Unsupervised Character Recognition in Cartoons

Nishad Mudras
nmudras@cs.umass.edu

Prateek Sharma
prateeks@cs.umass.edu

Abstract

In this project we present the design and implementation, and evaluate an unsupervised system for recognition of the main character *of an animated cartoon video. The main character (not to be confused with a text character!)is the person who occurs in the largest number of scenes in a video. Our approach eschews the commonly found supervised-learning based techniques, and instead is* completely unsupervised.

1. Introduction

The problem that we consider in this project is that of identifying *characters* in an animated cartoon video. In order to do this, we utilize a combination of image segmentation, clustering, and infer association-rules for recognizing the characters. Most importantly, we do everything in an unsupersived manner, and do not use any labelled/test data.

1.1. Unsupervised Learning

An unsupervised approach does not require labelled training data, a situation which is predominant because of the difficulties in acquiring high quality human-labelled training sets. While human-curated labelled data exist for a significant fraction of computer vision tasks (for example the SIFT descriptors), they are not universally available. The goal of this project was to explore and evaluate how far the unsupervised learning approach can be pushed in the character recognition domain.

On the flip-side, unsupervised learning also has many disadvantages, as we quickly learned. The lack of ground-truth presents a particularly nasty challenge. Without

1.2. Why Cartoons

This project focuses on recognizing characters in cartoons. Using cartoons as the environment in which we perform the recognition task has several important advantages which are manifested due to their simplistic visual nature.

Cartoons (see Figure **??** for a few example frames) have a relatively simple visual style. Textures are mostly non-existent, and colors are either solidly filled or simple gradients are used. The edges in the images are also smooth, with well defined curves. All of these features are artifacts of the cartoon creation process. While more modern cartoons may have a sophisticated visual style, we focus our attention to the simpler styles which are found in the vast majority.

1.3. Scope of this project

We study unsupervised techniques in cartoon character recognition. Our dataset consists of *SouthPark* videos, a popular animated-cartoon TV series. While it was our intent to explore the application of our technique to several other animated cartoons as well (for example *Simpsons, Family Guy, Pokemon*), we decided to restrain our focus to only SouthPark.

Given a video, our end-goal is to provide the list of scenes(or frames) in which any character appears. Alternatively, this can also be thought about as identifying characters in a given frame. Using this *appearance information*, we can provide analysis to answer questions such as:

- Which scenes does a character occur?
- Who is the main character in this video? The main character is the one who occurs in the largest number of scenes
- What is the percentage of scenes in which a character occurs? This can be used to improve the information retrieval in videos by providing an additional query parameter. For example, users might want to search for videos in which their favourite character occurs in atleast 50% of the scenes etc.

2. Background and Related Work

In order to detect people, the standard approach is to use a face recognition technique, since face recognition is a well understood problem. Because it relies on human face structure, face detection algorithms do no work with cartoon images at all because of the "cartoonish" nature of the faces with exaggerated features etc. As a result, one of the primary techniques for character recognition is ruled out.

Character detection in video sequences is a fairly well studied problem [2, 3, 5, 4]. Most of the work [1] relies on face recognition, since human face recognition is a

well studied problem as well. As noted earlier, face detection/recognition does not work for animated cartoons, and thus a vast majority of literature in this area is of little use to us.

For the animated cartoon domain, the results are not as numerous. [6] uses multi-scale shape detection for cartoons and a supervised machine learning approach, which is in contrast to our unsupervised approach.

3. Design

As mentioned previously, the focus of our design has been the use of unsupervised techniques. This section outlines the design of our image processing pipeline. We describe the high-level idea, the techniques used, and the alternatives to some the design decisions we took.

3.1. High-level pipeline

We assume that the entire video for which the character detection needs to be performed is given as the input. The pipeline makes 2 passes over the video, and thus a streaming approach would not work. We discuss an alternate solution which can work in the streaming setting later.

Below is the high-level pipeline, with each step explained in detail later:

1. Input: A video file
2. Process the video to extract frames. A sampling of frames is done to extract 2 frames per second instead of 24.
3. Scene detection. Even after the frame sampling, the changes in frames is small. We identify scene changes and pick a representative frame for a scene. All the rest of the frames are discarded.
4. For each frame, we perform segmentation and extract the image segments.
5. Each segment is processed to produce the features.
6. All the segments(and their features) from all the frames are then clustered to form a dictionary of segments.
7. Clusters which represent a whole or part of the characters are identified
8. The scene frames are processed again. They may be resegmented again, and we assign each segment to the appropriate cluster label.
9. From labelled segments, objects are recognized.
10. From detected objects, we recognize characters.

3.2. Scene detection

To detect changes in scenes, we inspect changes between consecutive frames. The frames are subtracted to form a *diff frame*. If the pixels dont change between frames, the diff frame is all zeroes. Any non-zeroes are indicative of a change. If the changes are above a certain threshold, the scene has changed. A small threshold yields a larger number of scenes, whereas a larger threshold is more sensitive to scene changes.

Using the scene detection, we are able to extract about 200 frames from a 20 minute episode.

3.3. Segmentation

Once frames from the video have been sampled (by using scene detection above), we process each frame independently to extract the *segments* from the image.

A crucial assumption/hypothesis of our project has been that *segments are good proxies for objects in animated cartoons*. We have evaluated this hypothesis for a large variety of segmentation algorithms (see Figure 1). Visual inspection (since we do not posses ground truth data) strongly suggests that objects (such as the characters' headgear, faces, clothes etc) are easily segmented from most frames. Ofcourse, we as yet do not know which segment corresponds to which object, which is considered in the next step.

We have the evaluated the effectiveness of **Normalized-cuts, Watershed and QuickShift** with different choice of parameters. Because its difficult to show the effectiveness of these without ground-truth, we do not include the results of the segmentation process, since a lot of it was done by visual inspection on a large number of frames to determine effectiveness. Both Watershed and normalized-cuts worked equally well, but use normalized-cuts as our primary segmentation algorithm because of the lack of any pre-processing (as required by watershed) and also because the number of segments can be given as a parameter to the normalized-cut algorithm.

The question then is how to determine the number of segments. Certain frames have a lot more segments than others. To solve this problem, we use the **Grab-cut** technique to extract the foreground of every frame. This eliminates the background noise in the images , and provides a filtered image for the segmentation algorithm. Grabcut requires a bounding box as the input. We provide the entire frame (minus a small border strip) to Grab-cut. Figure 3 shows the results of grabcut being applied on a frames. We can see that the background is removed from the images quite nicely. After grabcut, the maximum number of segments we segment the image into is *20*. If fewer number of segments are detected, then norm-cut outputs fewer segments. Too few segments clusters multiple objects into one. Whereas a larger number of segments will break down an object into many segments. A "perfect" segmentation for us yields a

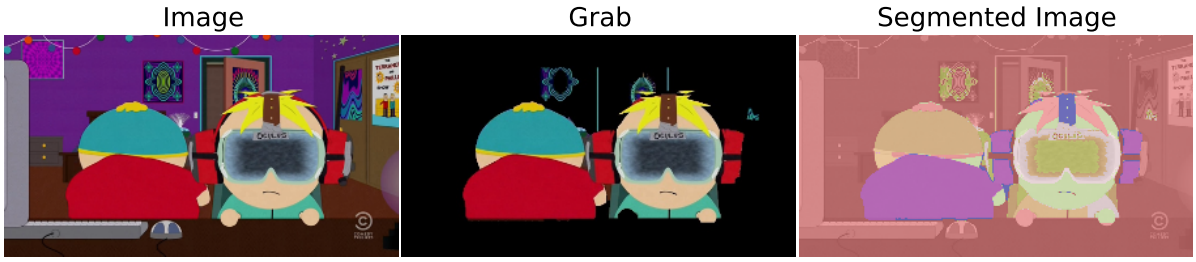

Figure 1. The result of the segmentation step of the pipeline. Grabcut is able to remove the background, and the norm-cut segmentation algorithm yields the segments. Each segment is represented by a different color overlayed on top of the original image.

one-to-one correspondence between objects and segments (an entire object is represented as one segment).

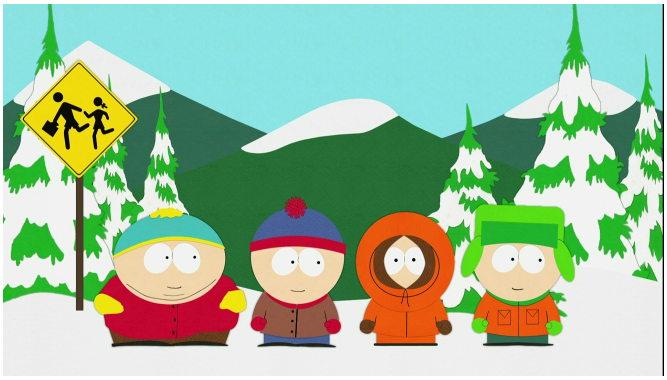

Figure 2. A sample frame from SouthPark. Notice the constant color-fills, smooth lines and edges, and lack of textures and complicated gradients.

Figure 3. Image after grab-cut being applied to the entire image. Grabcut is surprisingly good at removing a lot of the background. This reduces the number of segments present in the image, and particularly the larger, background segments.

3.4. Feature extraction

Each segment is processed independently to extract features from it. For animated cartoons the features that work best for animated cartoons are:

1. **The color of the segment.** We've observed that most segments are of predominantly one color. Thus, the average color across all the pixels of a segment can work as a feature. Another approach is to cluster the colors into a small number of colors (say 3) and use the cluster centres as features. The cluster centres are also weighted by their frequency (histogram). The intuition for using color as a feature is to detect clothing of characters, since characters mostly wear the same clothes. Thus the cap, jacket are segmented into their own segments, and are of uniform color throughout for a large number of scenes.

2. **Segment shape.** We can also use the segment *shape* as the a feature. We use the HOG features, and apply on each segment. This seems like an appealing at first, but does not seem very useful in increasing the accuracy of classification. Object shape has two problems. First, objects belonging to different characters are similarly shaped. The head,faces,jackets are all very similarly shaped. Thus we dont get specificity. Second, due to changing perspectives and angles, the shape does not remain constant over scenes either.

3.5. Dictionary creation using clustering

Once all the features from the images have been extracted, we store them for later use. The features are then fed to a clustering algorithm such as k-means. We have use 3 different clustering algorithms: kmeans, Agglomerative clustering, and DBSCAN. These are evaluated in the evaluation section.

The choice for the number of clusters is a very important one. While silhoutte scores guided us to use higher values for the number of clusters, having too many clusters makes the job of recognition more difficult. This is the key tradeoff. If there are too many clusters, then the association rules (cluster c maps to object d) become much harder, since we

lose the bijective property and need multiple clusters to represent the same object.

On the other hand, having fewer number of clusters means that the clusters are shared between objects. This is an even worse problem, since now we need to resort to other techniques. Accordingly, we have set the number of clusters in the 150-300 range.

3.6. HSV color representation for clustering

Specifically for cartoons, it was hypothesized that use of the HSV color space for clustering using k-means, instead of the RGB representation, would provide more tightly bound and accurate clusters. This was due to the fact that in cartoons, for a given object color (shirt, hat, etc.), the hue value remains almost same across multiple frames, while the saturation and value components might show few changes depending on the time of the day in a particular scene. As opposed to this, RGB can show variations across all its three components based on variations in light, which can skew the distance-based clustering process of k-means. The HSV color space enables the separation of the chrominance information from the luminance information, thus providing components such as color, vibrancy and brightness, as opposed to the RGB color space that doesnt provide a distinction between these factors.

As seen in Figures **??**, the difference between the largest of clusters sizes is more pronounced in results using HSV than using RGB. Also, the secondary cluster (i.e. the second largest cluster) represents the skin, which is as desired, since the component occurs across almost all the frames - while such a cluster in RGB representation is skewed; an unrecognizable color is obtained, which could be resultant of different colored segments clustered together.

3.7. Segment recognition

Once we have obtained a dictionary, the next step is to actually recognize the objects. We again grabcut and segment the image to obtain the segments, and get the feature vectors for each segment. The problem now is to assign segments to cluster centres.

A straightforward approach is to simply assign segments to their closest cluster-centre, a step supported by all the clustering algorithm implementations. This is a "greedy", local way of assigning *labels* to the segments. A more sophisticated strategy that we have tried is to instead consider pairs of neighbouring segments, and then do find a pair of clusters which minimize the distance to both the corresponding segments.

The problem now is : *How to recongnize objects from labels?*. Again, availability of ground-truth or tagged data would prove immensely useful here. An automated way of assigning labels(clusters) to objects is to use a frequency count of the labels in the clustering histogram.. If a label occurs very often, it is likely to belong to the leading characters of the episode since they occur in a larger majority of the scenes . An advantage of this method is that this is completely automated, and also robust to changes in style and between episodes , and can work equally well for all the episodes because we use a per-episode dictionary. This approach can also be used with a global multi-episode dictionary, wherein we train on a few episodes, create the dictionary, and then use this single dictionary on other(new, unprocessed) episodes.

But there are many characters, so how do we determine their "correct" labels? This is where we use an auxiliary authoritative dictionary. That is, we obtain a few "canonical" images for each character. Then, we do the segmentation and clustering on these canonical images, to identify the actual label of the segments. (Note that in our case, segments are proxies for objects). Thus, we now have a "gold-standard" mini-dictionary, using which we identify the clusters in the actual complete dictionary (obtained earlier by kmeans above). This gives us association rules like:

- If label==2 Then segment is Cartman's Hat
- If label==2 or label==3 Then segment is Cartman's jacket
- If label==2 and neighbouring label==5 Then ...

3.8. Character recognition

Now that we have recognized the segments, we need to combine them into actual characters, since a character can be composed of multiple segments, we use a simple agglomerative rules. These rules are of the form:

- If Cartman's hat is detected , then cartman exists in the scene

Obviously, if multiple objects are detected, that increases the confidence of our prediction. We ensure two objects are detected before concluding that the character exists in the frame.

4. Implementation

The pipeline is implemented by using the `scikit-image` and `OpenCV` libraries in `Python`. For clustering, the `scikit-learn` toolkit was utilized.

We have used `numpy` arrays to represent the images and feature vectors. The feature vectors are extracted from each video only once, after which they are `pickle`'d and compressed and stored on disk. This allows the very quick experimentation with clustering and object recognition. The entire pipeline is parallelizable, although we have not dealt with a large enough volume of data/processing to warrant multi-threading.

Parameter	Value
Number of scenes extracted from one episode	300
Segments per image	10
Feature vector length	3

Table 1. Average numbers for the experiments

In hindsight, `Matlab` would have been a better choice as a platform for the implementation of our pipeline. However, the initial plan was to run several large scale experiments (in parallel) on a small computing cluster, for which Matlab would have been unsuitable. The lack of stability in the image processing libraries and the wildly inconsistant implementation between versions was a huge source of unintended bugs and frustration. We must note here that `scikit-image` seems like a far easier to use and better developed image processing library for Python than the more popular and well-known `OpenCV`

Not counting the various test functionality for the imaging libraries, our core image processing pipeline is implemented in a little over 1000 lines of Python.

5. Evaluation

The primary difficulty in evaluation was the lack of ground-truth data at every step of the pipeline. As a result, most of the evaluation was done manually by "eye-balling" the output at each step and then trying to understand the results. Table 1 shows the average values for some the parameters.

We have broken down the

5.1. Segmentation Analysis

The normalized-cuts and watershed segmentation do almost equally well, whereas QuickShift is very sensitive on the input parameters, and was discarded after some initial evaluation.

5.2. Clustering Evaluation

In order to know how good the clustering is, we have used two primary approaches:

1. Histogram analysis of the cluster sizes.
2. Silhoutte scores

5.2.1 Cluster size Histograms

We inspect the cluster size (frequency of occurrence points in a given cluster) for the various cluster configurations and algorithms. For example, in Figure 6, we show the frequency histogram for kmeans run with 10 clusters. *The color of the bars represent the actual cluster centre color.*. Similar histograms for other clustering parameters are shown in Figures **??**.

5.2.2 Silhoutte Scores

To evaluate how good the clustering really is, we used the *silhoutte scores*. Table 2 gives the average silhoutte scores and the standard deviation for the various parameters. We can see that higher number of clusters yields scores closer to 1. Also note that HSV colorspace gives slightly higher scores.

6. Future Work and Conclusion

Some more clustering algorithms More robust and usable shape detection Efficient way to encode neighbour segment information Multi-episode dictionaries Application and testing on other animated cartoons Collection of some ground-truth data by using crowd-sourcing (Amazon Mechanical Turk etc)

Algorithm	Average(RGB)	Std(RGB)	Average(HSV)	Std(HSV)
agglo-200	0.458646443848	0.269008683093	0.465670488369	0.260424450724
agglo-100	0.427810049095	0.264761260823	0.445758372781	0.274797481827
agglo-50	0.39460106313	0.283348220964	0.407061643698	0.275471127872
dbscan	-0.59659098259	0.459189910187	Negative Score	
K=400	0.484954993451	0.269845063645	**0.49477624071**	0.269644110049
K=300	0.470033927835	0.255694124258	0.483288003653	0.259444221453
K=250	0.468510843432	0.254082589179	0.465240510022	0.261075742016
K=200	0.454357446669	0.248596212307	0.4609727129	0.246014493153
K=150	0.451752086146	0.251992960509	0.459139455038	0.240711074362
K=100	0.429384756877	0.236945657405	0.456732628513	0.237824162046
K=50	0.415098853996	0.233513415301	0.430250332809	0.223607063326

Table 2. Averages and standard deviations of the silhoutte scores. HSV colorspace gives a higher score.

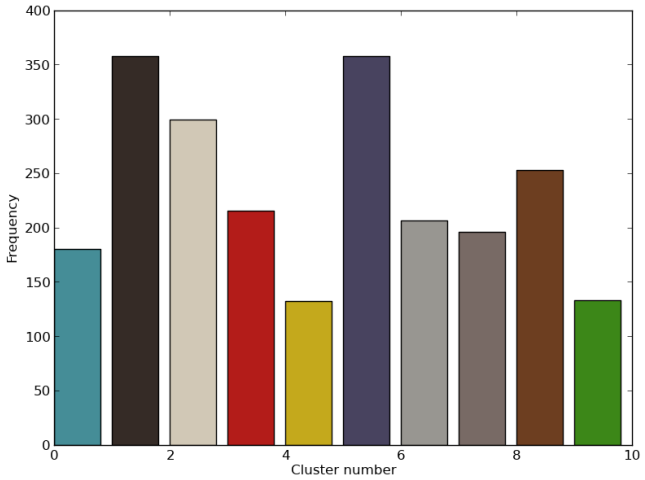

Figure 6. kmeans histogram with k=10. *The color of the bars represent the actual cluster centre color.*

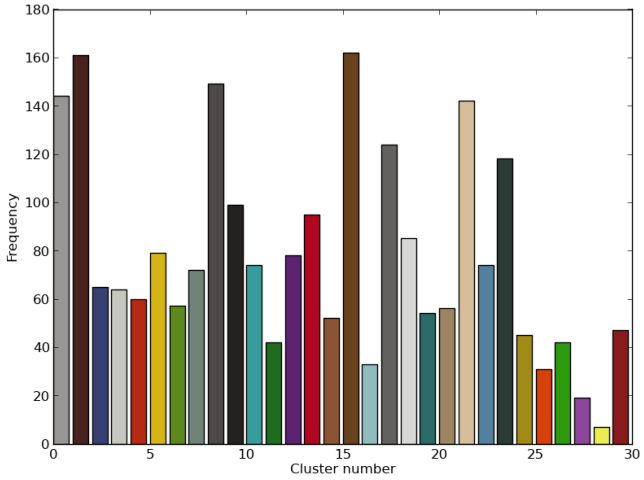

Figure 8. kmeans histogram with k=30. *The color of the bars represent the actual cluster centre color.*

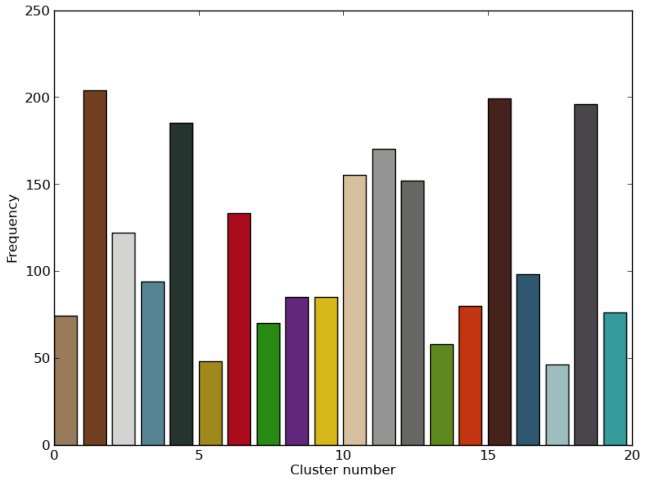

Figure 7. kmeans histogram with k=20. *The color of the bars represent the actual cluster centre color.*

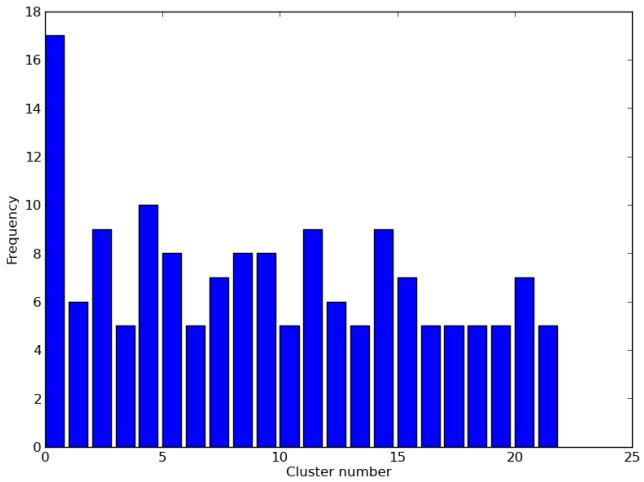

Figure 14. Cluster frequency histogram for the DBSCAN. Dbscan detects number of clusters automatically.

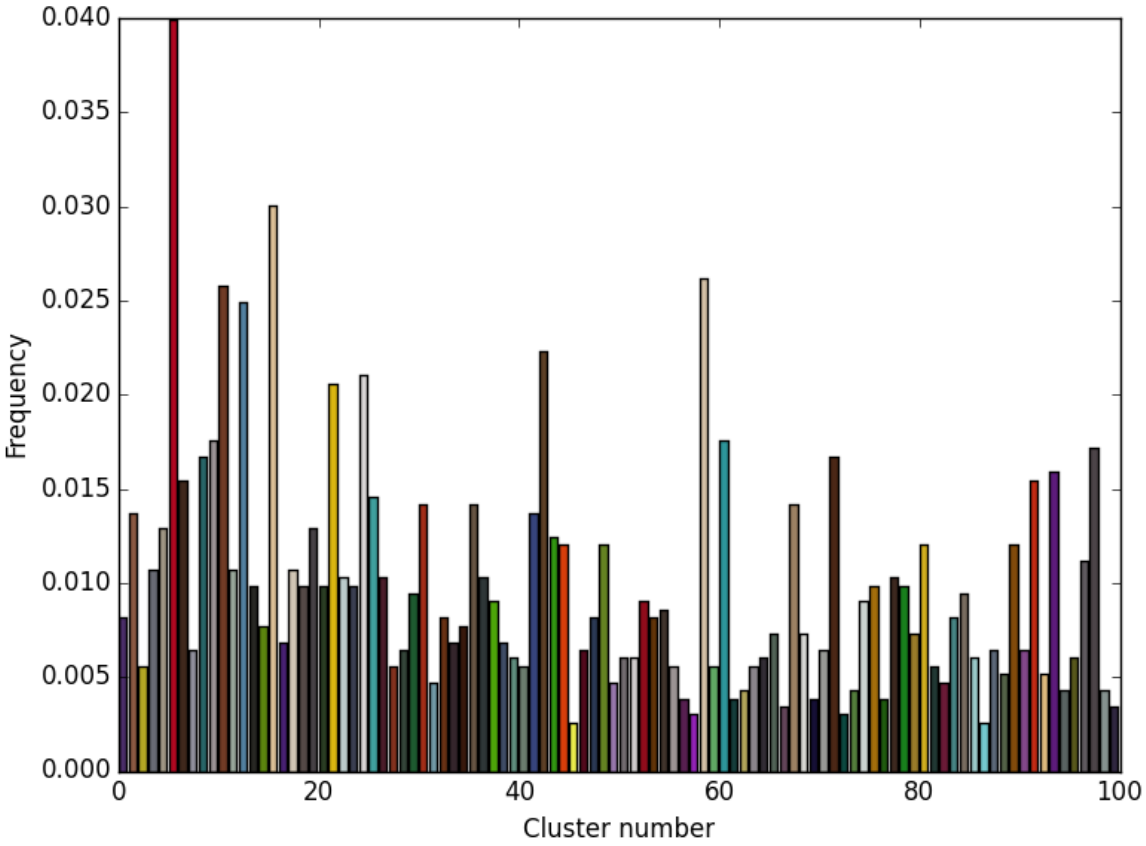

Figure 4. K-means histogram with the HSV feature vectors

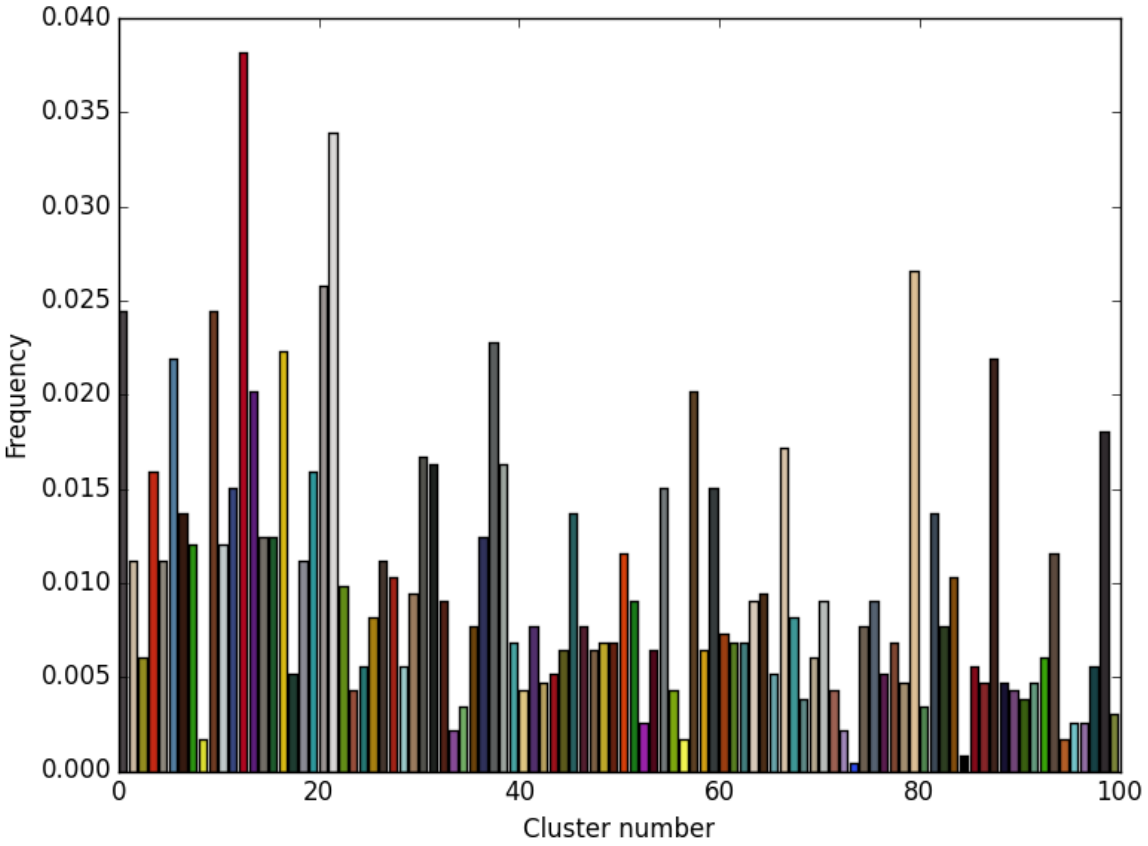

Figure 5. K-means histogram with the RGB feature vectors

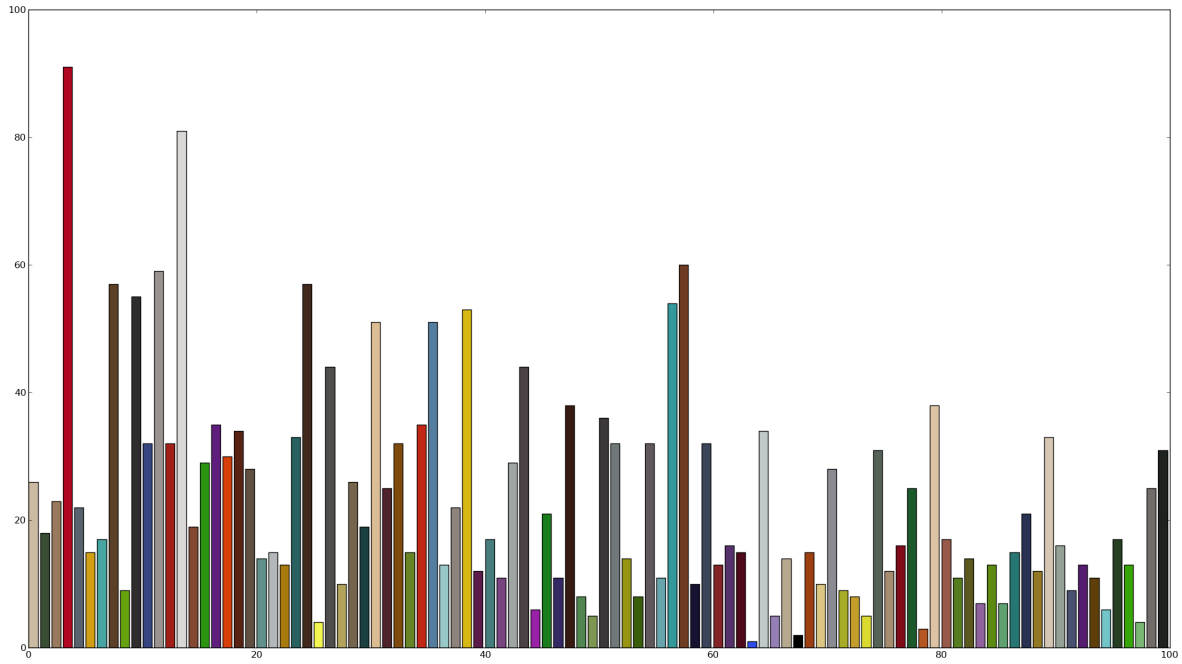

Figure 9. kmeans histogram with k=100. *The color of the bars represent the actual cluster centre color.*

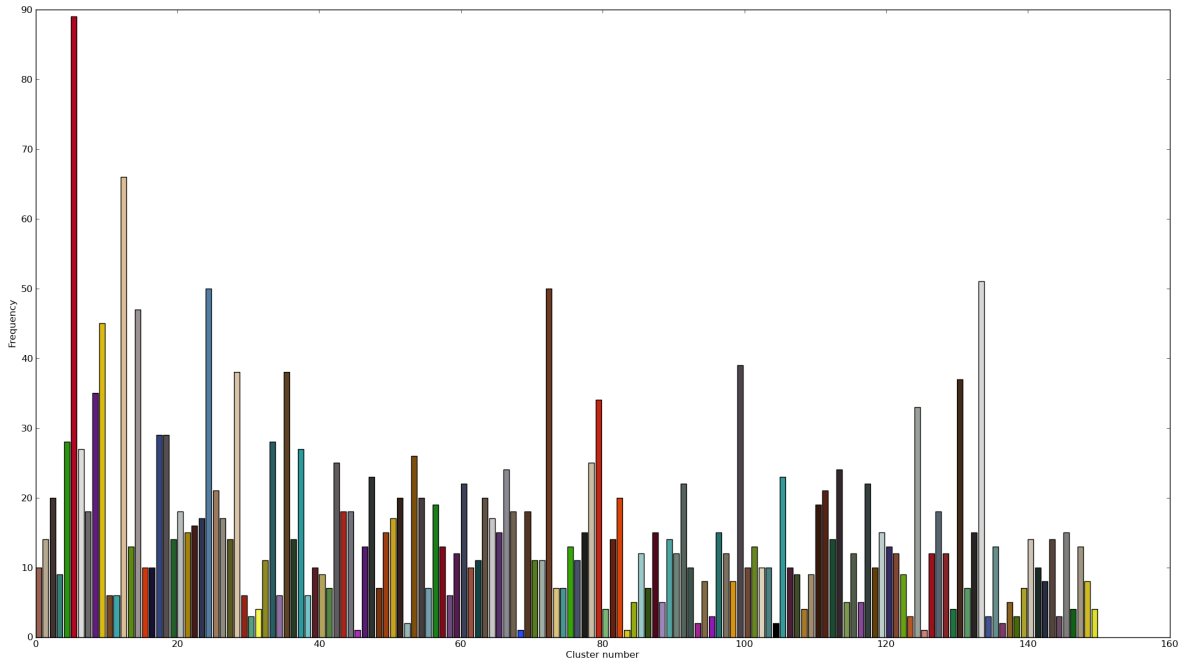

Figure 10. kmeans histogram with k=150. *The color of the bars represent the actual cluster centre color.*

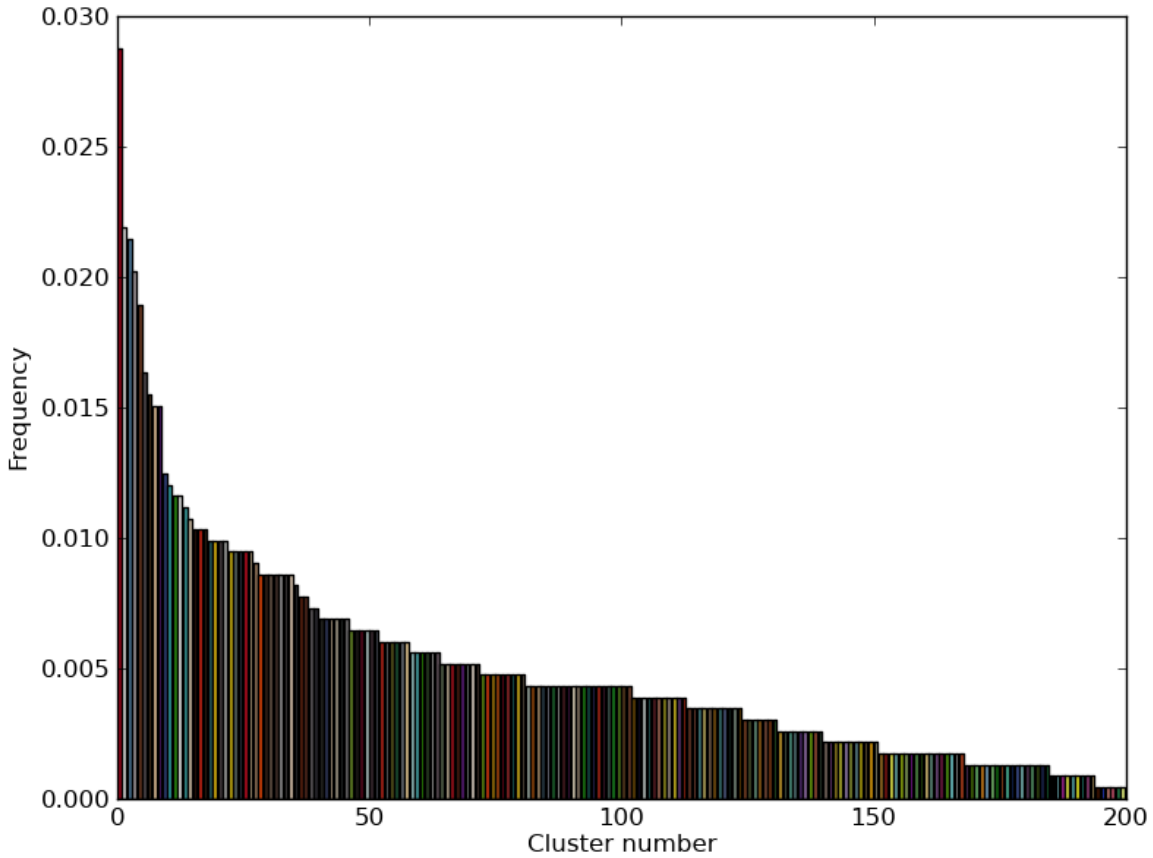

Figure 11. kmeans histogram with k=200. *The color of the bars represent the actual cluster centre color.*

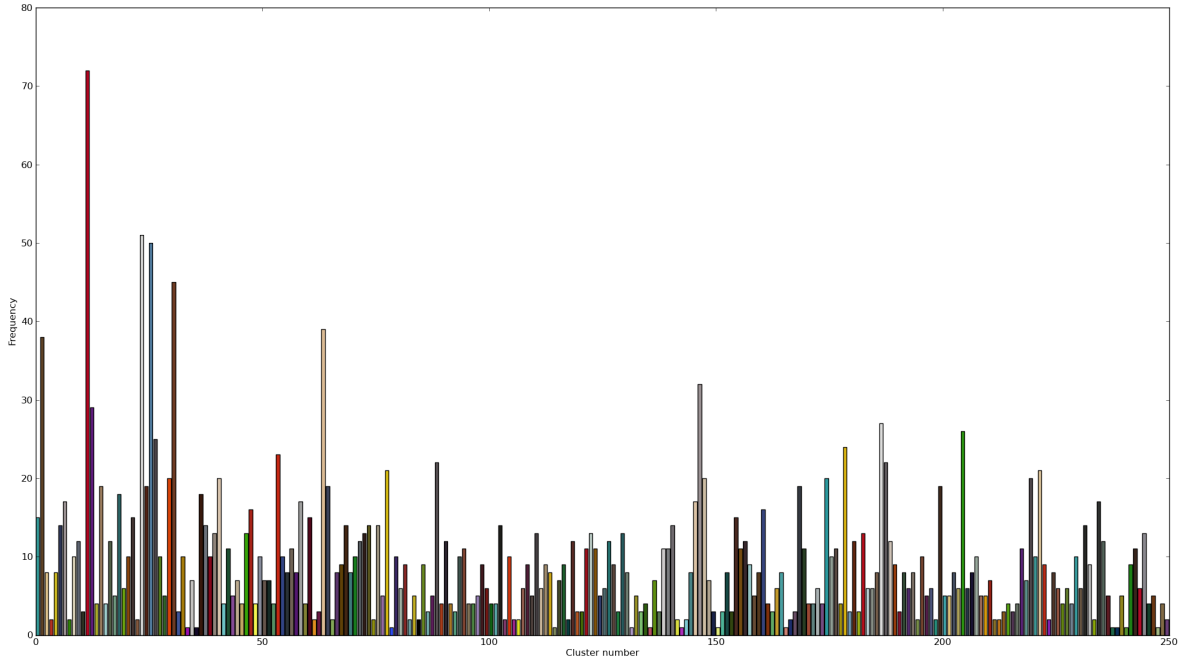

Figure 12. kmeans histogram with k=250. *The color of the bars represent the actual cluster centre color.*

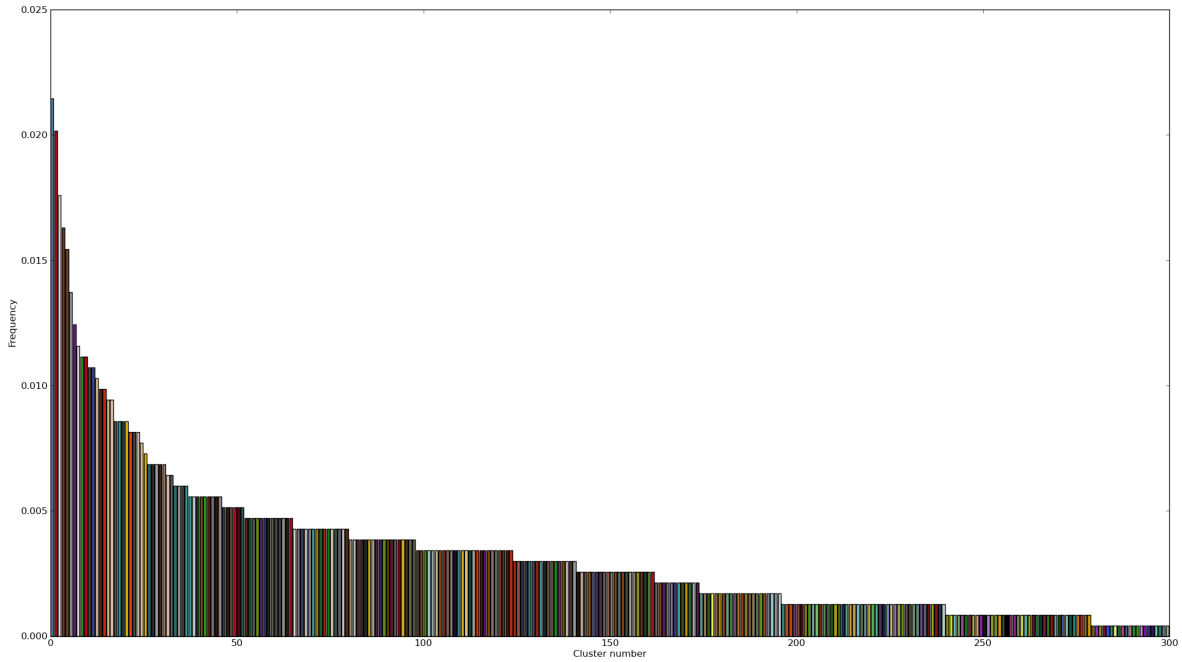

Figure 13. kmeans histogram with k=300. *The color of the bars represent the actual cluster centre color.*

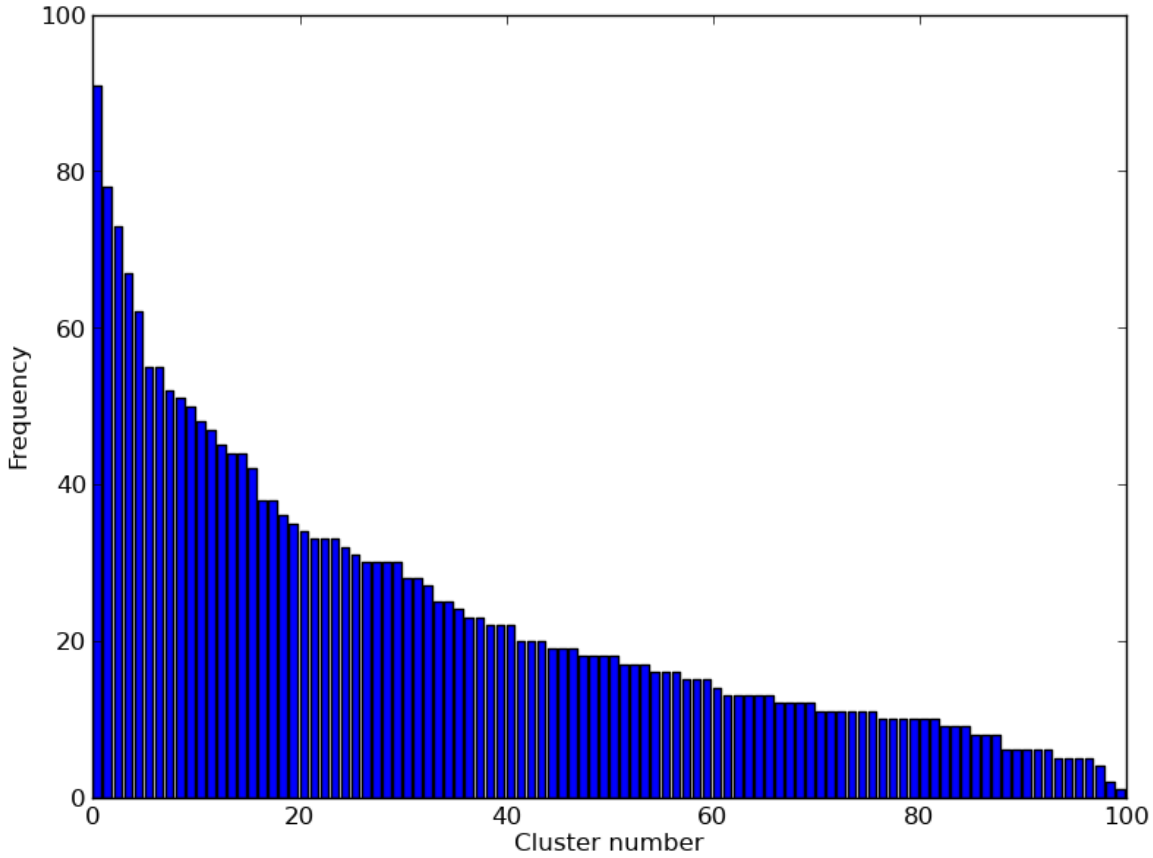

Figure 15. Agglomerative clustering with 200 clusters.

References

[1] O. Arandjelovic and A. Zisserman. Automatic face recognition for film character retrieval in feature-length films. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 860–867. IEEE, 2005. 1

[2] M. Everingham, J. Sivic, and A. Zisserman. Taking the bite out of automated naming of characters in tv video. *Image and Vision Computing*, 27(5):545–559, 2009. 1

[3] S. Satoh, Y. Nakamura, and T. Kanade. Name-it: Naming and detecting faces in news videos. *IEEE Multimedia*, 6(1):22–35, 1999. 1

[4] J. Sivic, M. Everingham, and A. Zisserman. Person spotting: video shot retrieval for face sets. In *Image and Video Retrieval*, pages 226–236. Springer, 2005. 1

[5] C. G. Snoek and M. Worring. Concept-based video retrieval. *Foundations and Trends in Information Retrieval*, 2(4):215–322, 2008. 1

[6] T. Zhang, Q. Han, A. A. Abd El-Latif, X. Bai, and X. Niu. 2-d cartoon character detection based on scalable-shape context and hough voting. *Information Technology Journal*, 12(12), 2013. 2