The Chan-Vese Multi-Phase Segmentation Model:

Implementation and Application to Noisy Images

Evan Murphy   Dr. Marco Viola

University College Dublin Undergraduate Summer Research Project

1. Abstract

The aim of this project is to describe and implement the Chan-Vese model for the multi-phase segmentation of images and test its performances in the presence of noise. We begin by discussing some basics of mathematical image processing relevant to this subject. We then consider an overview of existing models for two-phase segmentation, including the methods of Mumford and Shah, and Chan and Vese, which can be extended in a natural way to fit a multi-phase setting. We explicitly describe this for the case of four regions segmentation. We present the numerical algorithm developed in Python (based on a two-phase algorithm included in the scikit-image toolbox) for the solution of the four-phase Chan-Vese model and conduct some experimentation aimed at demonstrating the functionality of this code on a variety of simple and more detailed images. Finally, we address the issue of noise in images and show how cartoon-texture decomposition techniques can increase the robustness of the original model with respect to noise. [3]

2. Introduction to Mathematical Image Processing

In what follows we will introduce some basic concepts of image processing and some mathematical prerequisites relevant to the rest of this report.

2.1. What is an Image?

In mathematical image processing, one typically defines an image as a function $f : \Omega \to \mathbb{R}$ where $\Omega \subseteq \mathbb{R}^2$. In a theoretical setting, allowing the image domain to take on values from the entire plane opens the doors to some interesting possibilities. In practice however, images are made up of a finite number of pixels. To accommodate this, one can naturally define a discretisation of Ω through a grid of points (x_i, y_i) and represent the original (continuous) image as a matrix in which one assigns the value $f(x_m, y_n)$ to the $(m, n)^{\text{th}}$ entry of the matrix.

2.2. What is a Segmentation?

In many areas of mathematics, one may encounter the notion of a partition. In the context of mathematical image processing, a partition of an image Ω is a collection of pairwise disjoint subsets of Ω, (Ω_i) say, whose union returns the original image. That is,

1. $\Omega_i \subseteq \Omega,\ \forall i$,
2. $\overset{\circ}{\Omega}_i \cap \overset{\circ}{\Omega}_j = \varnothing\ \forall i \neq j$,
3. $\bigcup_i \Omega_i = \Omega$,

where we indicated as $\overset{\circ}{\Omega}_i$ the interior of the subset Ω_i. When we talk about a segmentation of an image, we are referring to a partition of the same image, with the additional

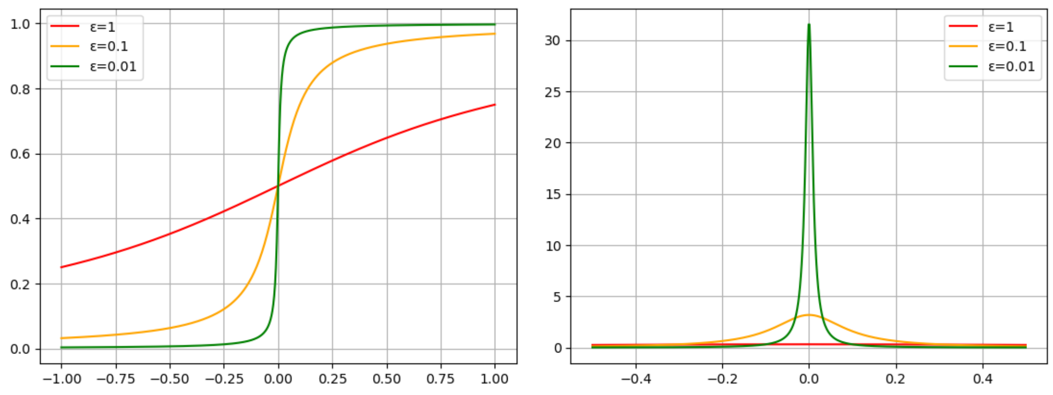

Figure 1: Visualisation of H_ε (left) and δ_ε (right) as ε decreases.

requirement that it reveals some information about the image. This somewhat vague stipulation can be interpreted in different ways. For example, we may wish for our segmentation to separate the foreground and background of an image, or to distinguish between different components of a single object in an image.

2.3. The Heaviside Function

Given a domain $D \subseteq \mathbb{R}$, one defines the Heaviside function $H : D \to \{0, 1\}$ given by

$$H(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases} .$$

Correspondingly, we define its derivative on D as

$$H'(x) = \delta(x) = \begin{cases} \infty, & x = 0 \\ 0, & x \neq 0 \end{cases} .$$

Whilst these two functions are useful definitions, in practice, their discontinuous nature can sometimes cause problems. To resolve this, we may regularise the functions in terms of some parameter ε. We write

$$H_\varepsilon(x) = \frac{1}{2}\left(1 + \frac{2}{\pi}\arctan\left(\frac{x}{\varepsilon}\right)\right)$$

for the regularised Heaviside function, and for its derivative, we write

$$\delta_\varepsilon(x) = H'_\varepsilon(x) = \frac{\varepsilon}{\pi(\varepsilon^2 + x^2)}.$$

As desired, one may notice that as $\varepsilon \to 0$, we have $H_\varepsilon \to H$ and $\delta_\varepsilon \to \delta$.

2.4. Level Set Functions

Another useful tool we will require is the notion of a level set function. A level set function is a representation of a closed curve Γ in a two dimensional plane by the zeroes of a function, φ, say. We express the curve Γ as

$$\Gamma = \{(x, y) : \varphi(x, y) = 0\}.$$

Typically, the function φ maps from a domain $\Omega \subseteq \mathbb{R}^2$ to $\mathbb{R}$ and has the following properties. For any $(x, y) \in \Omega$,

$$\begin{cases} \varphi(x,y) < 0, & \text{for } (x,y) \text{ outside } \Gamma \\ \varphi(x,y) = 0, & \text{for } (x,y) \text{ on } \Gamma \\ \varphi(x,y) > 0, & \text{for } (x,y) \text{ inside } \Gamma. \end{cases}$$

A detailed exploration of level set functions and their application to image segmentation is given in [4].

2.5. Discrete Gradient

As previously mentioned, one may discretise a continuous image (a function) into matrix form. It is thus important to define some analogue of the continuous gradient in the discrete setting. Given a matrix representation of an image $X = (x_{i,j})$, we define the gradient in three different ways, as follows.

Backward Gradient:

$$\begin{aligned} \nabla_x^- \, x_{i,j} &= x_{i,j} - x_{i-1,j} \\ \nabla_y^- \, x_{i,j} &= x_{i,j} - x_{i,j-1}. \end{aligned}$$

Forward Gradient:

$$\begin{aligned} \nabla_x^+ \, x_{i,j} &= x_{i+1,j} - x_{i,j} \\ \nabla_y^+ \, x_{i,j} &= x_{i,j+1} - x_{i,j}. \end{aligned}$$

Central Gradient:

$$\begin{aligned} \nabla_x^0 \, x_{i,j} &= \frac{1}{2}(x_{i+1,j} - x_{i-1,j}) \\ \nabla_y^0 \, x_{i,j} &= \frac{1}{2}(x_{i,j+1} - x_{i,j-1}). \end{aligned}$$

Each of these three definitions are useful in practice. On occasion, one may encounter issues when applying the gradient in a discrete setting. In particular, calculating the discrete gradient at the edge of the matrix. In this case one may choose the formulation of gradient that best suits the setting, using the forward gradient at the left/top edges and the backwards gradient at the right/bottom edges. Alternatively, one may 'pad' the matrix, adding an additional row/column at each edge of the matrix by duplicating the existing edge vales. With this solution, all three gradients are defined at the edges.

3. Chan-Vese Two-Phase Segmentation

We will now introduce a family of variational mathematical models for the solution of the image segmentation problem. The basis for this will be a two-phase segmentation, which will then be extended to fit a multi-phase setting.

3.1. Mumford-Shah

The aforementioned Chan-Vese model [2] is based upon the simpler Mumford-Shah approach [5]. The aim of this model is to produce a two-phase segmentation for an image $f : \Omega \to \mathbb{R}$ into regions $(\Omega_i)_i$. The idea behind this model is to build a function f^* approximating the image f and define the segmentation through the couple $S = (u^*, f^*)$, where $u^* : [0,1] \to \Omega$ is a curve representing the boundaries of the subsets composing the segmentation, associated to certain level sets of f^*.

Under these assumptions, one seeks to define an energy functional E, comprised of a fidelity term $\mathcal{F}$, which measures the segmentations faithfulness to the original image, and a penalty term $\mathcal{P}$, which can be used to promote certain properties of the segmentation. Hence we write,

$$S = (f^*, u^*) = \arg\min_{(f,u)} E(f,u) = \arg\min_{(f,u)} (\mathcal{F}(f,u) + \lambda\mathcal{P}(f,u)),$$

observing the that final segmentation is obtained by minimising the energy functional. Specifically, Mumford and Shah proposed the following fidelity and energy terms, writing the energy functional as

$$E(u,f) = \underbrace{\int_\Omega (f(x) - f_0)^2 \, dx}_{\mathcal{F}} + \lambda \underbrace{\int_{\Omega \setminus u} |\nabla I|^2 \, dx + \mu \mathrm{len}(u)}_{\mathcal{P}} .$$

The fidelity term $\mathcal{F}$ measures the difference between the segmentation and the original image, whilst the penalty term $\mathcal{P}$ simultaneously controls the variation of the segmentation and the length of the boundary.

3.2. Chan-Vese

Chan and Vese [2] improved this functional to better suit the setting of a two-phase segmentation. They considered the segmentation as a piecewise constant function, with a range of two non-negative values, c_1 and c_2. These two values represent the two regions of the segmentation. Their amended energy functional is described as

$$\begin{aligned} E(f, c_1, c_2) = {} & \mu \mathrm{Length}(C) + \nu \mathrm{Area}(\mathrm{insider}(C)) \\ & + \lambda_1 \int_{\mathrm{inside}(C)} (f(x) - c_1)^2 \, dx + \lambda_2 \int_{\mathrm{outside(C)}} (f(x) - c_2)^2 \, dx. \end{aligned}$$

Here, C represents the boundary between the two regions, and inside(C),outside(C) represent inside and outside the boundary C respectively.

3.3. Numerical Implementation

The first step to implementing an algorithm to minimise the Chan-Vese functional is to rewrite the expression in terms of level set functions. In this form, the energy functional is described as

$$\begin{aligned} E(c_1, c_2, \varphi) =& \mu \int_\Omega \delta(\varphi(x)) |\nabla \varphi(x)| \, dx + \nu \int_\Omega H(\varphi(x)) \, dx \\ & + \lambda_1 \int_\Omega |f(x) - c_1|^2 H(\varphi(x)) \, dx + \lambda_2 \int_\Omega |f(x) - c_2|^2 (1 - H(\varphi(x))) \, dx. \end{aligned}$$

where the function H represents the Heaviside function described in 2.3. Due to the non-convex and non-linear nature of the energy functional,minimising the above functional is a difficult problem, and finding a unique solution is not feasible. Instead, one may take a heuristic approach, implementing an iterative scheme with the goal of converging upon a reasonable solution. One begins by choosing an initial state for the level set function φ. The aim is then to update the values of c_1 and c_2, and φ, as follows.

Given an estimate for the level set function φ, we compute the derivatives

$$\begin{aligned} \frac{\partial E}{\partial c_1} &= -2\lambda_1 \int_\Omega (f(x) - c_1) H(\varphi(x)) \, dx := 0 \\ \frac{\partial E}{\partial c_2} &= -2\lambda_2 \int_\Omega (f(x) - c_2)[1 - H(\varphi(x))] \, dx := 0, \end{aligned}$$

which gives us the following formulation for updating the values of c_1 and c_2,

$$\begin{aligned} c_1 &= \frac{\int_\Omega f(x) H(\varphi(x)) \, dx}{\int_\Omega H(\varphi(x)) \, dx} \\ c_2 &= \frac{\int_\Omega f(x)[1 - H(\varphi(x))] \, dx}{\int_\Omega 1 - H(\varphi(x)) \, dx}. \end{aligned}$$

One may notice that these values are precisely the region averages for the areas of the segmentation. Correspondingly, for fixed values of c_1 and c_2, one may update φ by computing the derivative

$$\frac{\partial \varphi}{\partial t} = \delta_\varepsilon(\varphi(x)) \left[\mu \operatorname{div} \left(\frac{\nabla \varphi(x)}{|\nabla \varphi(x)|} \right) - \nu - \lambda_1 (f(x) - c_1)^2 + \lambda_2 (f(x) - c_2)^2 \right]$$

and integrating. It suffices to propose and initial state for φ, so one may begin the iterative process. Experientially, it has been shown that for a two-dimensional image, the level set function

$$\varphi(x,y) = \sin\left(\frac{\pi}{5}x\right)\sin\left(\frac{\pi}{5}y\right)$$

produces satisfactory results. This formulation can be implemented by discretising the operations required to compute c_1, c_2, and φ.

4. Extension of Two-Phase Segmentation

4.1. General Multi-Phase Case

The theory discussed in [3] explicitly describes a functional whose minimisation produces a two-phase segmentation of a given image. We may extend this functional in a natural way to produce multi-phase segmentations of images. In particular, we may segment images into 2^n regions, for $n \in \mathbb{N}$. As previously discussed, the two-phase functional is comprised of three components. We can update each of these to produce the multi-phase functional. Going forward we define $f : \Omega \to \mathbb{R}$ to be an image defined on some domain $\Omega \subseteq \mathbb{R}^2$.

In the two-phase model, the length term is given in terms of a single level set function as

$$L(\varphi) = \int\limits_{\Omega} \delta(\varphi(x))|\nabla\varphi(x)|\,dx.$$

For a multi-phase model, using n level set functions, we may extend the length term by summing over all level set functions as

$$L(\varphi_1,\cdots,\varphi_n) = \int\limits_{\Omega}\sum_{i=1}^{n} \delta(\varphi_i(x))|\nabla\varphi_i(x)|\,dx.$$

Correspondingly, we may extend the area term to fit a multi-phase setting. This is best done on a case-by-case basis.

For the region average terms, we make the natural extension of the two-phase model, where each term includes a single region, and omits the rest. We do this as follows. For a segmentation comprised of n level set functions (and hence 2^n regions), define the indicator functions

$$\chi_m(x) = \begin{cases} 1, x \text{ in region } m \\ 0, \text{ otherwise} \end{cases}.$$

This sequence of indicator functions is defined for all $x \in \Omega$ and $1 \leq m \leq 2^n$. Hence, in the multi-phase setting, the region average term for a single region is expressed as

$$\int\limits_{\Omega} |f(x) - c_m|^2\chi_m(x)\,dx,$$

for constants $(c_m)_m$. We then sum over all such terms, giving

$$\sum_{m=1}^{2^n} \int_\Omega |f(x) - c_m|^2 \chi_m(x)\, dx.$$

We may combine the above formulations to yield the general functional for a multi-phase segmentation,

$$\begin{aligned} E = \mu & \int_\Omega \sum_{i=1}^{n} \delta(\varphi_i(x)) |\nabla \varphi_i(x)|\, dx \\ & + \nu A(\varphi_1, \cdots, \varphi_n) \\ & + \sum_{m=1}^{2^n} \lambda_m \int_\Omega |f(x) - c_m|^2 \chi_m(x)\, dx, \end{aligned}$$

where $A(\varphi_1, \cdots, \varphi_n)$ represents the area term, and μ, ν, $(\lambda_m)_m$ are parameters used to control the weighting of each term.

4.2. Four-Phase Segmentation Model

We may now apply this theory to the specific example of a four-phase segmentation. This is the simplest multi-phase segmentation, and corresponds to the case $n = 2$. Written explicitly, the energy functional for this case takes the form

$$\begin{aligned} & \mu \int_\Omega \delta(\varphi_1(x)) |\nabla \varphi_1(x)| + \delta(\varphi_2(x)) |\nabla \varphi_2(x)|\, dx \\ & + \nu \int_\Omega H(\varphi_1(x)) + H(\varphi_2(x)) - H(\varphi_1(x)) H(\varphi_2(x))\, dx \\ & + \lambda_1 \int_\Omega |f(x) - c_1|^2 H(\varphi_1(x)) H(\varphi_2(x))\, dx \\ & + \lambda_2 \int_\Omega |f(x) - c_2|^2 \left[1 - H(\varphi_1(x))\right] H(\varphi_2(x))\, dx \\ & + \lambda_3 \int_\Omega |f(x) - c_3|^2 H(\varphi_1(x)) \left[1 - H(\varphi_2(x))\right]\, dx \\ & + \lambda_4 \int_\Omega |f(x) - c_4|^2 \left[1 - H(\varphi_1(x))\right] \left[1 - H(\varphi_2(x))\right]\, dx. \end{aligned}$$

From this, one may compute, in a similar fashion to 3.3, formulations for updating the two level set functions φ_1 and φ_2, and the region constants c_1, c_2, c_3, and c_4. By calculating derivatives in a similar fashion to the two-phase setting, one may write,

$$c_1 = \frac{\int_\Omega f(x) H(\varphi_1(x)) H(\varphi_2(x))\, dx}{\int_\Omega H(\varphi_1(x)) H(\varphi_2(x))\, dx}$$
$$c_2 = \frac{\int_\Omega f(x) [1 - H(\varphi_1(x))] H(\varphi_2(x))\, dx}{\int_\Omega [1 - H(\varphi_1(x))] H(\varphi_2(x))\, dx}$$
$$c_3 = \frac{\int_\Omega f(x) H(\varphi_1(x)) [1 - H(\varphi_2(x))\, dx}{\int_\Omega H(\varphi_1(x)) [1 - H(\varphi_2(x))]\, dx}$$
$$c_4 = \frac{\int_\Omega f(x) [1 - H(\varphi_1(x))][1 - H(\varphi_2(x))]\, dx}{\int_\Omega [1 - H(\varphi_1(x))][1 - H(\varphi_2(x))]\, dx}$$

for the updates to the region values, and

$$\begin{aligned}\frac{\partial \varphi_1}{\partial t} &= \delta_\varepsilon(\varphi_1(x)) \Big[\mu \operatorname{div}\left(\frac{\nabla \varphi_1(x)}{|\nabla \varphi_1(x)|} \right) - \nu + H_\varepsilon(\varphi_2) - \lambda_1 (f(x) - c_1)^2 \\ &\qquad + \lambda_2 (f(x) - c_2)^2 - \lambda_3 (f(x) - c_3)^2 + \lambda_4 (f(x) - c_4)^2 \Big] \\ \frac{\partial \varphi_2}{\partial t} &= \delta_\varepsilon(\varphi_2(x)) \Big[\mu \operatorname{div}\left(\frac{\nabla \varphi_2(x)}{|\nabla \varphi_2(x)|} \right) - \nu + H_\varepsilon(\varphi_1) - \lambda_1 (f(x) - c_1)^2 \\ &\qquad - \lambda_2 (f(x) - c_2)^2 + \lambda_3 (f(x) - c_3)^2 + \lambda_4 (f(x) - c_4)^2 \Big]\end{aligned}$$

for the level set functions.

4.3. Discrete Four-Phase

The above formulation is based upon a continuous definition of an image. In practice, however, we require an implementation fit for the discrete setting. In [3], Getreuer describes this for the two-phase model. We can update this approach to correspond to the four-phase case we seek to implement. From Getreuers work, we may write,

$$\operatorname{div}\left(\frac{\nabla \varphi}{|\nabla \varphi|} \right) \approx \nabla_x^- \frac{\nabla_x^+ \varphi}{\sqrt{\eta + (\nabla_x^+ \varphi)^2 + (\nabla_y^0 \varphi)^2}} + \nabla_y^- \frac{\nabla_y^+ \varphi}{\sqrt{\eta + (\nabla_x^0 \varphi)^2 + (\nabla_y^+ \varphi)^2}},$$

where the parameter η ensures a non-zero denominator. Using this approximation, we may formulate a discretisation for the four-phase model. We do this as follows.

$$\begin{aligned}\frac{\partial\varphi_1}{\partial t} &= \delta_\varepsilon(\varphi_1(x))\Big[\mu\left(\nabla_x^- \frac{\nabla_x^+\varphi_1}{\sqrt{\eta+(\nabla_x^+\varphi_1)^2+(\nabla_y^0\varphi_1)^2}} + \nabla_y^- \frac{\nabla_y^+\varphi_1}{\sqrt{\eta+(\nabla_x^0\varphi_1)^2+(\nabla_y^+\varphi_1)^2}}\right) \\ &\quad -\nu + H_\varepsilon(\varphi_2) - \lambda_1(f-c_1)^2 + \lambda_2(f-c_2)^2 - \lambda_3(f-c_3)^2 + \lambda_4(f-c_4)^2\Big] \\ \frac{\partial\varphi_2}{\partial t} &= \delta_\varepsilon(\varphi_2(x))\Big[\mu\left(\nabla_x^- \frac{\nabla_x^+\varphi_2}{\sqrt{\eta+(\nabla_x^+\varphi_2)^2+(\nabla_y^0\varphi_2)^2}} + \nabla_y^- \frac{\nabla_y^+\varphi_2}{\sqrt{\eta+(\nabla_x^0\varphi_2)^2+(\nabla_y^+\varphi_2)^2}}\right) \\ &\quad -\nu + H_\varepsilon(\varphi_1) - \lambda_1(f-c_1)^2 - \lambda_2(f-c_2)^2 + \lambda_3(f-c_3)^2 + \lambda_4(f-c_4)^2\Big].\end{aligned}$$

These equations may be used to implement an algorithm to update the values of φ_1 and φ_2, and are computed for each pixel at a time. For a given pixel $\varphi_{i,j}$, we write

$$\frac{\partial\varphi_{i,j}}{\partial t} \approx \frac{\varphi_{i,j}^{(n)} - \varphi_{i,j}^{(n-1)}}{dt},$$

where $\varphi_{i,j}^{(n)}$ represents pixel (i,j) of the n^{th} iteration of φ. This allows us to update each pixel of the level set functions φ_1 and φ_2

5. Implementation of Multi-Phase Algorithm (Four-Phase)

5.1. Overview

The implementation of the four-phase segmentation algorithm will be based upon [6], an existing Python script for the implementation of the two-phase Chan-Vese algorithm. This existing code is closely based upon the numerical implementation described in [3]. In this paper, a discrete approach is taken to implement a two-phase Chan-Vese algorithm. One can extend this approach to a four-phase setting, and correspondingly, update the Python script to implement it.

5.2. Demonstrations

The functionality of this implementation can now be demonstrated on a variety of images. A simple example of a square, divided into four regions of distinct colours demonstrated that the code produces segmentations which are natural and expected. We may also easily demonstrate the difference between a two-phase and a multi-phase segmentation.

The working code can now be used to segment a variety of more complicated images. For example, images of brain scans can be segmented to distinguish between different regions and types of matter. Whilst the two-phase segmentation can only construct the outline of the brain, the multi-phase segmentation shows off the regions in much greater detail.

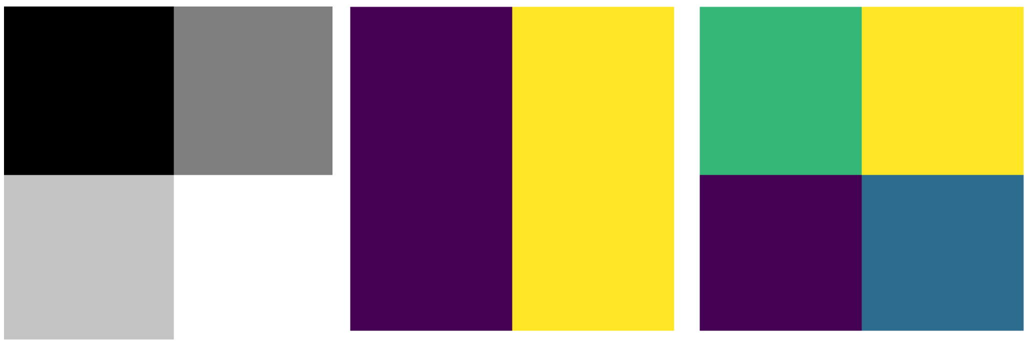
Figure 2: A simple example of a piecewise constant image (left), its two phase segmentation (middle), and its multiphase segmentation (right).

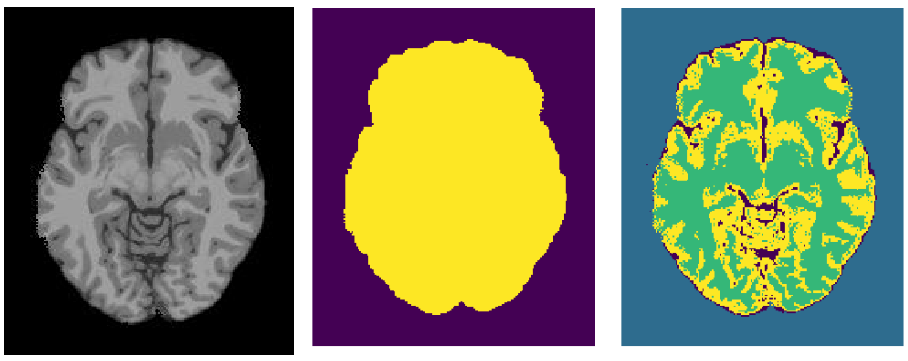
Figure 3: An image of a brain scan (left) with its two-phase segmentation (middle) and multi-phase segmentation (right).

$$\frac{1}{\pi}\begin{array}{|c|c|c|}\hline 0.019 & 0.208 & 0.019 \\ \hline 0.208 & 2.231 & 0.208 \\ \hline 0.019 & 0.208 & 0.019 \\ \hline \end{array} \to x_{i,j}$$

Figure 4: The ratios in which neighbouring pixels are combined in a standard Gaussian filter.

6. The case of Noisy Images

6.1. Noise

Under the definitions in mathematical image processing introduced in previous sections, we may introduce the notion of noisy images. In this context, noise is any unwanted perturbations in an image. In a mathematical sense, this translates to areas of high gradient of descent over a small area.

Noise can be rather harmful to the segmentation methods discussed above. Regions with a high presence of noise can be incorrectly split into different regions due to highly different pixel intensities. To combat this, one may consider applying a noise reducing filter over the image, before segmenting.

6.2. Filters

In mathematical image processing, a filter is a mapping $F : f \to g$ from an image f to a new image g. Filters may also be known as masks, and are generally used to change or emphasise certain properties of the original image. Common uses for filters are blurring, sharpening, and edge detection.

In the context of this report, we are interested in noise reduction filters. A particular example of such a filter is the Gaussian filter. This filter updates the value of each pixel with an average of itself and its surrounding pixels, weighted according to a Gaussian distribution. For example consider 3×3 filter, described approximately as follows. For a pixel $x_{i,j}$, we update its value as

$$\begin{aligned} x_{i,j} \to \frac{1}{\pi}\Big[&0.019(x_{i-1,j-1} + x_{i+1,j-1} + x_{i+1,j-1} + x_{i-1,j-1}) \\ &+ 2.231x_{i,j} + 0.208(x_{i-1,j} + x_{i,j-1} + x_{i+1,j} + x_{i,j+1})\Big]. \end{aligned}$$

An interesting application of Gaussian filter is in the Cartoon-Texture decomposition algorithm proposed in [1]. The idea is to separate an image into two components: a piece-wise smooth one, called the cartoon component, and a non-smooth one containing oscillatory information such as texture or noise. To obtain such a decomposition the authors in the paper exploit Gaussian filters in order to evaluate the local level of intensity variation for each pixel. Pixels of a higher local intensity variation are classified as texture or noise.

6.3. Effect of Denoising Filter on Segmentation

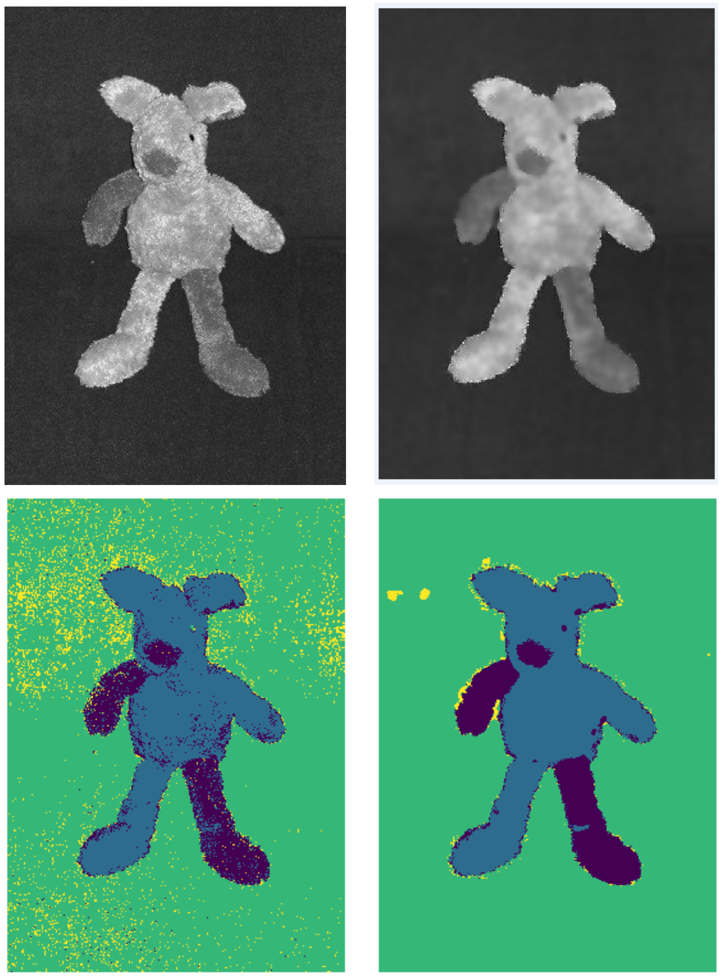
Figure 5: Original (left) and smooth (right) images with their respective segmentations.

Using a few iterations of the algorithm proposed in [1] (for which a MATLAB version is available), one may explore the effects of noise on segmentation techniques and the use of filters to improve segmentation quality. Consider the above figure and observe the noisy and smooth version of the same image above their respective segmentations. One can clearly notice a striking difference, with a significant improvement in the clarity and smoothness of the segmentation on the right

References

[1] A. Buades, T. M. Le, J. Morel, and L. A. Vese. Fast cartoon + texture image filters. *IEEE Trans. Image Process.*, 19(8):1978–1986, 2010.

[2] T. F. Chan and L. A. Vese. Active contours without edges. *IEEE Transaction on Image Processing*, 10(2):266–277, 2001.

[3] P. Getreuer. Chan-Vese Segmentation. *Image Processing On Line*, 2:214–224, 2012. https://doi.org/10.5201/ipol.2012.g-cv.

[4] A. Mitiche and I. Ayed. *Variational and Level Set Methods in Image Segmentation.* Springer, 2010.

[5] D. Mumford and J. Shah. Optimal approximations by piecewise smooth functions and associated variational problems. *Communications on Pure and Applied Mathematics*, 42(5):577–685, 1989.

[6] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors. scikit-image: image processing in python. *PeerJ*, 2:e453, jun 2014.

A. Multi-Phase Python Implementation

```
import numpy as np
pi=np.pi

import skimage as ski

from PIL import Image
from numpy import array
from skimage._shared.utils import _supported_float_type

def _mp_calculate_variation(image, phi1, phi2, mu1, mu2, nu,
        lambda1, lambda2, lambda3, lambda4, dt=0.5):

    eta = 1e-8
    P = np.pad(phi1, 1, mode='edge')
    Q = np.pad(phi2, 1, mode='edge')

    phi1xp = P[1:-1, 2:] - P[1:-1, 1:-1]
    phi1xn = P[1:-1, 1:-1] - P[1:-1, :-2]
    phi1x0 = (P[1:-1, 2:] - P[1:-1, :-2]) / 2.0

    phi1yp = P[2:, 1:-1] - P[1:-1, 1:-1]
    phi1yn = P[1:-1, 1:-1] - P[:-2, 1:-1]
    phi1y0 = (P[2:, 1:-1] - P[:-2, 1:-1]) / 2.0

    phi2xp = Q[1:-1, 2:] - Q[1:-1, 1:-1]
    phi2xn = Q[1:-1, 1:-1] - Q[1:-1, :-2]
    phi2x0 = (Q[1:-1, 2:] - Q[1:-1, :-2]) / 2.0

    phi2yp = Q[2:, 1:-1] - Q[1:-1, 1:-1]
    phi2yn = Q[1:-1, 1:-1] - Q[:-2, 1:-1]
    phi2y0 = (Q[2:, 1:-1] - Q[:-2, 1:-1]) / 2.0

    C11 = 1. / np.sqrt(eta + phi1xp**2 + phi1y0**2)
    C12 = 1. / np.sqrt(eta + phi1xn**2 + phi1y0**2)
    C13 = 1. / np.sqrt(eta + phi1x0**2 + phi1yp**2)
    C14 = 1. / np.sqrt(eta + phi1x0**2 + phi1yn**2)

    C21 = 1. / np.sqrt(eta + phi2xp**2 + phi2y0**2)
    C22 = 1. / np.sqrt(eta + phi2xn**2 + phi2y0**2)
```

```
C23 = 1. / np.sqrt(eta + phi2x0**2 + phi2yp**2)
C24 = 1. / np.sqrt(eta + phi2x0**2 + phi2yn**2)

K1 = (P[1:-1, 2:] * C11 + P[1:-1, :-2] * C12 +
      P[2:, 1:-1] * C13 + P[:-2, 1:-1] * C14)

K2 = (Q[1:-1, 2:] * C21 + Q[1:-1, :-2] * C22 +
      Q[2:, 1:-1] * C23 + Q[:-2, 1:-1] * C24)




Hphi1 = _mp_heavyside(phi1)
Hphi2 = _mp_heavyside(phi2)
Hphi1inv = 1. - Hphi1
Hphi2inv = 1. - Hphi2


(c10, c01, c11, c00) =
_mp_calculate_averages(image, Hphi1, Hphi2)



difference_from_average_term_1 =
(0.0 - lambda1 * Hphi2inv * (image-c10)**2+
lambda2 * Hphi2 * (image-c01)**2 -
lambda3 * Hphi2 * (image-c11)**2 +
lambda4 * Hphi2inv * (image-c00)**2)

difference_from_average_term_2 =
(0.0 + lambda1 * Hphi1* (image-c10)**2 -
lambda2 * Hphi1inv * (image-c01)**2 -
lambda3 * Hphi1 * (image-c11)**2 +
lambda4 * Hphi1inv * (image-c00)**2)

area_term_1 = Hphi1inv # 1.0 - Hphi1
area_term_2 = Hphi2inv # 1.0 - Hphi2

new_phi_1 = (phi1 + (dt*_mp_delta(phi1)) *
(mu1*K1 + difference_from_average_term_1
+ nu*area_term_1))
```

```
    new_phi_2 = (phi2 + (dt*_mp_delta(phi2)) *
    (mu1*K2 + difference_from_average_term_2
    + nu*area_term_2))


    return (new_phi_1 / (1 + mu1 * dt * _mp_delta(phi1)
     * (C11+C12+C13+C14)),
     new_phi_2 / (1 + mu2 * dt * _mp_delta(phi2)
     * (C21+C22+C23+C24)) )


def _mp_edge_length_term(phi1, phi2, mu1,mu2):
    P = np.pad(phi1, 1, mode='edge')
    Q = np.pad(phi2, 1, mode='edge')

    fy1 = (P[2:, 1:-1] - P[:-2, 1:-1]) / 2.0
    fx1 = (P[1:-1, 2:] - P[1:-1, :-2]) / 2.0

    fy2 = (Q[2:, 1:-1] - Q[:-2, 1:-1]) / 2.0
    fx2 = (Q[1:-1, 2:] - Q[1:-1, :-2]) / 2.0

    return (mu1 * _mp_delta(phi1)
    * np.sqrt(fx1 ** 2 + fy1 ** 2) + mu2 * _mp_delta(phi2)
    * np.sqrt(fx2 ** 2 + fy2 ** 2))


def _mp_difference_from_average_term(image, Hphi1, Hphi2,
 lambda1, lambda2, lambda3, lambda4):

    (c1, c2, c3, c4) =
     _mp_calculate_averages(image, Hphi1, Hphi2)
    H1inv = 1. - Hphi1
    H2inv = 1. - Hphi2

    return (lambda1 * (image-c1)**2 * Hphi1 * H2inv +
#in/out
            lambda2 * (image-c2)**2 * H1inv * Hphi2 +
#out/in
            lambda3 * (image-c3)**2 * Hphi1 * Hphi2 +
#in/in
            lambda4 * (image-c4)**2 * H1inv * H2inv)
#out/out
```

```
def _mp_area_term(Hphi1, Hphi2, nu):
    return nu * (Hphi1 + Hphi2 - (Hphi1 * Hphi2))

def _mp_calculate_averages(image, Hphi1, Hphi2):

    H1 = Hphi1
    H2 = Hphi2

    H1inv = 1. - H1
    H2inv = 1. - H2

    H1in_H2out = np.sum(H1 * H2inv)
    H1out_H2in = np.sum(H1inv * H2)
    H1in_H2in = np.sum(H1 * H2)
    H1out_H2out = np.sum(H1inv * H2inv)


    avg_inside_1_only = np.sum(image * H1 * H2inv)
    avg_inside_2_only = np.sum(image * H1inv * H2)
    avg_inside_both = np.sum(image * H1 * H2)
    avg_outside_both = np.sum(image * H1inv * H2inv)

    if H1in_H2out != 0:
        avg_inside_1_only /= H1in_H2out
    if H1out_H2in != 0:
        avg_inside_2_only /= H1out_H2in
    if H1in_H2in != 0:
        avg_inside_both /= H1in_H2in
    if H1out_H2out != 0:
        avg_outside_both /= H1out_H2out

    return (avg_inside_1_only, avg_inside_2_only,
     avg_inside_both, avg_outside_both)

def _mp_heavyside(x, eps=1.):

    return 0.5 * (1. + (2./np.pi) * np.arctan(x/eps))


def _mp_delta(x, eps=1.):
    return eps / (eps**2 + x**2)


def _mp_init_level_set_1(init_level_set ,
 image_shape , dtype=np.float64):

    res = _mp_checkerboard_1(image_shape , 5, dtype)

    return res.astype(dtype , copy=False)


def _mp_checkerboard_1(image_size , square_size ,
 dtype=np.float64):

    yv =
    np.arange(image_size[0] ,
    dtype=dtype).reshape(image_size[0] , 1)
    xv = np.arange(image_size[1] , dtype=dtype)
    sf = np.pi / square_size
    xv *= sf
    yv *= sf

    return np.sin(yv) * np.sin(xv)


def _mp_init_level_set_2(init_level_set , image_shape ,
 dtype=np.float64):

    res = _mp_checkerboard_2(image_shape , 5, dtype)

    return res.astype(dtype , copy=False)


def _mp_checkerboard_2(image_size , square_size ,
 dtype=np.float64):

    yv =
    np.arange(image_size[0] ,
    dtype=dtype).reshape(image_size[0] , 1)
    xv = np.arange(image_size[1] , dtype=dtype)
    sf = np.pi / square_size
    xv *= sf
    yv *= sf

    return np.cos(yv) * np.cos(xv)



def _mp_energy(image, phi1, phi2, mu1, mu2, nu, lambda1,
 lambda2,  lambda3, lambda4):

    H1 = _mp_heavyside(phi1)
    H2 = _mp_heavyside(phi2)
    avgenergy =_mp_difference_from_average_term(image, H1,
    H2, lambda1, lambda2, lambda3, lambda4)
    lenenergy = _mp_edge_length_term(phi1, phi2, mu1, mu2)
    areaenergy = _mp_area_term(H1, H2, nu)

    return np.sum(lenenergy) +
    np.sum(avgenergy) + np.sum(areaenergy)


def mp_chan_vese(image, mu1=1.0, mu2=1.0, nu=0.50,
lambda1=5e3, lambda2=5e3, lambda3=5e3, lambda4=5e3,
tol=1e-6, max_num_iter=1000, dt=0.5,
init_level_set='checkerboard', extended_output=False):

    if len(image.shape) != 2:
        raise ValueError("Input_image_should_be_a_2D_array.")

    float_dtype = _supported_float_type(image.dtype)
    phi1 = _mp_init_level_set_1(init_level_set, image.shape,
    dtype=float_dtype)
    phi2 = _mp_init_level_set_2(init_level_set, image.shape,
    dtype=float_dtype)

    if type(phi1) != np.ndarray or phi1.shape != image.shape:
        raise ValueError("Dimension_Error")

    image = image.astype(float_dtype, copy=False)
    image = image - np.min(image)
    if np.max(image) != 0:
        image = image / np.max(image)
```

```
i = 0
old_energy = _mp_energy(image, phi1, phi2, mu1, mu2, nu,
lambda1, lambda2,  lambda3, lambda4)
energies = []
phivars = []
phivar = tol + 1
segmentation1 = phi1 > 0
segmentation2 = phi2 > 0


while(phivar > tol and i < max_num_iter):
    # Save old level set values
    oldphi1 = phi1
    oldphi2 = phi2

    # Calculate new level set
    temp = _mp_calculate_variation(image, phi1, phi2,
    mu1, mu2, nu, lambda1, lambda2, lambda3, lambda4, dt)
    phi1 = temp[0]
    phi2 = temp[1]

    phivar1 = ((phi1-oldphi1)**2).mean()
    phivar2 = ((phi2-oldphi2)**2).mean()
    phivar = np.sqrt(phivar1 + phivar2)

    # Extract energy and compare to previous
    # level set and
    # segmentation to see if continuing is necessary
    segmentation1 = phi1 > 0
    segmentation2 = phi2 > 0


    new_energy = _mp_energy(image, phi1, phi2, mu1, mu2,
    nu, lambda1, lambda2, lambda3, lambda4)

    # Save old energy values
    energies.append(old_energy)
    old_energy = new_energy

    phivars.append(phivar)
    i += 1
    if extended_output:
        return (segmentation1,segmentation2, phi1, phi2,
        phivars, energies)
    else:
        return (segmentation1, segmentation2)
```