Flazam: Using ImageNet for Hierarchical Object Identification

Sophie Chesney
Tessa Koelewijn

Abstract (SC)

Flazam uses WordNet's (Miller, 1995) hierarchical structure in conjunction with a corpus of ImageNet (Deng et al., 2009) images to answer object queries. In this report, we will present two variations of the system, one of which uses a K-nearest neighbours approach to find the best image match from a corpus of images (Flazam-KNN), and one which compares the features of the input image to the learnt features of the ImageNet images and outputs the best classification of the query image (Flazam-SVM).The answer provided is the name associated with the most similar image in that category. We report the performance of the two systems, comparing their accuracy and run-time values. We will show that, overall, Flazam-SVM is the more practical solution of the two, with Flazam-KNN heavily prone to biases in the data and subject to a long run-time.

1 Introduction and Motivation (TK)

Flazam is an image recognition tool that uses WordNet's hierarchical structure to limit the search space. The system can be used to answer object queries of the type: "what __ is this?". The idea behind the system is the intuitive assumption that it is much more efficient to search if you know more or less what you're looking for. The user provides the system with a category, which lets it know what to search for. This does not only limit the search space, but also provides the user with a very intuitive way of formulating a query. If a person wants to know what kind of flower he or she is looking at, it seems very likely that this person will ask "what flower is this?", rather than "what is this?"; adults rarely come across entirely unknown object categories, at least on a fairly general level. We make use of a person's knowledge about the query object, by making sure that this knowledge is available in the query, directing the system's search process. Flazam is trained on images from ImageNet and uses the hypernym/hyponym relation from WordNet to perform its search. When the user asks a question, all of its hyponyms are considered as possible output candidates and the most likely one will be returned. The current implementation described in this paper is still very limited, and is only able to search for three types of flowers, but once a satisfying accuracy is reached for this set, expanding it with more categories is only a small extension.

2 Implementations

2.1 SIFT Features (TK)

In order to describe both implementations, some explanation about the extracted SIFT features (Lowe, 2004) is necessary. For every image in the corpus, a set of keypoints is extracted. Each of those keypoints has its own SIFT-descriptor, which is a feature vector belonging to its keypoint. Both the Flazam-KNN and Flazam-SVM are based on the assumption that some of these descriptors will contain information specific to the flower it belongs to.

2.2 Flazam-KNN (SC)

As a basis for Flazam-KNN, we used the FLANN-matcher from OpenCV's (Bradski, 2000) Feature Matching tutorial (Mordvintsev and Abid, 2013). FLANN (Fast Library for Approximate Nearest Neighbours) is an algorithm optimised for fast nearest neighbour search in large datasets, and is therefore well suited to this object recognition task. In simple terms, the FLANN matcher will look for good feature matches, finding approximate near neighbours rather than the optimal or best match for a given feature. This is in contrast to a Brute Force (BF) matcher which will go through every possible match and find the best one. Of course, this means that a BF-matcher may be more accurate, but this increase in accuracy is costly in terms of efficiency. FLANN may miss optimal matches but performs reasonably well at a much faster rate.

Following the tutorial, OpenCV's FLANN-matcher takes the descriptor of a feature in the input image and looks for near-neighbour matches

with the features in a given corpus image, using a calculation of the distance between the points to determine how good a feature match is. The number of good matches for a given image is used as the basis for the computation of how similar the input image is to a given corpus image (Mordvintsev and Abid, 2013).

Each image therefore receives a score using the following similarity measure (adapted from De Graaf & Dobnik (2015)), and the best image match is the highest scoring match:

$$S1 = \frac{good\ matches}{keypoints\ in\ corpus\ image} \tag{1}$$

$$S2 = \frac{good\ matches}{keypoints\ in\ input\ image} \tag{2}$$

These two sub-scores are combined to reduce the effects of an image with many features matching with images with fewer features (to reduce bias that a feature-rich image may cause):

$$Score = \frac{2 * S1 * S2}{S1 + S2} \tag{3}$$

This approach was a very simple conceptualisation of the object identification problem, and has the advantage that only one good image match can give a correct output result. However, we found that the process of comparing each corpus image to the input image as extremely time consuming, even with the faster FLANN-matcher, leading to a very long run-time. Furthermore, if there is a dominant class in the corpus (perhaps in relation to the number of keypoints found in each image), then this class appears to dominate match guesses, leading in some cases to a severe bias in the results, despite the use of the combined similarity score. Given that the similarity score looks at the number of keypoints in the images, we expect that images that contain more extracted keypoints than others will dominate in the calculation of this score, leading to a biased guess in the matching process.

In an attempt to address both the timing and bias issues, we developed an alternative system that classifies each image based on a pre-trained SVM classifier, as will be discussed in the following section.

2.3 Flazam-SVM (TK)

The approach for Flazam-SVM is very similar to document classification, using a bag-of-words approach, but instead of words, we use the SIFT-descriptors. In document classification all the different words in the texts form the vocabulary, so it seems logical to treat all the different descriptors as different words to form the vocabulary. The problem with this approach is that two descriptors that are extremely similar, but not identical, will be treated as two unrelated words. To avoid this problem, the descriptors are clustered and the vocabulary is formed in the following way: a descriptor (des1) in the corpus is matched against every descriptor (des2) in the vocabulary, using cosine similarity. If the similarity is 0.6 or larger, the two descriptors are considered similar and will be treated as one 'word'. The values of des2 will be slightly changed towards des1, using a weighted average. If des1 is not similar enough to any of the descriptors in the vocabulary, des1 will be added to the vocabulary as a new word. This process is repeated for every descriptor in the corpus. The starting point is an empty vocabulary and the first descriptor is the first word in the corpus. In order to make sure that there is no bias towards the beginning of the vocabulary list, this list is randomised after every step. This is necessary because the program will loop through the vocabulary list in the same order every time, and stop as soon as it has found a match. The probability of the program reaching the end of the list is quite small, especially when the vocabulary is large. By randomising the list, we try to prevent the program from mapping descriptors to the same clusters too often. The result is a vocabulary vector, containing the descriptors belonging to every cluster in the vocabulary.

In the training stage, every image is vectorised using the vocabulary. In the vectorisation process, every descriptor of the image is matched against every descriptor in the vocabulary, using cosine similarity. Every descriptor is mapped to the 'word' with the highest similarity score. These vectors are then used to train the LinearSVC classifier from scikit-learn (Pedregosa et al., 2011). In the testing stage, again every image is vectorised. The classifier then predicts a class for every image in the testing corpus. As we will show in the evaluation section, the system performs quite poorly and on a small set, it does not outperform the Flazam-KNN approach. However, the issues with timing and bias are solved with this approach. Training is still time consuming, but once the classifier has been trained, classification is very fast.

2.4 WordNet and Dialogue Interface (SC)

As mentioned above, in addition to the core matching systems, both implementations also use WordNet's hierarchical structure to retrieve further information about the flower designated the best match. For example, Flazam-KNN contains a dialogue interface that allows the user to type a query (e.g. "what flower is this?") and receive the name of the flower as the answer, after the matching process is complete. This mirrors the intuitive motivation and description of the task as mentioned in the first section of this report. Furthermore, the user can

then ask for additional information if they so wish, which demonstrates the system's mixed initiative capabilities, to which a description of the flower is provided.

3 Evaluation (SC & TK)

For training and evaluation, we limited our dataset to three types of flowers: sunflower, daisy and wallflower. We carried out three experiments, where we compare the two systems on a toy dataset (50 instances per flower for training, 10 for testing), and the Flazam-SVM system on a bigger dataset (800 instances per flower for training, 80 for testing). We chose not to run Flazam-KNN on the large dataset, as the run-time became unmanageably long: based on the toy dataset, we estimate that it would take almost two days to classify the full dataset. The current results are detailed in the following table:

Flazam-	Dataset	Accuracy	Run-Time
KNN	toy	46.67%	1244.47s
SVM	toy	33.33%	194.44s
KNN	full	–	–
SVM	full	47.06%	6732.28s

Table 1: Accuracy and run-time results for both Flazam systems

4 Discussion & Future Work (SC & TK)

As the results show, neither of the systems achieve a particularly high accuracy level, but Flazam-KNN outperforms Flazam-SVM in terms of accuracy on the toy dataset. However, the run-time prevents this from being a practical solution on a larger dataset, even though we expect the accuracy to improve further. The shorter run-time makes Flazam-SVM a more practical solution, despite its lower accuracy. Additionally, this system produces a more even spread of guesses in the toy evaluation as compared with Flazam-KNN, which favours one flower class over the others (wallflower: 19 guesses, daisy: 9 guesses, sunflower: 2 guesses).

Interestingly, Flazam-KNN fails to follow human intuition when comparing two very similar images. For example, consider Image 1 and Image 2 below. Although we as humans consider these two images to be very similar, and would therefore predict a high match score, Flazam-KNN gives a similarity score of only 0.0429 between these two images. This is likely due to a number of factors, but image background may play a key role in distorting the score.

We expect that the background of an image will greatly impact on the success of its classification in *both* systems, so removing the it could potentially lead to significant improvements across the board, as only the core 'flower-features' will then be compared. Furthermore, neither system takes colour information into account, instead looking only at contrast for example, so this too could be added to improve accuracy.

Image 1

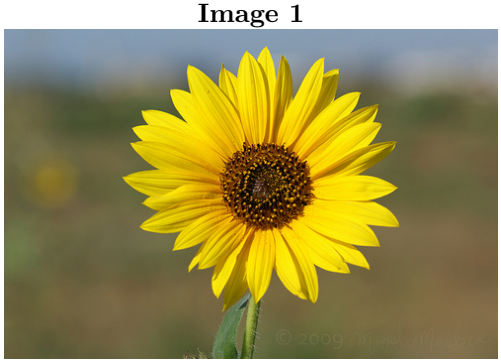

Image 2

Figure 1: *Two similar images with low match score*

More generally, given that we have used a random sample of the crowd-sourced ImageNet data, there are some very poor examples that appear in our dataset. Particularly in Flazam-SVM, which trains a classifier on all available training data, such 'bad examples' could also reduce the accuracy of the system. Ideally, with improved clustering methods and a more optimised approach, however, a much larger body of data could be used to train this system, potentially minimising the effects of this type of data driven discrepancy.

5 Conclusions (SC)

This paper presents two object identification systems. Though the current accuracy levels are not as high as desired, Flazam-SVM makes considerable improvements in terms of run-time, and lays a solid foundation for further development. Using a linguistically motivated design can reduce search space, leading to a theoretically intuitive system. Additionally, a dialogue system that makes use of the hierarchical structure gives a pleasing user experience.

References

G. Bradski. 2000. OpenCV. *Dr. Dobb's Journal of Software Tools*.

E. De Graaf and S. Dobnik. 2015. KILLE: Learning Objects and Spatial Relations with Kinect. In *Proceedings of goDIAL - Semdial 2015: The 19th Workshop on the Semantics and Pragmatics of Dialogue.*, pages 166–167. Gothenburg.

Jia Deng, Wei Dong, Richard Socher, Li jia Li, Kai Li, and Li Fei-fei. 2009. Imagenet: A large-scale hierarchical image database. In *In CVPR*.

David G. Lowe. 2004. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60:91–110.

George A. Miller. 1995. Wordnet: A lexical database for english. *Communications of the ACM*, 38:39–41.

A. Mordvintsev and K. Abid. 2013. OpenCV Feature Matching Tutorial. `http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html`. Accessed: 06/01/2016.

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. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.