DIGITAL CULTURE
ITMO UNIVERSITY

Image Comparison: Global Features

High School of Digital Culture
ITMO University
dc@itmo.ru

Contents

1 Image features and similarity: traditional computer vision

1.1 Why do you need to measure image similarity?

In the previous lectures we discussed low-level image processing techniques, which can be very useful for image enhancement. In those techniques both input and output are images, but the output is in some sense better than the input - less noisy or with higher contrast. These techniques can make it easier to analyze images and extract semantic information for both humans and computers. They are the basis for other more complex image analysis and image recognition algorithms, which we will be discussing next.

During the introductory part of the course we looked at different tasks and applications of image analysis and recognition: image retrieval, classification, segmentation, object detection and recognition, and many others. Many of these are based on the notion of similarity between the images. The definition of similarity may change from task to task, but an ability to compare two images with respect to task-specific similarity is paramount to all of these tasks.

Image retrieval For image retrieval we consider two images similar, if they have both visual and semantic similarity. But these are not necessarily pictures of exactly the scene. Like these two images in the first leftmost column - they are similar, they are both images of rural mountain areas during summer time. Here we care about similarity of the whole image.

Scene classification For scene classification task semantic similarity plays a more vital role than visual similarity. If we would like to classify scenes as indoor and outdoor, we would expect all images on this slide to be treated as similar, because there are all outdoor scenes. We would like a winter landscape with a lot of white snow to have higher similarity measure with a green grassy summer landscape, than with a clean white room.

Object detection and segmentation But if our task is object detection, and buses, trams and cars are objects of interest, these images will end up in different buckets. We would expect two images in the second column to be considered similar, and two images in the third column to be considered similar, because these pairs of images have objects of the same type. For object detection we typically care about similarity between fragments of the image depicting objects of interest, and not so much about background.

The task of semantic segmentation is similar to the object detection task, but requires much more precise localization of the object. Instead of approximate location of an object, the goal of image segmentation is to provide labels for every pixel - if it belongs to an object or not. For that we may want to measure similarity not only between any given fragment of an image under analysis and some golden template for the object of interest, but also between different fragments of the image we analyze - to make a judgment if two neighbor fragments belong to the same object or not.

Image annotation For image annotation we'll need to assign tags to both objects and background in the images. For example, many images here have sky, some have road. And here we'll need to compare image fragments to some representations of specific tags. This task is solved as multi-class classification problem - every image is allowed to have multiple labels, and we compare every given image to representations of every label in the dictionary of available labels.

Near duplicate detection, image alignment, 3D reconstruction, motion tracking For all these tasks we mentioned so far, two images may be considered similar even if they are not images of exactly the same scene or exactly the same object. These images of trams would belong to the same class and should be considered similar, if our task is tram detection. But for some other tasks, such as near duplicate detection, image alignment, 3D reconstruction, motion tracking we need to be able to identify when two images depict exactly the same scene or object. For these tasks images of green trams shouldn't be considered similar, but the two images of the yellow pickup and the two areal shots should. And for for many of these tasks we also need to identify positions of matching points.

1.2 How do you measure image similarity?

Now that we've seen that we need to measure similarity between images, the next question is how to do that? A computer needs numeric representation of images and a similarity metric defined in the given representation space. A numeric representation of an image is commonly called feature vector. The most basic feature vector you can use is the raw pixel intensities themselves. We could compare images using these basic feature vectors pixel-by-pixel, but this would only work well for identifying exact image copies. Pixel-wise similarity metrics aren't capable to capture semantic similarity between the images. Even two pictures of exactly the same scene from the same position would give us different pixel values due to potential differences in light conditions, exposure, or focus. That's why image representation with raw pixel intensities is usually just a starting point for extraction of more complex features, which are then used to compare images.

An output of a feature extraction transformation is a feature vector. A feature vector is just a vector that contains information describing important characteristics of an image. Feature vectors are typically a more compact representation of an image than the original pixel values, but at the same time more descriptive and preserves key semantic information of interest.

Once we have this new representation of images via feature vectors, we can compare feature vectors using task-specific similarity measures defined in the corresponding feature spaces.

Feature spaces So, feature vector is an abstraction of an image used to characterize and numerically quantify the contents of an image. Normally real, integer, or binary valued. Simply put, a feature vector is a list of numbers used to represent an image. In image processing, features can take many forms. Traditional image processing and computer vision algorithms often rely on very sophisticated collections of features. And the choice of features in a particular computer vision system may be highly dependent on the specific problem at hand.

Choosing informative, discriminating and independent features is a crucial step for effective algorithms in pattern recognition, classification and regression. Features are usually numeric, but structural features such as strings and graphs are sometimes also used.

The vector space associated with feature vectors and with a similarity function defined for this space is often called the feature space. In other words, a feature space is just the set of all possible feature vectors with a function than can be used to compute similarity (or distance) between any given two feature vectors.

In some applications, it is not sufficient to extract only one type of features to obtain the relevant information from the image data. Instead two or more different feature vectors are extracted. And each of these feature spaces can have its own associated similarity measure.

These feature vectors can be concatenated into one long joint feature vector, and then we can define a similarity function for this new feature space of a higher dimension.

Combination of feature vectors One simple way to define a similarity function for a combination of different feature vectors is to use a linear combination of similarity functions defined for each of the original feature spaces. If we have a similarity function d_1 defined for features X, a similarity function d_2 defined for features Y and so on, we can define a similarity function for the combined feature space as a weighted sum of all d_i. So when measuring similarity between two images, we will take into account different features, and we can also define how

important each feature is for a particular task and adjust weights of our weight sum accordingly.

1.2.1 Deep Learning vs Traditional Computer Vision

Feature extraction step can be designed and programmed manually. It is also known as feature engineering. It's a combination of art and science. It requires the experimentation of multiple possibilities and builds on the intuition and knowledge of the domain expert. Major achievements in traditional computer vision are due to more and more sophisticated feature extraction techniques. But over the past several years manual feature engineering has been replaced by automated feature learning for many applications of traditional computer vision. Deep learning made it possible for a machine to not only use hand-crafted features for learning, but to learn the features itself. Deep learning encapsulates both labor-intensive feature engineering and automated decisioning. For many applications today it is possible to feed raw pixel values into a machine learning algorithm and get more accurate predictions than ever before. We will talk about deep learning in computer vision in the upcoming lectures. But today let's focus on traditional algorithms. Deep learning is the most common solution today for many image recognition tasks, but others, like image alignment and 3D reconstruction, are still often rely on traditional hand-crafted feature extraction techniques.

1.3 Image features taxonomies

Textual and visual Now let's look at different types of image features. There are different ways to categorize them. First, image features can be divided to textual and visual. Textual features are comprised of textual annotations created by the user and image metadata, such as geo tags, create date, camera settings, etc. These are not derived from image pixels, but still can e be very useful for different tasks and are often used in combination with visual features.

Visual features are features derived from raw pixel values. These are content-based features. The can characterize color distribution in an image, image texture, shapes of objects, relative positions of objects, color or texture patterns.

Global and local Visual features can be further classified to global and local. Global features describe an image as a whole. It's typical for global features to be computed based on all pixels values of an image. Some examples are color histogram of an image, or an average intensity of an image. These features can be used image retrieval and scene classification tasks, but these days are mostly replaced by deep learning.

Local features describe a fragment of an image. Many local features are computed for regions in the neighborhood of so called keypoints or points of

interest. Keypoint detection and local features are still widely used for many tasks such as image alignment, 3D reconstruction, motion tracking. Just a few years ago these manually designed local features have been a foundation for progress in object detection and segmentation too, but these days learned features and deep learning have replaced manually engineered features for these tasks. We will get to deep learning a bit later, and now let's talk about content-based features in general and learn a couple of most prominent local feature detectors still used today for a number of computer vision tasks.

1.4 Content-based features: color, texture, shape

There are many different classifications of content-based image features. Color, texture and shape are among the most intuitive features for humans. And there are many approaches in computer vision how to describe and represent these important properties of an image. Sometimes feature vectors are also called descriptors. Color, texture and share descriptors describe color, texture and shape of a whole image or of some part of an image. For some tasks spatial arrangement of image parts is also important, and in then spatial features can also be used to incorporate information about spatial layout of image fragments with specific color, texture of share properties.

Color Color is arguably the most important feature for human visual perception of an image. That's why color descriptors play very important role in image analysis. A natural way to compare color characteristics of two images is to compare color distributions in those images. And thus many color features are descriptors of a probability distribution and common metrics used to compare color features are distance functions for probability distributions.

The most straightforward representation if a distribution is a histogram. Histograms are often used as image features. Color histogram is a representation of the distribution of colors in an image. It is obtained as the normalized count of the number of pixels in the image that fall in each of possible colors in a given color space. So when color histograms are used as color features, images will be represented by vectors of a length equal to a total number of colors in a chosen color space, and each element of this feature vector is a fraction of pixels of a particular color in a given image. Then two images can be easily compared by computing distance between their color histograms. Various metrics can be used for that: L_1, or just a sum of element-wise differences of two vectors, L_2 - Euclidean metric, L_∞, Chi-squared distance, Earth Mover's Distance (or Wasserstein distance) - to name just a few.

Because we are talking about distributions, standard statistical measures are also sometimes used for a more compact representation of image color distributions, such as the first three moments: mean, variance and skewness. And then

again standard metrics can be used to measure distance between feature vectors.

Common functions for histogram matching We've mentioned many different functions which can be used to compare two histograms. In practice, histogram intersection and Chi-squared are used most frequently. To compare color histograms of two images of different sizes, it's important to normalize histograms first.

Histogram intersection is the easiest and the fastest to compute metric. It's defined as

$$histint(H_1, H_2) = 1 - \sum_{i=1}^{K} min(H_1(i), H_2(i)) \tag{1}$$

where H_1 and H_2 are two histograms with K bins. This distance function is attractive because of its ability to handle partial matches when the areas of the two histograms (the sum over all the bins) are different. When the histograms are normalized (when areas of the two histograms are equal), the histogram intersection is equivalent to the normalized L_1 distance.

Another commonly used function is Chi-squared distance. It is more expensive to compute, but often works better. It is defined as

$$\chi^2(H_1, H_2) = \frac{1}{2} \sum_{i=1}^{K} \frac{(H_1(i) - H_2(i))^2}{H_1(i) + H_2(i)} \tag{2}$$

The main advantage of the Chi-squared distance is that it's calculated on relative counts. So a small difference between two small values will have greater impact than exactly the same small difference between two large values.

Quantization and histograms Choice of a distance function for histograms is not the only choice which influence the results. Another important aspect is quantization. Although we are dealing with digital images and thus all the values are already discrete, often more coarse-grained quantization of the color space is needed. It will reduce the size of feature vectors and may also lead to better end-to-end results with distance functions only relying on bin-to-bin comparison (like the once we've discussed before).

The choice of the right quantization scheme might be tricky. With bin-to-bin distance functions, too fine-grained quantization will result in large distances even between images with similar, but not exactly the same colors. Too coarse-grained quantization will lead to different colors being assigned to the same bin, and thus we will end up with small distance between differently colored images.

Histograms in multidimensional spaces Another tricky aspect with quantization and histograms is related to multidimensional spaces. Color spaces are usually 3-dimensional: R, G, B coordinates in RGB space, H, S, V - in HSV, etc. In a multidimensional space we can build joint or marginal histograms. Joint histograms of data in an N-dimensional space are N dimensional. You just put the data points into N-dimensional bins. Joint histograms often have many empty bins, especially with fine-grain quantization. Coarse-grain quantization helps to reduce total number of bins, and thus there will be fewer empty bins, but coarse-grain quantization means loss of resolution and not-that-similar values will end up in the same bin.

Due to this, sometimes marginal histograms are used instead in multidimensional spaces. Marginal histograms are less than N-dimensional histograms where one or more dimension has been ignored. On this slide we have two-dimensional space, and an example of a joint histogram om the left and two marginal histograms on the right. Joint and marginal histograms are very similar to joint and marginal distributions. With marginal histograms we will get more data points per bin compared to joint histogram with the same bin sizes along every feature.

Quantization via clustering An alternative way to quantize a multidimensional space is via feature clustering. A number of bins is defined by a number of clusters. This technique allows for non-uniform data-based quantization of feature space. Cluster centers are usually defined using a set of available images-examples. And then when a new previously unseen image should be processed, features of that image are assigned to the closest bins.

On this slide we again see a two-dimensional feature space. We can quantize this space into two bins using a clustering algorithm with two clusters. X points on the right image are places in the centers of two clusters.

Quantization via clustering is a good alternative for high-dimensional feature spaces. It's not frequently used for simple color histograms, because color spaces are typically only 3-dimensional. But we will see at least one example later in the course when this technique is used with other feature spaces.

What about color similarity? Histograms with bin-to-bin distance functions are a powerful and frequently used tool for color comparison of two images. But it is not perfect. Even with smart color space quantization techniques, it still doesn't take into account similarities between different color bins. A human would expect a distance between histograms H_1 and H_2 to be less than a distance between histograms H_1 and H_3 on this slide, because although H_1 and H_2 don't have intersecting bins, the overall color distribution as perceived by humans is similar - the have bins if similar colors. With any bin-to-bin distance function the distance

between H_1 and H_2 will be greater than the distance between H_1 and H_3, because H_1 and H_3 have one common bin.

One possible way to fix this problem is to use cumulative histograms and still keep bin-to-bin distance function of your choice. The cumulative histogram is a histogram in which every bin counts values for that bin plus values of all previous bins. Cumulative histogram corresponding to H_1 histogram from our previous example will look like this. The first bin will be the same, the second bin contains sum of the first and second bins of the original histogram, and so on. Because second, third and forth bins in the original histogram are empty, the first four bins of the cumulative histogram will have the same value. Now in fifth bin we need to add content of the previous, forth, bin and the fifth. And so on. I kept colors on the slide, so it's easy to follow.

So, the cumulative histogram corresponding to the original histogram H_1 looks like this. Now let's build cumulative histograms corresponding to H_2 and H_3. I painted them with different colors so it's easier to compare. You can see that a simple bin-to-bin distance such as L_1 will now give us a result which better corresponds to our intuition. Distance between the cumulative histograms H_1 and H_2 is less than between H_1 and H_3.

It's important to note, that this approach will work only for the case when neighbor histogram bins correspond to similar colors.

One cal also use Earth Mover's Distance to address cross-bin similarity. It's also known as Wasserstein distance or Kantorovich–Rubinstein metric. Intuitively, if each histogram is viewed as a pile of soil distributed across bins, the metric is the minimum "cost" of turning one pile into the other, which is assumed to be the amount of earth that needs to be moved times the distance it has to be moved. So coming back to our original example, all three histograms, H_1, H_2 and H_3 have three units of "earth" in three different bins. To transform H_1 to H_2 we need the content of all three bins one step to the right. So the cost of moving is 3 unit-steps. To transform H_1 to H_3, the red bin stays where it is, but two greenish bins need to be moved larger distances: 3 steps for salad green and 6 steps for emerald green. So, total of 9 unit-steps. Wassertstein distance between H_1 and H_2 is less than between H_1 and H_3, which is closer to human perception.

As with cumulative histogram, EMD "works" as expected only if neighbor histogram bins correspond to similar values (similar colors in case of color histograms).

Another option is to adopt a different distance function which can measure cross-bin similarity. An example of such distance function is quadratic-form distance, defined on the slide. This distance was suggested in Niblack et al. (1993) for color-based image retrieval. It is a generalized Euclidean distance with a middle term matrix A, which incorporates cross-bin similarity information. It contains similarity coefficients for all the bins of the quantized space.

What about layout? Another problem with color histograms is that they don't take into account spatial layout of colors. They represent color distribution without any spatial information. For example, color histograms of these three synthetic images are exactly the same - all three images have equal number of white and black pixels.

And here is another example of three different images with the same color histogram, but which are clearly very different and won't be considered as similar by a human.

Color spatial layout In early days of computer vision people sometimes partitioned the images into fixed or fuzzy blocks and constructed color histograms for each block independently. This trick added spatial information into color features, but it is not adaptive - block positions are fixed for all images.

That's why color on its own is not enough for image comparison and texture and other features are often used in combination with color.

Histograms summary: quantization Summarizing, color histograms is a very simple but at the same time expressive color descriptor. It has been one of the first and most important descriptors. Although it has been replaced by more sophisticated techniques for image recognition tasks, it can be powerful enough for specific sets of images. Histograms in general are widely used as descriptors till these days.

A choice of quantization is important for histograms to work well. With fewer bins the probability of two histograms having non-zero values for the same bins is higher, even when there are not as many data points, so bin-to-bin metrics will work better. But this also means coarser representation of the data, when non-similar values might be placed into the same bin.

With many bins we get fine-grained representation of the data, but it becomes harder to match histograms - with few data points many bins will be empty, and two histograms might have zero intersection, so it won't be possible to compare them with bin-to-bin metrics. The distance between two images that have similar but not identical colors will be large. The histograms with more bins are very sparse and, thus, sensitive to noise.

Quantization also can be uniform and data-independent, grid-based, when we partition the whole space into grids. For multidimensional space it becomes a Cartesian product of uniform partitions of each dimension. This is easy to implement, but will lead to very large number of bins for high-dimensional space. And because this partition is not data-driven, many bins may end up empty.

Another way to partition the space is using clustering. This approach is data-driven and will create non-uniform partitions of space in accordance to data distribution. This is more expensive - to find centers of the bins we need to cluster

a representative subset of the data. But at the same time it can lead to fewer bins with enough granularity for high-dimensional spaces.

Histograms summary: matching Different functions can be used to compute distance between two histograms. Majority of distance functions used today are bin-to-bin distance functions, which don't incorporate cross-bin similarity. Most frequently used metrics are histogram intersection, which is fast to compute, and Chi-squared distance which is more expensive to compute, but often works better, than simple histogram intersection or L_1.

For cross-bin comparisons, when nearby histogram bins represent similar values, one can use cumulative histograms with standard bin-to-bin distance functions, or switch to Earth Mover's distance.

Quadratic-form distance is a more general distance function for cross-bin comparisons. It incorporates arbitrary similarity between bins via similarity matrix A.

Texture Although color descriptors contain important information about contents of an image, it's often not enough. Another important image feature for visual human perception is texture. It gives us information on structural arrangement of surfaces and objects on the image. We won't go into as many details here as with color descriptors, but will discuss a couple of the most important texture descriptors of the past.

What is a texture? Everyone is capable of recognizing texture; however, it is difficult to give a formal definition of this concept. Intuitively texture descriptor provides measures of properties such as smoothness, coarseness, and regularity. Let's look at three examples on the slide. The white squares on the three images on the slide mark, from left to right, smooth, coarse, and regular textures. Texture descriptors are designed to quantify the perceived texture of an image. Image texture gives us information about the spatial arrangement of color or intensities in an image or selected region of an image. Texture is not defined for a separate pixel; it depends on the distribution of intensity over the image.

Texture analysis plays an important role in comparison of images supplementing the color feature. There exist many methods for texture representation and comparison of images on the basis of texture. Texture features can be classified into statistical, geometrical, model, and spectral.

Statistical features describe distribution of intensity over the image by means of various statistical parameters. Features from this category are among the first texture descriptors in computer vision. One of the simplest approaches for describing texture is to use statistical moments of the intensity histogram of an image or region. Other statistical texture features include descriptors based on the co-occurrence matrices, and texture histograms built upon the Tamura features.

Geometrical features describe texture with texture elements, or primitives. This class includes methods for representing texture by Voronoi diagrams and structural methods. The latter specify texture by defining texture primitives and placement rules for these primitives relative to one another. Structural methods are not appropriate for describing irregular textures.

Model methods of texture analysis rely on construction of a model that can be used not only for description but also for texture generation. Descriptors based on Markov Random Fields and fractals belong to this category.

Spectral features describe texture in the frequency domain. They are often based on decomposition of a signal into a weighted sum of basis functions and using the weights as elements of the feature vector. Among spectral approaches Gabor filters deserve special attention. Gabor filters have been widely used in myriad of image processing applications. The characteristics of certain cells in the visual cortex of some mammals can be approximated by these filters. These filters have been shown to possess optimal localization properties in both spatial and frequency domain and thus are well suited for texture segmentation problems.

Gabor filters Gabor filters are linear filters used in many image processing applications for edge detection, texture analysis, feature extraction, etc. With a Gabor filter one can analyze whether there is any specific frequency content in the image in a specific direction. Gabor filters are special classes of band-pass filters, i.e., they allow a certain 'band' of frequencies and reject the others. Frequency and orientation representations of Gabor filters are claimed by many vision scientists to be similar to those of the human visual system.

In the spatial domain, a 2D Gabor filter is a Gaussian kernel function modulated by a sinusoidal wave of particular frequency and orientation. One such 2D Gabor filter is shown on the slide: a 2-D Gaussian kernel modulated by a sinusoid oriented 30∘ with X-axis. A sinusoid is sometimes called the carrier, and a 2-D Gaussian-shaped function is called the envelope. Filtering an image with such a filter will detect edges in a particular direction. In practice a set of Gabor filters with different scales and orientations is commonly used. This set of filters is often called a filter bank.

For example, this slide shows a Gabor filter bank of 40 filters for 5 scales and 8 orientations. Simply speaking, Gabor filters detecting image gradients of a specific orientation. The convolution of image patches with a number of different scales and orientations allows for extraction and encoding of local texture information into a low dimensional feature vector.

The filters are convolved with the original image, resulting in a so-called Gabor space. Then response energy can be computed for every filter, and these energies can form a texture feature vector for a given image.

ICA filters Although Gabor filters are considered to be a good approximation of the characteristics of certain cells in the human visual cortex, they are not data-driven. One example of data-driven filters (prior to wide spread of deep learning) is ICA filters, where ICA stands for Independent Components Analysis. These filters can also be used for texture analysis. They are constructed using Independent Component Analysis. A representative set of images is used to "learn" "independent components". In its simplest form, ICA is an algorithm that search for a linear transformation that minimizes the statistical dependence between the components of an input vector. When ICA is applied to natural images, it produces sets of visual filters which look like simple cells in primary visual cortex. They are spatially localized, oriented and selective to structure at different spatial scales. These filters are primitives that represent the spatial patterns occurring in the different scenes. Each image patch is can be represented as a combination of a set of these primitives. The slide shows examples of ICA filters extracted by the authors from patches of natural images collected from the web.

ICA filters - response model After filters are obtained, with N total filters each image can be characterised by a collection of N responses. The energetic responses of an image $I(x, y)$ to the selected pool of filters are estimated as squares of convolution of an image and every filter. The squaring operation, which correspond to the energy of the response, results from the intrinsic ambiguity about the sign of the signals that are estimated with ICA. The absolute value can also be used instead of squaring.

So, after filtering with N filters we have N responses. We can choose different descriptors (or signatures) now: mean value of responses only, mean and variance, or histogram. Histograms preserve more information about distribution, and this representation was chosen by the authors.

Textures of two images can now be compared by comparing distributions of the responses of those images to ICA filters. The authors of the ICA filters suggested to use Kullback-Leibler divergence to compare response distributions. This is another commonly used measure of how one probability distribution is different from another. It is sometimes also called relative entropy. It is defined as:

$$KL(f_1, f_2) = -\int_{-\infty}^{\infty} f_1(x) \log\left(\frac{f_1(x)}{f_2(x)}\right) \mathrm{d}x \tag{3}$$

Thanks to the concavity of the logarithm function, this measure is positive when f_1 and f_2 are different, and is zero if f_1 is equal to f_2. But it is neither symmetric nor fulfils the triangle inequality. The first drawback can be solved using a symmetric version of this measure:

$$KL_s(f_1, f_2) = KL(f_1, f_2) + KL(f_2, f_2) \tag{4}$$

When histograms are used as the signatures of images, the dissimilarity between the images represented by histograms H_1 and H_2 is computed as:

$$KL_s(H_1, H_2) = \sum_{b=1}^{B} H_1(b) \log\left(\frac{H_1(b)}{H_2(b)}\right) + \sum_{b=1}^{B} H_2(b) \log\left(\frac{H_2(b)}{H_2(b)}\right)$$
$$= \sum_{b=1}^{B} ((H_1(b) - H_2(b)) \log\left(\frac{H_1(b)}{H_2(b)}\right) \tag{5}$$

Here, B is a total number of bins in histograms - both histograms H_1 and H_2 has the same number of bins.

But each image is characterized not by a single histogram, but by a collection of N histograms. Each response can be considered as a density of an independent variable, and the set of N responses as multidimensional distribution. The Kullback-Leibler divergence between two multidimensional distributions with independent components is the sum of the Kullback-Leibler divergences between each component. So with each image represented by N histograms, the dissimilarity between images can be computed as:

$$KL_I(I_1, I_2) = \sum_{i=1}^{N} KL_s(H_{1,i}, H_{2,i}) \tag{6}$$

So for every filter $i = 1 \dots N$, we compute Kullback-Leibler divergence between responses of image I_1 and image I_2 to that filter, and than sum up all these dissimilarities for every filter.