Tracking Viral Videos on YouTube: Tool Set Construction

David Hehir
u4402855

November 3, 2011

Abstract

This paper documents the implementation of a tool set calculates a set of near matches of an input set of images using the colour correlogram as a feature extractor. This tool set is a subset of tools that can be used to extract common short segments of video, or visual memes, from a set of videos. The software described in this paper is able to extract features and match similar images into sets of similar images, and sits within a large more complicated process for visual meme tracking. This paper serves as a design and performance testing document, illustrating that the two command line tools created work with an extraction rate of up to 94% precision and 87% recall, matching the performance of the base algorithm this tool set is based on.

1 Introduction

YouTube is currently the largest social video platform on the Internet, containing a huge number of videos with over 6.9 billion video views in the U.S alone (comScore 2011). Along with this, 48 hour of new content is uploaded every minute (Search Engine Watch 2011). With all this content existing in the one place, much of it is rehashes of existing content. Currently, there is no easy way to track the reuse of content easily within the YouTube ecosystem. This is for two main reasons, comparing two videos to check if they share nearly-matches of sections of contents is non-trivial to do reliably. The second reason is the huge amount of data to process. In order to run any checks, the data must be subsetted in some way. The only way to do this is currently using text searches that have the potential to miss large amounts of reuse, in particular if the video is untagged or tagged in a different language. Thus, how much of it that is reused is not known and is the basis of this project.

This project's aim is to investigate how can the reused be measured between videos without actually watching these videos. Work on this has already been undertaken to track small sections of remixed videos throughout YouTube (Xie, Natsev, Hill, Kender & Smith 2011). This paper discussed a methodology that has been tested on YouTube with success, but only investigates videos of current affairs, leaving a gap for further research into other genres. However the tools to research other genres of videos are

not readily available, leading to the main topic of this project.

In order to fit this project into a single semester, this project focused mainly on the construction of the tools to allow others to further research the propagation of remixed videos throughout YouTube. This paper focuses on the design of the tools (section 3.2.1) as well as the implementation of these into functioning tools (section 3.3 and 3.4). The tools were then benchmarked against the same test data set as in (Xie et al. 2011) to compare implementation results. A set of correct matches allow the calculation of precision and recall values, that allow tuning of the tools as well as evaluation of the overall performance and thus suitability for further research.

2 Prior Work

In Xie's paper (Xie et al. 2011), there is discussion on the need for a cultural unit for videos as there is for still based and text based mediums. Websites such as Know Your Meme (Cheezburger Inc 2011) index many know memes including video but have limited actual measures of a meme's spread other than video views, Google insight or observation.

Other sites offer ways of visualising the spread of particular news stories or search terms (Leskovec, Backstrom & Kleinberg 2011). However this is only for text and not for video.

This project focuses more on the problem of comparing sections of video, represented by key frames, together and finding videos with near-matches. The idea of Content-based image querying is not new, with several implementations being commercially used such as for image search engines (Bloore 2008) and Google Goggles (Google 2010)

3 Method

In this section, the methodology to create the tools to solve the problem described in 1 is described.

3.1 Overview

The basis for the solution to the problem specified is given in (Xie et al. 2011). This paper outlines a process of finding short segments of video that is replicated in many videos within YouTube. In this paper, the term *visual meme* is defined as "a short segment of video that is frequently remixed and reposted by more than one author" (Xie et al. 2011) . In order to trace the propagation of visual memes throughout Youtube a process shown in figure 1.

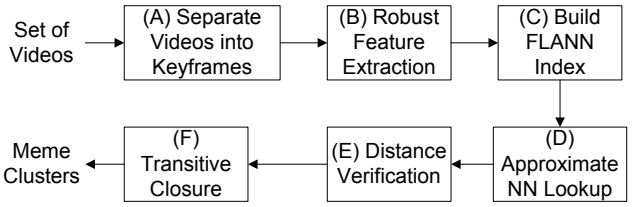

Figure 1: High Level Process of Visual Meme Matcher

(A) Separate Videos into Keyframes The first step within the process is to convert the set of videos into a set of still images, representing scenes within the video. This process is known

as keyframe extraction. For the sake of simplicity for this project, keyframe extract is not incorporated in the toolset construction. This was done with since other research is currently undergoing to complete this stage.

For this project, the starting point is the output of step (A), with a set of images the input for the rest of the toolset.

(B) Robust Feature Extraction The first step that the tool set needs to calculated is the feature vector for each image. A feature vector is essentially n dimensional vector containing information about the image in each entry. The colour correlogram is discussed in more detail in section 3.3. The output of this step is a set of feature vectors representing each image.

(C) Build FLANN Index Once the feature vectors have been generated, clusters of matches need to be generated. In this project, Fast Library for Approximate Nearest Neighbours (FLANN) is used to calculated which images are near neighbours to each other. This in turn allows grouping of near-match images together to form groups of *memes*.

FLANN performs an approximate nearest neighbour by clusting the images into trees, by storing images that are 'close' in a similar section of the tree. By storing the images in trees and only search a subset of the tree (e.g. only specific tree branches based on a query image), a faster but approximate nearest neighbours can be found. Further detail on how FLANN works is given in section 3.4.

The first step involving FLANN in this project is the indexing step. This step builds the tree based on a given input data set of feature vectors. The output of this step is a FLANN matcher that is ready to be used to calculate matches.

(D) Approximate NN Lookup Step (D) in the process is to calculate the matches. This step is where all the feature vectors produced in step (B) are matches producing a large number of sets of pairs.

(E) Distance Verification In Xie's paper, a match is only considered valid if the distance between the two feature vectors are less than a specific threshold. This threshold is defined as

$$T_q \triangleq \tau \frac{|f_q|_2}{|f_{max}|_2} \tag{1}$$

All matches that satisfy this requirement are kept. The result from this step is a subset of the pairs found in step (D)

(F) Transitive Closure The final step in the process is group the pairs together. This process groups the pairs together as shown in figure 2

This process is run over every pair, creating a set of clusters of similar images.

3.2 Tool set Design

3.2.1 Problem Decomposition

Figure 1 outlines the 6 main steps involved in calculating sets of meme clusters from YouTube videos. In this project, only a subset of this flow chart has been considered. The main assumption is that the videos have already been

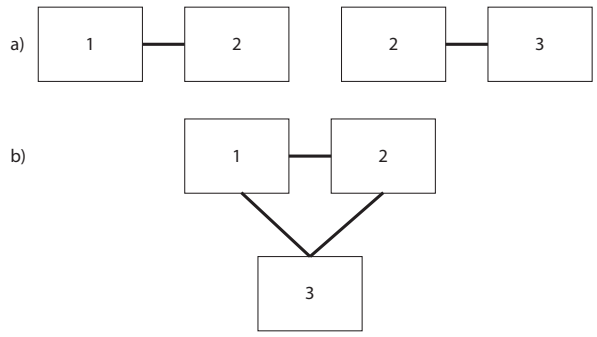

Figure 2: Transitive Closure Example

Figure 3: High level block diagram of developed tool set

extracted from YouTube and have been subsequently had their keyframes extracted.

From the highest level, the tool set ideally operates as a black box as illustrated in figure 3. From the diagram, all that is required is a set of images, along with some optional tuning parameters.

The assumption that keyframes are already extracted allows the overall process to be simplified and split into two main processing section; calculating the colour correlogram feature descriptors and matching based on these descriptors as illustrated in figure 4.

3.2.2 Design Criteria

In the initial design stages of the tool set, a set of criteria were written

- Memory Efficiency
- Modular
- Tunable

Memory Efficiency The given problem for this project deals with a large number of images that require processing. One of the biggest challenges with dealing with images on this scale is memory usage. After a certain number of images it is no longer possible to keep them in memory, thus any processing over the whole data set needs to be in stages keeping the minimal amount of information about the image in memory at an period of time.

Tunable In order to get the best performance out of the tool set, several tuning parameters for both the colour correlogram and feature matcher need to be exposed. The two obvious parameters that can be exposed are the search distance k for the colour correlogram and τ, the global threshold distance value.

Modular The design constraint that the tool set should modular was chosen since it is likely that this tool set may only be used in part. That is, a different feature descriptor extractor may be used to compare to the results of the colour correlogram. Alternatively, the colour correlogram may be used as a feature extractor for other uses than meme clusters. Therefore, the tool set was split into two separate executables,

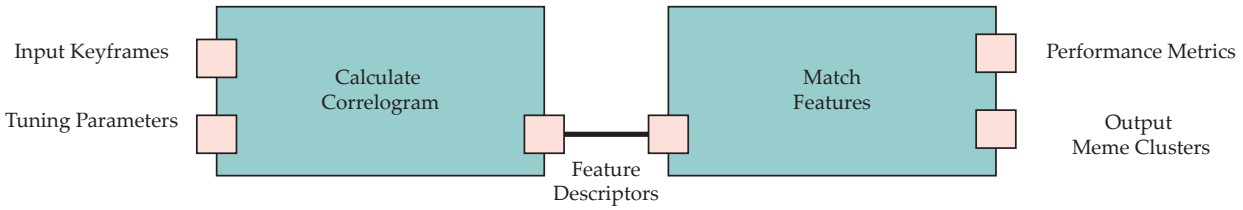

Figure 4: Proposed tool set decomposition

one for the colour correlogram and another for the feature matching.

This design choice allows faster tuning of the feature matcher. This is because the colour correlogram can be executed once only (for a set distance value), with the feature matcher being able to reuse the correlogram values.

3.3 Colour Correlogram

3.3.1 Background

The colour correlogram can be defined as follows, given a pixel of colour c_i in an image, the colour correlogram defines the probability that a pixel within a distance k is of colour c_j (Huang, Kumar, Mitra, Zhu & Zabih 1997). The autocorrelogram is defined as a correlogram with $c_i = c_j$. That is, the probability that a pixel within distance k shares the same colour as the reference pixel. This can mathematically be expressed as

$$\gamma^{(k)}_{c_i,c_j} \triangleq Pr_{p_1 \in I_{c_i}, p_2 \in I_{c_j}} \left[p_2 \in I_{c_j} \middle| |p_1 - p_2| = k \right] \tag{2}$$

With the autocorrelogram defined as

$$\alpha^{(k)}_c \triangleq \gamma^{(k)}_{c,c} \tag{3}$$

Thus, the feature vector calculated from the colour correlogram is a vector containing a probability value $\alpha^{(k)}_c$ at index c. Therefore, the feature vector with the same dimension as the number of quantization bins.

Figure 5, illustrates a small image section consisting of 9 pixels and two colours, c_1 and c_2. When examining the middle pixel with $k = 1$, the autocorrelogram will be equal to $\alpha^{(1)}_{c_1} = \frac{1}{4}$.

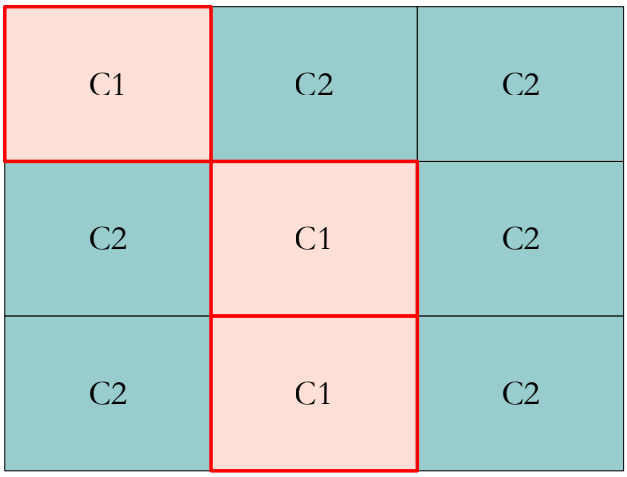

Figure 5: Small Subset of Pixels with representative colour values

The autocorrelogram was selected as the feature detector since it can be computed quickly

and can tolerate changes in zoom, viewing position as well as being rotation invariant (by definition). These properties make the color correlogram well suited for use in visual meme matching since it is fairly common for small transforms to be undertaken by the remix author (e.g. cropping or resizing).

To illustrate the autocorrelogram, figure 6 shows three images with their correlogram underneath. The left and centre images are considered matches, while the image on the right is not a match. The correlograms plotted underneath illustrate this.

The fact that the correlogram is computationally simple with a runtime of $O(n^2k)$ (Huang et al. 1997) make it suitable for use with large data sets, such as the ones used in this project and (Xie et al. 2011).

Cross Pattern Colour Correlogram A modification of the original colour correlogram feature vector is to calculate a cross pattern colour correlogram. The idea is the only pixels that are considered are those within a cross mask as shown in figure 7.

The reason for doing so is to remove any boarders or logos that may be present (e.g. news feeds down the bottom), that are common across many different images. The colour correlogram used in the reference paper (Xie et al. 2011) utilised the cross colour correlogram. However, for simplicity, the regular correlogram was used.

3.3.2 Design

At the core of this tool is the use of OpenCV (Bradski 2000). This library handles the image loading and simple manipulations involved with this step. This library was chosen since . Although there exist bindings for C,C++ and python, the C bindings were chosen due to the language familiarity and potential performance gains over a python implementation.

The process the program will go through to calculate the correlogram is illustrated in the flow chart in figure 8

The process outlined in the figure allows much of the work to be delegated to the OpenCV or other libraries. OpenCV provides facilities to read an image (step A) into a form such that direct access to the pixel values are possible.

Quantisation (step B) is achieved by reading each individual pixel value, based on its value setting it to a new value inside a quantisation bin. The following formulas implement this:

$$bin = \lfloor \frac{pixel\ value}{interval} \rfloor$$
$$p_{new\ value} = interval/2 + interval \times bin$$

Where the value of *interval* is defined as $interval = Max_{channel}/N_{channel\ bins}$. This definition allows different number of quantisation bins in each channel. In this project 162 bin quantisation is used, with 18 hue channels, 3 saturation channels and 3 value channels.

In Huang et. al. (Huang et al. 1997) an algorithm is proposed for calculating the correlogram in $O(n^2d)$ for small values of search radius d. The algorithm uses the following formula

$$\Gamma^{(k)}_{c_i,c_j}(I) \triangleq \left| \{p_1 \in I_{c_1}, p_2 \in I_{c_j}\} | |p_1 - p_2 = k| \right| \tag{4}$$

$$\gamma_{c_i,c_j} = \frac{\Gamma^{(k)}_{c_i,c_j}}{h_{c_i}(I)8k} \tag{5}$$

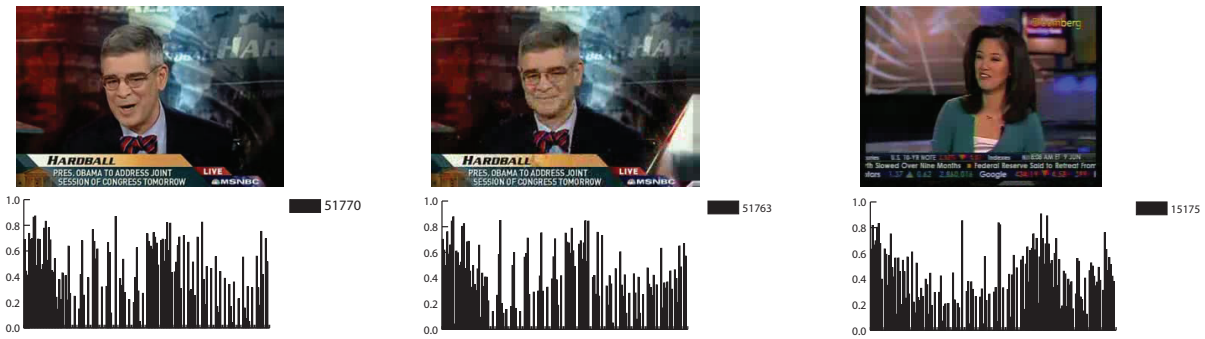

Figure 6: Three images and their respective colour correlograms

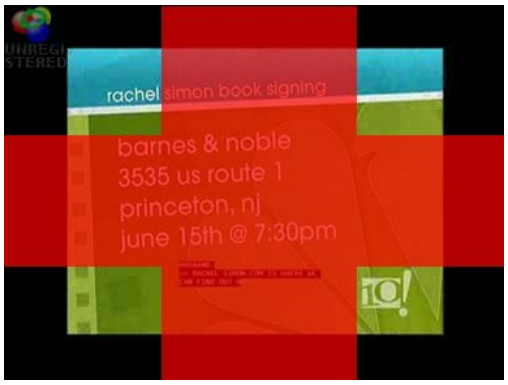

Figure 7: Area taken into account for correlogram calculation

Where $h_{c_i}(I)$ is the image histogram. The function for $\Gamma^{(k)}_{c_i,c_j}$ is define as follows for the $O(n^2d)$ solution

$$\Gamma^{(k)}_{c_i,c_j} = \sum_{(x,y)\in I_{c_i}} \Big(\lambda^{c_j,h}_{(x-k,y+k)}(2k) + \lambda^{c_j,h}_{(x-k,y-k)}(2k) + \lambda^{c_j,h}_{(x-k,y-k+1)}(2k-2)\lambda^{c_j,h}_{(x+k,y-k+1)}(2k-2) \Big)$$

Where $\lambda^{c,h}_{(x,y)}(k)$ is defined as:

$$\lambda^{c,h}_{(x,y)}(k) = \lambda^{c,h}_{(x,y)}(k-1) + \lambda^{c,h}_{(x+k,y)}(0)$$

with an initial condition

$$\lambda^{c,h}_{(x,y)}(0) = \begin{cases} 1 & \text{if } p \in I_c \\ 0 & \text{otherwise} \end{cases}$$

The calculate correlogram stage (step C) is implemented using the $O(n^2d)$ method explained above. This process is repeated for each of the input images, with each value of c_i representing a quantisation bin, thus giving a feature vector with 162 bins in this project. It should be noted that as each bin is calculated, the max feature vector is updated if the result is larger than the previous stored value.

The final step in this process is to serialize the feature vectors (step D). In this case, the feature vectors were chosen to be serialized in JSON since it is human readable (i.e. nicer to debug) and lightweight than XML, with the potential to be significantly smaller files than XML.

Figure 8: Process for calculating the correlogram for a single image

3.3.3 Implementation

The Colour Correlogram tool was the first of the two tools to be constructed. It was primarily written in C since this was a language option that has bindings to OpenCV. OpenCV was chosen as the image library since it is not only free but has the simple functions to read images and calculate histograms.

The JSON serialization library, Jansson (Jansson 2011), was chosen since it has no external library dependencies as well as being simple to use.

With OpenCV and Jansson, the tool set was relatively straightforward in implementation of the Colour Correlogram feature extractor.

Throughout the tool, there are three main C structs that are used to store the set of feature vectors shown below.

```
typedef struct Correlogram
{
    char* fileName;
    double* FeatureVector;
} Correlogram;

typedef struct CorrelogramArray
{
    Correlogram** array;
    int elements;
} CorrelogramArray;

typedef struct CorrelogramCollection
{
    CorrelogramArray* correlograms;
    double* MaxFeatureVector;
    int NumBins;
    int searchDistance;
} CorrelogramCollection;
```

The proposed design has met the design criteria specified in section 3.2.2. The program has been written with memory use in mind. For example, the program will only have on image in memory at any point in time, this is essential since images are large and by keeping the image in memory is unnecessary and reduces scalability of the program.

The program has satisfied the tunable design criteria s

3.4 Fast Library for Approximate Nearest Neighbours

3.4.1 Background

Approximate nearest neighbour matching is a method used to approximately find matches of points (in our case vectors) in high dimensional space. A linear search typically is not used since this takes too long to execute with large datasets of high dimensionality; the time complexity of a linear search is $O(n^2d)$, where n is the number of images and d is the dimensionality.

As mentioned previously, approximate nearest neighbour solutions reduce this complexity, allowing the matching process to be done in less than $O(n^2d)$ time. The FLANN library which uses randomised k-d trees, can be build in $O(nlog^2n)$ and queried in $O(n^{1-1/k} + m)$ (Cormen, Leiserson, Rivest & Stein 2009*b*). The FLANN library has been found to not only be faster than linear search but also a magnitude faster than other approximate nearest neighbour match algorithms (Muja & Lowe 2009), making it ideal for the purposes needed in this project.

The actual FLANN algorithm is not discussed in depth here and more information can be found in Muja et. al. (Muja & Lowe 2009). Instead, this project just interacts with libraries that implement the algorithm. There are two main libraries that implement FLANN, OpenCV (Bradski 2000) and the Point Cloud Library (Rusu & Cousins 2011).

3.4.2 Design

The Feature Matcher design is far more simple than the Colour Correlogram design in section 3.3 above. This is because the majority of the calculation is done within external libraries, reducing this second executable to just interacting with the libraries.

The overall process is illustrated in figure 9. The first major section in the feature matcher tool is to read in the file (step A) and deserialize this file into feature vectors (step B). The deserialization process is tightly bound to the serialization process of the Colour Correlogram (section 3.3). The order of extracting the objects from the JSON file needs to be done in the same order in which they are packed by the Colour Correlogram process.

Step C is where the bulk of the work is done in this tool, where each individual image is matched with a set of various other images within a specified search radius, given by equation 1. This is done using a FLANN based matcher that matches a query image, img_q, to other images within the image database that have a distance to img_q that is less than or equal to the threshold value, calculated from 1.

Each iteration of the matching process produces a set, containing image filenames that are considered matches. With the set of match pairs calculated, the transitive closure step needs to executed. It was decided for simplicity to do this within the same application as the feature matching so that the amount of intermediate data serialized to disk is minimised.

Thus, after matching has occurred for each query image (i.e. all the images in the database), the sets of images need to be merged. The merging step is done such that if image $A \in S_a$ and $A \in S_b$ then the all images $I \in S_a \cup S_b$ are considered a match. The one disadvantage to this method of joining is that the worst case distance between two images is nT_q, where n is the number of images in $S_a \cup S_b$, which may lead to some false positives. However, this method of joining has the advantage of being able to use the disjoint set data structure with each match results being a set off the parent represented by its query image. This allows the merging operations to only take $O(m\alpha(n))$ where $\alpha(n)$ is the inverse Ackermann function and m is the number of disjoint set operations (join, find, etc) (Cormen, Leiserson, Rivest & Stein 2009*a*).

The matches are then serialized using a simple JSON file. This allows the results of the

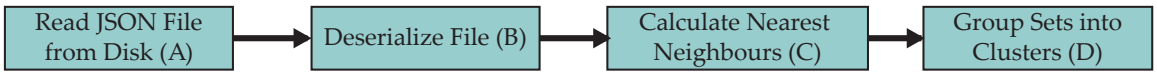

Figure 9: Feature Matcher process

matching to be checked at a later date.

3.4.3 Implementation

The implementation of the feature matcher was required to be written in C++ due to interfaces of various descriptor matchers (including the FLANN based matcher) only exist in the C++ OpenCV bindings and not C.

OpenCV was chosen over the Point Cloud Library since the colour correlogram tool was written using OpenCV, thus not adding in a new system dependency. The Point Cloud Library is focused more on robotics while OpenCV has more an image processing focus, giving extra helper functions making this tool easier to implement.

The initial step of deserialization was able to be completed again with Jansson. With the feature vectors deserialized, the vectors needed to be converted to a format that is suitable for the OpenCV FlannBasedMatcher object. This meant that the feature vectors (originally double[]) needed to be converted to an OpenCV specific format, cv::Mat.

This deserialization step was found to take a significant amount when each feature vector was serialized in a single file (approximately 10 minutes) with the test data set discussed in 4. A modification to the Colour Correlogram tool was made such that the feature vector was serialized into a single file which reduced the time taken 20 fold to approximately 30 seconds on the same data set.

Using the FlannBased matcher gives three different methods of querying the dataset; *match*, *knnMatch* and *radiusMatch*. The first, *match*, returns only a single but best result and is thus inapplicable for the tool set needs. *knnMatch* returns the best k matches. This method could be used by selecting the best 50 results then culling the results based on their distance to the query vector as in Xie (Xie et al. 2011). This method would be preferable if there are too many result within the search radius T_q, thus limiting to a maximum of 50 results. However for simplicity, the *radiusMatch* method was used since this only returns results that are within the search distance away from the query image.

At this stage a disjoint set is built from the results of the FlannBasedMatcher.radiusMatch method call. C++11 does have a disjoint set but C++11 is not in widespread use. Thus, the boost implementation of disjoint set was used (Siek, Lee & Lumsdaine 2001). The boost implementation uses both path compression and union by rank so that the $O(m\alpha(n))$ is achieved.

4 Results

Once the program was constructed as described in section 3.2.1, the tools needed to be verified that they work as expected.

4.1 Methodology

The testing methodology was to use the same tuning dataset as in Xie (Xie et al. 2011) and compare the recall and precision results. Along with the test dataset a set of true/false pairs were given to calculate the recall and precision values.

A python script was used to piece together the two tools and batch run the tools with different parameters. The python script also does the calculation of precision and recall values as follows:

```
for pair in Pairs:
  file1 = FindPairIdInFile(pair(0))
  file2 = FindPairIdInFile(pair(1))
    if pair is a true pair:
      if file1 = file2:
        numerator++
                recallDenominator++
                precisionDenominator++
          else
                recallDenominator++
        if pair is a false pair:
          if file1 = file2:
            precisionDenominator++
end loop
```

Recall is then calculated as

$$recall = \frac{numerator}{recallDenominator}$$

Precision is given as

$$precision = \frac{numerator}{precisionDenominator}$$

4.2 Observed Results

Using the methodology presented above, the recall and precision values were found for a varying number of τ and search radius r values.

τ	Recall	Precision	$N_{matches}$
1	0.744	0.995	1416
2	0.739	0.994	1595
3	0.553	0.982	2394
4	0.818	0.970	4172
5	0.875	0.941	6289
6	0.915	0.511	4504
7	0.978	0.358	1201
8	0.991	0.342	282
9	0.996	0.339	84
10	1.0	0.339	41
11	1.0	0.339	23
12	1.0	0.339	15
13	1.0	0.339	10
14	1.0	0.339	3
15	1.0	0.339	3

Table 1: Results for $r = 1$

From table 1 similar results as compared to the reference material was found. The operating point chosen in (Xie et al. 2011) was $\tau = 11.5$ with recall value $R = 80.1\%$ and precision value $P = 98.2\%$. The closest value occurs somewhere between 4 and 5. The tool was run again using a finer interval of τ values (0.1) giving the results shown in table 2

From the tables, there is no direct match for the operating point found in the reference material. Thus, the maximum F_1 score will be taken ($F_1 = \frac{2RP}{R+P}$). The maximum value is found to be at $\tau = 5$

Compared to (Xie et al. 2011), the results found using the tool build in this project are comparable but ultimately are less precise than the equivalent values found in the reference paper. However, it should be noted that this tool

τ	Recall	Precision
4.1	0.826	0.970
4.2	0.826	0.965
4.3	0.828	0.963
4.4	0.821	0.960
4.5	0.831	0.960
4.6	0.841	0.954
4.7	0.851	0.950
4.8	0.850	0.946
4.9	0.859	0.944
5.1	0.879	0.93
5.2	0.878	0.925
5.3	0.875	0.905
5.4	0.875	0.881
5.5	0.884	0.804
5.6	0.896	0.76
5.7	0.909	0.708
5.8	0.909	0.651
5.9	0.911	0.54

Table 2: Results for $r = 1$ continued

set produces slightly higher recall values. One explanation for this is that this tool set makes larger groups than the equivalent in the reference paper. This is likely to result in higher recall values and lower precision as found here.

The reason for this can be attributed to differences in the preprocessing and implementation of the colour correlogram. For the results found, no preprocessing was undertaken. However, in the reference results, the extra preprocessing steps included the removal of blank frames, removal of boarders, normalization of the aspect ratio, de-noising and contrast and gamma correction. This would mean two different feature vectors since they are now two different images.

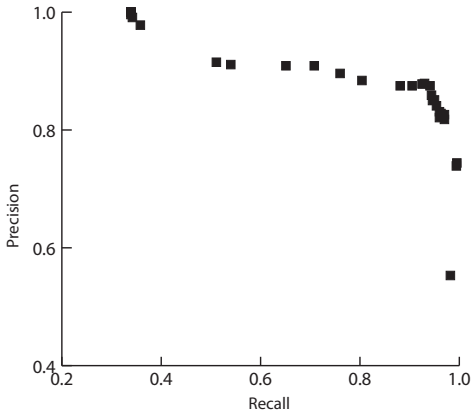

Figure 10: Plot of Recall against Precision for $r = 1$

This may also explain the differences between operating points.

The tool set used in the reference set utilised the cross-correlogram as explained in 3.3. This again changes the feature vector resulting in different matches when compared to those in found above.

Other than τ it is also possible to tune the colour correlogram stage with a larger search radius.

When the same process was run over $r = 2$, the precision and recall values where found to be as shown in table 3. The results found using this new radius value are similar though with a different operating point. When the values are plotted over figure **??**, the values form a similar pattern as shown in figure 11.

4.3 Performance

Performance was not a major criteria for the design of this tool set. However, since it may

τ	Recall	Precision
1	0.744	0.995
2	0.739	0.994
3	0.738	0.994
4	0.582	0.987
5	0.624	0.977
6	0.822	0.971
7	0.853	0.953
8	0.864	0.924
9	0.914	0.766
10	0.940	0.450
11	0.978	0.361
12	0.984	0.344
13	0.996	0.342
14	1.0	0.339
15	1.0	0.339

Table 3: Results for $r = 2$

be used for potentially very large dataset, some idea of timing is required. Table 4 shows the wall time taken to execute the individual tools at different tuning values. The tests were run on a laptop with an i5-2410M 2.30GHz with 4GB of RAM.

Further values of τ are not shown since the tool only took 35 seconds regardless of the value of τ or r.

From the timing results and recall and precision results shown above, varying the search radius ,r, provides no benefit.

5 Conclusion

The original aim of this project was to not only develop tools that allow feature extraction and matching of key frames in videos but to extend

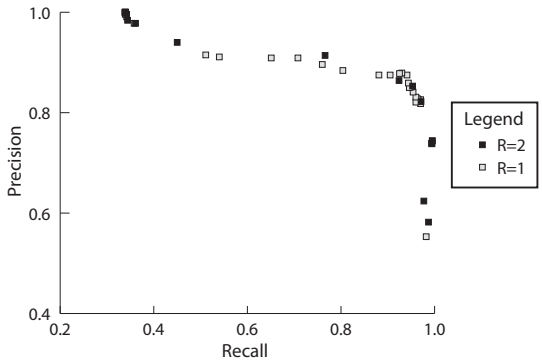

Figure 11: Plot of Recall against Precision for $r = 1$ and $r = 2$

Colour Correlogram	Time (s)
$r = 1$	3618
$r = 2$	7964
$r = 3$	14046
Feature Matcher	
$\tau = 1$	35
$\tau = 2$	35
$\tau = 3$	35

Table 4: Timing of individual tools

the investigation that was initially started in (Xie et al. 2011) into finding and observing how visual memes propagate through the YouTube ecosystem. This goal was not achieve for various reason but mainly due to time constraints. However, this project report has illustrated that the tools required to do so have been constructed and require only extra preprocessing of images or implementing the cross colour correlogram to begin the original projects aims investigation.

5.1 Further Work

From here, there are several obvious extensions to the work outlined in this report. One is to rewrite the colour correlogram such that it extends the OpenCV abstract class cv::FeatureDetector. This would allow easier interaction with the rest of the OpenCV library, making the first tool that calculates the colour correlogram much simpler.

A second obvious extension of this work is to begin the investigation into visual memes on YouTube. In (Xie et al. 2011), only current affair videos have been investigated as visual memes. However, many other genres of videos exist on YouTube that would provide interesting results. For example, music videos and gaming videos could be used to find out another measure of popularity, rather than just video views. This tracking could be used in conjunction with traditional measures (views) to document the popularity of video memes in a similar way to that used by text and other mediums.

6 Code

The code has been kept on github available at `https://github.com/davidhehir/Comp3750Project`. The code, installation guide and user guide can all be downloaded from this repository.

Acronyms

ANN Approximate Nearest Neighbour

FLANN Fast Library for Approximate Nearest Neighbours

References

Bloore, P. (2008), 'Tineye'. Accessed 20/10/2011. Available Online `www.tineye.com`.

Bradski, G. (2000), 'The OpenCV Library', *Dr. Dobb's Journal of Software Tools* .

Cheezburger Inc (2011), 'Know your meme'. Accessed 20/10/2011 `http://knowyourmeme.com`.

comScore (2011), 'comscore releases august 2011 u.s online video rankings'. Accessed 20/10/2011. Available Online `http://www.comscore.com/Press_Events/Press_Releases/2011/9/comScore_Releases_August_2011_U.S._Online_Video_Rankings`.

Cormen, T., Leiserson, C., Rivest, R. & Stein, C. (2009*a*), *Introduction to Algorithms*, MIT Press, chapter 21.4 Analysis of union by rank with path compression.

Cormen, T., Leiserson, C., Rivest, R. & Stein, C. (2009*b*), *Introduction to alogrithms*, MIT Press, chapter 10.

Google (2010), 'Google goggles'. Accessed 20/10/2011. Available Online `http://www.google.com/mobile/goggles/`.

Huang, J., Kumar, S. R., Mitra, M., Zhu, W.-J. & Zabih, R. (1997), Image indexing using color correlograms, *in* 'Proceedings of the 1997 Conference on Computer Vision and Pattern Recognition (CVPR '97)', CVPR '97, IEEE Computer Society, Washington, DC, USA, pp. 762–.

Jansson (2011), 'Jansson'. Accessed 18/10/2011 http://www.digip.org/jansson/.

Leskovec, J., Backstrom, L. & Kleinberg, J. (2011), 'Meme-tracking and the Dynamics of the News Cycle'. Accessed 20/10/2011 Avaliable Online http://memetracker.org/quotes-kdd09.pdf.

Muja, M. & Lowe, D. G. (2009), Fast approximate nearest neighbors with automatic algorithm configuration, *in* 'International Conference on Computer Vision Theory and Application VISSAPP'09)', INSTICC Press, pp. 331–340.

Rusu, R. B. & Cousins, S. (2011), 3d is here: Point cloud library (pcl), *in* 'International Conference on Robotics and Automation', Shanghai, China.

Search Engine Watch (2011), 'New YouTube Statistic: 48 hours of video uploaded per minute'. Accessed 20/10/2011. Available Online http://searchenginewatch.com/article/2073962/New-YouTube-Statistics-48-Hours-of-Video-Uploaded-Per-Minute-3-Billion-Views-Per-Day.

Siek, J. G., Lee, L.-Q. & Lumsdaine, A. (2001), *The Boost Graph Library: User Guide and Reference Manual (C++ In-Depth Series)*, Addison-Wesley Professional.

Xie, L., Natsev, A., Hill, M., Kender, J. & Smith, J. R. (2011), 'Visual Memes in Social Media: Tracking Real-world News in YouTube Videos', *ACM Multimedia* . To Appear.