tvID: Identifying Characters in a TV Show

Philip Meyers
Northwestern University
Evanston, Illinois
philipmeyers2017@u.northwestern.edu

1. Introduction

tvID seeks to label character's faces in an episode of a television show using just the episode's transcript, subtitles, and video frames. Each plays an important role in being able to guess which character is "on screen" throughout the course of the episode. An episode transcript serves as a record of *who* is saying *what*. It additionally offers a list of all character names in the episode, from which a subset to identify is later chosen. Subtitles for an episode contain *what* is being said *when*. Finally, video frames show *who* is on the screen *when*. By aligning the *what*'s of the transcript and subtitle and using the *when* alignment of the subtitle and video frames, the *tvID* can make a guess of *who* is on the screen and attempt to learn his or her facial features.

2. Data

2.1. Transcript

The transcript of an episode contains scene and setting information as well as all character dialogues. Transcripts are generally not released by television studios. Instead, transcripts are sourced from websites and wiki pages where dedicated fans post their transcriptions of episodes. *The Big Bang Theory S01E01* was chosen as a test material specifically because it was the first TV show that found in a search a fanbase that transcribed all 248 episodes over 11 seasons. Using fan-produced work presents a few challenges:

1. **Spelling errors** A fan's dedication to a show is not always matched by a dedication to proper spelling. Correct spelling of words, especially rarer words, is essential to lining up an episode transcript with subtitles. Spelling mistakes are manually corrected in each transcript by copying the transcript into Microsoft Word and performing a spellcheck over the document.

2. **Formatting differences** Authors used slightly different characters to indicate auxiliary scene information. For example: `(Raj looks at her, looks back at his food, takes a mouthful)` and `[Raj looks at her, looks back at his food, takes a mouthful]`. Since this scene information is relatively infrequent (less than 10 per script), they are removed manually at the same time as the spell checking process.

3. **Accents and non-standard punctuation** Some transcripts contain accents and non-standard punctuation, like curly apostrophes and quotation marks. Curly apostrophes and quotation marks are substituted for their non-curly counterparts and any remaining non-UTF-8 characters are ignored.

After manual and automatic pre-processing steps, a transcript can be parsed as a list of character dialogues of the form `CHARACTER: Line`. The set of `CHARACTER`'s are treated as the list of possible labels. To reduce noise, it is paramount that every item in the set refers to a real main character. Two steps of filtering are used to remove non-characters:

1. The `CHARACTER` component is occasionally used to designate an abstract character concept like `ALL` or `TOGETHER` or even some `SCENE` for more specific setting information. Such dialogues are ignored by checking if they are members of a list of hand constructed stopwords.

2. Dialogues from characters with fewer than n total dialogues are removed under the assumption that these characters are not main characters and have too little airtime to properly identify. In practice, a threshold of $n = 5$ is sufficient.

An example of a portion of a processed script is shown in Figure 1.

2.2. Subtitles

Unlike fan-produced transcripts, subtitles follow a consistent format. Each subtitle file is composed of discrete subtitle data chunks which contain an index, start and end timestamps, and display text. A sample of subtitle chunks

```
Penny:  Oh, you're inviting me over
to eat?

Leonard:  Uh, yes.

Penny:  Oh, that's so nice, I'd love
to.

Leonard:  Great.
```

Figure 1: Sample of processed transcript of *The Big Bang Theory E01S01* [2]

are shown in 2. The subtitle file is entirely noise free, except for the last chunk which contains source information as well as default display settings like font color and size. Given the regularity of the format, subtitles are easy to parse. Furthermore, when the frame rate of the video is known, the subtitle chunk display times are converted to frame ranges. Subtitle chunks generally correspond to the dialogue of a single speaker. When a subtitle chunk contains dialogue for multiple speakers, each dialogue line is prefixed by a dash (−). These subtitle chunks are ignored because it is impossible to determine to a finer degree which character's line is being spoken at what time.

2.3. Video

Video frames are extracted from the TV episode itself. Sound is ignored because it does not provide any more useful information for the alignment of transcript to video. TV episodes were obtained from various online (and likely unsanctioned) websites. Although it is easy to find streams for almost every episode of any television show online, locating safe, high quality downloads was more challenging. A useful trick is to prepend `index of` to a query for an episode and resolution (i.e. `index of big bang theory s01e01 720p`). Doing so often return results of indices from streaming sites where shows can be downloaded safely and quickly.

3. Pipeline

The process of identifying characters in a TV episode consists of eight steps, as outlined in Figure 3.

3.1. Identifying Subtitle Speakers

Aligning an episode's transcript with its video subtitles is necessary in order to be able to make a guess of which character is on the screen at certain times throughout the episode. This alignment is accomplished in the process of identifying the character speaking during a subtitle:

```
105
00:05:43,200 --> 00:05:44,900
You're inviting me over to eat?

106
00:05:47,300 --> 00:05:49,300
Oh, that's nice.
I'd love to.

107
00:05:49,500 --> 00:05:50,500
Great..
```

Figure 2: Sample of subtitle chunks from *The Big Bang Theory E01S01* [1]

1. **Load and process data** Data from both the respective transcript and subtitle files are read in and processed. Noisey data like non-character dialogues and multi-speaker subtitle chunks are discarded. Non-UTF-8 characters are ignored.

2. **Tokenize text** All dialogue lines and subtitle text are tokenized. Stopword tokens from NLTK's English stopword set and empty tokens are removed. All punctuation is also discarded.

3. **Count tokens** Frequencies for tokens in the dialogue set D and subtitle set S are calculated and compared. Tokens t that occur an equal number of times are recorded with their occurrence count c in E as tuples (t_i, c_i).

4. **Find correspondences** For each token and occurrence count $(t, c) \in E$ and for each $i = 1, ..., c$, locate the dialogue line d_i and subtitle chunk s_i that contains the i-th occurrence of token t. Vote for the character of d_i as the speaker of s_i.

5. **Tally votes** For each subtitle chunk $s_i \in S$, if all speaker votes for s_i are for the same character, then choose that character as the speaker of s_i.

Although simple to implement and cheap to compute, this algorithm offers great performance. When evaluated on the transcript and subtitles for *The Big Bang Theory S01E01* [2, 1], it offered a precision of 1.00 and recall of 0.87 on 409 total subtitle chunks (356 total identified). Subtitles are not present throughout an entire episode. In the aforementioned episode, subtitles are displayed during 60% of the episode (13.8 minutes of 22.7 total minutes). Tagged subtitles account for 89% of the period during which subtitles are displayed. Tagged subtitle chunks are well distributed, enabling a thorough sampling of characters and

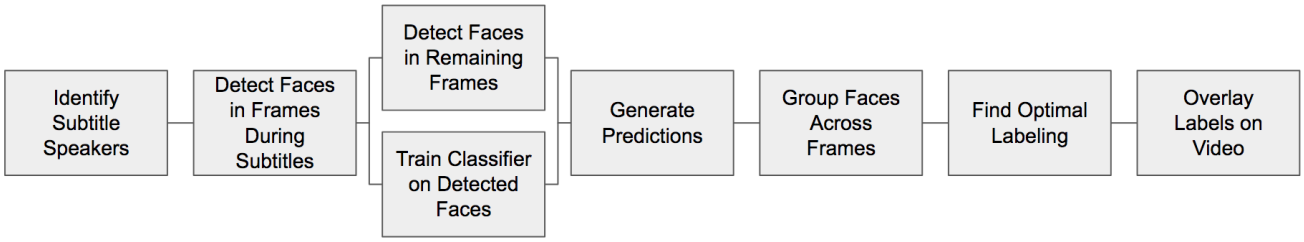

Figure 3: Overview of processing pipeline (executed left to right). *Detect Faces in Remaining Frames* and *Train Classifier on Detected Faces* are stacked because they are independent processes and can be executed concurrently to reduce overall computation time.

faces throughout the video (Figure 4). Note that this rate is slightly higher than the reported recall rate. Subtitle chunks with more tokens have a higher rate of being tagged because the greater number of tokens offers more opportunities to find equally occurring tokens. Subtitle chunks with more tokens are also generally present for longer durations, leading to a marginally higher overall tagged duration as seen in Figure 5.

3.2. Preliminary Face Detection

The tagged subtitle chunks are used to construct training data for a facial classifier. This training data relies on the assumption that whenever a character is speaking, that character is the most likely character to be shown on the screen. Humans intuitively focus their gaze on a person when that person is speaking to look for additional facial cues [FIND REFERENCE]. In visual entertainment, there are many cases that break this assumption:

- Character A has dialogue but the screen pans to character B to show B's reaction.
- Character A is offscreen and we only see character B listening and responding to A.
- Character A has dialogue but there are multiple other characters in the same shot.

An effective classifier can still be trained from the constructed data because these failure cases are relatively infrequent. The case of the latter case (the most common case, by observation) is further minimized by the use of sample weighting during training.

3.2.1 Facial Recognition Model

Construction of the dataset is achieved with a facial recognition model. Though many models exist, *dlib*'s deep learning model was selected for the following reasons: 1. The model is based on ResNet architecture proposed by [3] and achieves state-of-the-art accuracy of 99.38% on the Labeled Faces in the Wild dataset [4]. 2. The *dlib* implementation offers interfaces for locating the bounds of faces in an image and for extracting 128-dimensional features for a face within a supplied boundary. 3. The model has a nice Python interface via the package `face_recognition` [**?**]. 4. My testbed's high performance graphics card (*Nvidia GTX 1080 Ti*) allows for accelerated GPU computing, which yields lower computation times than less performant CPU-based facial recognition models (like HOG). Additionally, offloading facial recognition to the GPU frees up the CPU for concurrent training of the model. The model accepts three hyperparameters: 1. `num_times_to_upsample` selects how many times the image is upsampled before being sent through the model to locate face boundaries. Higher values result in the detection of smaller face boundaries (characters further from the camera plane) but are computationally more expensive. Setting `num_times_to_upsample` $= 1$ resulted in reasonable computation time while still offering a high recall rate. 2. `num_jitters` determines how many times a face is resampled when calculating an encoding for the face. Again, higher values lead to greater accuracy at the cost of linear increase in computation time. Retaining the default value `num_jitters` $= 1$ offered satisfactory performance without any increases in computation time.

3.3. Training Face Classifier

3.3.1 Training Dataset

The training dataset is composed of faces detected in frames during labeled subtitle chunks. Each video frame during a subtitle chunk with a labeled speaker is fed into the *dlib* model and boundaries for detected faces (if any) are returned. Batch processing with a batch size of 64 (limited by the *GTX 1080 Ti*'s 11GB memory) to reduce computation time. Then each located face boundaries and its respective frame is fed back into the model to extract 128-dimension encodings for the face within the provided location. The label of each 128-dimension encoding is simply the character

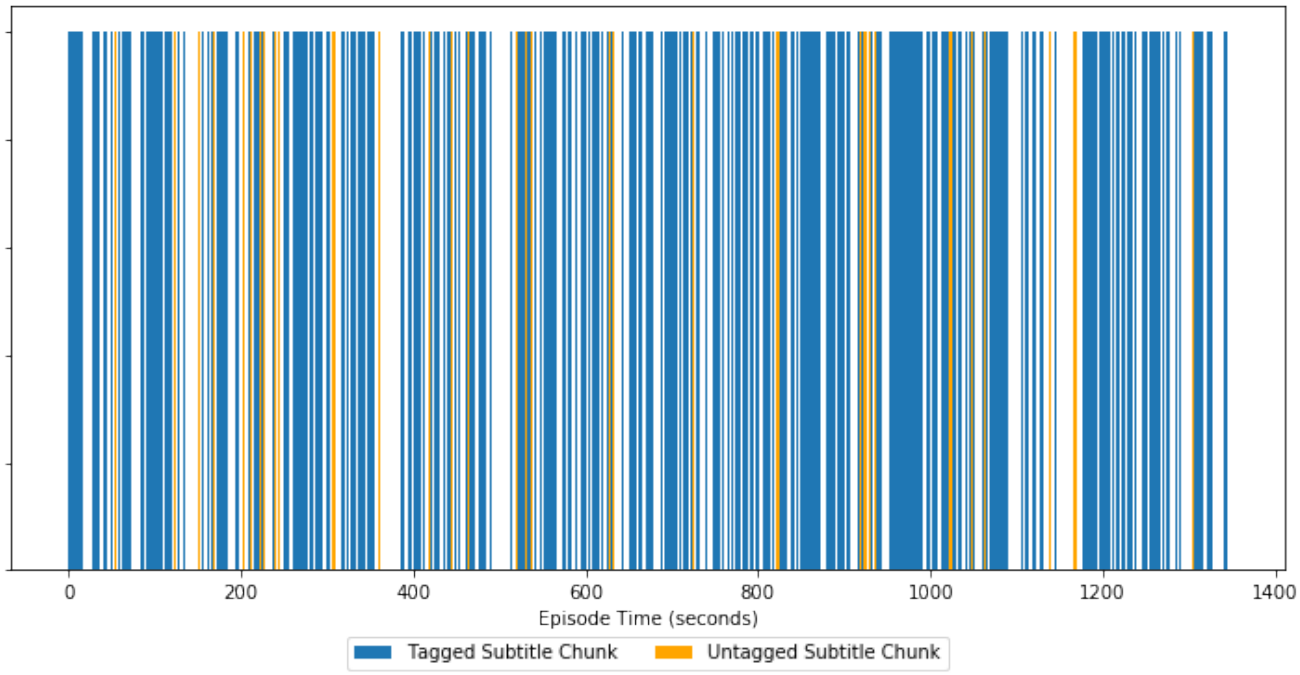

Figure 4: Temporal distribution of subtitle tag state throughout the duration of an episode of *The Big Bang Theory S01E01*. White regions have no subtitles.

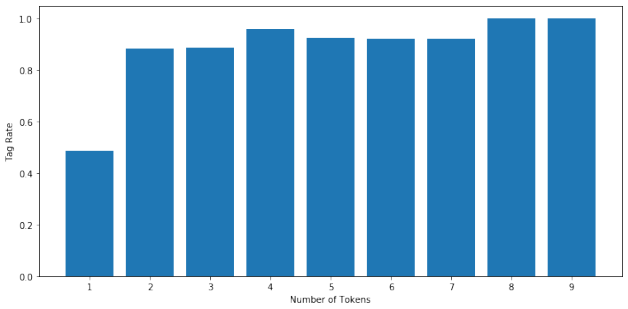

Figure 5: Distribution of subtitle chunks tagged based on their number of tokens

identified as the speaker for the subtitle chunk during which the encoding's face occurs.

This dataset is full of noise as many data points will inherently have incorrect labels. For example, if three faces are identified in a frame that occurs while character A is speaking, all corresponding face encodings are labeled as character A when only a maximum of 1 can be correct. An initial attempt to remove this noise was to construct the training data exclusively from faces in frames in which no other faces were identified. The intuition being that if a character is speaking and there is only one face on the screen, then that face belongs to the speaking character. This filtering significantly reduced the size of the training dataset and yielded far worse results. The best results were achieved when I used all data points and weighted them by the number of faces on the screen:

$$w(x) = \frac{1}{e^{C(x)-1}}$$

where $C(x)$ is the number of faces detected during the same frame as x. It is likely that the intuition did not hold true, rather using a weighted aggregate of all faces actually produces a distribution that more closely resembles that of the true distribution by reducing the impact of noise from incorrect labels. A total of 22,920 data points are collected from 17,763 frames in this manner for *The Big Bang Theory S01E01*.

3.3.2 SVM Classifier

A multi-class Support Vector Machine (SVM) classifier is used for facial classification. SVM was chosen from among other supervised classifiers for its performance on high-dimensional data, readily available implementation in `scikit-learn` [6], and ability to use sample weighting. Default settings for the classifier were used except for `probability` parameter which was set to `True`. By setting `probability = True`, the classifier outputs a probability distribution over all training classes for a given input. Getting a prediction distribution (as opposed to a sin-

gle class prediction) is essential to later steps when predictions are aggregated and for cases when the most likely class cannot be chosen.

The classifier is trained on the collected data points using the sample weighting scheme discussed previously. From here on, the terms 'label' and 'class' are used interchangeable to refer to a character.

3.4. Secondary Face Detection

While the classifier is being trained on the CPU, the idle GPU is leveraged to obtain encodings for faces in all other frames. The process for obtaining these encodings mirrors that of the initial collection of encodings for training the classifier. Each frame that does not occur during a tagged subtitle chunk is fed into the face detection model. Locations for faces and their corresponding 128-dimensional encodings are obtained and recorded. All face encoding data is stored in a dictionary indexed by frame number for convenient processing down the road. No model parameters are changed.

3.5. Prediction Generation

At this point in the pipeline, all frames have been searched for faces and encodings have been collected for each identified face region. Predictions are generated by feeding face encodings into the trained classifier and recording the returned probability distribution over all classes. To reduce computation time, predictions are performed in batches of 128.

3.6. Facial Grouping

With predictions in hand, a nave next step would be to assign the most likely class (character) as each face's label (character). The results of such an approach are particularly noisy and exhibit 'flickering' among numerous labels for the same face. In other words, if a character's face is on screen for 8 frames, then choosing the most likely label at each frame yields a labeling sequence like

Frame	1	2	3	4	5	6	7	8
Label	B	C	B	C	A	A	B	D

Here A, B, C, and D are a subset of all possible labels. Noisy labeling is particularly prevalent when a character's head is rotated or further from the camera as seen in Figure 6. Not only is such labeling visually distracting, but also inherently incorrect most of the time. A face can only have a single label (no actor plays two characters) so at most only a single label from the set is correct. Interestingly, the correct face label is almost always among the subset of labels.

3.6.1 Temporal Grouping

Using these observations, an intermediate step is introduced in the pipeline to smooth and improve face predictions. Rather than treating each face in a frame as independent from all other faces in frames, faces can be grouped across adjacent frames. Introducing face groups eliminates 'flickering' predictions because each face group is assigned single label. The label is propagated to all member faces over the frames in which the group appears (ideally the whole duration that the character's face is on screen). These single labels are also of higher quality than the individually determined labels. A face group produces a new prediction distribution from the individual predictions of all member faces. Recall that quality of a face's prediction is diminished by rotation of the head or distance from the camera. Because TV actors are generally dynamic, collecting predictions over the entire time that a face is temporarily on screen increases the likelihood that at least some predictions will be of high quality.

Grouping faces is a relatively simple process. Once a new face boundary is detected, the next frame is searched for a face such that the intersection of the current face and the face in the next frame has at least $\textit{OVERLAP\_MIN} = 60\%$ the area of the current face bounds. If such a face is detected, that face is given the same group id as the current face, and then treated as the current face. The processes continues until no such faces are found in $\textit{MISS\_MAX} = 12$ consecutive frames (0.5 seconds of video) or the end of the video is reached.

3.6.2 Prediction Aggregation

Once groups have been determined, an aggregate prediction for the group is produced by combining all of its member predictions. Multiple strategies were explored for combining predictions. Included with each strategy is the result of its application to the group predictions:

	1	2	3
A	0.20	0.30	0.45
B	0.70	0.50	0.30
C	0.10	0.20	0.25

1. Choose the member prediction with the greatest value (highest probability for the most likely class).

A	0.20
B	0.70
C	0.10

2. Sum across each class and renormalize.

A	0.32
B	0.40
C	0.28

Figure 6: 'Flicking' assignments over a 3 second clip of the side of a character's head. The correct label, `Howard` appears multiple times.

Algorithm 1 Face Grouping Pseudocode

```
procedure GROUPFACES(frames)
    groups ← {}
    id ← 0
    for f in frames do
        for face in f do
            if face in groups then
                continue
            else
                groups[face] ← id
                id ← id + 1
                EXTENDGROUP(f, face, groups)
    return groups
procedure EXTENDGROUP(frame, face, groups)
    misses ← 0
    for f in FRAMESAFTER(frame) do
        if misses ≤ MISS_MAX then return
        o ← FINDOVERLAPPINGFACE(f, face)
        if o is None or AREAOVERLAP(face, o) ≤ OVERLAP_MIN then
            misses ← misses + 1
        else
            misses ← 0
            groups[o] ← groups[face]
```

3. Choose the highest probability for each class and renormalize.

A	0.32
B	0.50
C	0.18

4. Sum the ranking of each class across member predictions and normalize. For example, the rankings of

A	0.39
B	0.44
C	0.27

Of the four strategies, the third was chosen because it observationally yielded the best results.

3.7. Optimal Labeling

Even after aggregate predictions have been produced for each group, the problem of assigning each group a class from among its predictions remains. Again, simply choosing the most likely class produces results that are impossible in the real world. Consider the case when two faces are on screen simultaneously where the aggregate predictions for each face's group is one of the examples produced above. Class B is the most likely class even though each distribution is unique. Choosing the most likely class for the two face groups would then result in a situation where both faces are given B's label. In the same way that one face cannot be multiple characters, one character cannot be in multiple places on the screen at the same time. A greedy forward pass could be applied to choose the highest possible class that does not conflict with previously determined assignments. However, given that the results will be viewed by humans, there is a high cost of entering local a local maxima and choosing one or more incorrect labels. It is preferable to find the globally optimal labeling scheme that maximizes the total likelihood of all label assignments subject to the constraint that no two faces in the same screen can share a label.

This global maxima can be obtained through constrained linear optimization. By reformulating the face labeling problem as a linear program the maximum likelihood assignment scheme can be solved for deterministically. A linear program is composed of variables, linear constraints, and a linear objective function that is maximized or minimized. Let G and L be the sets face groups and face labels, respectively. Then the formulation used to solve for the optimal assignment scheme is:

- **Variables** Each possible assignment of a label to a group is a binary variable.

$$\text{Let } a_g^l = \begin{cases} 1, & \text{if group } g \text{ is assigned label } l \\ 0, & \text{otherwise} \end{cases}$$

  There are a total of $|G| \times |L|$ variables.

- **Constraints** Two types of constraints are enforced: 1. Each group is assigned exactly one label 2. If two groups are ever on screen at the same time, then those groups cannot be assigned the same label. The first set of constraints is constructed as

$$\forall g \in G, \sum_{l \in L} a_g^l = 1$$

  Note since all variables are binary, each constrained ensures that one and only one summand $a_g^l = 1$. Define

$$C = \{(i, j) \mid g_i \text{ and } g_j \text{ appear concurrently}\}$$

  Then the second set of constraints is constructed as

$$\forall (i, j) \in C, \forall l \in L, a_i^l + a_j^l \leq 1$$

  The inequality ensures that the a label is assigned to at most one face group at any given time.

- **Objective** Let P be the distributions produced in Section 3.6.2 so p_g^l is the likelihood of group g being label l. Then the objective is to maximize the sum of likelihood assignments

$$\text{maximize} \sum_{g \in G} \sum_{l \in L} p_g^l a_g^l$$

The `PuLP` library [5] is used to determine the optimal labeling within a Python environment. `PuLP` does not directly solve the linear program. Instead it provides a programmatic interface to construct a linearly program, generates a text representation of the problem, passes the representation to a dedicated linear program optimizer (COIN Branch and Cut Solver), and parses the returned solution. The optimal labeling is determined by identifying all variables in the solution with value 1.

3.8. Label Overlay

The final step in the pipeline is to overlay labels in video. `OpenCV`'s `rectangle()` and `putText()` are used to draw a blue bounding box around each face identified region and write the chosen label as well as prediction confidence. Red bounding boxes are drawn around faces whose prediction confidences were lower than a chosen threshold. A black bar and character name are also are overlaid at the top of the screen throughout tagged subtitles (the blue regions in Figure 4).

4. Results

Confusion matrices for *The Big Bang Theory S01E01* and *Seinfeld S01E05* are presented in Figure 7 and Figure 8, respectively. Evaluation *The Big Bang Theory S01E01* is quite good except in the cases of characters *Raj* and *Receptionist*. The *Receptionist* only appears in a few scenes in the beginning of the episode, but has a fair number of lines during those scenes. On the contrary, *Raj* appears in more scenes at the end of the show, but his character has a certain restriction that he cannot speak in the presence of women. Thus although he appears more often near the end of the episode, he is almost always in the presence of at a woman (*Penny*) and hence he has very few dialogue lines that can be tagged and used to identify his face. On the contrary, *Man* has perfect performance because he appears once in the episode and speaks alone for almost the entire duration, allowing the model to closely learn his facial attributes.

Performance on the *Seinfeld* episode leaves much to be desired. Only *Jerry* and *George* have similar performances to those of the previously discussed episode. The performance disparity was so great that it warranted another look at the processing of the episode. A closer inspection of the episode, script, and subtitles revealed that the subtitles times were not properly aligned with the video! Given the misalignment of data sources, the results make a lot more sense and are, to a certain extent, surprisingly good.

Labeled frames from the two episodes are displayed in Figure 9.

5. Implementation Details

The above pipeline was implemented in Python using Jupyter notebooks. Total total implemented spanned almost 800 lines of code and used the following the libraries:

1. `NumPy` (`http://www.numpy.org/`)
2. `scikit-learn` (`http://www.scikit-learn.org/`)
3. `scikit-video` (`http://www.scikit-video.org/`)

	Raj	Man	Sheldon	Receptionist	Penny	Howard	Leonard
Raj	**0.08**	0.00	0.67	0.00	0.00	0.17	0.08
Man	0.00	**1.00**	0.00	0.00	0.00	0.00	0.00
Sheldon	0.00	0.00	**0.92**	0.01	0.04	0.01	0.02
Receptionist	0.00	0.00	0.43	**0.57**	0.00	0.00	0.00
Penny	0.00	0.00	0.02	0.00	**0.98**	0.00	0.00
Howard	0.04	0.00	0.00	0.00	0.04	**0.88**	0.04
Leonard	0.00	0.00	0.13	0.00	0.06	0.03	**0.78**

Figure 7: Confusion matrix for *The Big Bang Theory S01E01*.

	George	Jerry	Dry Cleaner	Kramer	Elaine	Vanessa
George	**0.73**	0.27	0.00	0.00	0.00	0.00
Jerry	0.00	**0.92**	0.02	0.02	0.03	0.02
Dry Cleaner	0.00	0.67	**0.33**	0.00	0.00	0.00
Kramer	0.00	0.60	0.00	**0.40**	0.00	0.00
Elaine	0.00	0.42	0.00	0.00	**0.58**	0.00
Vanessa	0.00	0.50	0.00	0.00	0.00	**0.50**

Figure 8: Confusion matrix for *Seinfeld S01E05*.

4. `OpenCV` (`https://opencv.org/`)

5. `face_recognition` (`https://github.com/ageitgey/face_recognition`)

6. `PuLP` (`https://github.com/coin-or/pulp`)

All code and data sources (excluding video which is copyrighted content) are hosted online at `https://github.com/feelmyears/tvID`. Result video is available on request.

6. Future Work

Given more time, there are four opportunities for further innovation:

1. **Automated preprocessing** Much of the preprocessing of the script is done by hand and it would be easier, faster, and generalize the approach to be able to convert any script format into a standard, usable format for the pipeline.

2. **Concurrency** Although *Detect Faces in Remaining Frames* and *Train Classifier on Detected Faces* are shown to be concurrent in Figure 3, doing so in code was beyond the scope of the project. Executing them concurrently would cut down on computation time, but given the experimental nature of this project, fast execution was not necessary. Still, implementing concurrency would be a good learning experience and a final optimization for the pipeline.

3. **Object-tracking face grouping** In the the current approach of grouping faces across screens, if the camera switches from one angle to another and two different faces are in similar locations in the different shots, then those faces will be grouped into a single face group. Raising the area overlap threshold and lowering the frame miss threshold can prevent some of this confusion, but only to a point, after which overall performance is degraded. A superior solution would be to look for the first occurrence of a face and then track it until it leaves the screen (or the camera angle switches) using an object tracker. This approach also has the benefit of continuously tracking a face throughout a shot, rather than relying on the face detection model to locate the bounds of a face in every frame the face exists (which it sometimes fails to do). Following the face throughout each frame will produce smoother, most consistent bounds to train from, identify, and label.

4. **Augmented facial recognition** Using more than one facial recognition model could improve face detection rates as well as labeling performance by offering new encodings based on different features on which the classifier can train.

There is room for improvement on the evaluation side. It is quite tedious and time consuming to produce confusion matrices for an episode. The approach used in Section 4 requires the evaluator to watch the episode at half speed and frequently pause the video to record all labeling data. This makes it near impossible to search the hyperparameter space to find the best possible pipeline configuration.

A preferable approach would be to somehow pre-label all faces in an episode and automatically compare the generated labels with the ground truth labels.

Another unexplored evaluation method is the transitivity of a classifier on another episode. In other words, how well does a classifier trained on episode A perform when applied to episode B.

Finally, it would be interesting to see if the face encodings learned in multiple episodes and TV shows can be combined to attempt to label the actual actors and actresses playing the characters (assuming that some TV shows have an intersection of common actors and actresses.

7. Conclusion

This project was a fantastic learning experience and offered a lot of insight into how to integrate academic and engineering principles to solve a problem with real data.

References

[1] The big bang theory s01e01 english subtitles. www.tvsubtitles.net.

[2] The big bang theory series 01 episode 01 - pilot episode. https://bigbangtrans.wordpress.com/series-1-episode-1-pilot-episode/.

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[4] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller. Labeled faces in the wild: A database for studying face recognition in unconstrained environments. Technical Report 07-49, University of Massachusetts, Amherst, October 2007.

[5] S. Mitchell, S. M. Consulting, and I. Dunning. Pulp: A linear programming toolkit for python, 2011.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Figure 9: Screenshots of results from *The Big Bang Theory S01E01* (top) and *Seinfeld S01E05* (bottom)