Homework 2: Multimedia Information Retrieval

Deadline February 22th 2022

Deniz Sen

Unbuntu 64-bit 20.04.1

1 Feature detection and matching

Feature detection and matching plays a fundamental part in many computer vision problems, including object or scene recognition, 3D stereo reconstruction from multiple images, stereo correspondence and motion tracking [1]. To exemplify the importance of feature detection and matching, let us consider a camera application, which tries to seamlessly stitch together two images of a mountain as shown in Figure 1 to create a larger panorama image. To do this, what features or regions within the image should the algorithm be sensitive to?

Keypoint features or *interest points*, which describe the appearance of local, salient regions in an image are of particular interest. That is the algorithm should be sensitive to unique key points or locations with the most distinctive features (i.e. mountain peaks, building corners or other interestingly shaped patches), on each image. However, finding such salient regions is not an easy task as can be seen from the examples in Figure 1, where three arbitrary patches have encased in red with their corresponding ground truth location in the second image. Magnifications of each patch can be seen below the main images.

The first magnified image patch, which correspondents to the image patches from the sky illustrate the challenge of detecting keypoint features. The texture and colors are homogeneous at both locations. Comparing the both magnified patches, we also notice that the illumination and color changes from, making it harder to find corresponding feature locations. While the second patch, which was extracted from the smaller mountain, provides more information about the orientation along the curve, similar regions may exist at different location of the image. Therefore, the region is not unique defined, providing another challenge. The last patch, which corresponds to the corner of the mountain top however, is a salient region. It is uniquely defined. Compared to the other two examples, illumination, color are consistent and the slight change in perspective does not make it harder to to find the corresponding location of the mountain peak in the second image.

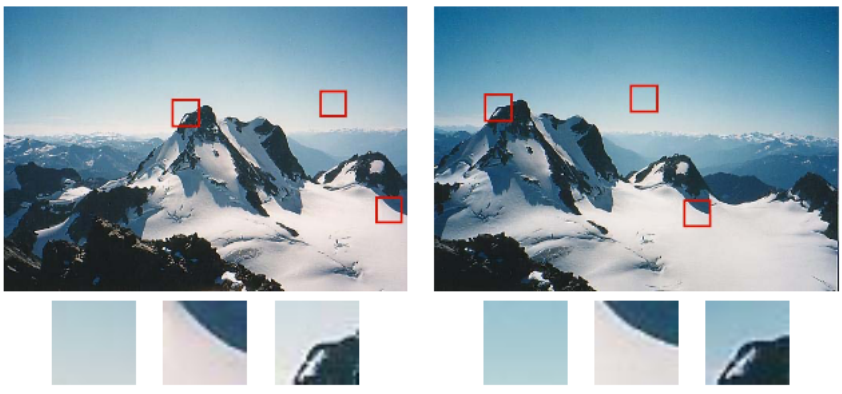


Figure 1: Two images pairs with example regions magnified below to illustrate the difficulty of salient feature detection.

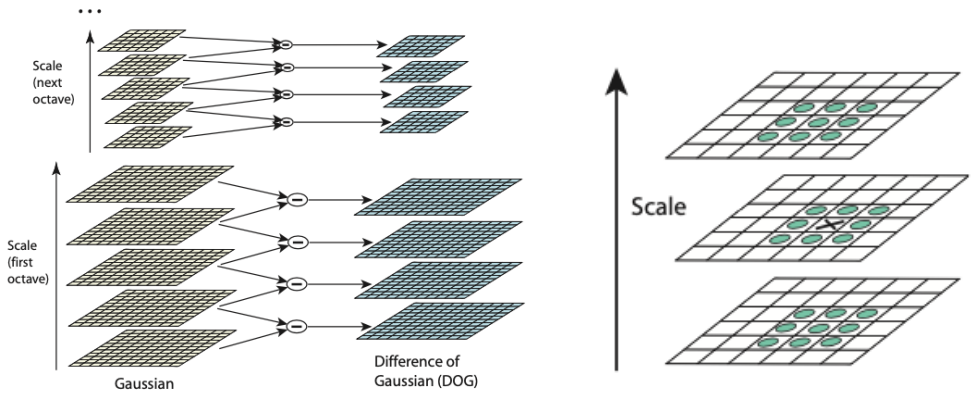
.

As such, features should be invariant to perspective effects and illumination, we wish to find the same features as in the original image, so that they can be extracted into a feature vector, that is distinct and matches to the correct location in the other image.

1.1 Scale Invariant Feature Transform

Scale Invariant Feature Transform (SIFT) [2] is a popular method to perform feature detection and matching. It is not only invariant to rotation or perspective effects, but also to a scale. This is achieved by constructing a scale space by iterative filtering the image with a Gaussian filter. We start with the original image and filter it with a Gaussian filter multiple times, resulting in the filters getting more and more blurred, scaling down the image creating a next octave (see Figure 2a). Once we have a scale pyramid consisting of the first octave at the original image and

the subsequent next octaves ad a down-scaled image, we compute the difference between adjacent Gaussian (DoG). An interesting property of DoG is that if we detect extrema or interest points, the interest points are blobs in the image. As such SIFT is a blob detector at different scales, as we apply a DoG at each octave [3]. We then find these extrema, by looking at the DoG scale space and finding maximum and minimum points in that space within the entire volume. That is we determine exterma by examining neighboring scales as well as shown in Figure 2b.



(a) Visual representation of the scale pyramid, consisting of scale space octaves. For each octave the image is repeatedly convolved with Gaussian filters to produce the set of scale space images shown on the left. Adjacent Gaussian images are subtracted to produce the difference of Gaussian (DoG) images on the right. After each octave, the Gaussian image is down-sampled by a factor of 2, and the process repeated.

(b) Visual representation of the in the DoG scale volume. Extrema of the DoG images are detected by comparing a pixel, represented as Xs to its 26 neighbors in 3x3 regions at the current and adjacent scales, represented as circles.

Figure 2: Visualisation of SIFT points on the Notre Dame cathedral in Paris, France.

After having detected potential salient points in the image, SIFT then examines the interest points at the respective scale and then examines gradients by computing a the filter response. It then aligns the descriptor based on the dominant gradient orientation, making it rotation invariant. The resulting feature is now not only scale but rotation invariant.

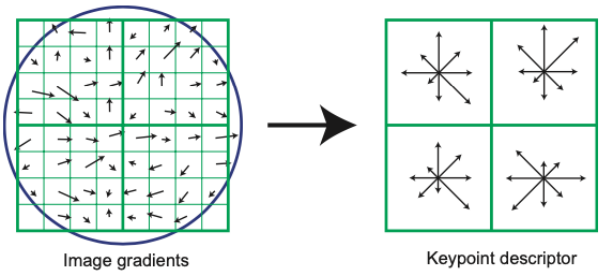


Figure 2: Graphical representation of how a key-point descriptor is created by computing the gradient magnitude and orientation around the keypoint location, as shown on the left, which are are weighted by a Gaussian window as indicated by the overlaid circle. Samples from key-point regions the are then accumulated into orientation histograms summarizing the contents over 4x4 subregions, as shown on the right. The length of an arrow corresponding to the sum of the gradient magnitudes near that direction within the region.

.

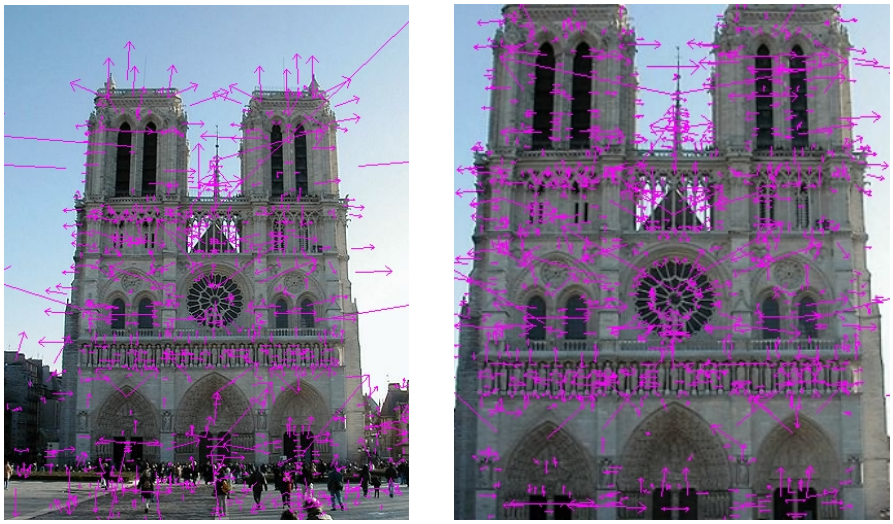
After rotation, gradient histograms are computed for local sub-regions which form the descriptor, where all histograms are concatenated and normalized to form a 128D feature vector [3]. The reason we look at these histogram, is because they are illumination invariant. Meaning if we examine the gradients we obtain in-variance to relative brightness changes, while a histogram is in addition also transformation invariant, since it aggregates information over a larger region.

1.1.1 Comparing Notre Dame images using SIFT

A demonstration of a SIFT algorithm can be seen in Figure 3. We extracted 846 feature points in the first image and 1117 feature points in the second image, with 316 total matches between the two. If we consider that some of the 846 salient points in the first image are noise factors (i.e. surrounding buildings and pedestrians as we are

primarily interested in the cathedral itself and in addition consider that the second image, has such noise factors omitted due to its closer perspective, 316 matches seems a lot. Even-though, the first image is not able to capture as many salient points due to noise, and a closer perspective to the building with noise factors removed can help with point detection, I believe that the algorithm did a fairly good job.

As such even-though salient points sometimes found in the first image, due to the influence of noise and distance to the building, moving closer to the building and removing such noise factors as in the second image leads to salient points to be found often. Most matches between the the first and second image within the cathedral regions should be found.



(a) Visualisation of SIFT points on an image of the Notre Dame cathedral.

(b) Visualisation of SIFT points on an another image of the Notre Dame cathedral.

Figure 3: Visualisation of SIFT points on the Notre Dame cathedral in Paris, France.

1.2 Random Sample Consensus

RANdom SAmple Consensus (RANSAC)[4] is a robust technique that can remove outliers and discount erroneous feature matches. That is, in practice once we already have some hypothetical matches using SIFT, RANSAC can verify which matches are inliers and which are outliers. Ransac randomly chooses a pair of edges to form line hypothesis and then tests how many other edges fall onto this line. Lines with sufficiently large numbers of matching edges are considered to be desired line segments and thus inliers. Lines with insufficently number of matches are regarded as outliers and can be removed. [1].

In more detail, RANSAC starts by selecting at random a subset of k correspondences, which are used to compute an initial estimate p. The residuals for a full set of correspondences are then computed using:

$$\boldsymbol{r}_i = \tilde{\boldsymbol{x}}_i'(\boldsymbol{x}_i; \boldsymbol{p}) - \hat{\boldsymbol{x}}_i' \tag{1}$$

, where $\tilde{x}_i^{'}$ are the estimated locations and $\hat{x}_i^{'}$ are the detected feature points. After computing the residuals, RANSAC then counts the number of inliers that are within the parameter ϵ of their predicted locations (i.e. $\| r_i \| \leq \epsilon$). The random selection process is then repeated S times and the sample set with the largest number of matching edges, which are inliers is kept for the final solution.

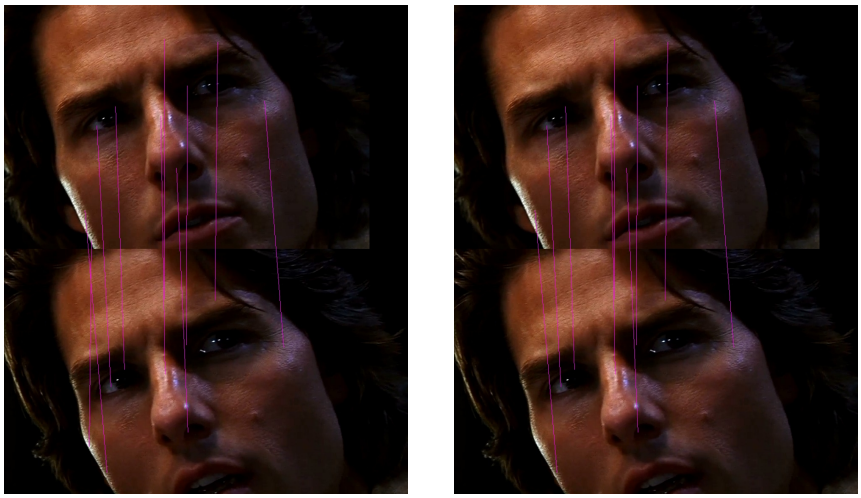
1.2.1 Compare matching methods: SIFT vs RANSAC

A demonstration of SIFT followed by RANSAC is shown in Figure 4 and Figure 5. We find 311 feature points in the first image and 417 in the second image. After RANSAC the number of matches between the two images are reduces from 14 to 13, removing a outlier. This can be visually verified by comparing Figure 5a vs. Figure 5b. A matching line near the left cheek (from the viewers point of view), has been removed. That feature point previously corresponded to two locations, which means SIFT was not able to match the point from the original image to a unique correspondence. RANSAC was able to detect this as an outlier an remove it.

(a) Visualisation of SIFT points on an image of the Tom Cruise.

(b) Visualisation of SIFT points on an another image of Tom Cruise.

Figure 4: Visualisation of SIFT points on famous action movie actor Tom Cruise.



(a) Visualisation of SIFT matches between the two Tom Cruise images.

(b) Visualisation of RANSAC matches between the two Tom Cruise images.

Figure 5: Comparison between SIFT matches and RANSAC matches on the same two images of Tom Cruise.

1.3 Image similarity

Feature matching and detection methods like SIFT can also be used to determine how similar two images are by examining the number of matches between the two images. If two images are similar to each other, it would make sense that they have the same or a similar quantity of matching features. To examine this, we expanded the `getsift` method: `Usage: getsift [image1.jpg] [image2.jpg]` to a `similarimage` method: `Usage: getsift [image1.jpg] [image2.jpg] [image3.jpg]`, which compares the second and third image to the first one and determines which one is more similar. The `similarimage` method was tested using the first Notre Dame image, the second Notre Dame image and the first Tom Cruise image. An example run of this can be seen in Figure 6, the last line of the console output is manually highlighted and returns the similarity evaluation. That is whether image 2 or image 3 is more similar to image 1. console. Results of the matching can be seen in Figure 7.

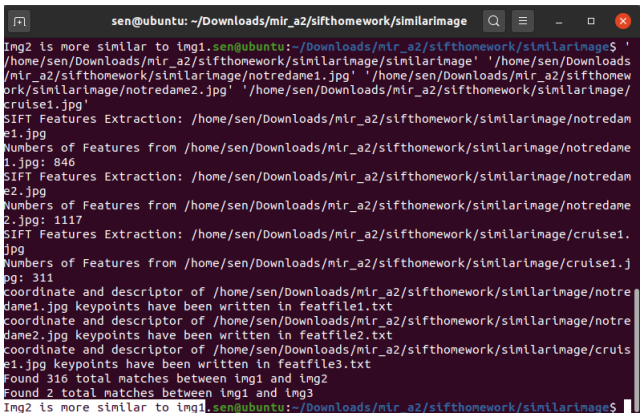
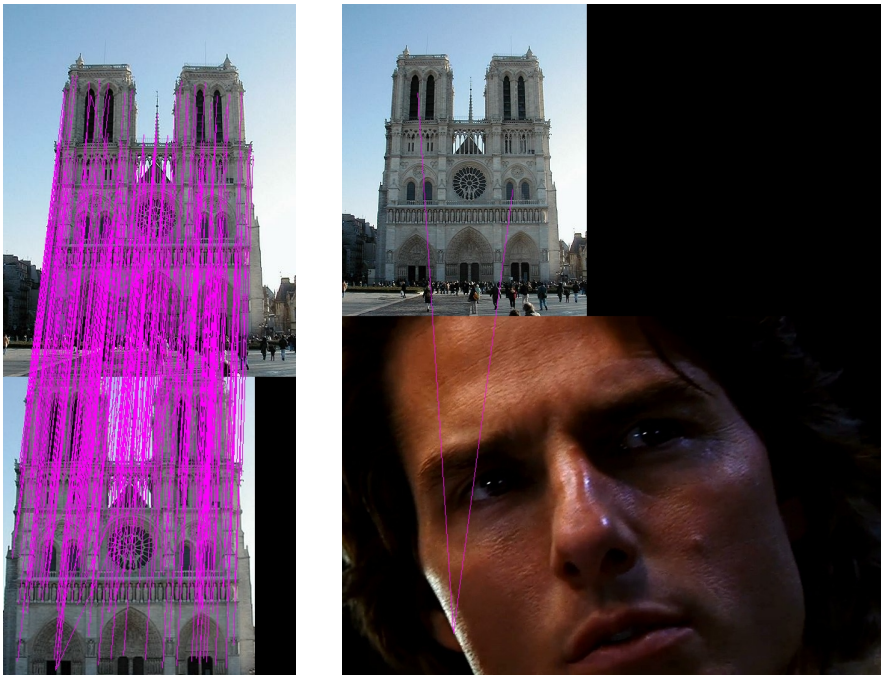


Figure 6: Terminal or console process of our similarimage method.

As the output in the terminal states, based on matches the second image (notredame2.jpg) is more similar to the first image (notredame1.jpg) than the third input image (i.e. notredame2.jpg). Interesting to note is that our matching based similarity method, matches a region near the earlope and cheek to the windows of the cathedral. If we were to magnify the image, we could see that the white reflection on the cheek and the black pixel values at the discontinuity between cheek and ear-lope, are corresponding to features near the white cathedral stone and its black tinted windows. Since SIFT features are scale, rotation and illumination invariant, such a match between two seemingly unrelated images can also occur.



(a) Visualisation of matches between the first and second input image.

(b) Visualisation of matches between the the first and third input image.

Figure 7: Results of our similarimage method.

2 Bag of Words

A challenging task within the computer vision community is the recognition of general class categories, which often involves recognizing instances of varied classes. A popular technique to approach this problem is to rely on the presence of features known as Bag of Words (BoW) models. A BoW is a sparse vector of occurrence counts of local image features. In a typical processing pipeline for BoW category recognition system, features are first extracted at keypoints over multiple images and then used to obtain a distribution histogram over the learned feature cluster centers (See Figure 7). The feature distribution histograms can then be used to learn a decision boundary using a generic classifier such as a support vector machine.

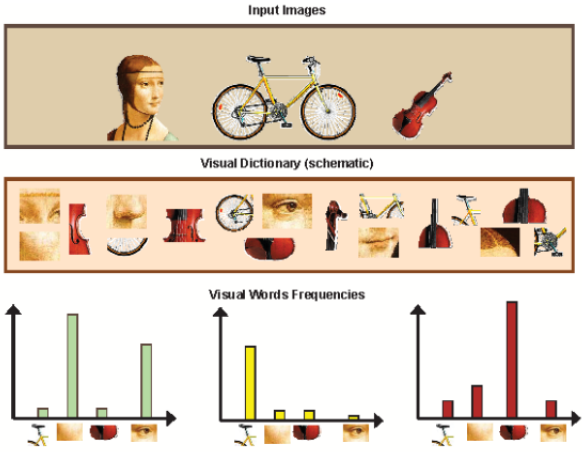


Figure 7: Schematic representation of Bag-Of-Words system.

.

To achieve this a BoW model generally follows three steps: 1) a feature detection stage, 2) feature description stage and 3) a code-book generation stage. At the feature detection stage feature points are localised in each image and subsequently represented as vectors referred to as feature descriptors in the feature descriptions stage. Since good feature descriptor is illumination, rotation and scale invariant, SIFT can be used for this step. In the final code-book generation stage, the feature description vector is converted into code-words, where a code-word can be thought of as aggregated representation of similar image patches or features. Since this is done over multiple features,

BoW generates a code-book or code-word dictionary, which contain aggregated representations of similar keypoints, clustered using a clustering method (i.e., k-means clustering).

2.1 Bag of words image similarity and retrieval

To showcase a BoW image similarity and retrieval system, we query for four different images (an image of a pizza, a beaver, a passenger plane and a military plane), where the BoW model was trained on 25 images of similar cases. An example of our retrieval system can be seen in Figure 8, where images with similar features are (ideally) ranked in order. To benchmark the system's sensitivity to varying dictionary sizes, the model was trained using a dictionary size of 100, 50 (default setting), 15 and 2. As a performance evaluation metric for our similarity system, we define the ranking accuracy as the percentage of relevant images in the top 5 ranks. The results of this analysis can be seen in Table 1.

As dictionary size decreases, the accuracy tends to drop in particular for the pizza, beaver and passenger plane classes. That is as dictionary size decreases, less features can be stored within the codebook dictionary, resulting in a decrease in the ability to detect similarities between images of the same category as can be seen in Table 1). It seems that the similarity between different pizzas is the hardest to detect, since it has the lowest accuracy with a dictionary size of 15, while the accuracy of most other classes only drops with a dictionary size of 2.

Figure 8: Example of our BoW image retrieval system based on similarity with the query image at the top and retrieved examples below.

If we relax inter-class separation and assume that a passenger plane and a military plane are similar enough as not to consider a military plane an erroneously retrieved image when querying for a passenger plane, the top 5 accuracy for both the passenger plane query and the military plane query increase as shown in Table 1. Since both classes share considerable features with each other, finding features that are unique to their respective class is difficult. Passenger planes and military planes are often considered to be similar, when in fact there are part of different classes. We are able to describe and retrieve the general class (i.e., plane), which involves recognizing instances of varied classes (i.e., various planes), but have difficulty detecting and describing the specific case (i.e., military plane and passenger plane). We note that the the system was consistently able to retrieve the image that was identical to its query image as the most similar image (i.e. rank one) regardless of dictionary size. Accuracy was never 0% .

Table 1: BoW image retrieval system based on similarity: Ranking accuracy as the percentage of relevant images in the top 5 ranks.

Dict size	100	50	15	2
Pizza img 1	0.60	0.60	**0.40**	**0.20**
Beaver img 2	0.60	0.60	0.60	**0.20**
Passenger plane img 3	0.40	0.60	0.60	**0.20**
Military plane img 4	0.60	0.60	0.60	0.60
Total accuracy	0.55	0.60	0.55	**0.3**

a) Respecting inter-class separation (i.e. assuming that a Passenger plane and a combat plane are part of a distinct class.)

Dict size	100	50	15	2
Pizza img 1	0.60	0.60	0.40	0.20
Beaver img 2	0.60	0.60	0.60	0.20
Passenger plane img 3	**1.00**	**1.00**	**1.00**	**0.80**
Military plane img 4	**0.80**	**0.80**	**0.80**	**0.80**
Total accuracy	**0.75**	**0.75**	**0.70**	**0.50**

b) Relaxing inter-class separation (i.e. assuming that a Passenger plane and a combat plane are both planes and therefore of part same class.)

2.2 Bag of words classifier

Subsequently, we examine the performance of a BoW classifier under varying dictionary sizes. A SVM classifier was trained on the code-book dictionary to obtain a decision boundary to separate three classes from each other (i.e., accordions, wildcats and soccer-balls). The results and subsequent accuracy of such a classifier are shown in Table 2. An example of the console output and process is shown in Figure 9. Given our classification problem, where the classifier has to find some non-linear decision surface across the features within the code-book dictionary to separate

Table 2: BoW SVM classifier: One-hot encoded predictions with ones representing a correct and 0 an incorrect classification.

Dict size	100	50	15	2
Accordion img 1	1	1	1	**0**
Wild cat img 2	1	1	1	**0**
Soccer ball img 3	1	1	1	1
Total accuracy	1.00	1.00	1.00	**0.33**

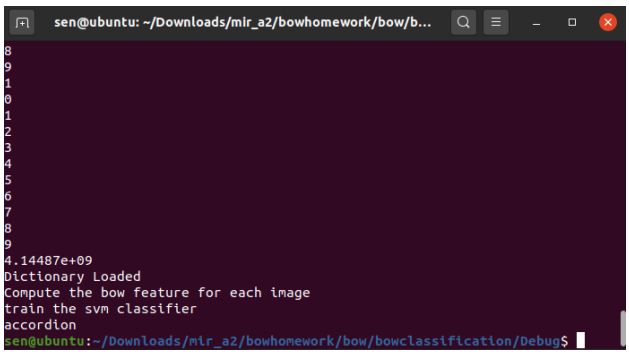


Figure 9: Terminal or console process of our BoW SVM classifier.

the three classes, we observe that a BoW SVM classifier is quite robust to decreased dictionary sizes. Dictionary sizes of $100, 50$ and 15 result in a classification accuracy of 100%. The classification accuracy is only affected once the dictionary size decreases to 2. With a dictionary size of 2 the classification accuracy drops to 33%. This is presumably, due to the non-linearity of separating the three classes. With a dictionary size of 2, the SVM is not able to find a non-linear decision surface to accurately assign images to their correct class correspondence.

2.3 Detecting Labradors

To further tax the BoW classifier, we expand the number of classes by adding a fourth class "Labradors". Results of whether we were able to detect a Labrador or not are shown in Table 3 with an example classification output from the console process in Figure 10. Since all input images are of Labradors false positives (i.e., falsely classifying a Labrador as Labrador) is not possible. False negatives, are however possible as our classifier may incorrectly assign one of the input images (all Labradors) to a false class. As such, the classifier obtains a false positive rate of zero regardless of dictionary size, but a false negative rate of 4 with a dictionary size of 15 and a false negative rate of 7 with a dictionary size of 2. The classification accuracy for a dictionary size of 15 is 60%, dropping to 30% with a dictionary size of 2.

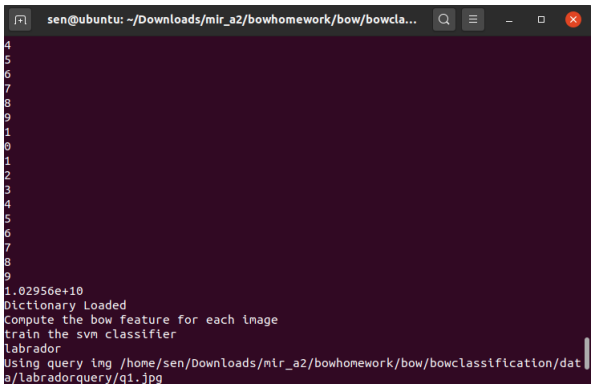


Figure 10: Terminal or console process of our BoW SVM classifier for classifying Labradors.

Given a classifier, which randomly guesses whether the input image is a Labrador or not obtains a classification accuracy of 50%, a dictionary size of 15 is sufficient to do better than random guessing. A dictionary size of 2 does worse than random guessing. Considering that a random guessing classifier would also have a chance of making a false negative error, 5 out of the 10 input images would be false negatives. As such, a dictionary size of 2 with a false negative rate of 7 also incurs more false negative errors compared to a dictionary size of 15 with 4 false negatives.

Table 3: BoW SVM classifier for Labradors: One-hot encoded predictions with ones representing a correct and 0 an incorrect classification.

Dict size	15	2
Labrador img 1	**0** (wild cat)	**0** (accordion)
Labrador img 2	**0** (wild cat)	**0** (accordon)
Labrador img 3	1	1
Labrador img 4	**0** (accordion)	1
Labrador img 5	1	**0** (soccer ball)
Labrador img 6	**0** (accordion)	**0** (soccer ball)
Labrador img 7	1	**0** (soccer ball)
Labrador img 8	1	1
Labrador img 9	1	**0** (soccer ball)
Labrador img 10	1	**0** (soccer ball)
Total accuracy	0.60	**0.30**

References

[1] R. Szeliski, "Computer vision algorithms and applications," 2011.

[2] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, pp. 91–110, Nov. 2004.

[3] D. Lowe, "Object recognition from local scale-invariant features," in *Proceedings of the Seventh IEEE International Conference on Computer Vision*, vol. 2, pp. 1150–1157 vol.2, 1999.

[4] M. Fischler and R. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.