Reverse Image Search

Piyush Kansal and Vinay Krishnamurthy

Stony Brook University

Adviser: Prof. Tamara Berg, Stony Brook University

Fall 2012

Abstract

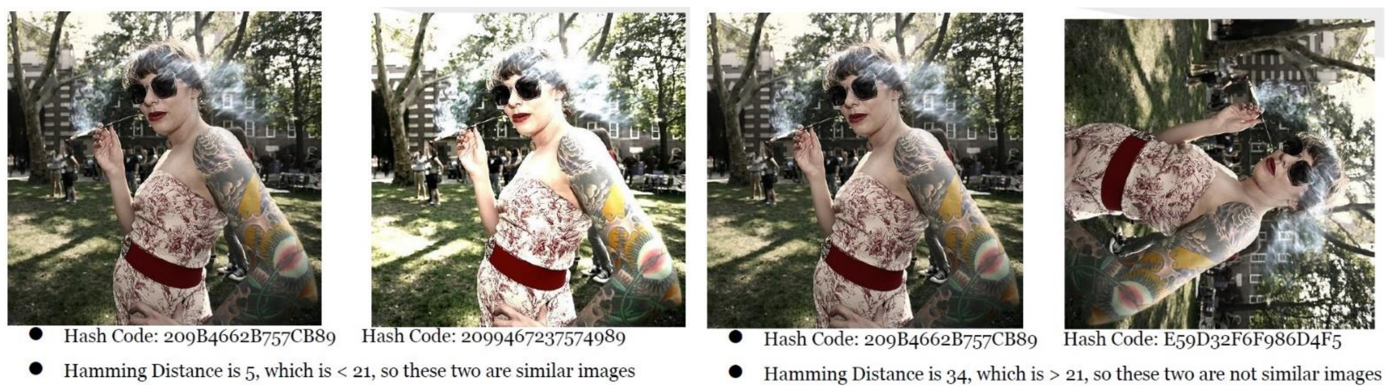
A regular image search on any search engine like Google begins by entering a keyword like Eiffel Tower, which is followed by a set of images related to Eiffel Tower in Paris. However, a lot of times, there is a need for searching with a given image and finding related images i.e. instead of giving a text as an input to search, an image is given as an input. For example, an image of Eiffel Tower is given as input and then we would like to see images not only of Eiffel Tower but also of similar towers. Existing techniques generates results which contains exactly matching towers i.e. only Eiffel Tower in Paris. They do not generate similar towers from around the world in the result.

We have developed a system which addresses this requirement. The test results varied as per the image to be searched and number of similar images in the dataset. Overall, the results were satisfactory.

1 Introduction

Image search engines like TinEye.com and Google' Similar Image are capable of doing image search on an input image i.e. an image given as input by user. These search engines then find the exactly matching images from all over the web from different sources. TinyEye.com uses an algorithm called Perceptual Hash [1].

In an image, the high frequencies give us the details while the low frequencies give us the structure. A large detailed picture has lots of high frequencies and a small picture lack details, so it is all low frequencies. And this is the concept on which Perceptual Hash (pHash) works. It reduces size of image to 8*8, converts it to grayscale, computes Discrete Cosine Transform (DCT), computes the mean value of 64 DCTs and set the bits of 64 bit integer to 1 or 0 depending on whether a particular DCT value was more or less than mean DCT value. This generates a 64 bit hash value of an image. Since Perceptual Hash belongs to the class of comparable hash functions, unlike other hash algorithms like MD5 and SHA1, this algorithm never has to deal with collisions and, two pHash can be directly compared using Hamming Distance. So, in pHash, a hamming distance value of less than 21 is considered as similar images.


Hash Code: 209B4662B757CB89
Hash Code: 2099467237574989
Hamming Distance is 5, which is < 21, so these two are similar images
Hash Code: 209B4662B757CB89
Hash Code: E59D32F6F986D4F5
Hamming Distance is 34, which is > 21, so these two are not similar images

(a) (b)

Fig. 1: Perceptual Hashes of Images. Fig. 1(a): The left image is the original image, the right image is changed by increasing the brightness and contrast. Fig. 1(b): The left image is the original image, the right image is changed by rotating it by 90 degrees.

Perceptual Hash algorithm has another interesting property: images can be scaled larger or smaller, have different aspect ratios, and even minor coloring differences (contrast, brightness, etc.) and they will still be matched as similar images (Fig. 1(a)). However, if the image is rotated, then pHash will generate a hash value whose hamming distance with the original image will be much more than 21, thus regarding the image as dissimilar (Fig. 1(b)). Besides, these search algorithms only rely on image analysis and do not consider text associated with an image which can also be helpful to generate more results.

The system we have developed solves these limitations as follows:

1. We take into account the text associated with the image. In our case, we used tags to find images having similar text. This step helps us in finding potential images which can be similar to the input image. We used an algorithm called Latent Dirichlet Allocation (LDA) [2] to find similar tags/words. For example, in one of the topics LDA classified water, sea, ocean and blue as similar words etc

2. We process the potential images using Spectral Hashing [3] which uses a better technique and does Principal Component Analysis on the potential images which is immune to rotations and minor structural differences and thus handles similar images better

The rest of the report is organized as follows: section 2 describes the design of the system, section 3 evaluates the performance of the system and section 4 describes the future work and section 5 lists down the references.

2 Design

2a. Goal

We started with the following goals for our system:

- Using the text file associated with the query image, we should be able to find the closest matching text files. Besides, we should also be able to find the text files which contains words which are semantically similar

- To search for the similar images as quickly as possible

- To generate better results by searching on huge dataset. We initially started with 25k images, however, finally expanded it to 400k images (all from MIRFLICKR [5])

2b. Algorithms Used

- **LDA (Latent Dirichlet allocation)**: LDA is a generative probabilistic model (three level hierarchical Bayesian model), which we have used in our project to discover similar topics. LDA can be best represented by a plate notation as depicted in (Fig. 2):

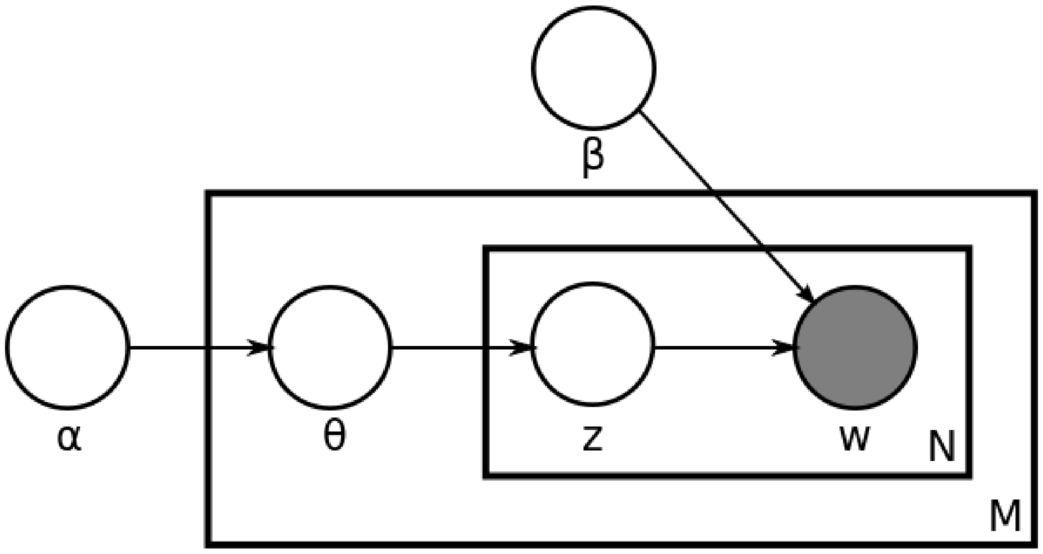


Fig. 2: Graphical model representation of LDA

Here, the boxes are plates representing "replicates", and the outer plate denotes documents, while the inner plate denotes repeated choice of words and topics within a document. The meaning of each symbol, represented in the above diagram, is given below:

- α – Parameter of the Dirichlet prior on the per-document topic distributions. We used $\alpha = 0.5$ in this project
- β – Parameter of the Dirichlet prior on the per-topic word distribution
- θ_i – Topic distribution for document *i*

    - z_{ij} – Topic for the j^{th} word in document i
    - w_{ij} – Specific word
    - N – Number of words in a document
    - M – Number of documents

- **Spectral Hashing**: In order to detect similar images, we have used the Spectral Hashing algorithm to reduce the image data into a 32-bit hash value. This algorithm also calculates hash value to detect similarity (similar to pHash algorithm), but this algorithm is resilient to rotation of images, as it tries to correlate code-words with semantic similarity of the image. The working of Semantic Hashing can be summarized in three steps:
    - First step is to calculate PCA for a given set of data points for a given image. We considered GIST descriptor as data points of an image, for this project
    - Next step is to calculate k smallest single-dimension analytical eigenfunctions of L_p (where k is the no. of bits to encode the hash value, and L_p is the Laplace-Beltrami operator). Since, encoded the image as a 32-bit hash value in our project, k in this case was 32
    - The last step is to obtain the binary codes i.e. value of each bit in the hash value. This was obtained by thresholding the eigenfunction values at zero. So, a negative value of eigenfunction was encoded as 0 in the hash value, while a positive eigenfunction value was encoded as 1

2c. Working

Following is the step-by-step process of how the system works:

1. **Preprocessing:** It is an important step before starting the image search and helps in reducing the amount of time taken to find similar images. We do preprocessing on the text associated with the images in the dataset, which in our case are the tags associated with the images and generate following data files:
    a. A vocabulary of all the unique words (required as input for LDA)
    b. A dictionary of a word to file mapping i.e. in what all files a particular word was found (required as input for LDA)
    c. Word count of each unique word among all the words (required as input for LDA)
    d. Input files for LDA (generated from the three files (a), (b), (c) mentioned above)
    e. Topics and topic weightages using LDA (generated by running LDA). We create 10, 20, 30 and 40 topics because the larger the number of topics, the more accuracy one gets during text processing). We tried going for higher values of topics, however, the LDA program crashed due to excessive memory usage and so we could not go further.

    In this step, we only do text processing and create the required data files.

2. **Processing during Query time:** In this step, we take input from user, a query image and number of topics to search the image in and then do following:
   a. **Text Processing:**
   Using the preprocessed data and text associated with the query image, we find 3000 most similar text files as follows:

      1. Using the text associated with the query image, we calculate SSD on all the text files in the dataset. We sort the SSD values in ascending order and pick the first 1000 text files. This gives us a set of text files which contain *exactly matching* combination of one of more words in the query image
      2. As per the number of topics chosen by user during input, we select a topic weightage file generated during preprocessing step and then we find two topics of highest weightage. The words present in these two topics helps us in finding text files which contain *similar words* and thus can be a good candidate for further processing. For each topic, we pick 1000 files and thus in total 2000 files containing similar words

   The images corresponding to these 3000 text files becomes our candidate for final image processing and are given as an input to the next step.

   b. **Image Processing:**
   3000 chosen images along with the query image are processed using Spectral Hashing to compute a 32 bit hash code for each single image. Since Spectral Hashing uses GIST and PCA, it takes care of the variations like rotation, contrast, brightness, aspect ratio etc. and is thus able to generate good results. We then calculate the Hamming Distance between hash code of the query image and hash code of the 3000 images and pick all the images as our final result, which falls under a Hamming radius of 7. We experimented with lesser Hamming distances as well, but the value of 7 generated better results compared to the other values.

   All these resultant images represent the images most similar to the query image in our dataset.

3 Evaluation

We did evaluation on MIRFLICKR dataset [5]. We downloaded 400k images and the associated tags. We experimented on 3 categories. Following are the categories and a graph showing accuracy (Fig. 3):

1. Test 1 – Scenery
2. Test 2 - Water bodies and
3. Test 3 - Animals

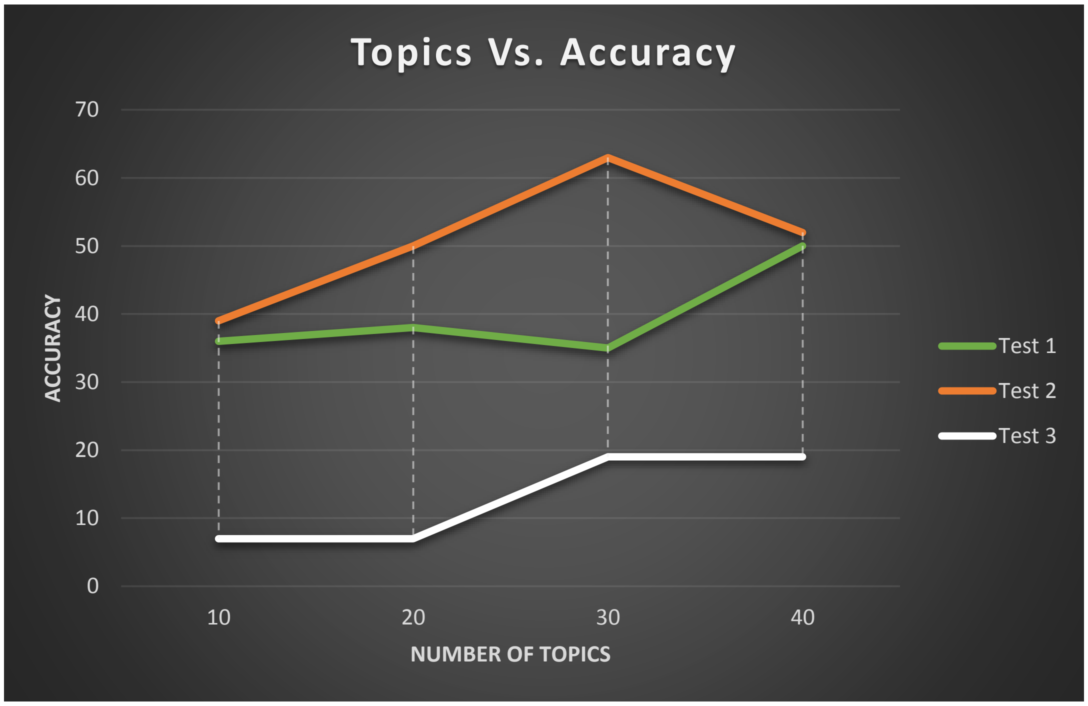


Fig. 3. Graph showing accuracy vs. number of topics

Evaluation results varied depending on the query image and the number of similar images already present in the dataset. Following are the results for the 3 categories (Fig. 4, 5 and 6):

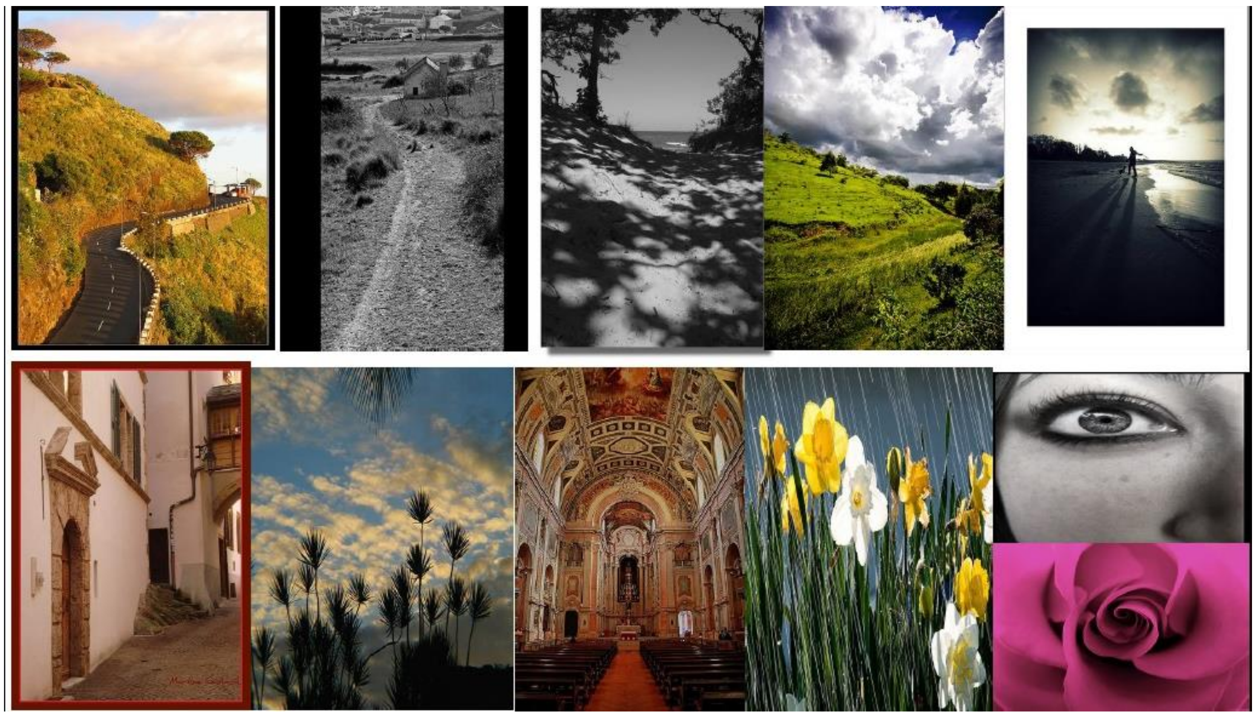


Fig. 4. Test 1 – Scenery

Test results for sceneries (Fig. 4) were good. We see lots of images of various kinds of sceneries present in our dataset. Moreover, text processing of such an image yield substantial number of good results because of lots of words similar to water, trees, sky etc.

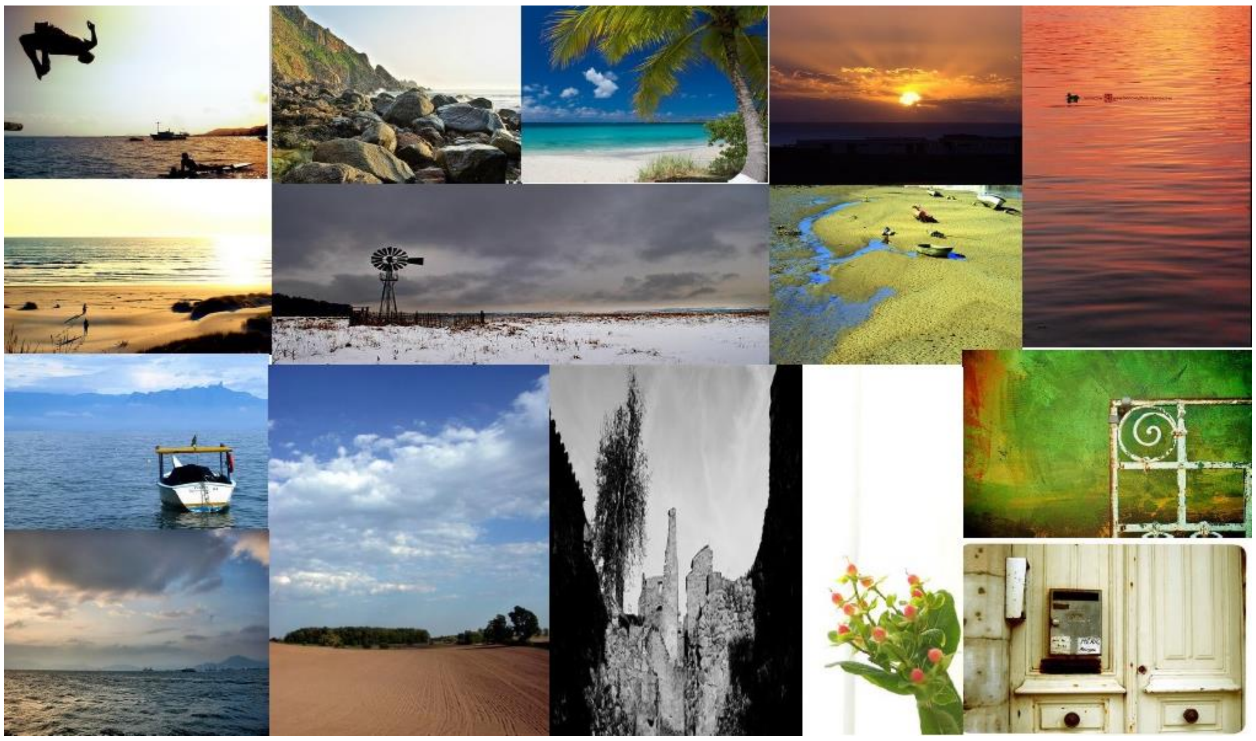


Fig. 5. Test 2 – Water Bodies

Test results for the water bodies (Fig. 5) were the best as we have lots of images of water bodies in our data set and as already mentioned, a substantial number of good images get matched during text processing as well.

Test results for horses (Fig. 6, on next page) did not have many images having horses as we do have many images for animals in our dataset however not especially for horses. And thus the graph (Fig. 3) shows a lower accuracy value for this test result.

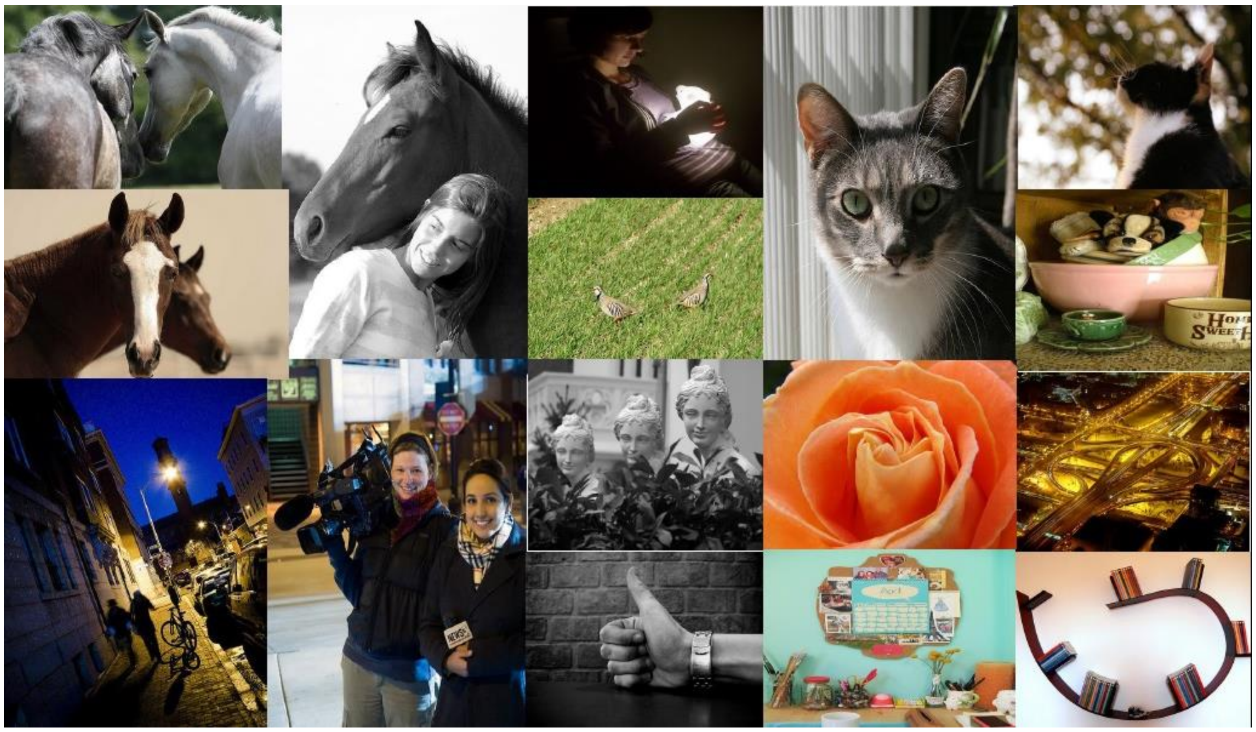


Fig. 6. Test 3 – Animals

4 Future Work

During the evaluation of the test results, we came across few things which we believe should be able to improve the test results to a good extent. They are as follows:

1. Use of Multidimensional Spectral Hashing (MDSH) [4]:
   Spectral Hashing performs well for the close neighbors, however does not work good for distant neighbors, thus giving inaccurate results. Multidimensional Spectral Hashing takes care of distant neighbors, and thus generate better results

2. A bunch of images in the test results do not match at all with the query image. It happens when the text files associated with these images are empty which causes them to get selected during the phase of using SSD. An area of improvement in this case would be to not to take into account such empty tag files

5 References

[1] http://www.hackerfactor.com/blog/index.php?/archives/432-Looks-Like-It.html
[2] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet allocation. In: ANIPS. (2002)
[3] Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: NIPS. (2008)
[4] Weiss, Y., Torralba, A., Fergus, R.: Multidimensional Spectral hashing. In: ECCV. (2012)
[5] http://press.liacs.nl/mirflickr/