A Fast Approximation Algorithm for the Gaussian Filter

Kentaro Imajo (Yamamoto Laboratory)
I propose a fast approximation algorithm for the Gaussian filter, which computes one Gaussian-filtered pixel in $O(1)$ in the area size $n \propto \sigma^2$. The naïve method requires $O(n)$ time.
Application: Binary Classifier

Extending the key algorithm,

- The proposed method speeds up prediction of SVMs with the Gaussian kernel,
- The proposed method speeds up training with small loss of accuracy.

Training (pre-computation):
\[ O(n^2) \rightarrow O(4^d n \log^d n) \]

Prediction:
\[ O(n) \rightarrow O(4^d \log^d n) \]
Outline

1. **Key Algorithm:**
   A fast approximation algorithm for the Gaussian filter
   Kentaro Imao, "Fast Gaussian Filtering Algorithm Using Splines,"
The 21st International Conference on Pattern Recognition, 2012.

2. **Application:** Binary classification
   Kentaro Imao, Otaki Keisuke, Yamamoto Akihiro,
   "Binary Classification Using Fast Gaussian Filtering Algorithm,"
   Algorithms for Large-Scale Information Processing in Knowledge Discovery, 2012.

3. Conclusions
1. Key Algorithm: Gaussian Filtering
2. Application: Binary Classification
3. Conclusions
Summary of Key Algorithm

It computes one Gaussian-filtered pixel in constant time in the area size \( n (\propto \sigma^2) \) within 3% error.

The Gaussian filter is convolution with the 2D Gaussian Function:

\[
\text{where } \frac{1}{\sqrt{2\pi\sigma}} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right), \text{ the parameter } \sigma \text{ is variance.}
\]
Problem

To compute one Gaussian-filtered pixel,

- The Naïve method takes linear time in the area size of the Gaussian function.
- An FFT method cannot compute pixels apart because FFT depends on all pixels.
- A down-sampling method has 52% error.
Solution

We approximate the Gaussian function using a spline function:

• A spline function becomes discrete if it is differentiated some times.
  – Convolution of an image and a discrete function is computed fast.

A spline function is a piecewise polynomial function that is written as

\[ \tilde{\psi}(x) = \sum_i a_i (x - b_i)^n_+, \]

where \((\cdot)_+\) is \(\max(\cdot, 0)\).
\[ \tilde{\psi}(x) = 3(x + 11)^2_+ - 11(x + 3)^2_+ + 11(x - 3)^2_+ - 3(x - 11)^2_+ \]

\[ 12 \times 2.657 \times 10^2 \exp \left( -\frac{x^2}{5.2720^2} \right), \]

where \((\cdot)_+\) is \(\max(\cdot, 0)\).
\[ \tilde{\psi}(x) = 3(x + 11)^2_+ - 11(x + 3)^2_+ + 11(x - 3)^2_+ - 3(x - 11)^2_+ \]

\[ \geq 2.657 \times 10^2 \exp \left( -\frac{x^2}{5.2720^2} \right) , \]

where \( (\cdot)_+ \) is \( \max(\cdot, 0) \).
The convolution can be reduced by differentiating a spline function $\tilde{\psi}$ until it becomes discrete.

Intuition:

$$\psi \ast [I](x) = \left( \frac{d}{dx} \psi \right) \ast \left( \int I(x) \, dx \right)(x)$$

This can be computed fast.
Nature of a Spline Function

Differentiating a spline function, we can get a discrete function:

The Gaussian-like approximation function can also become a discrete function!
Evaluation

Approximation error is about 2%.

\[
\left( \int |\tilde{\psi}(x) - \psi(x)| \, dx \right) / \left( \int |\psi(x)| \, dx \right)
\]

Higher order improves more:

Approximation error of 4\textsuperscript{th} order approximation is about 0.3%.

\[
\tilde{\psi}(x) = 70(x + 22)^4_+ - 624(x + 11)^4_+ + 1331(x + 4)^4_+ - 1331(x - 4)^4_+ + 624(x - 11)^4_+ - 70(x - 22)^4_+.
\]
Approximation of 2D

\[ \tilde{\psi}(x, y) = (3(x + 11)^2 + 11(x + 3)^2 + 11(x - 3)^2 - 3(x - 11)^2) \\ (3(y + 11)^2 + 11(y + 3)^2 + 11(y - 3)^2 - 3(y - 11)^2) \]

Contour lines

- Gaussian
- Approximation
- Control point
2D Convolution

The 2D Gaussian function is decomposed:

\[
\exp \left( -\frac{x^2 + y^2}{2\sigma^2} \right) = \exp \left( -\frac{x^2}{2\sigma^2} \right) \exp \left( -\frac{y^2}{2\sigma^2} \right)
\]

The approximation can be defined as:

\[
(\tilde{\psi} \ast I)(x, y) = \sum_{i=1}^{m} a_i \sum_{j=1}^{m} a_j J(x - b_i, y - b_j)
\]

where \( J(x, y) = \sum_{(\Delta x, \Delta y) \in \mathbb{Z}_+^2} \Delta x^n \Delta y^n I(x - \Delta x, y - \Delta y) \)

\( O(m^2) \) linear combination, where \( m \) is \# of knots for each dimension and \( m \) is equal to 4.
### Experiments

#### Figure

Computational time for one pixel on average

<table>
<thead>
<tr>
<th># of application pixels of the filter</th>
<th>Computational time (sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$10^0$</td>
<td>$10^{-8}$</td>
</tr>
<tr>
<td>$10^1$</td>
<td>$10^{-7}$</td>
</tr>
<tr>
<td>$10^2$</td>
<td>$10^{-6}$</td>
</tr>
<tr>
<td>$10^3$</td>
<td>$10^{-5}$</td>
</tr>
<tr>
<td>$10^4$</td>
<td>$10^{-4}$</td>
</tr>
</tbody>
</table>

For 70+ pixels our algorithm is faster

Diameter of 9 has 69 pixels

---

#### Key Algorithm: Gaussian Filtering

- **Naïve method**
  - $O(NM)$
  - $O(M)$

- **FFT method**
  - $O(N \log N)$
  - N/A

- **Proposed method**
  - $O(N)$ at first
  - $O(N)$
  - $O(1)$

Table 1. Comparison of computational order

$N$ and $M$ is the size of a source image and the Gaussian filter respectively.

N/A indicates that the FFT method cannot compute values apart.

#### Experiments

Comparing with the naïve method and the Fast Fourier Transform method, we evaluate the proposed method with a $1024 \times 1024$ gray-scale image and the Gaussian filter whose the parameter $\sigma$ is 47 and the application area is $99 \times 99$ pixels. We implemented them in C++ and compiled them with GNU C++ Compiler and execute them on Mac Pro (6-core 2.93GHz Xeon). As the result of computation, the naïve method took 13.7 seconds, and the FFT method took 0.373 seconds, and the proposed method took 0.102 seconds. In this case, the proposed method can do it more than 100 times as fast as the naïve method does.

Next, we evaluate computational time to figure out a filtered value of one pixel. Table 1 shows computational order of the precomputation and computation of all pixels and computation of one pixel for each method. Computational time of the naïve method depends on application area of the filter. Figure 4 shows the proposed method is faster than the naïve method if application area of the filter is larger than $8 \times 8$.

#### Conclusion

We proposed a fast method to apply the Gaussian filter to a 2D image. In order to compute one Gaussian filtered pixel value, the naïve method spends $O(M)$ time, where $M$ is the size of the Gaussian filter. While the FFT method compute all Gaussian filtered pixel values simultaneously in $O(N \log N)$ time, where $N$ is the size of a source image, it cannot compute them sparsely.

The proposed method computes one Gaussian filtered pixel value in constant time on the size of a source image and the Gaussian filter.

The proposed method approximates the Gaussian filter function using a spline function whose number of the control points is small. Although precomputation takes time linear in the size of a source image, the precomputation does not depend on the size of the Gaussian filter, so we do not need to recalculate the precomputation even if we use a different size of the Gaussian filter. As a result, the proposed method is more than 100 times as fast as the naïve method when application area is $99 \times 99$ pixels. Furthermore, unlike the FFT method, the proposed method can compute a filtered pixel value at any position with any size of the Gaussian filter in constant time.

#### References


Conclusions of Part 1 (1/2)

We approximate the Gaussian function using a 2\textsuperscript{nd} order spline function:

- It has only 2\% approximation error,
- It has only 4 knots,
- It has unequal intervals of control points while general approximation functions often have equal intervals.
  – It is sufficiently optimized as a spline.
Conclusions of Part 1 (2/2)

Using spline approximation and an integrated image of an input image, we can compute every pixel apart in constant time to the area size.

\[ \int\cdots\int d\mathbf{x}^3 d\mathbf{y}^3 \ast \]
1. Key Algorithm: Gaussian Filtering
2. Application: Binary Classification
3. Conclusions
Extending the key algorithm,

- **The proposed method speeds up prediction of SVMs with the Gaussian kernel,**
  - \(O(n) \rightarrow O(4^d \log^d n)\)

- It speeds up **training** with **small loss** of accuracy.
  - \(O(n^2) \rightarrow O(4^d n \log^d n)\)
Problem

An SVM is a good binary classifier, but
– prediction takes $O(n)$ time,
– training takes $O(n^2) \sim O(n^3)$ time.

We cannot use SVMs if # of data gets large.

Solution

• For prediction, we use the key algorithm.
• For training, we treat data equally with small loss of accuracy.
Key Algorithm

It assumes that an image is given. Since we cannot integrate an image, we propose to use range trees.

Image

Every pixel has a value

Few pixels have a value

Binary Classification

Application: Binary Classification
A range tree [Bentley 1979] is a data structure that can compute the total sum in a box $x_1-x_2$ in $O(\log^d n)$ time with $O(n \log^d n)$ space.

We can calculate the total sum with the sum of $O(\log^d n)$ regions.
The proposed method can decide the class of any point in $O(4^d \log^d n)$ time, and the pre-computation is $O(4^d n \log^d n)$ time.
A range tree can give:

\[ K_{i,j}(x, y) = \sum_{(\Delta x, \Delta y) \in \{(0,0), \ldots, (x,y)\}} \Delta x^i \Delta y^j I(\Delta x, \Delta y). \]

Using 9 range trees, \( J \) can be computed with:

\[
J(x, y) = \begin{pmatrix}
y^2 \\
-2y \\
1
\end{pmatrix}^T \begin{pmatrix}
K_{0,0}(x, y) & K_{1,0}(x, y) & K_{2,0}(x, y) \\
K_{0,1}(x, y) & K_{1,1}(x, y) & K_{2,1}(x, y) \\
K_{0,2}(x, y) & K_{1,2}(x, y) & K_{2,2}(x, y)
\end{pmatrix} \begin{pmatrix}
x^2 \\
-2x \\
1
\end{pmatrix}.
\]
Experiments on Time

Time consumption of prediction:

<table>
<thead>
<tr>
<th># of training data</th>
<th>1,000</th>
<th>10,000</th>
<th>100,000</th>
<th>1,000,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed method</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prediction (Precomputation)</td>
<td>2.43 ms (0.13 s)</td>
<td>2.48 ms (1.48 s)</td>
<td>2.55 ms (17.8 s)</td>
<td>2.96 ms (203 s)</td>
</tr>
<tr>
<td>LIBSVM</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Prediction (Training)</td>
<td>0.02 ms (0.05 s)</td>
<td>0.22 ms (4.47 s)</td>
<td>2.29 ms (442 s)</td>
<td>23.6 ms (1000+ s)</td>
</tr>
</tbody>
</table>

The proposed method is faster when # of training data is over 150,000.
Experiments on Accuracy

If we treat input data equally, **accuracy** of a new classifier and LIBSVM:

**Table** Accuracy for Iris Flower Data Set

<table>
<thead>
<tr>
<th>Method</th>
<th>$\sigma = 0.1$</th>
<th>$\sigma = 0.3$</th>
<th>$\sigma = 1$</th>
<th>$\sigma = 3$</th>
<th>$\sigma = 10$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proposed</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$C = 0.1$</td>
<td>93%</td>
<td>93%</td>
<td>93%</td>
<td>86%</td>
<td>84%</td>
</tr>
<tr>
<td>$C = 1$</td>
<td>58%</td>
<td>83%</td>
<td>93%</td>
<td>90%</td>
<td>85%</td>
</tr>
<tr>
<td>$C = 10$</td>
<td>68%</td>
<td>94%</td>
<td>95%</td>
<td>94%</td>
<td>90%</td>
</tr>
<tr>
<td>$C = 10^6$</td>
<td>69%</td>
<td>93%</td>
<td>93%</td>
<td>95%</td>
<td>93%</td>
</tr>
</tbody>
</table>

The loss of accuracy is small.
Conclusions of Part 2

- It speeds up prediction of SVM: $O(n) \rightarrow O(4^d \log^d n)$.
- With small loss of accuracy, it speeds up training: $O(n^2) \rightarrow O(4^d \log^d n)$.

The proposed method consists of:
- The key algorithm: A fast approximation algorithm for the Gaussian filter,
- Range trees.
1. Key Algorithm: Gaussian Filtering
2. Application: Binary Classification
3. Conclusions
Conclusions

**Key Algorithm:** A fast approximation algorithm for the Gaussian filter:
- It uses a spline function for approximation.
- It speeds up Gaussian filtering, which is often used in image processing.

**Application:** Binary classification
- It extends the key algorithm with range trees.
- It speeds up prediction of SVMs.
- It speeds up training with small loss of accuracy.
Related Achievements

International Conference (Reviewed):


Other Achievements

International Conference (Reviewed):
• Takuya Akiba, Kentaro Imajo, Daisuke Okanohara, "Engineering Parallel String Sorting Algorithms,” Algorithms for Large-Scale Information Processing in Knowledge Discovery, 2011.

Journal:

Programming Contests:
• ICFP Programming Contest, the 2nd place, September 2011.
• ACM/ICPC Asia Regional Contest in Fukuoka, Japan, the 5th place, November 2011.
• ACM/ICPC Asia Regional Contest in Hsinchu, Taiwan, the 3rd place, November 2011.
• ACM/ICPC World Finals 2012, an 18th place tie, May 2012.