News broadcast analysis

Christine K. C. Cheng

1 Shot detection

Sum of absolute differences

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

Histogram differences

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

Figure 1: Sum of absolute differences scores of clip 1

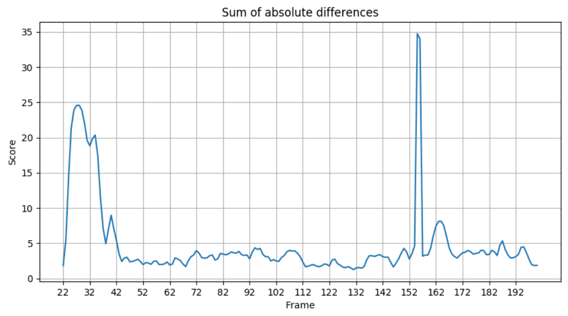


Figure 2: Histogram differences scores of clip 1

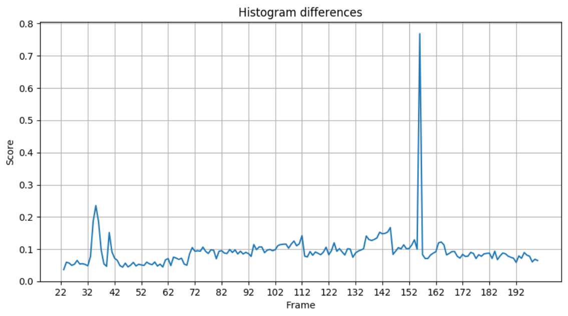


Figure 3: Sum of absolute differences scores of clip 2

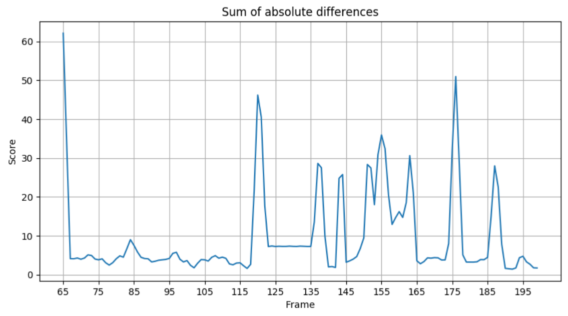


Figure 4: Histogram differences scores of clip 2

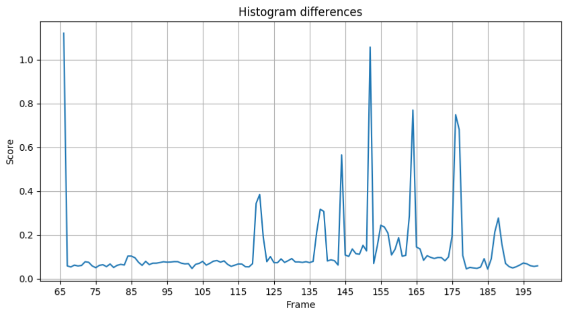


Figure 5: Sum of absolute differences scores of clip 3

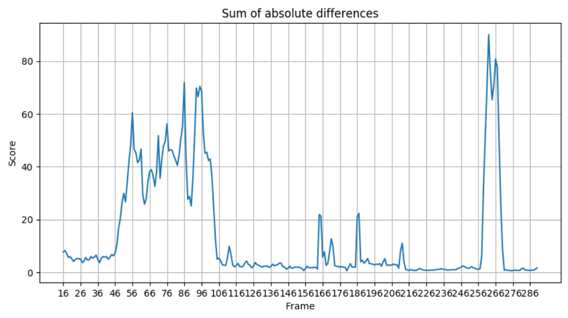
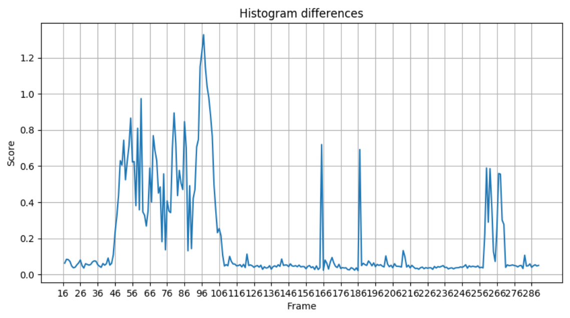


Figure 6: Histogram differences scores of clip 3



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

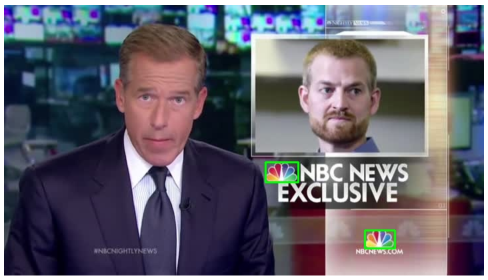
```
python3 run.py shot_detection -t <type> -i <path to frames>
```

2 Logo detection

Template matching is an object detection algorithm which is translation invariant but not scale or rotation invariant. As we are detecting a logo, it is assumed that the target of detection will be in a known orientation. Also, template matching is run on templates of different sizes because the size of the logo in a frame is unknown.

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 and a frame.

Figure 7: Logo detection on clip 1



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

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

3 Face detection

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

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

Color detection

The initial attempt to detect faces is to use skin detection - trying to filter out skin in images. Figure 7 and Figure 8 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 9, it is not successful in general. It fails to detect a large area of the face of the man on the right in Figure 10. Also, this model is especially poor when other things in the frame are very similar to human skin tone.

Figure 8: RGB distribution of fe male training images

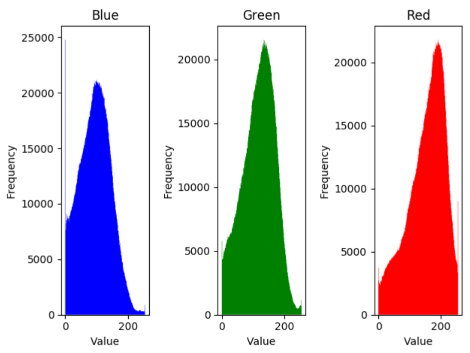


Figure 9: HSV distribution of female training images

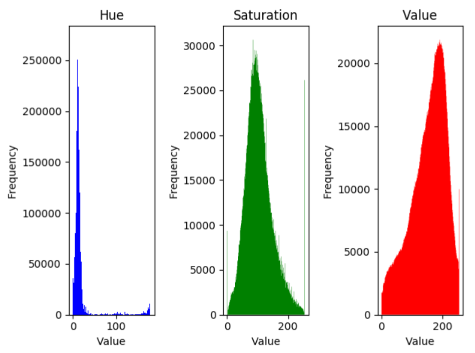


Figure 10: HSV distribution of female training images

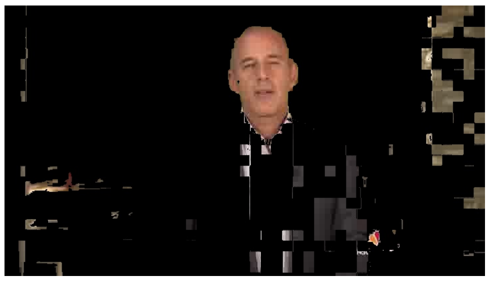
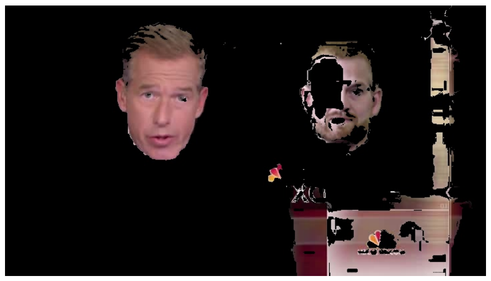


Figure 11: HSV distribution of female training images



Classifier

I ended up using `cv2.CascadeClassifier` for detecting faces. It works fairly well. However, it was not able to detect the face in Figure ????? despite trying different parameters.

4 Gender classification

90% of the images were used for training, whereas the other 10% were 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.

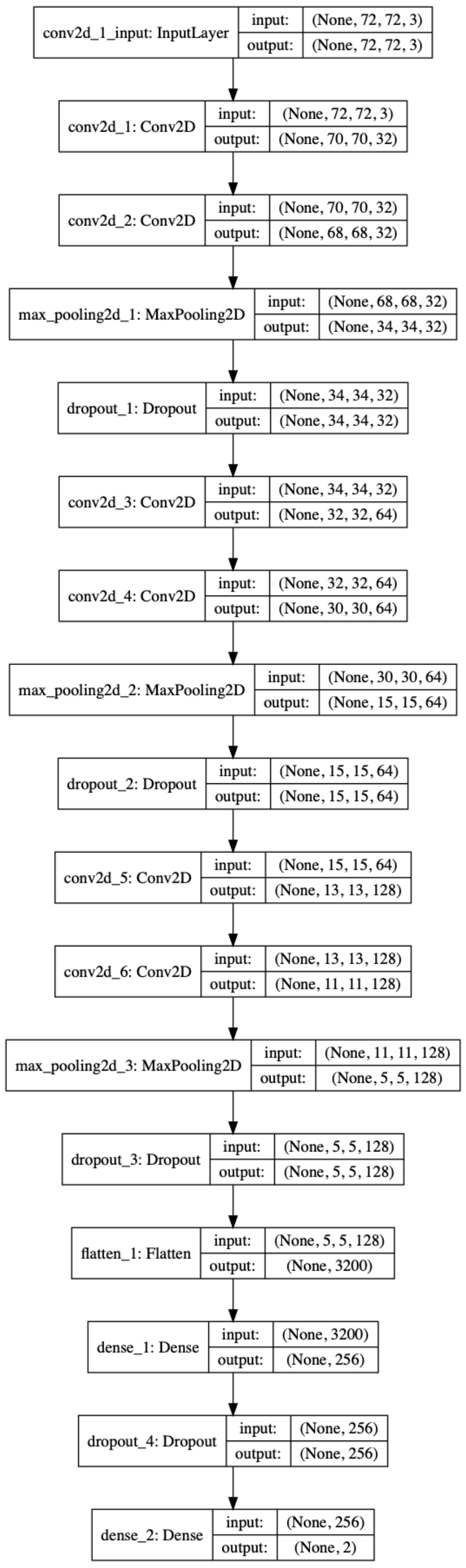
Neural network

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 11 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 12: 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 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) is in `face_detection()` in `face.py`. Run the command below.

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

Performance

Table 1 shows the test accuracies of the three models. As expected, CNN performed poorly, as good as random guesses, due to the very small training data size of 468.

Table 1: Gender classification performance

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

5 References

Video shot boundary detection based on color histogram

Wikipedia - Shot transition detection