Image Filtering and Object Identification

Alfano Caterina Berardi Angelo Cappelli Dario Fratocchi Emanuele

October 30, 2020

Contents

1 Image Filtering

Digital image filtering methods are usually applied for restoration of noise contaminated images and for the extraction of useful information, like edges.
An important example technique of linear filtering is given by the Gaussian one, which is used to smooth pictures; considering a 2-D convolution, the general formula of the Gaussian linear filter is presented as the following:

$$\frac{1}{2\pi\sigma^2} \exp -\frac{x^2 + y^2}{2\sigma^2}$$

So, our first task was about the development of a method which computes the values of a 1-D vector of Gaussian generated values with a given standard deviation σ , and then to implement a 2-D Gaussian filter, that should be defined for all integer values x in the range [-3 σ, 3σ].
We decided to do it in two different ways:

1. convolving the image with a 2-dimensional kernel created by multiplying our Gaussian vector for his transpose

2. leveraging the separability of Gaussian filtering and therefore computing first the multiplication of every row for our horizontal vector of gaussian values and the the multiplication of every column for the transpose of the vector (vertical vector). The formula above is in fact the combination of these two:

$$\frac{1}{\sqrt{2\pi\sigma}} \exp\left(\frac{x^2}{2\sigma^2}\right) \quad (1) \qquad\qquad \frac{1}{\sqrt{2\pi\sigma}} \exp\left(\frac{y^2}{2\sigma^2}\right) \quad (2)$$

Here is an example of our smoothing filter using the made as in (1) and (2):

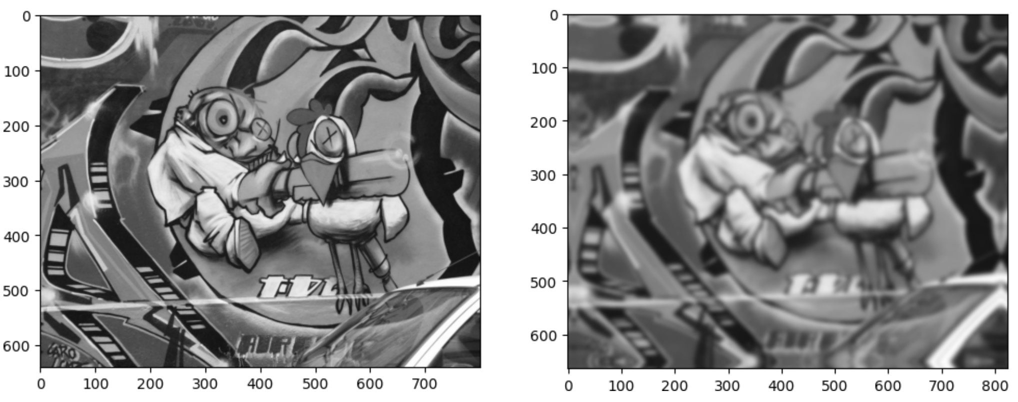


Figure 1: Smoothing with 2-d kernel

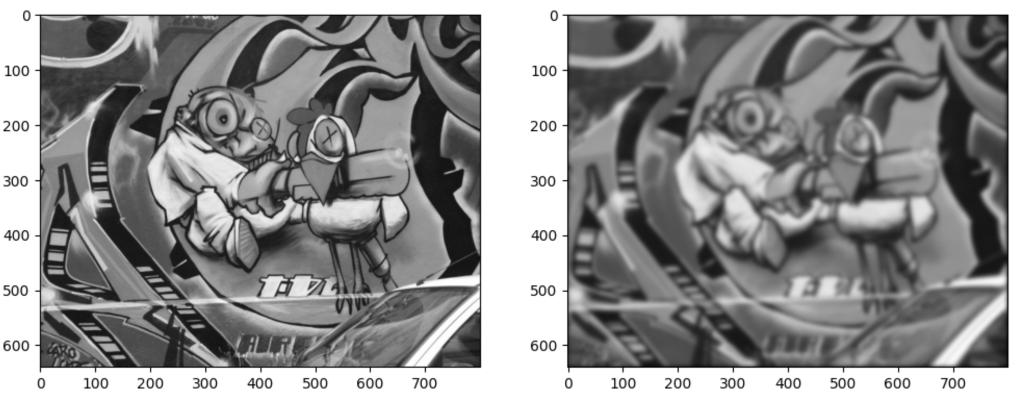


Figure 2: Smoothing using separability

Let's look at the output below as we reason on what it represents. We have to keep in mind that given two Gaussian functions, G^s and G^t where t and s are two values of σ, the convolution of these two functions will produce the same result of applying a single Gaussian kernel with this parameter:

$$G^{\sigma} = G^{\sqrt{s^2+t^2}}$$

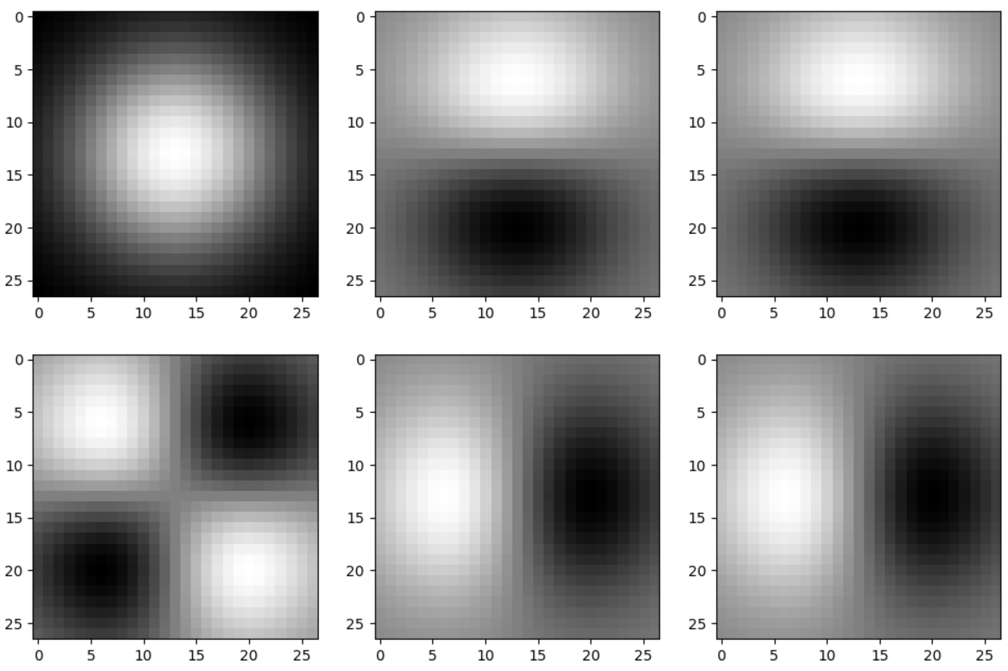
Moreover, in convolution these properties are valid: linearity, associative property, commutative property. This is why the first picture in the top left corner is the result of applying a Gaussian filter with given σ value. In fact, the image was convoluted with a vector of Gaussian values over x first, and with its transpose afterwards (a Gaussian over y with the$\sqrt{}$same σ parameter). We know this is exactly the same as considering a single Gaussian Kernel of shape $G = \sqrt{\sigma^2\sigma^2}$. So, is the Kernel we are applying in this case is the following: $G^\sigma = \overline{G^{\sqrt{s^2+t^2}}} = \overline{G^{\sqrt{\sigma^2+\sigma^2}}} = \bar{G}^{\sqrt{2\sigma^2}}$, so the kernel we are applying in this case is the following:

$$\frac{1}{2\sigma\sqrt{\pi}}\exp\left(\frac{x^2+y^2}{4\sigma^2}\right)$$

This picture can also be seen as the two-dimensional representation of a 3-d Gaussian seen from above, where the white represents the peak.

For the following cases we take in consideration all the previous considerations.
The second and third image are the results of applying the filter only along the y axis. They are in fact obtained by making two convolutions: first with the Gx vector and later with the transpose of the Dx derivative (DxT can be considered as: *$y/\sigma^2 * Gx$*, basically the Dy derivative). The order in which the two operations are computed is not important, in fact the two pictures are the same; this is because convolution is associative.
Going back to the 3-d representation, the white part would be a peak, while the black one a lower point of the curve. The same thing applies for the last two pictures of the second row, but this time we have the Dx derivative (*$x/\sigma^2 * Gx$*).

Finally, the first picture of the second row is the convolution of the two derivatives (one along the x axis and the other one along the y axis) that results in a second derivative along x an dy; this is represented by a more complex shape with two peaks and two low points. We can see a better representation of the 3D structures below (source http://campar.in.tum.de/Chair/HaukeHeibelGaussianDerivatives)



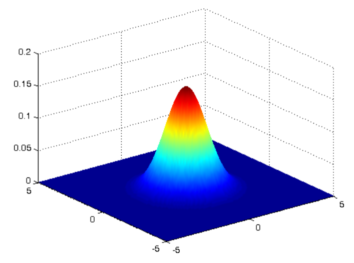


Figure 3: 3D representation of a Gaussian

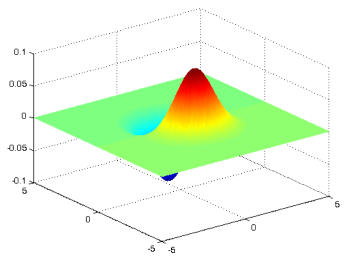


Figure 4: 3D representation of of Dx derivative of a Gaussian

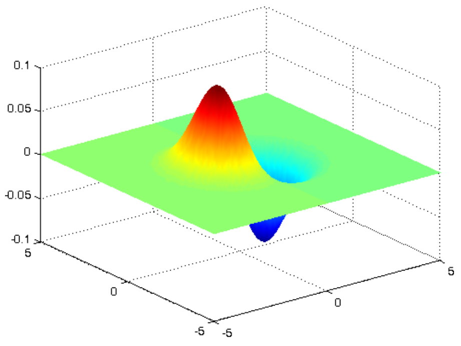


Figure 5: 3D representation of of Dy derivative of a Gaussian

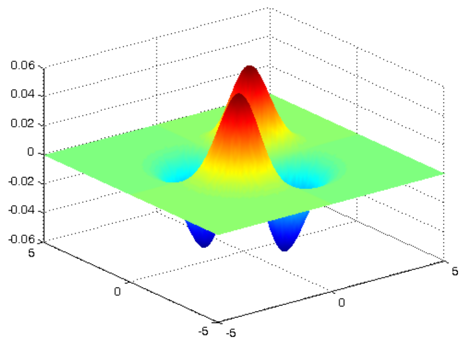


Figure 6: 3D representation of a second derivative along both axis of a Gaussian

Gauss derivatives

Derivatives are important for edge detection. To do this we used the gaussdx function to have a vector of values according to the Gauss derivative along the x axis given by:

$$\frac{1}{\sqrt{2\pi\sigma^3}}\mathrm{x} * \exp - \left(\frac{x^2}{2\sigma^2}\right)$$

Then we were able to use this vector in convolution with the image to get the Dx derivative, and its transposein convolution with the image to get the Dy derivative.
It should be noted that the image we were working with is not the one passed as input but it is its blurred version. In fact, blurring the image with a Gaussian filter before computing the derivative it's really important because it reduces the noise and allows us to detect only the important edges. We can see the difference right below. Smoothing the image had a huge impact on the image identification performance of the dxdy histograms. Looking with our eyes of course we can see a lot more details without the smoothing, but when it comes to image identification this are all unnecessary details that are going to make our results worse.

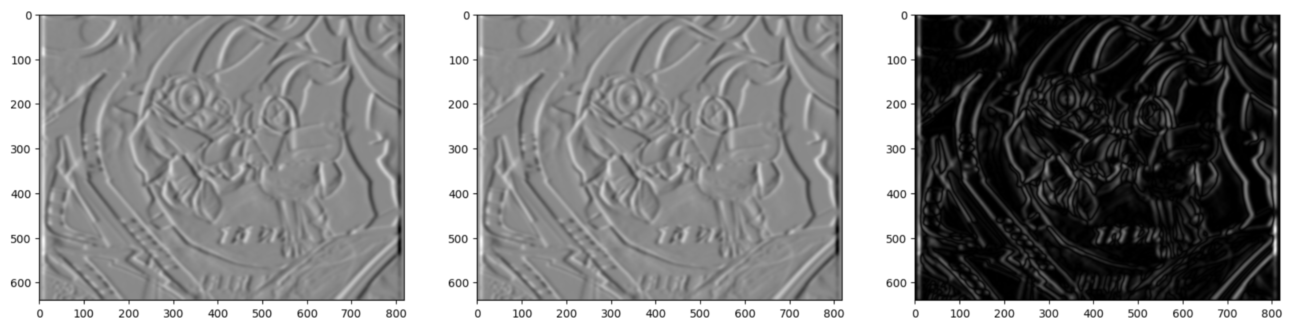


Figure 7: Derivatives with smoothing

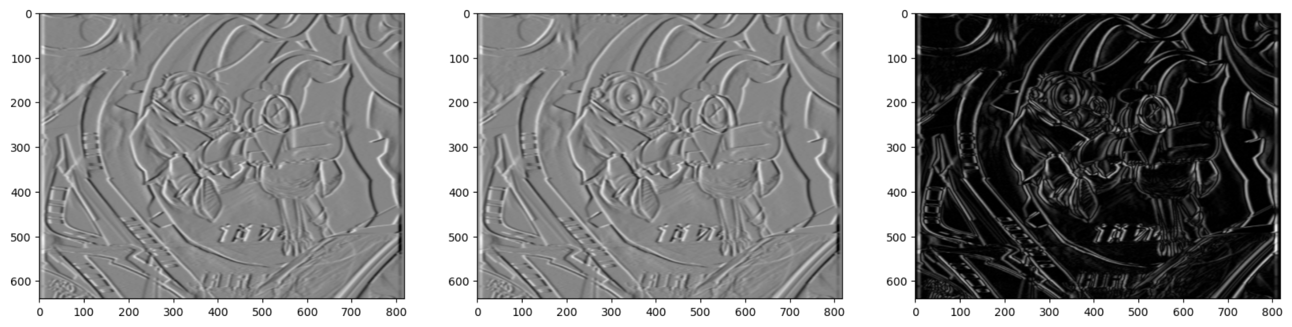


Figure 8: Derivatives without smoothing

2 Image Representations, Histogram Distances

The histogram module is the core part of the script, it allows to study the images and find characteristics that will later be compared. All the histogram functions have the same structure even if the data structures have different dimensions:

1. Computing the bin edges to store in a vector
2. Looping into every pixel of the image
3. Checking in which bin the pixel (or channel) belongs to by checking the rest of the division of the pixel (or channel) value by 255.
4. Adding a +1 to the count related to that specific bin
5. Flattening the array of the histogram (when necessary)
6. Normalizing the results

The dxdy histogram function has some more steps. In fact, before looping through the pixels we have to apply the gaussderiv function so that we can detect the edges and cap the values that exceed the interval [-6,6]. Finally we can loop through each pixel of the two images at the same time. This time, since the values could also be negative we decided to use the bisect function to find out the correct bin for the pixel.

For the distanced module we implemented the given distance formulas simplifying them when possible. Specifically, the intersect function could be reduced to a single sum, because the denominators are equal to 1 (the arrays are normalized). We would then have the sum of two identical numbers and a division by two. This is why we can directly compute just one sum. The two other formulas are exactly as stated in the instructions.

3 Object Identification

In this section we focused on using the histograms to analyze the similarities between pictures. The find- best-match function works by calling the correct histogram function (thanks to compute-histograms), having previously identified if the picture should be colored or in black and white. After computing the histograms of every query and model image, it proceeds to fill the distance matrix (according to the specified function) finding the distance of every possible couple query-model. Finally, the function looks for the minimum value in every column and returns is as the best match.

For question 3.b we used the previously implemented find-best-match function to get the matrix of distances amongst all queries and model images. Then for every query received as input we looked for the 5 most similar images and plotted them alongside the query.

Playing with combinations

We decided to test our script on 36 different combinations, to see how good all the histograms and distance functions performed with 3 possible numbers of bins: 10, 15 and 20. The script we used to perform this task is inside the ”precisino.py” file.

These are the results we obtained:

COMBINATION	RECOGNITION RATE
chi2, dxdy, 10	0.55
chi2, dxdy, 15	0.48
chi2, dxdy, 20	0.51
l2, dxdy, 10	0.54
l2, dxdy, 15	0.48
l2, dxdy, 20	0.51
intersect, dxdy, 10	0.65
intersect, dxdy, 15	0.65
intersect, dxdy, 20	0.67
chi2, rg, 10	0.58
chi2, rg, 15	0.6
chi2, rg, 20	0.54
l2, rg, 10	0.58
l2, rg, 15	0.58
l2, rg, 20	0.48
intersect, rg, 10	0.7
intersect, rg, 15	0.83
intersect, rg, 20	0.73
chi2, rgb, 10	0.66
chi2, rgb, 15	0.64
chi2, rgb, 20	0.53
l2, rgb, 10	0.61
l2, rgb, 15	0.6
l2, rgb, 20	0.48
intersect, rgb, 10	0.79
intersect, rgb, 15	**0.89**
intersect, rgb, 20	0.8
chi2, grayvalue, 10	0.46
chi2, grayvalue, 15	0.4
chi2, grayvalue, 20	0.39
l2, grayvalue, 10	0.44
l2, grayvalue, 15	0.4
l2, grayvalue, 20	0.38
intersect, grayvalue, 10	0.51
intersect, grayvalue, 15	0.45
intersect, grayvalue, 20	0.52

As we would expect we got the best result (0.89) in identifying the pictures while using the rgb histograms (that consider all three channels) combined with the ”intersect” distance function. In fact, as shown also by 4.2, the precision-recall curve of the intersect function is the one that performs best in rg and rgb histograms. We can also see that on average the dxdy or grayvalue histograms have worse results. This is because not considering colors has a huge impact on the identification problem (even if shapes are considered, as for dxdy). Another interesting fact is that, even though increasing the number of bins is also a key factor in improving the precision, when the bins become too much the recognition rate gets smaller, at least for rgb and rg function. The dxdy and grayvalue histograms, which are less precise, can instead still get some benefits from the addition of more bins.

4 Performance Evaluation

For this last section we implemented the plot-rpc function which computes the number of true positive, false positive and false negative on a given distance matrix using different thresholds for the distances. We decided to use 1000 thresholds to see in details how the precision recall curve behaved. Since every distance function has a different range of possible outcomes we couldn’t have fixed thresholds for the three curves. This is why the thresholds are different every time and based on the values inside the matrix. But to ensure an equal amount of thresholds and a proportional step between two consecutive thresholds while using different distance function we used the difference between the maximum and minimum value, divided it by 1000 and computed steps of this length.

We tested the histogram functions with three different bin values. Initially we tried using 20 bins and as we can see from these plots, while working with rgb and rg histograms the intersect function is the one that performs better even with really small thresholds. On the other hand, chi2 and l2 start off in a similar way but, with growing thresholds, chi2 start having best results than l2 until the threshold becomes so big that chi2 and l2 have again a similar behaviour.

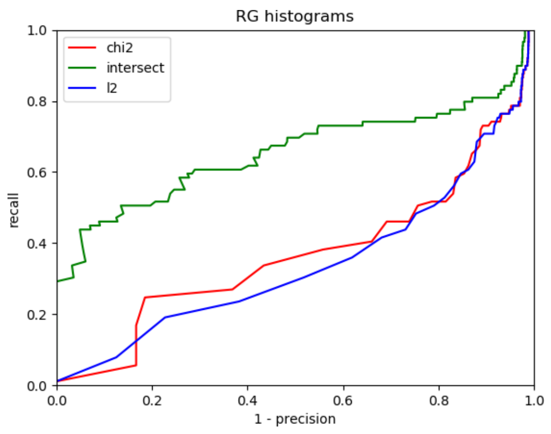


Figure 9: Performance assessment with RG histograms and 20 bins

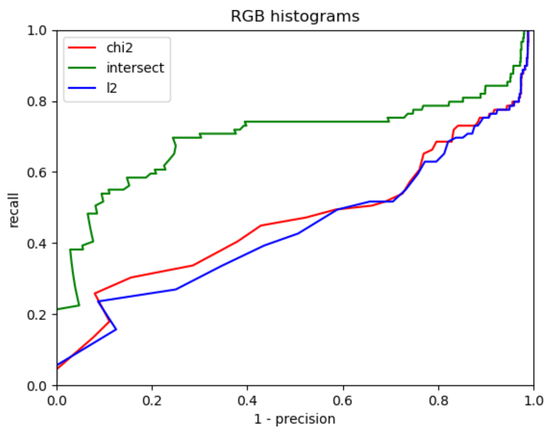


Figure 10: Performance assessment with RGB histograms and 20 bins

Something different happens if we decide to use the dxdy histograms. In this case the three functions have a similar behaviour at the beginning. As thresholds' values increase, chi2 and l2 start having better results than intersect, until the situation reverses itself once again with the biggest thresholds.

Generally speaking the precision and recall values are higher if we use color channel instead of black and white photos, and this is why the curves of the dxdy and grayvalue plot are lower that the rgb/rg ones. We can also notice that the grayvalue histograms only work in an acceptable way when using bigger thresholds.

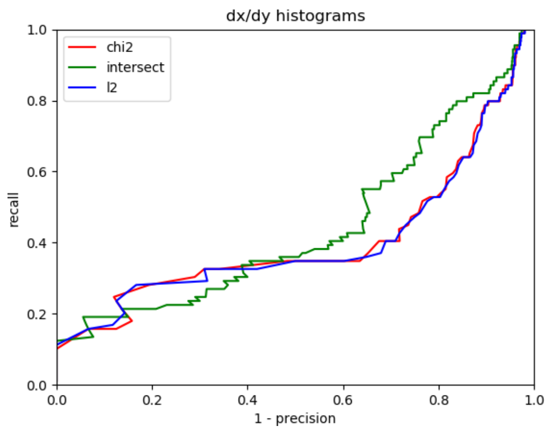


Figure 11: Performance assessment with DxDy histograms and 20 bins

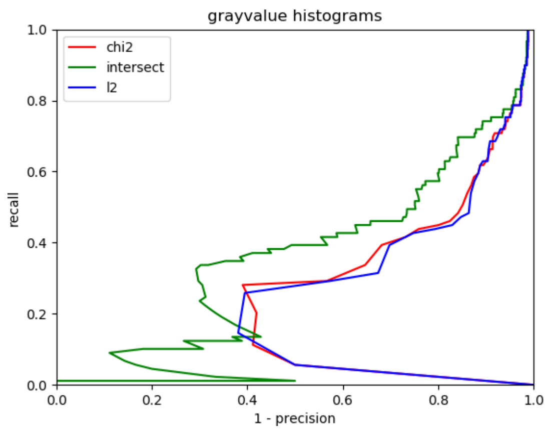


Figure 12: Performance assessment with grayvalue histograms and 20 bins

Then we decided to run the same functions using 15 bin, which according to our table in 3.c is the optimal quantity for most combinations. From the results that the curves are actually performing better, especially the intersect ones. The grayvalue function also has a better overall behaviour, with a uniform result from the three distance functions.

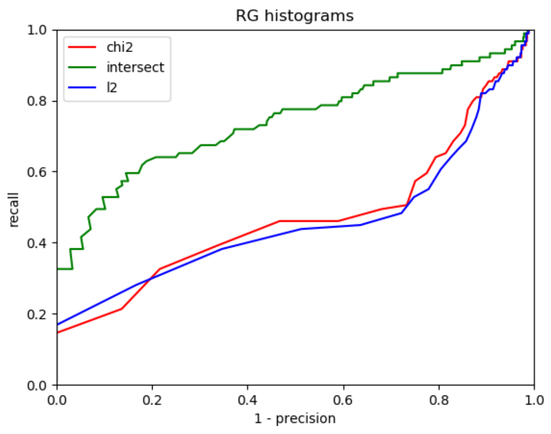


Figure 13: Performance assessment with RG histograms and 15 bins

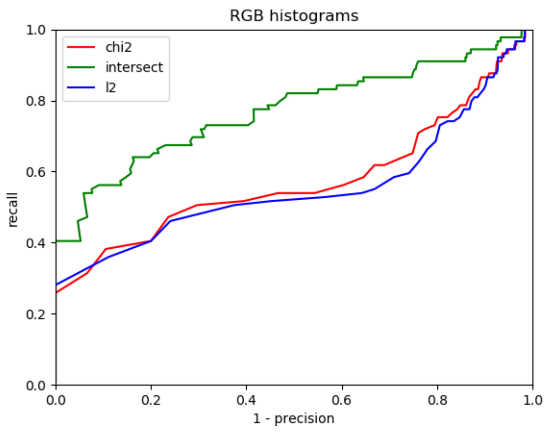


Figure 14: Performance assessment with RGB histograms and 15 bins

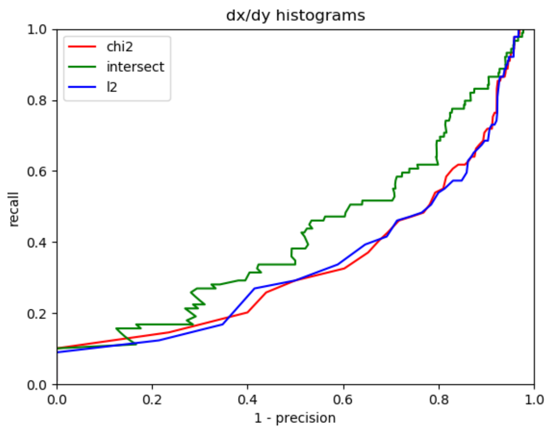


Figure 15: Performance assessment with DxDy histograms and 15 bins

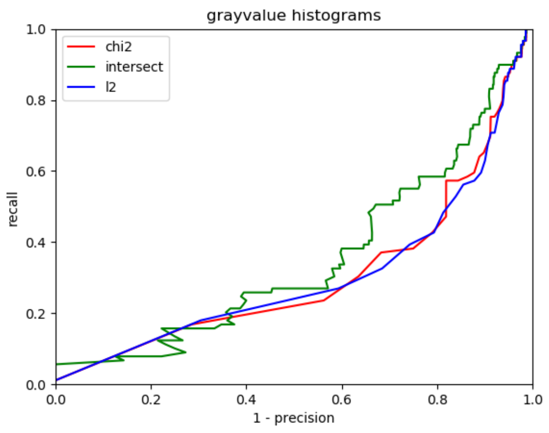


Figure 16: Performance assessment with grayvalue histograms and 15 bins

Finally we did the same things with 25 bins to see how adding more precision and details in the categorization of images will impact the precision-recal curves. We found out that the RG and RGB histograms weren't affected much (only the chi2 and l2 curves have some differences). Even with the dxdy and grayvalue histograms the difference are not many, and it's hard to see at a glance how the curves are altered and in which points.

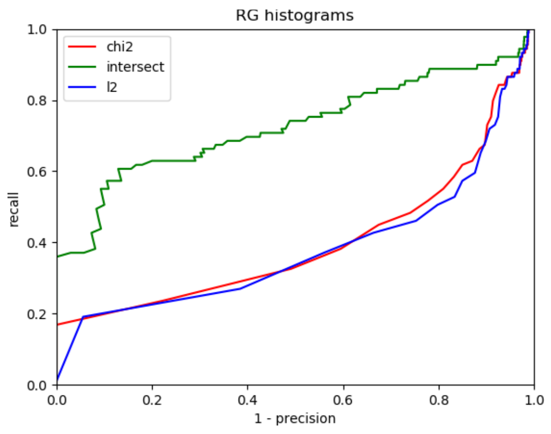


Figure 17: Performance assessment with RG histograms and 25 bins

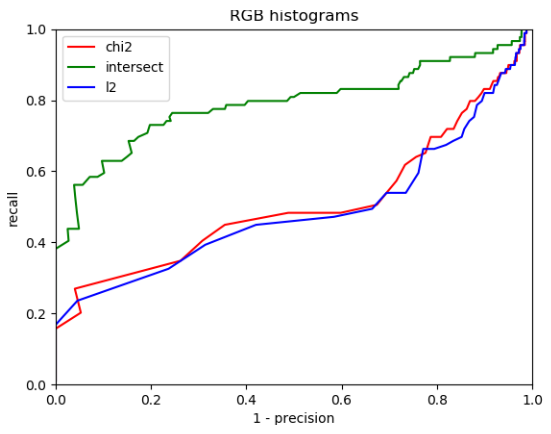


Figure 18: Performance assessment with RGB histograms and 25 bins

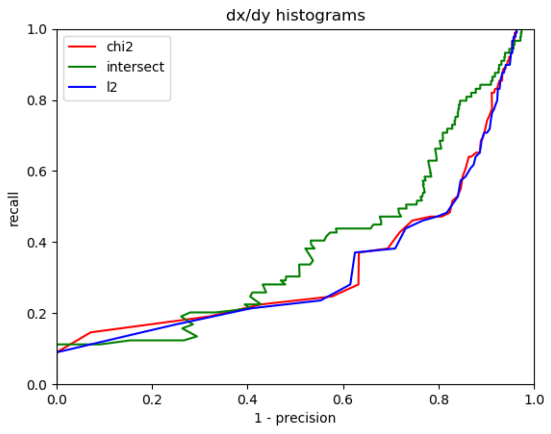


Figure 19: Performance assessment with DxDy histograms and 25 bins

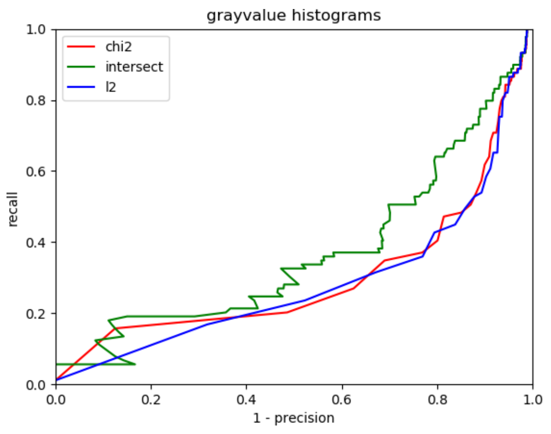


Figure 20: Performance assessment with grayvalue histograms and 25 bins