CS231A Project Final Report

Character Identification in TV Series from Partially Labeled Data

Benjamin Paterson and Arthur Lacoste
{paterben,alacoste}@stanford.edu

March 19, 2014

Abstract

We present a framework for finding and identifying character faces in TV series. This method requires only a video and a set of subtitles containing the names of the speaking characters and the time at which they are speaking. It relies on a processing pipeline incorporating face detection, face tracking, mouth and speaker detection, featurization of face tracks and finally discriminative learning. We show an almost-complete implementation of this pipeline, where only speaker labeling has been done manually. By itself, our learning algorithm achieves an average accuracy of 74.8% on a publicly available dataset of featurized face tracks from 6 episodes of a TV series where speakers have already been labeled. Using our own face tracker and manually labeled speakers, the learning algorithm achieves 85.23% accuracy on the first episode.

1 Introduction

In this paper we attempt to further the method and results presented in [1]. Specifically, we are interested in the problem of automatic character identification in TV series. Results from character identification can be used in higher-level multimedia analysis applications, such as video summarization or indexing. The problem is as follows: we are given one or more video episodes from a TV series or movie, as well as synchronized and named subtitle files for each of the videos. Using this data we wish to locate all the face tracks in the video, and label each track with the name of the character that it belongs to.

The basic idea is to use the timing and name information present in the subtitles to label a subset of the face tracks. For every frame in a face track, we determine if the face is speaking, and if it is the case we look at the subtitle file to determine the name of the character that is speaking at that instant. Ideally, every face track that corresponds to a character that is speaking can be identified in this way. Face tracks corresponding to characters that are silent, however, will remain unlabeled. A learning algorithm can then be trained on the labeled faces and used to classify the remaining unlabeled faces.

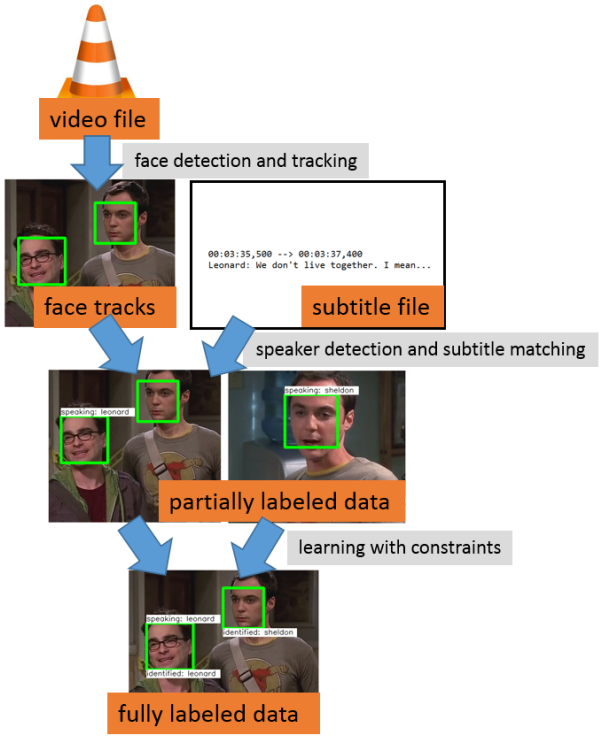

Figure 1: The data processing pipeline. A face tracker processes the video to find all face tracks. Speakers are then detected by analysis of the mouth region of the face, and are matched with names from the subtitle file. Finally, the faces are featurized using Discrete Cosine Tranform and the data is run through a learning algorithm which outputs a name for each face track.

The basic approach outlined above suffers from several problems. First, there can be mistakes in the speaker detection algorithm, caused by misaligned subtitles, several characters speaking at the same time, a character open-

ing his mouth without speaking, the speaking character not being present in the frame, or by mistakes made by the face or mouth detection algorithms. We would like to have an opportunity to correct these mistakes. Second, since all the unlabeled faces are available at training time, we would like to use them in some way to improve the learned parameters. And finally, the learning algorithm as suggested above does not incorporate some very rich information given by the face tracker, namely that all frames in the same track should belong to the same character, and that frames in different face tracks that overlap in time are unlikely to correspond to the same character (unless the character is looking at himself in the mirror or facing an evil twin).

The learning algorithm presented in [1] and which we analyze here attempts to address these problems in a unified way. The learned parameters are used to classify labeled and unlabeled face tracks indiscriminately, which allows recovery from mistakes made by the speaker detection algorithm. The algorithm incorporates unlabeled data into the learning process by penalizing parameters that give a high uncertainty as to how to classify unlabeled instances. And finally, the algorithm adds *soft constraints* which incorporate track information. These soft constraints penalize frames which lie in the same track but are classified differently, as well as frames which lie in overlapping tracks but are classified the same.

A summary of the data processing pipeline is shown in Fig. 1.

2 Related Work

There have been several attempts at automatic character identification in the last few years, most of them (including [2] and [3]) focusing on recognizing face tracks. Extracting face tracks from a video is not trivial and we have built our own face detection and tracking system from the Viola-Jones [6] and KLT [7] primitives. To make the whole identification process completely autonomous, [2] proposed a method to weakly label tracks by detecting speaking characters and matching them to named subtitles (obtained by aligning standard subtitles and transcripts). The second part of the problem is to learn the identity of all tracks from these unreliable labels. [12] propose loss functions that allow to learn from both weakly labeled and unlabeled tracks. In this work, we also make use of must-link and cannot-link constraints that arise from the structure of the face tracks and integrate those in a discriminative learning framework.

3 Character Identification

3.1 Face Detection

General approach

We are using the Viola-Jones algorithm [6] (cascade of classifiers based on Haar-like features) to detect faces in static images. Our implementation uses two pre-trained cascades of classifiers: one detecting frontal faces and one detecting profile faces. Viola-Jones is not rotation invariant and our initial implementation performed poorly at detecting inclined faces, so we are actually running both our classifiers on several slightly rotated versions of the image ($\{-6, -4, -2, 0, +2, +4, +6\}$ degrees). Thus, even after the built-in non-max supression of the Viola-Jones primitive we are using, each face can yield multiple detections. We perform a custom aggregation, in the process of which we filter out some of the false positive detections.

Aggregation of detections

For R_1, R_2 two axis-aligned rectangles and $R_1 \cap R_2$ the rectangle over which they overlap, we define the following metrics:

- A measure of how similar they are:

$$\text{sim}(R_1, R_2) = \frac{2 \cdot \text{area}(R_1 \cap R_2)}{\text{area}(R_1) + \text{area}(R_2)}$$

  which takes values in $[0, 1]$, 0 if and only if R_1 and R_2 do not overlap, 1 if and only if $R_1 = R_2$.

- A measure of how much they intersect:

$$\text{inter}(R_1, R_2) = \frac{\text{area}(R_1 \cap R_2)}{\min(\text{area}(R_1), \text{area}(R_2))}$$

  which takes values in $[0, 1]$, 0 if and only if R_1 and R_2 do not overlap, 1 if and only if one rectangle is completely inside the other one.

We use the following algorithm to produce consolidated and non-redundant face detections:

1. For each detection, rotate back the bounding rectangle to be axis aligned if necessary (keeping its center and size constant).

2. Given a similarity threshold D_{sim} (empirically set to 65%), select the detection R^* that is at least D_{sim} similar with the most other detections. This yields a group of k detections that are all at least D_{sim} similar to a single detection R^*.

3. If k is less than a given threshold D_{count} (empirically set to 3), discard all remaining detections and return the current set of consolidated detections.

4. Otherwise, compute the average of all these detections R_{avg} and add it to the set of consolidated detections.

5. Then, given an intersection threshold D_{inter} (empirically set to 20%), discard all detections whose intersection with R_{avg} is at least D_{inter} (this will remove all the detections that were R_{avg}, but will also remove slightly off yet overlapping detections).

6. Go to 2.

This aggregation technique is fairly simple but proved very accurate and robust, so we did not deem worth it to further improve it.

3.1.1 Performance

In our dataset (images for the the first episode of 'The Big Bang Theory' Season 1), Viola-Jones alone (without rotations) cannot achieve a high detection rate without also picking up a lot of false positives. At constant detection rate, introducing rotations and aggregation of detections slightly reduces the number of false positives. Tracking will allow us to further prune spurious detections, so we sacrificed some precision to improve recall in our thresholds tuning.

3.2 Face Tracking

General approach

We first detect the time boundaries of the camera shots in our video (instants at which the camera switches to another view), at which point shots can be treated independently from each other with regard to face tracking. We run our face detection technique on a subset of frames in the shot. These "anchor" frames are regularly spaced throughout the shot, and there are $\max(6, nShotFrames/10)$ of them. Each consolidated face detection inside an anchor frame is then tracked across the whole shot using the Kanade, Lucas & Tomasi algorithm. A face is likely to be detected several times in different anchor frames and the corresponding tracks should overlap, so that we can again perform a custom aggregation to produce consolidated tracks and filter out some of the false positive consolidated detections.

Detecting shot boundaries

We run the following algorithm on every pair of neighbor frames $f^{(1)}$ and $f^{(2)}$ to determine whether there is a shot change between them:

1. Change the encoding of the frames to be HSV (Hue, Saturation, Value) instead of RGB (Red, Green, Blue).

2. Split each frame in four quadrants (top-left, top-right, bottom-left, bottom-right).

3. For each quadrant q in both frames, compute the histogram of colors $H_q(h, s)$. The hue is discretized into 30 bins, the saturation into 32 bins, and the value is ignored.

4. Define the "distance" d between the two frames to be:

$$d = \sum_{q=1}^{4} \sum_{h=1}^{30} \sum_{s=1}^{32} |H_q^{(1)}(h, s) - H_q^{(2)}(h, s)|$$

5. Compare d to an empirically set threshold.

This algorithm tolerates camera movements inside a shot (up to a certain speed), and can detect shot boundaries even if the overall color histogram of the image remains roughly the same (even then, the histograms taken quadrant by quadrant are likely to be significantly different). We tested it on a substantial amount (about 50) of shot boundaries in our video file and it did not make any mistake (the worst positive example was about 50% over the threshold and the worst negative example about 50% under the threshold). Therefore, we did not try to make it even more robust to complex situations (such as a flash of light inside a shot).

Tracking a single face across a shot

To track a consolidated face detection across the whole shot, we first find the 50 strongest corners inside the corresponding initial bounding rectangle using the method from [8]. These points are then tracked frame by frame forwards and backwards inside the shot using the Kanade, Lucas & Tomasi algorithm [7].

We first compute the initial displacement v_i of each point p_i to the center c of the provided bounding rectangle: $v_i = c - p_i$. At each step of the tracking, the reported tracking error is used to evict badly tracked points (using an empirically set threshold). Then, we fit the best translation + scaling transformation between the initial and new points to reconstruct the bounding rectangle for the face at that point. Specifically, find s, c'_x, c'_y solving the least squares problem:

$$\min_{s,c'} \sum_i \|p'_i - (c' - s \cdot v_i)\|^2$$

We then further filter points that are tracked well but strongly diagree with others: we associate a vote for c' to each point $c'_i = p'_i + s \cdot v_i$ and compute $d_i = \|c'_i - c'\|$, then eliminate points for which $d'_i > 3.5\sigma + 1px$, σ being the standard deviation of the d_i. This allows to eventually eliminate points that are attached to the background instead of the actual face.

We use the number of tracking points remaining as a measure of the quality $q(j)$ of our tracking at each frame j. We think the significance of that measure might have been improved by taking a non linear (sigmoid-type) function of the number of remaining points (the tracking is still very good with 40 points remaining, but from practical testing its quality drops significantly in the 23-30 points region), but we did not have time to implement it. The standard deviation of the votes for c' also bears meaning as to the quality of the tracking, so it could have been used as basis for a metric as well.

Reconstructing all the face tracks of a shot

We define a merge procedure that takes two tracks t_1, t_2 and returns a merged track t_m such that, for each frame j in the shot

- The quality of the merged track is $q_m(j) = q_1(j) + q_2(j)$
- The rectangle of the merged track is the weighted average of the rectangles of the merged tracks, i.e. it has center and size:

$$c_m(j) = \frac{q_1(j)c_1(j) + q_2(j)c_2(j)}{q_1(j) + q_2(j)}$$

$$s_m(j) = \frac{q_1(j)s_1(j) + q_2(j)s_2(j)}{q_1(j) + q_2(j)}$$

Note that this operation is associative, i.e. the final result of our aggregation procedure will be the same as long as we eventually merge the same tracks together.
To decide on which tracks to merge, we define a similarity measure among tracks:

$$\mathrm{sim}(t_1, t_2) = \frac{\sum_j q_1(j) \cdot q_2(j) \cdot \mathrm{sim}(r_1(j), r_2(j))}{\sum_j q_1(j) \cdot q_2(j)}$$

This measure takes its values in $[0, 1]$, 0 if and only if both tracks's rectangles have zero-similarity in every frame where both tracks have non-zero quality, and 1 if and only if the two tracks perfectly agree (identical rectangles) in every frame where both tracks have non-zero quality. The intuition behind it is that it does not matter much if both tracks only overlap shortly, as long as they totally agree when they do, and conversely tracks that slightly agree but across a long period of time should probably not be merged. Given these two definitions, our aggregation algorithm is a follows:

1. While there exist a pair of tracks with similarity s_{max} greater than a threshold T_{sim} (empirically set to 60%), merge the two tracks with highest similarity.
2. When no pair of tracks is similar enough anymore, retain all tracks with top-quality greater than a threshold $T_{quality}$ (empirically set to 2.0). The top-quality Q of a track is defined as: $Q = \max_j q(j)$.

While the merging order can theoretically matter (some mergings may only be allowed depending on the order), we could not find any situation where it actually proved to be the case. Similarly, the retaining threshold might seem strange, but in our testing it turned out to be almost equivalent to selecting all tracks that are the result of the merging of at least 3 original tracks (and slightly easier to implement).

3.3 Speaker Detection and Subtitle Matching

After face tracks have been identified, we wish to determine which of them are speaking. For this, we use a method close to that presented in [2]. First, the mouth region is identified in each frame using a pre-trained Haar cascade. The mouth regions are resized to a unique size, and the mean pixel-by-pixel squared difference between two consecutive frames is computed. If it is above a certain threshold, then the character is deemed to be speaking at that instant. There are two sources of noise that then need to be eliminated. First, to make the algorithm robust to small translations of the detected mouth region, we recompute the mean pixel-by-pixel squared difference for several positions of the second detected mouth region in a small search window around its original position, and take the minimum of the detected values. We also require that the mean squared difference between frames be higher than the threshold for several consecutive frames to eliminate noise.

As we were implementing the above algorithm, we soon realized that the pre-trained mouth detector we were using had very likely been trained to identify only closed mouths, and this meant that we needed a different method for detecting the mouth region. Due to lack of time, we decided to forego speaker detection, and label speaking characters manually.

3.4 Learning with Constraints

Featurization of Face Frames

Each face frame is featurized using the Discrete Cosine Transform [4]. First, each face is resized to a 56-by-56 pixel image and converted to grayscale. Then the Discrete Cosine Transform is applied to each of the 49 8-by-8 pixel blocks in the image. For each transformed block, we keep only the entries corresponding to the 5 lowest frequencies (at the top-left corner of the image), disregarding the top-leftmost component which represents average brightness of the image. This results in a $d = 49 \times 5 = 245$-dimensional feature vector $\mathbf{x}$ for each frame. In [11], the authors show that these low-frequency components carry most of the information useful for facial recognition.

Model

For learning, we use a Multinomial Logistic Regression model inspired from [1]. The set of all possible names is determined using the output from speaker detection and each name is mapped to a number from 1 to K. For each frame $\mathbf{x}$ and given a parameter vector $\theta \in \mathbb{R}^{d\times K}$, the model tries to predict the probability of the frame belonging to character k, $P_\theta^k(\mathbf{x})$, using the formula:

$$P(y=k|\mathbf{x}:\theta) = P_\theta^k(\mathbf{x}) = \frac{e^{\theta_k^T\mathbf{x}}}{\sum_{z=1}^K e^{\theta_z^T\mathbf{x}}}$$

Supervised loss

Let $\mathcal{X}^{(l)}$ be the set of labeled frames and $\mathcal{X}^{(u)}$ be the set of unlabeled frames. Let $N = |\mathcal{X}^{(l)}|$ and $M = |\mathcal{X}^{(u)}|$ be the cardinalities of these sets. We define the regularized supervised loss to be the negative log-likelihood of the labeled frames:

$$\mathcal{L}_l(\theta) = -\frac{1}{N}\sum_{i=1}^N\sum_{k=1}^K \mathbb{1}_{[y_i=k]}\ln P_\theta^k(\mathbf{x}_i^{(l)}) + \lambda\|\theta\|^2$$

Its gradient with respect to the k-th column of θ is given by:

$$\frac{\partial\mathcal{L}_l}{\partial\theta_k} = -\frac{1}{N}\sum_{i=1}^N(\mathbb{1}_{[y_i=k]} - P_\theta^k(\mathbf{x}_i^{(l)}))\mathbf{x}_i^{(l)} + 2\lambda\theta_k$$

This loss function operates only on the labeled frames. However as outlined earlier we have two extra sources of information: the unlabeled frames and the information about which track each frame belongs to. This motivates the use of the following loss functions.

Unsupervised loss

Intuitively, we would like the unlabeled data to lie far away from the decision boundaries that result from the parameters θ, so that the labeling decision is more robust. We can achieve this by incorporating a loss term that penalizes uncertainty in the classification decision for the unlabeled data. We define the unsupervised loss to be the average entropy of the classification probability distribution of the unlabeled frames:

$$\mathcal{L}_u(\theta) = -\frac{1}{M}\sum_{i=1}^M\sum_{k=1}^K P_\theta^k(\mathbf{x}_i^{(u)})\ln P_\theta^k(\mathbf{x}_i^{(u)})$$

Its gradient with respect to the k-th column of θ is given by (we correct here a minor mistake in [1]):

$$\frac{\partial\mathcal{L}_u}{\partial\theta_k} = -\frac{1}{M}\sum_{i=1}^M\Big[\mathbf{x}_i^{(u)}P_\theta^k(\mathbf{x}_i^{(u)}) \sum_{c=1}^K(\mathbb{1}_{[k=c]} - P_\theta^c(\mathbf{x}_i^{(u)}))(1+\ln P_\theta^c(\mathbf{x}_i^{(u)}))\Big]$$

Positive and negative constraints loss

Finally, we would like to incorporate a penalty whenever we make a classification decision that is incoherent with the structure of the face tracks: different frames in the same face track should be labeled the same, and if two face tracks overlap in time, the frames in those two tracks should be classified differently. To achieve this, we generate two lists of constraints. Between any two frames $\mathbf{x}_{i_1}$ and $\mathbf{x}_{i_2}$ in the same face track, we generate a *positive constraint* $c_i^{(p)} = (\mathbf{x}_{i_1}, \mathbf{x}_{i_2})$. Let $L^{(p)} = |c^{(p)}|$. Let $P(y_{i_1} = y_{i_2})$ be the probability that $\mathbf{x}_{i_1}$ and $\mathbf{x}_{i_2}$ belong to the same class:

$$P(y_{i_1} = y_{i_2}) = \sum_{k=1}^K P_\theta^k(\mathbf{x}_{i_1})P_\theta^k(\mathbf{x}_{i_2})$$

We then define the positive constraints loss:

$$\mathcal{L}_p(\theta) = -\frac{1}{L^{(p)}}\sum_{(\mathbf{x}_{i_1},\mathbf{x}_{i_2})\in c^{(p)}}\ln P(y_{i_1} = y_{i_2})$$

The smaller the probability that $\mathbf{x}_{i_1}$ and $\mathbf{x}_{i_2}$ belong to the same class, the higher their contribution to the loss.

Similarly, between any frames belonging to two different tracks whose timespans have a non-empty intersection, we generate a *negative constraint* $c_i^{(n)} = (\mathbf{x}_{i_1}, \mathbf{x}_{i_2})$. Let $L^{(n)} = |c^{(n)}|$. We then define the negative constraints loss:

$$\mathcal{L}_n(\theta) = -\frac{1}{L^{(n)}}\sum_{(\mathbf{x}_{i_1},\mathbf{x}_{i_2})\in c^{(n)}}\ln P(y_{i_1} \neq y_{i_2})$$

The gradients with respect to θ_k of the above loss functions are given by (we again correct a mistake in the original paper):

$$\frac{\partial\mathcal{L}_p}{\partial\theta_k} = -\frac{1}{L^{(p)}}\sum_{(\mathbf{x}_{i_1},\mathbf{x}_{i_2})\in c^{(n)}}\Bigg[(\mathbf{x}_{i_1}+\mathbf{x}_{i_2})\frac{P_\theta^k(\mathbf{x}_{i_1})P_\theta^k(\mathbf{x}_{i_2})}{P(y_{i_1}=y_{i_2})} - (\mathbf{x}_{i_1}P_\theta^k(\mathbf{x}_{i_1}) + \mathbf{x}_{i_2}P_\theta^k(\mathbf{x}_{i_2}))\Bigg]$$

$$\frac{\partial\mathcal{L}_n}{\partial\theta_k} = \frac{1}{L^{(n)}}\sum_{(\mathbf{x}_{i_1},\mathbf{x}_{i_2})\in c^{(n)}}\Bigg[(\mathbf{x}_{i_1}+\mathbf{x}_{i_2})\frac{P_\theta^k(\mathbf{x}_{i_1})P_\theta^k(\mathbf{x}_{i_2})}{P(y_{i_1}\neq y_{i_2})} - (\mathbf{x}_{i_1}P_\theta^k(\mathbf{x}_{i_1}) + \mathbf{x}_{i_2}P_\theta^k(\mathbf{x}_{i_2}))\frac{P(y_{i_1}=y_{i_2})}{P(y_{i_1}\neq y_{i_2})}\Bigg]$$

The number of constraints generated per face track is quadratic in the number of frames in that track. This is undesirable since it gives a much higher weight to longer face tracks. Instead, we subsample a number of constraints that is only linear in the length of each face track. This also helps keep the number of constraints under control (in our implementation, an episode of 20 minutes generated over a million constraints).

3.4.1 Increasing the dimensionality of the feature space

The dimensionality of the feature space ($d = 245$) is quite small relative to the number of data points, especially considering the number of constraints - in the tens of thousands. However, kernelization here would be prohibitively expensive. Instead, we increase the parameter d by taking a random sample of the features corresponding to the L2 kernel. This allows to fine-tune the dimensionality of the feature space depending on considerations such as available memory or speed of the learning algorithm. In practice, adding these new features helps performance significantly.

Training and prediction

The final loss function is given by:

$$\mathcal{L} = \mathcal{L}_p + \mu\mathcal{L}_u + \gamma^{(p)}\mathcal{L}_p + \gamma^{(n)}\mathcal{L}_n$$

where μ, $\gamma^{(p)}$ and $\gamma^{(n)}$ are hyperparameters to be tuned. The loss function can be minimized using a pseudo-Newton method such as L-BFGS [5]. Once a prediction has been made for each frame, we need to decide on a prediction for the entire track. According to the model, the prediction should be:

$$y_t = \arg\max_k \sum_{i=1}^{|t|} \ln P_\theta^k(\mathbf{x}_i^{(t)})$$

However, the authors of [1] suggest discarding the logarithm in the expression above, as summing probabilities seems to work better in practice than multiplying them. We find that this is indeed the case. A possible interpretation is that summing probabilities is more robust than multiplying them, since a mistake in a single frame can radically change the product but not the sum.

4 Experimental Results

4.1 Face detection and tracking results

We implemented the described face detection and tracking process in C++ using the OpenCV framework, where an implementation for several primitives we use is available (Viola-Jones detection with pre-trained cascades for faces, corner detection, KLT tracking using the implementation described in [9]). Because the end-goal of our project is learning performance and because ground truths must be manually collected, most of our results regarding this part of the project are qualitative, but we still collected quantitative recall and precision figures of our overall face tracking results on the first episode of 'The Big Bang Theory' Season 1.

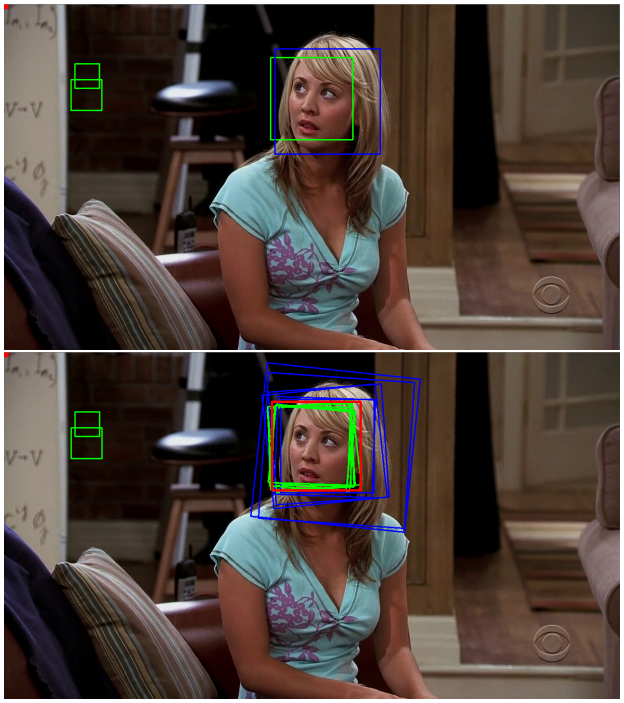

Figure 2: Result of Viola-Jones face detection without (top) vs. with (bottom) rotations to the original image. Green rectangles are frontal face detections, blue rectangles are profile face detections, and red rectangles are consolidated face detections.

Face detection

Fig. 2 shows an example of a situation where applying Viola-Jones not only to the original image but also on slightly rotated versions of it and aggregating the results is able to eliminate false positive detections.

Face tracking

Fig. 3 shows the result of the technique we use to track a single detection throughout a shot (using corner detection + KLT algorithm + bounding rectangle reconstruction). It is very robust and the number of remaining points in the tracking provides a clear and reliable signal as to the quality of the track.

Fig. 4 shows the results of the technique we use to aggregate tracks. In this case, the face of Sheldon was not detected in any of the anchor frames, so there is

Figure 3: Tracking of a single face throughout a shot. The top image shows the initial detection (green) and the corresponding strong corners (red), and the bottom image shows the corners after tracking and the reconstructed face bounding rectangle.

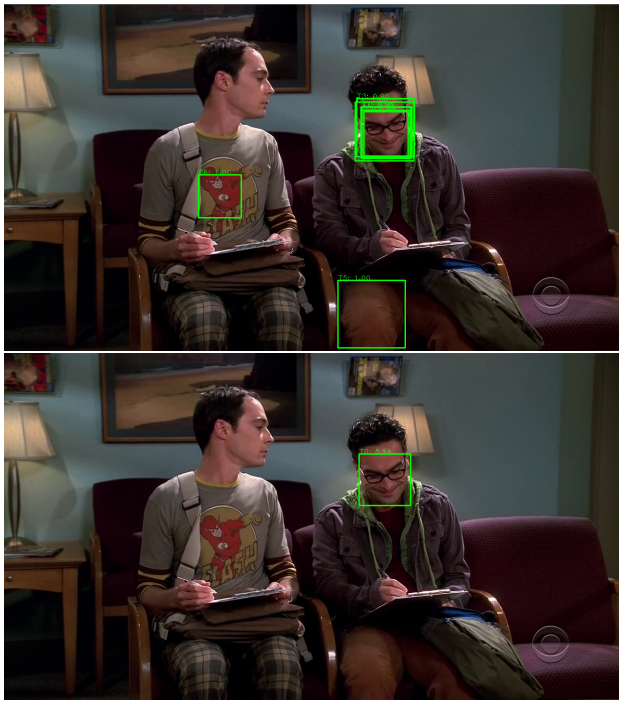

Figure 4: Result of the track-level aggregation. The top image shows all the tracks at a specific frame in the shot, and the bottom image shows the consolidated tracks for the same frame.

no way to recover from that. However, there were two false positives among the consolidated detections that are correctly evicted in the aggregation procedure, and all tracks that correspond to detections of Leonard's face are correctly merged together.

Quantitative results

We evaluated the performance of our overall face detection and tracking on the first episode of 'The Big Bang Theory' Season 1. We counted a total of 553 actual apparitions of faces inside a shot, out of which 423 were correctly picked up by our algorithm, and we counted 50 false positive tracks. This yields the following performance:

- Recall: 76.5%
- Precision: 89.4%

Note that:

- Characters facing away from the camera for the integrality of the shot were not counted.
- Face tracks that are split in two in the middle of a shot (because a character faces away from the camera for a moment or moves his head too fast) were counted as successes.
- Face tracks that fail to capture the integrality of a face screen time (but don't miss it completely) were counted as successes if the face is tracked for at least half of its screen time.

4.2 Learning results

We developed our learning algorithm using a dataset made public by the authors of [1]. This dataset contains face tracks that have been featurized frame by frame, and ground truth labels as well as labels from speaker detection, for 6 episodes of 'The Big Bang Theory' Season 1. Hyperparameter tuning was executed on the first episode of the series. Our results are summarized in Table 1. The optimal hyperparameters we found were:

- $\lambda = 5.10^{-4}$
- $\mu = 0.7$
- $\gamma^{(n)} = 1.5$
- $\gamma^{(p)} = 1$

In our experience, adding the soft constraints loss functions had a considerable beneficial impact, in particular for the positive constraints. The authors of [1] report a much smaller impact, though it is unclear whether they incorporated both positive and negative constraints.

Some examples of classifications made by our algorithm are shown in Fig. 6. The learning algorithm is able to recover from mistakes made by speaker detection. For characters which speak often, the classification is almost always correct. For characters which rarely speak, classification is very often incorrect. Characters which never speak cannot be identified.

Using the hyperparameters found above, we report results on the first 6 episodes of 'The Big Bang Theory' Season 1 in Table 2. The results are most impressive on Episode 1, where there is the smallest number of characters. Of course, since our parameters were optimized to work well on this episode, overfitting accounts for part of the high performance reported on it. We obtain an average track classification accuracy of 74.78% over these 6 episodes.

For 'The Big Bang Theory' Season 1 Episode 1, we manually labeled groud truth and correct speaker for the 473 face tracks we had detected. We labeled false positive detections in a separate 'ignore' ground truth category and marked them as not speaking. We then ran our learning algorithm with the same hyperparameters on this dataset. We obtained the following results:

- Frame labeling accuracy: 79.93%
- Track labeling accuracy: 85.23%

It is important to note that since false positives are always labeled as not speaking, the learning algorithm has no chance of ever correctly labeling them (it does not have any examples of the 'ignore' category). This bounds the theoretical maximum track accuracy to 89.4%. The authors of [1] report an accuracy of 95.18% when speakers are labeled manually instead of being found automatically. However, in the dataset they provide, there is not a single false positive detection, which is likely a result of manual pruning and helps accuracy considerably.

We show the resulting confusion matrix for track identification in Fig. 5. Most errors come from not identifying false positives ('ignore' category), which as we pointed out above is not possible.

Loss function	Accuracy (frames)	Accuracy (tracks)
Max prior	37.39	37.94
$\mathcal{L}_l$	70.72	82.15
$\mathcal{L}_l + \mu\mathcal{L}_u$	71.77	81.67
$\mathcal{L}_l + \mu\mathcal{L}_u + \gamma^{(n)}\mathcal{L}_n$	72.86	83.44
$\mathcal{L}_l + \mu\mathcal{L}_u + \gamma^{(n)}\mathcal{L}_n + \gamma^{(p)}\mathcal{L}_p$	80.28	**89.55**

Table 1: Frame and track classification accuracy in 'The Big Bang Theory' Season 1 Episode 1 for various loss functions.

Episode	Accuracy (frames)	Accuracy (tracks)
BBT-1	80.28	89.55
BBT-2	75.18	82.48
BBT-3	67.57	70.15
BBT-4	74.46	73.49
BBT-5	62.63	63.98
BBT-6	69.97	69.02
Average	71.68	**74.78**

Table 2: Frame and track classification accuracy for various episodes in 'The Big Bang Theory' Season 1.

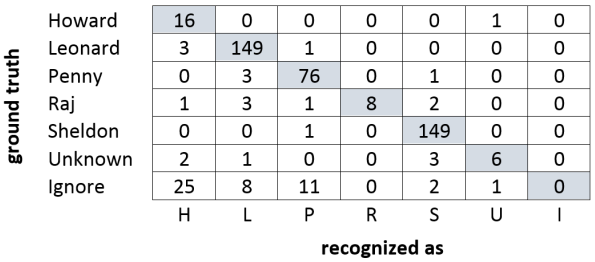

Figure 5: Confusion matrix of the learning algorithm on 'The Big Bang Theory' S01E01, using our face tracker and manually labeled speakers and ground truths.

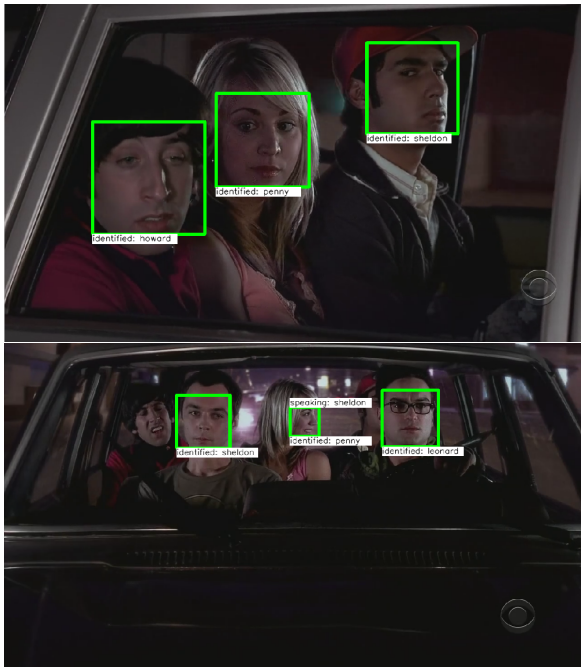

Figure 6: Examples of classification results. Characters which rarely speak (such as Raj in the first image above) are often misidentified. However, the learning algorithm also corrects some mistakes made by the speaker detection algorithm (such as Penny in the second image).

5 Conclusion

We have presented a full pipeline which takes only a raw video file and subtitle file and which outputs labeled face tracks. We have built a robust face tracker based on Viola-Jones detector and KLT tracker, implemented a learning algorithm that incorporates loss terms from various sources and tested it both on a publicly available dataset and on a dataset we built ourselves. We achieve performance close to that of [1].

In practice, most subtitle files do not include the names of the speaking characters, and to automate the process further the names should be obtained by matching subtitle files (which contain timing information) with episode transcripts (which contain no timing information but do contain the names of the speaking characters). We did not implement this but it should be relatively straightforward. More importantly, the next step to full automation would be to implement a working speaker detection algorithm. This could either be done using the method we detailed or using a threshold on the average pixel intensity of the mouth region as detailed in [10].

Another possible area of improvement is in the learning algorithm itself. As implemented, the learning algorithm is agnostic of the tracks themselves except for the positive and negative constraints we impose, and these constraints cannot be hard constraints because there is no way to enforce it in this framework. We could modify our model to use a hidden class variable Y for each track specifying its character. Then, weak labels are priors on the class of certain tracks, and frame by frame features $\mathbf{x}$ are noisy observations of the class variable, for example with distribution (with uniform prior on $\mathbf{x}$):

$$P(\mathbf{x}|Y=k) = \frac{P(Y=k|\mathbf{x}) \cdot P(\mathbf{x})}{P(Y=k)} \propto P(Y=k|\mathbf{x})$$

Positive constraints inside a track are automatically enforced, and negative constraints can be added as mutual exclusion constraints on the hidden class variables of two tracks. Parameters of such a graphical model can then be learned using Expectation Maximization, and performing a prediction for all tracks consists in finding the MAP assignment to all hidden class variables.

References

[1] M. Bauml, M. Tapaswi and R. Stiefelhagen. Semi-supervised Learning with Constraints for Person Identification in Multimedia Data. In *CVPR*, 2013.

[2] M. Everingham, J. Sivic and A. Zisserman. "Hello! My name is... Buffy" - Automatic Naming of Characters in TV Video. In *BMVC*, 2006.

[3] J. Sivic, M. Everingham and A. Zisserman. "Who are you?" Learning person specific classifiers from video. In *CVPR*, 2009.

[4] H. Ekenel and R. Stiefelhagen. Analysis of Local Appearance-Based Face Recognition: Effects of Feature Selection and Feature Normalization. In *CVPR Biometrics Workshop*, 2006.

[5] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. In *Mathematical Programming*, 45:503528, 1989.

[6] P. Viola and M. Jones. Rapid Object Detection using a Boosted Cascade of Simple Features. In *CVPR*, 2001.

[7] B. Lucas and T. Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, pp. 674-679, 1981.

[8] J. Shi and C. Tomasi. Good Features to Track, 1994.

[9] J.-Y. Bouguet. Pyramidal Implementation of the Lucas Kanade Feature Tracker. In *Intel Corporation Microprocessor Research Labs*, 2000.

[10] S. Siatras, N. Nikolaidis, M. Krinidis and I. Pitas. Visual Lip Activity Detection and Speaker Detection Using Mouth Region Intensities. In *IEEE Trans. Circuits Syst. Video Techn.*, 19(1):133-137, 2009.

[11] K. Manikantan, V. Govindarajan, V. Kiran and S. Ramachandran. Face Recognition using Block-Based DCT Feature Extraction. In *Journal of Advanced Computer Science and Technology*, 1(4):266-283, 2012.

[12] M. Kostinger, P. Wohlhart, P. M. Roth, and H. Bischof. Learning to Recognize Faces from Videos and Weakly Related Information Cues. In *AVSS*, 2011