_faces = {}
419
420     for filename in img_names:
```

```
421        img = cv2.imread(os.path.join(input_dir, filename))
422        img_g = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
423
424        ############################
425        ### Cascade face detection ###
426        ############################
427        face_cascade = cv2.CascadeClassifier(XML_FILENAME)
428        faces = face_cascade.detectMultiScale(img_g, scaleFactor=1.3,
429            minNeighbors=5, minSize=(min_size, min_size))
430        curr_faces = {}
431
432        for x, y, w, h in faces:
433            ####################
434            ### Face tracking ###
435            ####################
436            face = img[y:y+h, x:x+w]
437            kp1, des1 = sift.detectAndCompute(face, None)
438            found_face = False
439            for (x2, y2, w2, h2), index in prev_faces.items():
440                face2 = img[y2:y2+h, x2:x2+w]
441                kp2, des2 = sift.detectAndCompute(face2, None)
442                if len(kp1) < 2 or len(kp2) < 2:
443                    continue
444                matches = flann.knnMatch(des1, des2, k=2)
445                good_matches = []
446                for m, n in matches:
447                    if m.distance < 0.6*n.distance:
448                        good_matches.append(m)
449                num_kps = 0
450                if len(kp1) <= len(kp2):
451                    num_kps = len(kp1)
452                else:
453                    num_kps = len(kp2)
454                score = len(good_matches) / num_kps * 100
455                if score > 45:
456                    found_face = True
457                    face_number = index
458                    curr_faces[(x, y, w, h)] = index
459            if not found_face:
460                face_number = face_count
461                curr_faces[(x, y, w, h)] = face_count
462                face_count += 1
463
464            ############################
465            ### Gender classification ###
466            ############################
467            if classification == "SVM":
468                # Get SIFT descriptors
469                kps, des = sift.detectAndCompute(face, None)
470                # Predict female or male
471                result = model.predict(des)
472                f_count = np.count_nonzero(result == -1)
473                m_count = np.count_nonzero(result == 1)
474                # Plot color box
475                if f_count > m_count:
476                    text = str(face_number) + " " + F_TEXT + ": " +\
477                        format(f_count/(f_count + m_count)*100, ".2f") + "%"
478                    color = F_COLOR
479                elif m_count > f_count:
```

```
480              text = str(face_number) + " " + M_TEXT + ": " +\
481                  format(m_count/(f_count + m_count)*100, ".2f") + "%"
482              color = M_COLOR
483          else:
484              text = str(face_number) + " " + O_TEXT
485              color = O_COLOR
486
487      elif classification == "NN_SIFT":
488          assert w == h
489          kps, des = sift.detectAndCompute(face, None)
490          prediction = model.predict(des)
491          f_count = np.sum(prediction < 0.5)
492          m_count = np.sum(prediction > 0.5)
493          if f_count > m_count:
494              text = str(face_number) + " " + F_TEXT + ": " +\
495                  format(f_count/(f_count + m_count)*100, ".2f") + "%"
496              color = F_COLOR
497          elif m_count > f_count:
498              text = str(face_number) + " " + M_TEXT + ": " +\
499                  format(m_count/(f_count + m_count)*100, ".2f") + "%"
500              color = M_COLOR
501          else:
502              text = str(face_number) + " " + O_TEXT
503              color = O_COLOR
504
505      elif classification == "CNN":
506          assert w == h
507          face = cv2.resize(face, (size, size))
508          prediction = model.predict(np.array([face]), verbose=0)
509          prediction = prediction[0]
510          if prediction[0] > 0.5:
511              text = str(face_number) + " " + F_TEXT + ": " +\
512                  format(prediction[0]*100, ".2f") + "%"
513              color = F_COLOR
514          elif prediction[0] < 0.5:
515              text = str(face_number) + " " + M_TEXT + ": " +\
516                  format(prediction[0]*100, ".2f") + "%"
517              color = M_COLOR
518          else:
519              text = str(face_number) + " " + O_TEXT
520              color = O_COLOR
521      cv2.rectangle(img, (x, y), (x+w, y+h), color, thickness=2)
522      cv2.putText(img, text, (x, y-10), color=color,
523          fontFace=cv2.FONT_HERSHEY_PLAIN, fontScale=1)
524
525  prev_faces = curr_faces
526  cv2.imwrite(os.path.join(output_dir, filename), img)
527  # cv2.imshow("img", img)
528  # cv2.waitKey(0)
```

Listing 4: utils.py

```
1  import cv2
2  import os
3  import numpy as np
4
5  def make_video(imgs_dir, vid_name, fps):
6      """
7      Make vid_name.mp4 using images in imgs_dir.
8      """
```

```
 9        OUTPUT_DIR = "output/"
10        if not os.path.isdir(OUTPUT_DIR):
11            os.mkdir(OUTPUT_DIR)
12        exts = [".jpg", ".png"]
13        imgs = [img for img in os.listdir(imgs_dir) if img.endswith(tuple(exts))]
14        # Sort images by name in ascending order
15        imgs.sort(key=lambda img: int("".join(filter(str.isdigit, img))))
16        frame = cv2.imread(os.path.join(imgs_dir, imgs[0]))
17        h, w, _ = frame.shape
18
19        # Make sure vid_name does not include a file extension
20        index = vid_name.find(".")
21        if index != -1:
22            vid_name = vid_name[:index]
23        file_path = OUTPUT_DIR + vid_name + ".mp4"
24        vid = cv2.VideoWriter(file_path, cv2.VideoWriter_fourcc(*"MP4V"), fps, (w, h))
25        for img in imgs:
26            vid.write(cv2.imread(os.path.join(imgs_dir, img)))
27        vid.release()
28        print("Video is now in ", file_path)
29
30    def shuffle(x, y):
31        """
32        Shuffle data x and their labels y.
33        """
34        assert len(x) == len(y)
35        idx = np.random.permutation(len(x))
36        x, y = np.array(x)[idx], np.array(y)[idx]
37        return x, y
```

Listing 5: run.py

```
 1    import click
 2    from pathlib import Path
 3
 4    @click.group()
 5    def main():
 6        pass
 7
 8    @main.command()
 9    # @click.option("--model_path", "-m", type=Path, default="models/cnn_model.h5")
10    @click.option("--model_path", "-m", type=Path, default="models/svm_model.sav")
11    @click.option("--classification", "-c", default="SVM", help="SVM or NN_SIFT or CNN")
12    def train(**kwargs):
13        from face import train_model
14        train_model(**kwargs)
15
16    @main.command()
17    # @click.option("--model_path", "-m", type=Path, default="models/nn_model.h5")
18    @click.option("--model_path", "-m", type=Path, default="models/svm_model.sav")
19    def predict(**kwargs):
20        from face import predict_model
21        predict_model(**kwargs)
22
23    @main.command()
24    @click.option("--model_path", "-m", type=Path, default="models/svm_model.sav")
25    # @click.option("--model_path", "-m", type=Path, default="models/nn_model.h5")
26    @click.option("--classification", "-c", default="SVM", help="SVM or NN_SIFT or CNN")
27    # Clip 1
28    @click.option("--input_dir", "-i", type=Path, default="original_data/clip_1/")
```

```python
29    @click.option("−−output_dir", "−o", type=Path, default="output/clip_1_face/")
30    @click.option("−−min_size", "−s", default=70, type=int, help="Min size for face
          ↪ detection")
31    # Clip 2
32    # @click.option("−−input_dir", "−i", type=Path, default="original_data/clip_2/")
33    # @click.option("−−output_dir", "−o", type=Path, default="output/clip_2/")
34    # @click.option("−−min_size", "−s", default=70, type=int, help="Min size for face detection")
35    def face_detection(**kwargs):
36        from face import face_detection_hsv, face_detection_cascade
37        # face_detection_hsv(**kwargs)
38        face_detection_cascade(**kwargs)
39
40    @main.command()
41    # Clip 1
42    # @click.option("−−input_dir", "−i", type=Path, default="original_data/clip_1/")
43    # @click.option("−−output_dir", "−o", type=Path, default="output/clip_1_logo/")
44    # @click.option("−−logo_path", "−d", type=Path, default="data/clip_1_logo2.png")
45    # @click.option("−−min_threshold", "−t", type=float, default=0.87)
46    # Clip 2
47    # @click.option("−−input_dir", "−i", type=Path, default="original_data/clip_2/")
48    # @click.option("−−output_dir", "−o", type=Path, default="output/clip_2/")
49    # @click.option("−−logo_path", "−d", type=Path, default="data/clip_2_logo.png")
50    # @click.option("−−min_threshold", "−t", type=float, default=0.82)
51    # Clip 3
52    @click.option("−−input_dir", "−i", type=Path, default="original_data/clip_3/")
53    @click.option("−−output_dir", "−o", type=Path, default="output/clip_3/")
54    @click.option("−−logo_path", "−d", type=Path, default="data/clip_3_logo.png")
55    @click.option("−−min_threshold", "−t", type=float, default=0.82)
56    def logo_detection(**kwargs):
57        from logo import logo_detection
58        logo_detection(**kwargs)
59
60    @main.command()
61    @click.option("−−input_dir", "−i", type=Path, default="original_data/clip_3/")
62    @click.option("−−method", "−m", default="SAD2", help="SAD2 or HD")
63    @click.option("−−k", "−k", default=1, help="k used in threshold")
64    def shot_detection(**kwargs):
65        from shot import shot_detection
66        shot_detection(**kwargs)
67
68    @main.command()
69    @click.option("−−imgs_dir", "−i", default="original_data/clip_1/", type=Path)
70    @click.option("−−vid_name", "−v", default="clip_1")
71    @click.option("−−fps", "−fps", default=6, type=int, help="Frame per second")
72    def make_video(**kwargs):
73        from utils import make_video
74        make_video(**kwargs)
75
76    @main.command()
77    # Clip 1
78    @click.option("−−input_dir", "−i", type=Path, default="original_data/clip_1/")
79    @click.option("−−output_dir", "−o", type=Path, default="output/clip_1_shots/")
80    @click.option("−−shots", "−s", type=list, default=[22, 156])
81    # Clip 2
82    # @click.option("−−input_dir", "−i", type=Path, default="original_data/clip_2/")
83    # @click.option("−−output_dir", "−o", type=Path, default="output/clip_2_shots/")
84    # @click.option("−−shots", "−s", type=list, default=[65, 120, 138, 144, 152, 164, 177, 188])
85    # Clip 3
86    # @click.option("−−input_dir", "−i", type=Path, default="original_data/clip_3/")
```

```python
 87    # @click.option("−−output_dir", "−o", type=Path, default="output/clip_3_shots/")
 88    # @click.option("−−shots", "−s", type=list, default=[16, 51, 57, 59, 61, 63, 65, 67, 69, 71, 72, 74, 78, 80, 84, 91,
          ↪ 95, 102, 165, 187, 260, 267])
 89    def add_shot_number(**kwargs):
 90        from shot import add_shot_number
 91        add_shot_number(**kwargs)
 92
 93    @main.command()
 94    # Clip 1
 95    # @click.option("−−input_dir", "−i", type=Path, default="output/clip_1_shots/")
 96    # @click.option("−−output_dir", "−o", type=Path, default="output/clip_1/")
 97    # @click.option("−−logo_path", "−d", type=Path, default="data/clip_1_logo.png")
 98    # @click.option("−−min_threshold", "−t", type=float, default=0.87)
 99    # @click.option("−−model_path", "−m", type=Path, default="models/svm_model.sav")
100    # @click.option("−−classification", "−c", default="SVM", help="SVM or NN_SIFT or CNN")
101    # @click.option("−−vid_name", "−v", default="clip_1")
102    # @click.option("−−fps", "−fps", default=6, type=int, help="Frame per second")
103    # @click.option("−−min_size", "−s", default=70, type=int, help="Min size for face detection")
104    # Clip 2
105    @click.option("−−input_dir", "−i", type=Path, default="output/clip_2_shots/")
106    @click.option("−−output_dir", "−o", type=Path, default="output/clip_2/")
107    @click.option("−−logo_path", "−d", type=Path, default="data/clip_2_logo.png")
108    @click.option("−−min_threshold", "−t", type=float, default=0.82)
109    @click.option("−−model_path", "−m", type=Path, default="models/svm_model.sav")
110    @click.option("−−classification", "−c", default="SVM", help="SVM or NN_SIFT or CNN")
111    @click.option("−−vid_name", "−v", default="clip_2")
112    @click.option("−−fps", "−fps", default=6, type=int, help="Frame per second")
113    @click.option("−−min_size", "−s", default=70, type=int, help="Min size for face
          ↪ detection")
114    # Clip 3
115    # @click.option("−−input_dir", "−i", type=Path, default="output/clip_3_shots/")
116    # @click.option("−−output_dir", "−o", type=Path, default="output/clip_3/")
117    # @click.option("−−logo_path", "−d", type=Path, default="data/clip_3_logo.png")
118    # @click.option("−−min_threshold", "−t", type=float, default=0.82)
119    # @click.option("−−model_path", "−m", type=Path, default="models/svm_model.sav")
120    # @click.option("−−classification", "−c", default="SVM", help="SVM or NN_SIFT or CNN")
121    # @click.option("−−vid_name", "−v", default="clip_3")
122    # @click.option("−−fps", "−fps", default=8, type=int, help="Frame per second")
123    # @click.option("−−min_size", "−s", default=40, type=int, help="Min size for face detection")
124    def run_all(input_dir, output_dir, logo_path, min_threshold, model_path, \
125        classification, vid_name, fps, min_size):
126        from face import face_detection_cascade
127        from logo import logo_detection
128        from utils import make_video
129        logo_detection(input_dir=input_dir, output_dir=output_dir,\
130            logo_path=logo_path, min_threshold=min_threshold)
131        face_detection_cascade(input_dir=output_dir, output_dir=output_dir,\
132        # face_detection_cascade(input_dir=input_dir, output_dir=output_dir,\
133            model_path=model_path, classification=classification, min_size=min_size)
134        make_video(imgs_dir=output_dir, vid_name=vid_name, fps=fps)
135
136    if __name__ == "__main__":
137        main()
```