Final Project
CSC420

Hongyi Sun, Hantang Li

April 19 2022

Project 1

Shot Detection (Hantang Li)

For code please view Shot_boundary_Qa_Qb.ipynb and Shot_boundary_Qa_Qb.pdf

Definition

By definition, a shot is a continuous footage or sequence between two edits or cuts. To detect shots, we will need to determine whether a frame or frame is a boundary between two shots based on the dissimilarities. We need to use some methods to compute the dissimilarities between every two frames of a video and

Dissimilarity Score

We will use two methods to compute the dissimilarity score:

- Histogram differences[9]: Here, we tried to use two different histogram differences. And later, we found they returned almost the same value and proceeded only to use RGB scale histogram difference.
  - Grayscale histogram: For image 1 and image 2, after transforming both images to grayscale, we calculate their Histogram differences by first computing each image's colour histogram using the "cv2.calcHist" function. Since grayscale pixel values range from 0 to 255, the histogram size is 256 and ranges from 0 to 255, and each histogram bar counts one gray colour. Then we compute the euclidian difference between two images' histograms and use that difference as the score to evaluate the differences between two frames.
  - RGB scale histogram, we compute one histogram for each colour channel and then compute Euclidean distance for each channel between two images. Then we calculate the average of all channels differences as the score.

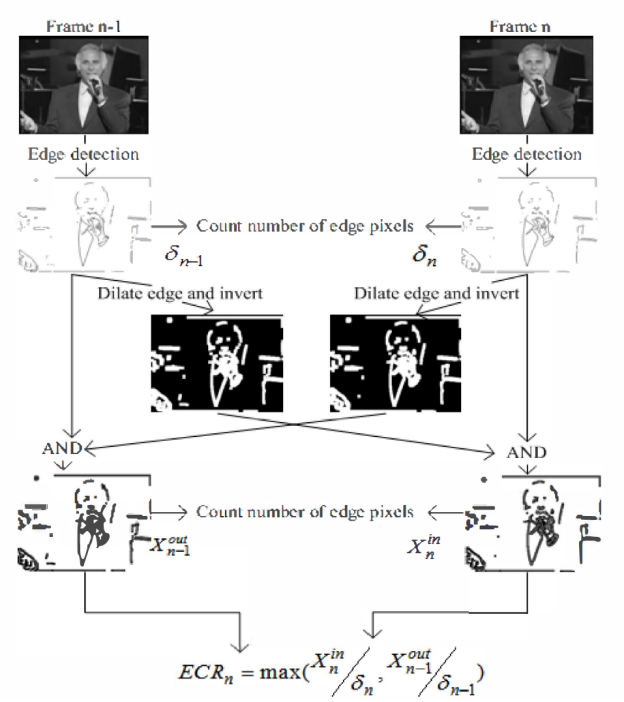

Figure 1: Calculation graph of Edge change ratio [4]

- Edge change ratio[4]: It's a ratio to evaluate how two frames' edges overlap. For image 1 and image 2, we first run canny edge detection for each image by setting the hysteresis threshold to be 0 and 200. The lecture taught that the hysteresis threshold is used to accept edges greater than some threshold and connect edges that are between the threshold. We adjusted the value by experiment so that the edge we detected contains some details of the objects in a scene and facial features but is not detailed enough to capture all the details. Then we dilate the edge by using 'cv2.dilate'. The algorithm is by convolving a filter through the image and setting the filter's centre to be the pixel with a maximum value under the filter. By doing that, we make the edges thicker, and it would be convenient to compute the intersection between two edges which we will do in the next step. To compute the intersection, we count the number of image 1's edges that are within image 2's dilated edge and then divide by image 1's edges. We do that for image 2 as well and take the maximum between those two values.

Algorithm

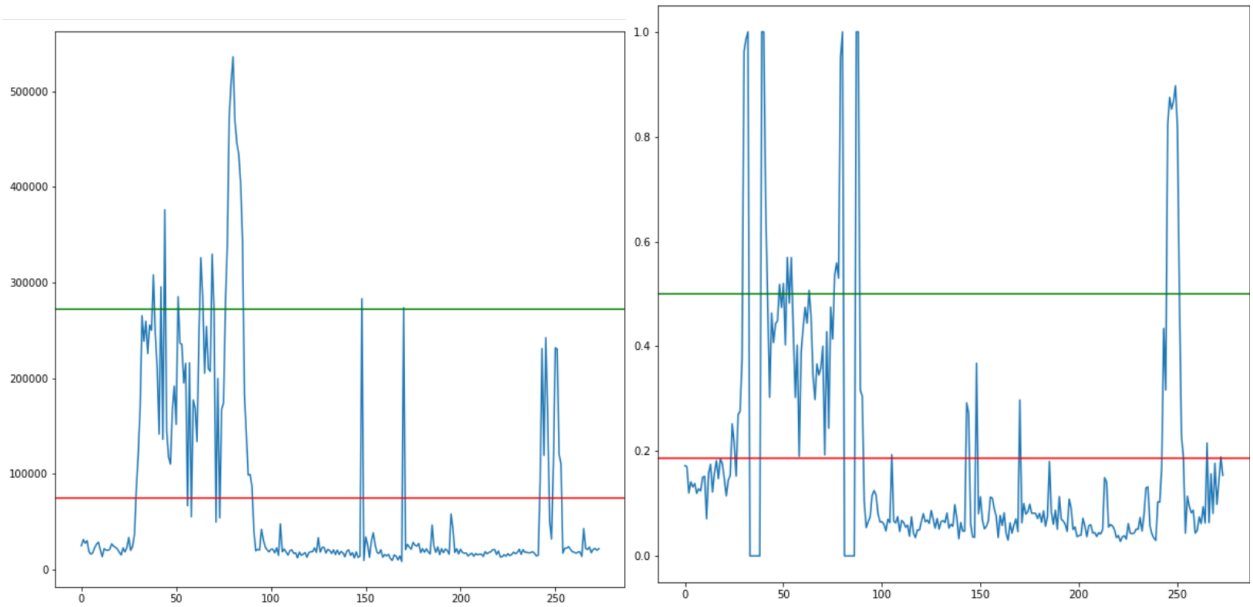

Figure 2: Dissimilarity score across all clip 3 frame pairs, left is Histogram differences right is Edge change ratio

A shot can be classified into abrupt Transitions and gradual transitions. Considering that, we will need not only to consider abrupt dissimilarity score changes but also need to consider a gradual increase and decrease in the score. So I designed my algorithm as follows:

- Step 1: Compute the score between the two consecutive frames using one of the above methods.
- Step 2: By observing the plotted scores across all frame pairs, we see that frames between two shots differ a lot. For abrupt transition, the score for the transition is much higher than the score for all the other frame pairs; for a gradual transition, if the transition is animated, then the animated scene tends to start and ends with large changes. So our goal is to detect frame pairs with dissimilarity scores. Due to differences in filming technique, scores may distribute differently across different scenes; thus, the threshold could be difficult to decide. So I designed the algorithm in the following way, I sorted all the scores and calculated the standard deviation for frame pairs with changes less than 30% quantile, then I used the standard deviation as the threshold to find scores with local standard deviation higher than a certain threshold. I define frame pairs that are involved in computing those scores with high local standard deviation as a shot boundary.

- Step 3: To filter out gradual transition scenes, I find frames between two shot boundaries found in step 2. If the scores between those frames have a higher standard deviation and the number of those frames is small, then those frames are part of gradual transition shot boundaries. Because frames within transition animations still have relatively high dissimilarities.

Manual Annotation (Hantang Li)

Annotated shot boundaries are recorded in Shot_boundary_Qa_Qb.ipynb and its printed PDF Shot_boundary_Qa_Qb.ipynb

The shot boundary is annotated by counting all the pairs of frames with abrupt changes and a consecutive pair of frames with gradual changes or shot change animations.

Evaluation

We will use Recall and Precision as well as the F1 score to evaluate how well we are detecting the shots. By definition,

- Recall: The probability that the algorithm will detect a correct boundary (manually labelled boundaries). This score is calculated by counting the number of frame pairs detected as part of correct shot boundaries and dividing it by the number of correct shot boundaries frame pairs.
- Precision: The probability that a correct cut (manually labelled cut) will be detected. This score is calculated by counting the number of frame pairs detected as part of correct shot boundaries divided by the total number of frame pairs that are detected as shot boundaries.
- F1: Is computed using both Precision and Recall and is used to evaluate how well the shot detection algorithm works in general. The equation is $2 * precision * recall/(precision + recall)$

Histogram differences

	Clip 1	Clip 2	Clip 3
Precision	100.00%	93.33%	54.84%
Recall	100.00%	82.35%	87.18%
F1	100.00%	87.5%	67.33%
Time	1.29 sec	1.13 sec	2.12 sec

Table 1: Quantitative results using Histogram differences

Edge change ratio

	Clip 1	Clip 2	Clip 3
Precision	5.00%	42.86%	82.14%
Recall	100.00%	52.94%	58.97%
F1	1.00%	47.37%	68.66%
Time	1.76 sec	1.59 sec	3.35 sec

Table 2: Quantitative results using Edge change ratio

For Edge change ratio algorithm we used same hysteresis threshold for all three clips, since Edge change ratio only performs better in clip 3, we used the hysteresis threshold that performs better on clip 3. From the above result, Histogram differences takes less time and performs best on the clip 1 and clip 2. While Edge change ratio performs best on clip 3.

Figure 3: Clip 3 frame 96 and 97

The reason is when two frame has similar colour distribution such as the frames 96 and 97 from clip 3, the colour histogram is not able to differenciate their differences. Those two frames are clearly a shot boundary but colour histogram failed to detect it. While Edge change ratio works better in this case because there is a lot movement between two frames.

Figure 4: Clip 2 frame 162 and 163

However Edge change ratio performs badly when there is a lot of movements within a shot, such as frame 162 and frame 163 from clip 2. While in this case colour histogram performs better, because the colour does not change a lot between those two frames. And both clip 1 and clip 2 are filled with shots with many move ments, hence Edge change ratio performs not well on clip 1 and clip 2.

Logo Detection (Hongyi Sun)

For the logo detection problem, we explored various methods.
Note: All accuracy tests below don't count for partially visible logos. Please review `logo_feature_match.py` and `logo_template_match.py` for code.

Machine Learning

First we looked at machine learning based approaches, such as training a CNN for object detection on an annotated logo dataset. However, the major limitation of this approach is the lack of annotated data for our target logo. Some popular logo dataset includes FlickrLogos[7], Logo-18/Logo-160[3] and LogoDet-3K[8], but we found the target logos "NBC news", "Clevver" and "Flicks And The City" (especially the last two) rarely appear in these datasets.

Template Matching

We then decide to use classical template matching, where the reference image is a cropped logo.

Figure 5: Cropped logos used for template matching.

We use `cv2.matchTemplate` method by OpenCV, we choose the `cv2.TM_CCOEFF_NORMED` option, that is, for each frame, it calculates the normalized cross-correlation coefficient between the logo and the frame for each position, and we draw bounding boxes at the pixel location where the coefficient is greater than a threshold. We find threshold= 0.69 works well for all clips.

	Clip 1	Clip 2	Clip 3
Accuracy	93.52%	100% on Clevver, 50% on NBC	100%
Time	4.44 sec	5.02 sec	11.83 sec

Table 3: Quantitative results using template matching, we calculate accuracy by using 100 percent minus the percentage of false negatives (logos present but not detected).

Figure 6: Qualitative results, logos are surrounded by blue bounding boxes.

Figure 7: What doesn't work well.

As we can see, template matching does not work well when there's even a slightest augmentation (rotation, affine transformation, scale up/down, etc...) of the template image. In our code, we have the option to match

based on edges of each image, however this can only solve the problem of color difference between templates and target image.

Feature Matching

Because of the disadvantage of template matching, we also implemented a SIFT-based logo detection. In my implementation, I first check the number of reliable SIFT keypoints matches (closest & second closest matches' distance ratio < 0.75) in both images, if the number of good matches is greater than 3, then I use RANSAC to calculate the optimal homography matrix. Finally I use this homography matrix to transform the 4 corners of the template image to the target image, and draw the bounding box based on the transformed corners. For finding matches, I implemented the option to use either the brute-force matching or FLANN based KNN matching (which is faster).

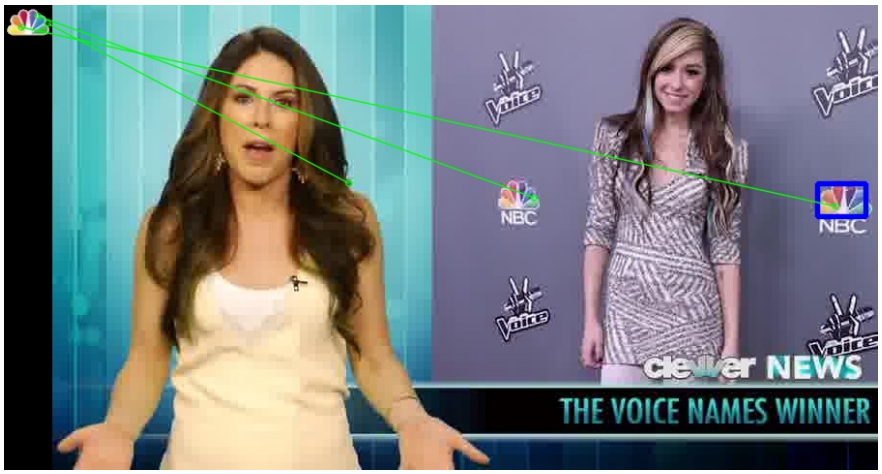

Figure 8: What doesn't work well for SIFT-based logo detection.

However, problem of SIFT-based approach arises when there are multiple matches on the target image. Since we don't know how many potential matches are there on the target image, we do not know how many matches are for one logo, and how many matches are for other logos. My implementation only account for one match. One possible solution for this problem is to crop an image into grid of several sections, and perform sift detection on each section, where the number of sections is a hyperparameter.

Grid-based Feature Matching

By using grid-based feature matching, we can successfully detect most of the augmented logos. A potential limitation of this approach might be that users need to manually test which grid resolution works best.

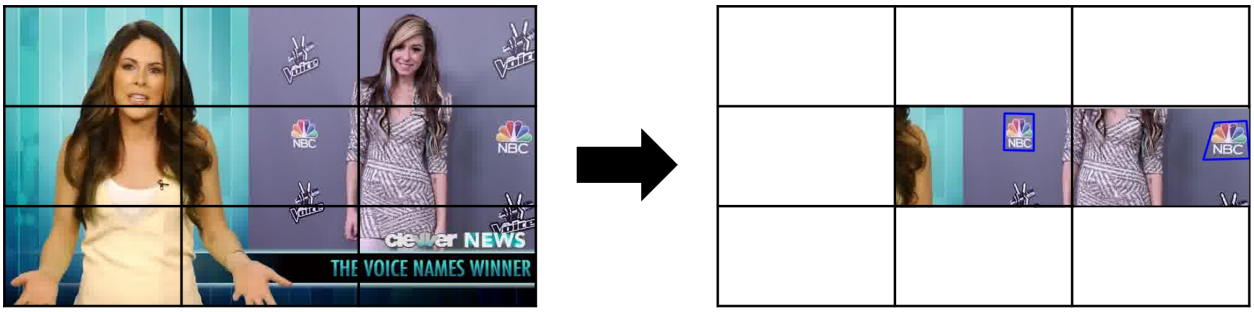

Figure 9: We first separate the image into grids, then perform feature matching on each

Figure 10: Lastly, we re-combine those image tiles. Notice that in the right image we can even detect partial logo sometimes (although this won't be included in the accuracy calculation).

	Clip 1	Clip 2	Clip 3
Accuracy	N/A	100% on NBC	N/A
Time (FLANN)	N/A	75.95 sec	N/A
Time (Brute Force)	N/A	132.12 sec	N/A

Table 4: Quantitative results using feature matching on the NBC logos for clip 2.

Final Results

After all types of implementation, we decided to use template matching first, to detect non-augmented logos, then we perform grid-based feature matching to detect augmented logos.

Figure 11: Final qualitative results. Achieving 100% accuracy.

Face Detection (Hongyi Sun)

Note: All accuracy tests below don't count the carton women's face in clip 3 as face.
Please review `train_face_detection.ipynb` for code on how to train, and `detection.ipynb` for its usage.

Existing solutions

To solve this problem, we first looked at existing solutions online. There are plenty of face detectors online, such as MTCNN and FaceNet. For MTCNN, we found its performance quite underwhelming, it detects faces perfectly, however it often place bounding box where there's no face. FaceNet requires MTCNN to first align the faces, thus we didn't test it.

	Clip 1	Clip 2	Clip 3
Accuracy	100%	99.28%	100%
Time	3.3min	2.7min	5min
False Positives	2	21	31

Table 5: Quantitative results using MTCNN face detector, we calculate accuracy by using 100 percent minus the percentage of false negatives (faces present but not detected divided by total faces). False positives count the number of bounding boxes that are placed where there's no face.

Another problem we noticed using MTCNN is that it's extremely slow. Detecting all faces in clip 3 which contains 275 images takes 5 minutes.

Figure 12: Qualitative results for MTCNN face detector where it detects false positives.

Figure 13: It also failed to detect heavily occluded faces.

Our implementation

We decided to use an existing object detection framework - YOLOv5 [6], and fine-tune it on the CelebA dataset [5], which contains around 200k annotated images with faces that covers a large variation of poses and background clutter.

	Clip 1	Clip 2	Clip 3
Accuracy	100%	95.2%	97.9%
Time	2.71sec	2.03sec	3.85sec
False Positives	0	0	0

Table 6: Quantitative results using our implementation. (YOLOv5s + CelebA)

We choose to use the medium-sized YOLOv5, the YOLOv5m which contains approximately 21.2 million parameters. We first preprocess our images from the CelebA dataset. Due to the amount of images available, we found that only using around 70000 training images can achieve pretty good results. Therefore we select and use the first 70000 images as training data, and the next 1000 images as validation data to fine-tune our YOLOv5m model. Another preprocess procedure includes changing the annotated bounding box value from CelebA's format to YOLO's required format. The original bounding box value includes the coordinates of the top-left corner and bottom right corner. But YOLO requires us the coordinate of the center of each bounding box, plus the dimension (height $\times$ width) of the bounding box, all normalized. Lastly, we change

the config file for YOLOv5, to enable it to detect for only 1 class ("face" in this case), and specify the training and validation data's path.

Figure 14: Qualitative results for our implementation.

From the quantitative and qualitative results, we can observe two advantages of our implementation: 1. Faster speed. Our implementation renders one frame in 0.015 seconds, and achieves an overall 75x speedup comparing to MTCNN implementation. 2. No false positives. By allowing us to freely edit the include threshold, we can adjust the detector to include more low possibility predictions and vice versa. This allows us to completely eliminate all false positives.
However, since YOLO is not specifically designed for face detection, the accuracy is not perfect. Right now we are only using 50k images which is 35% of the dataset, and we choose the medium YOLOv5 model due to time and hardware constraints. It is possible that it can achieve higher accuracy given more data, a larger model and carefully adjusting various hyper-paremeters.

Face Association (Hantang Li)

For code please view Face_Association_Q_g.ipynb and Face_Association_Q_g.pdf

Method 1: Accurate Scale Estimation for Robust Visual Tracking

Here I used the method introduced in the paper 'Accurate scale estimation for robust visual tracking.' Proceedings of the British Machine Vision Conference BMVC. 2014. [2]. This paper uses Correlation Filter Based Tracking[1] with Scale Estimation so that the algorithm can track an object and draw a bounding box that will adjust the size based on the object's size in the frame. Correlation Filter Based Tracking is similar to the template matching we told in the lecture, we input a frame with a bounding box, and the algorithm calculates a correlation filter based on the bounding box and then convolves the correlation filter in the next frame to find the pixel with a peak. The paper uses an additional algorithm to compute correlation filters with different sizes, so the algorithm is able to handle large-scale variations. We implemented it using dlib.correlation_tracker. This function is an implementation of the algorithm above.

Algorithm:

Loop over all the frames in order, and keep a list of dlib.correlation_tracker for each bounding box. If there is no tracker, then we run the face detection algorithm and obtain the bounding box for that frame. After getting bounding boxes for that frame, for each bounding box, we initialize correlation_tracker by inputting the frame and the bounding box. Then we keep tracking for all the following frames and evaluate the Peak To Sidelobe Ratio. This ratio is a metric that measures the peak sharpness of the correlation plane. If the ratio is lower than a threshold, then we abandon that tracker and assign all the bounding boxes that the tracker assigned for each frame belongs to same person (I assign an unique id for each person associated with the bounding box). Until we abandon all the trackers, we run the face detection algorithm and do the things above again.

	Clip 1	Clip 2	Clip 3
Time	2.24	5.40	4.90
Num frames ran face detection	2	13	8

Table 7: Quantitative results for Method 1

Method 2 Compare the similarity between two frames bounding boxes and/or their content

Inspired by the shot detection algorithm, we can compare all the bounding boxes within a frame to all the bounding boxes in the following frame, and if they are similar or in a similar position, then we can say those two bounding boxes bound the same person. I tried to use two ways of calculating the similarity of the two bounding boxes:

- Centroid of the bounding box.
- Histogram differences between two bounding boxes' content.

Algorithm:

Loop over all the frames in increasing order. Inside the loop, run the face detection algorithm and obtain bounding boxes. Loop over all those bounding boxes and compare them to previous frames bounding boxes. If no previous frame existed or previous frames have no bounding box, then assign a unique person ID to each bounding box. If the previous frame existed, then compute the distance between this frame's bounding boxes and the previous frame's bounding boxes.

- If two frames contain the same amount of bounding boxes or the current frame contains fewer bounding boxes, then assign the existing bounding box to the bounding box's ID in the previous frame, which two bounding boxes have the smallest distance.
- If the current frame has more bounding boxes after all the previous frame's bounding boxes' id has been assigned. We assign a new person ID for all the rest of the current frame's bounding boxes.

	Clip 1	Clip 2	Clip 3
Time	0.61	0.58	0.82
Num frames ran face detection	179	135	275

Table 8: Quantitative results for Method 2

Result

Method 1 needs to run less face detection, and Method 2 requires run face detection on all frames. So people could choose which algorithm to use based on the face detection speed and accuracy.

For disadvantages, method 1 requires each face detection to be highly accurate. Otherwise the tracker is unable to track the undetected face. Such as the following:

Figure 15: Left method 1, right method 2

For method2, one disadvantage is if face detection fails to detect a person within a shot, the algorithm will assign that person another ID.

Gender Detection (Hongyi Sun)

Note: All accuracy tests below don't count the carton women's face in clip 3 as face.
Please review `detection.ipynb` for its usage.

Initial Approach

Our initial approach to this problem is simple. We use a pretrained ResNet-18, replace its classifier layers to a set of linear layers with batch norm and random dropout. We give its feature extraction layers (convolution layers) with pretrained weights a lower learning rate, and give the classifier a higher learning rate. We use the SGD optimizer and train on the given dataset for 6 epochs. We also augmented the dataset in 4 ways: horizontal flip, perspective change, blur and color jitter. Each of these transformations will be applied to input data once to increase model robustness.

	Clip 1	Clip 2	Clip 3
Accuracy	61.7%	72.6%	63.1%

Table 9: Quantitative results with our initial approach.

Figure 16: Qualitative results for our initial approach. Cases that our method works well. From left to right: Clip 1, Clip 2, Clip 3.

Overall, our method performed poorly, especially on scenes where not all facial features (eyes, nose) are available. Our method performed quite well on front-facing faces, however front-facing faces are rarely found among all clips.

Improved Approach

After carefully examing our training process, since our model is achieving 95% training accuracy while having a low accuracy on real dataset, this implies that our model is overfitting on our tiny training data (around 250 images for each class, 1000 after 4 augmentations), and doesn't generalize well to unseen data. Also, since our training data only contains aligned faces with no head movement, this explains why our model only works when classifying front-facing faces. Lastly, the prediction is not very stable, meaning that same face with tiny movements/changes across frames can be predicted with different genders. This also implies our model is still not robust.

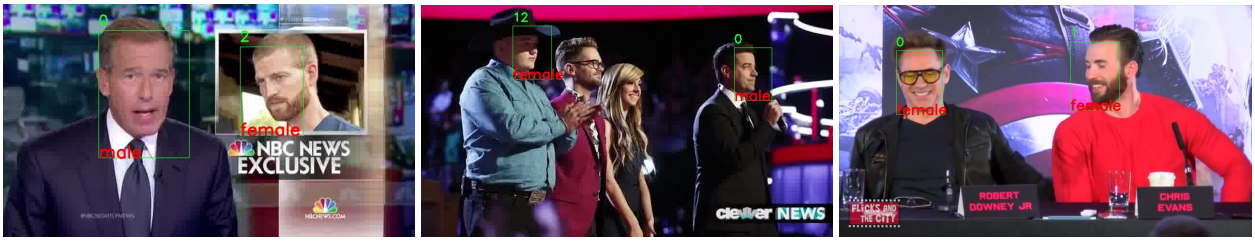

Figure 17: Cases where our initial approach failed. Notice how as soon as the face angle changes (Clip 1), or has any occlusion (shadows on the left guy's face on Clip 2), or facial feature changes (Robert Downey Jr. and Chris Evans show their teeth), their gender prediction changes.

The problem of prediction instability can be alleviated by utilizing face track described above, where we determine a full face track's gender based on which gender appears more. However, we can also increase the robustness of the current model, allowing it to detect same gender for the same face across all angles. Thus in our second attempt, we incorporate a much larger dataset[1] obtained from kaggle, with more than 23000 faces from various viewing angles. We also copy this dataset 4 times and perform 4 different augmentation to it to increase model robustness.

	Clip 1	Clip 2	Clip 3
Accuracy	100%	98.8%	67.1%

Table 10: Quantitative results with our improved approach.

From the result, we can see the model perform much better on clip 1 and 2 than the initial approach. The prediction is very stable (no switching gender for the same face across frames) meaning the model now is more robust than the initial model. The only misclassification happened in clip 2 is due to face occlusion (See Fig. 13).

However, since the prediction is stable, once a gender is classified, even if it's incorrect, all the same faces across frames will be classified as that incorrect gender. This is what happened at clip 3, where Scarlett Johansson is classified as "male" and the prediction carried over to almost all frames containing her. Since Scarlett appeared in most of the frames, this results in a low accuracy for Clip 3.

Other than this error, almost all other faces across all frames are classified correctly. Model's performance in terms of accuracy improved 62%, 36% for Clip 1 and Clip 2 respectively.

Also, because the prediction is stable, we don't have to use the face track information to determine the gender. Instead we can detect gender for every frame.

Demonstration (Both worked on this)

Figure 18: A few qualitative results from our final product.

[1] https://www.kaggle.com/datasets/cashutosh/gender-classification-dataset

For full video results, please visit links below:

Clip 1	Google Drive Link 1
Clip 2	Google Drive Link 2
Clip 3	Google Drive Link 3

We also include the clips inside our MarkUs submissions.

References

[1] Vishnu Naresh Boddeti. *Advances in Correlation Filters: Vector Features, Structured Prediction and Shape Alignment*. PhD thesis, USA, 2012. AAI3536331.

[2] Martin Danelljan, Gustav Hauml;ger, Fahad Shahbaz Khan, and Michael Felsberg. Accurate scale estimation for robust visual tracking. In *Proceedings of the British Machine Vision Conference*. BMVA Press, 2014.

[3] Steven CH Hoi, Xiongwei Wu, Hantang Liu, Yue Wu, Huiqiong Wang, Hui Xue, and Qiang Wu. Logo-net: Large-scale deep logo detection and brand recognition with deep region-based convolutional networks. *arXiv preprint arXiv:1511.02462*, 2015.

[4] Xiaocui Liu, Jiande Sun, Ju Liu, Jiande Sun, and Ju Liu. Shot-based temporally respective frame generation algorithm for video hashing. In *2013 IEEE International Workshop on Information Forensics and Security (WIFS)*, pages 109–114, 2013.

[5] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[6] J Redmon, S Divvala, R Girshick, and A Farhadi. You only look once: Unified, real-time object detection. arxiv 2015. *arXiv preprint arXiv:1506.02640*, 2015.

[7] Stefan Romberg, Lluis Garcia Pueyo, Rainer Lienhart, and Roelof Van Zwol. Scalable logo recognition in real-world images. In *Proceedings of the 1st ACM International Conference on Multimedia Retrieval*, pages 1–8, 2011.

[8] Jing Wang, Weiqing Min, Sujuan Hou, Shengnan Ma, Yuanjie Zheng, and Shuqiang Jiang. Logodet-3k: A large-scale image dataset for logo detection. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 18(1):1–19, 2022.

[9] Ziyou Xiong, Regunathan Radhakrishnan, Yong Rui, Ajay Divakaran, Tsuhan Chen, and Thomas S. Huang. A unified framework for video indexing, summarization, browsing, and retrieval. *The Essential Guide to Video Processing*, page 437–472, 2009.