平成２３年度　特別研究報告書

イラスト画像における対象物抽出と機械学習を用いた自動分類の研究

龍谷大学　大学院　理工学研究科　情報メディア専攻

学籍番号 **T10M098**　奥村 亮仁

指導教員　三好 力　教授

内容梗概

本研究ではイラスト画像に対するグラフカットを用いた対象物抽出およびその手法と自己組織化マップを用いた画像分類手法について検討する。イラスト画像における対象物抽出について、過去に動的輪郭モデルを用いた手法と変分ベイズ法を用いた手法を提案したが、さらに精度が良く処理時間が短い顕著性を示す画像とグラフカットによるセグメンテーションを用いた手法を新たに提案する。また、この対象物抽出の提案手法と自己組織化マップを組み合わせた画像分類の自動化手法を提案する。ブースティングなどの統計的手法を用いた画像分類の手法が一般的であるが、学習させるために大量の画像を必要とする。大規模な画像のデータベースを個人で用意するのは困難である。少ないサンプルで学習させ分類を行うことが出来れば個人のコンピュータで学習、分類が可能となるが、対象を限定しない一般画像識別は困難である。本研究ではイラスト画像に限定して、さらに作品等で限定すれば、少ないサンプルでもうまく分類できるのではないかと考えた。この事から画像分類の提案手法に関して、対象を限定して数枚程度の学習画像で画像の分類を可能とすることを目的とする。

abstract

We discuss about an automatic object extraction using graph-cuts for illust images(ex.Anime and Manga) and an image classification method using self-organizing map.

We already proposed the methods for an illust images object extraction, a method using active countour models, and a method using variational byesian in the past. In this paper, we newly propose an automatic object extraction using saliency images and graph-cuts segmentation. We showed by examination that this approach is better the accuracy and faster the processing time than past approaches.

We also propose an automatic image classification using an automatic object extraction using graph-cuts and self-organizing map. A image classification is generally used statistical methods, but, in order to teach the image classifier need very large number of learning images. It is difficult to prepare very large number of images personaly, so if a image classifier can be learned and classified by small images, it is possible to learn and classify at a personal computer and personal use. We propose a image classification method confined to illust images and productions and to be able to learn from small amount of images. We consider that an approach which we propose is confined to illust images and productions.

目 次

第1章　はじめに

今日、インターネットの利用者数の増加と発展により、多くの情報がやり取りされるようになった。中でも World Wide Web(以下、Web) は、電子メールと並ぶインターネットを利用する主な目的の 1 つとなっており多くの人に利用されている。Web 上には多くの画像が存在し、絵を描いて画像ファイルを個人サイトで公開している人も多い。最近では個人サイトだけでなく、ブログのようなサイトよりも簡単に公開できるものや pixiv 等の絵や漫画のアップロードと公開に特化したサイトなどが存在しており、より公開しやすい環境が整ってきている。

公開される画像の数が多くなるとともに、それらを私的に集める人も多くなり、集める量も多くなっていると考えられる。集めた画像を分類する作業は現在ほぼ手作業でするしかない。そこで、この作業を自動化できれば便利であると考えた。

画像を分類するための手法は様々あり、例えばサポートベクタマシン [2] を用いたものやブースティング [3] を用いたものがある。これらは統計手法を用いて大量のデータを学習させてから分類を行うものである。しかし、個人では大量のデータを用意することが困難である。したがって本研究では、イラスト画像に限定することで数枚の画像を学習することで分類できるような分類器を作成することを目的とし、個人で使えるものを考える。

また、分類の前に対象物抽出を行う。これによって、分類に必要のない背景部分を排除することができ、分類時には必要な対象物だけで考えればよくなる。

本研究は画像を分類するための手法として、グラフカットを用いた対象物抽出とそれを組み込んだ自己組織化マップを用いた画像分類の手法を提案する。グラフカットを用いた対象物抽出では、2 つの既存手法と提案手法の説明をし、2 つの既存手法と提案手法のそれぞれで実験を行なって比較検討する。自己組織化マップを用いた画像分類では、手法の説明をし、提案手法で実験を行なって分類ができているかどうかを検討する。

第2章　基本事項

2.1　イラスト画像

本研究では、人が鉛筆、シャーペンを用いて紙に描いた、またはパソコンでSAIやPhotoshopのようなソフトを用いて描いた絵であり、漫画やアニメのあるシーンの静止画像、一枚絵と呼ばれるようなイラストなどの画像をまとめてイラスト画像とする。

イラスト画像には、人物や生物、物体等の一般的にキャラクタと呼ばれるような対象となるものと、それ以外の領域が含まれている事が多い。前者を対象物、後者を背景と呼ぶとする。ただし、対象物が含まれておらず背景のみの画像も少なからず存在する。

イラスト画像を一般的な写真と比べると、対象物や背景は抽象化されており、色がまとまって存在し、色の数が少ないといえる。しかし、写実的に描いたものになるほど写真に近くなっていくともいえる。

対象物が特定の人物またはそれに相当するようなキャラクタの場合、実在する人と比べると、キャラクタは描く人によって変わるが、実在する人はほとんど一意に決定することができる。例えば、顔画像認識において、実在する人をある特定の個人であることを顔だけから判定することが出来るが、複数人がそれぞれ同じ特定のキャラクタを描いたとすると、顔だけでは特定のキャラクタと判定することは難しい。一方で、実在する人とは違って髪や服などの身なりが変わることが少ない事が多いことから、対象物全体を捉えることができれば対象物が特定のキャラクタと判定することができると考えられる。季節によって正月なら着物、夏なら水着、クリスマスならサンタクロースに扮したものになることがあるが、本研究では除外する。

2.2 対象物抽出

対象物抽出とは、画像処理の一種で入力画像の背景を取り除いて対象物だけを取り出すことであり、これを自動化する手法が様々考えられている。

イラスト画像における対象物抽出について、過去に動的輪郭モデルを用いた手法と変分ベイズ法を用いた手法を提案したが、さらに精度や処理時間が良いものがないかと検討した結果、顕著性を示す画像とグラフカットによるセグメンテーションを用いた手法を新たに提案する。

グラフカットによるセグメンテーションとは、入力画像をグラフとみなしてグラフ理論を用いてセグメンテーションを行う手法である。

既存のグラフカットによるセグメンテーションは既存手法より処理時間、精度が良かったが、手動で対象物と背景の選択しなければならなかった。ここで、手動の代わりに顕著性を示す画像で対象物と背景の選択を行うことで自動化した。

顕著性を示す画像とは、画像中の目立つ場所を示す画像である。2.1 よりイラスト画像において色はまとまって存在しているという仮定を用いて、色について顕著性を示す画像を作成すれば対象物と背景がおおまかに決定できる。グラフカットでは、対象物と背景の選択はおおまかでよく、指定を少し間違える程度ならば正しく抽出できるという利点があり、顕著性を示す画像が合うと考えられる。

本研究では、グラフカットを用いて対象物抽出を行う手法を提案し、分裂する動的輪郭モデルを用いた手法、変分ベイズ法を用いた手法を既存手法として提案手法と比較検討する。

2.3 画像分類

画像分類とは、画像を特定の条件で仕分けることである。

画像の分類を人間は容易に行うことができるが、これを計算機で自動化することは多くの困難が伴う。しかし近年、機械学習の発展によってある程度可能になっている。

画像分類の自動化では、例えばサポートベクタマシンやブースティングのような機械学習を用いた手法がよく使われている。しかし、これらの手法は学習させるサンプルを大量に用意しなければならず、Google のように大規模なデータベースを持つ企業では問題ないが、個人で用意するのは困難である。少ないサンプルで学習させ分類を行うことが出来れば、個人で気軽に学習させて分類を行うことができるようになるが、少ないサンプルでは対象を限定しない一般画像識別は非常に難しい。したがって、本研究ではイラスト画像に限定して、さらに作品等で限定すれば、少ないサンプルでもうまく分類できるのではないかと考えた。

本研究では対象物抽出の提案手法と自己組織化マップを用いた画像分類の手法を提案する。

2.4　グラフ理論

グラフ理論[1)]は、ノードの集合とノード間を接続するエッジの集合で構成されるグラフと呼ばれるものの理論である。

ノードとノードを接続するエッジに向きがあるグラフを有向グラフ、向きを考えないグラフを無向グラフという。図 2.1 に有向グラフ、図 2.2 に無向グラフの例を示す。グラフのノードを 2 つの部分集合に分割することをカットという。エッジに重みがある有向グラフをネットワークと呼ぶ。このとき重みを容量と呼ぶ。

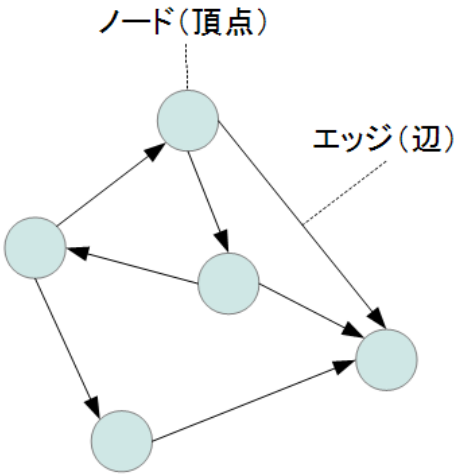

図 2.1: 有向グラフ

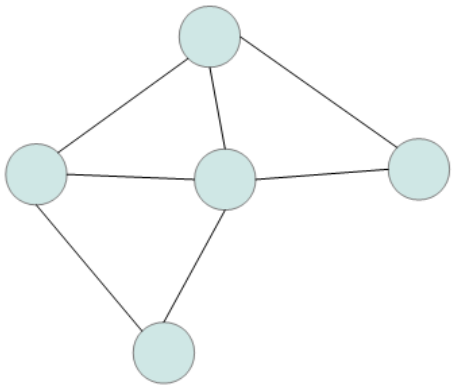

図 2.2: 無向グラフ

本研究では、対象物抽出の提案手法 3.3 でグラフカットによるセグメンテーションを用いる。このとき、最大フロー最小カット定理を考える。以下に最大フロー最小カット定理とグラフカットによるセグメンテーションについて示す。

最大フロー最小カット定理

始点と終点が存在するネットワークがあるとする。このとき、始点から終点へどれだけフローを増やすことが出来るかを考える。ネットワークに対してフローを限界まで増やした時、最大フローといい、最大フローを求める問題を最大フロー問題または最大流問題という。最大フローのとき、エッジの容量が最小になるカットが存在して、その容量は最大フローの容量と一致する。これを最大フロー最小カット定理という。最大フロー最小カット定理によって、最大フロー問題を解くで最小カットも同時にわかる。

グラフカットによるセグメンテーション

最小カットによって始点を含む部分集合と終点を含む部分集合に分割することが出来る。これを応用すると、画像をグラフとみなし、始点と終点をそれぞれ対象物と背景に対応させて対象物と背景を分割することができる。これをグラフカットによるセグメンテーション [21)22)] と呼ぶ。

図 2.3 に画像の画素をノードとみなして始点と終点を接続したときのグラフを示す。

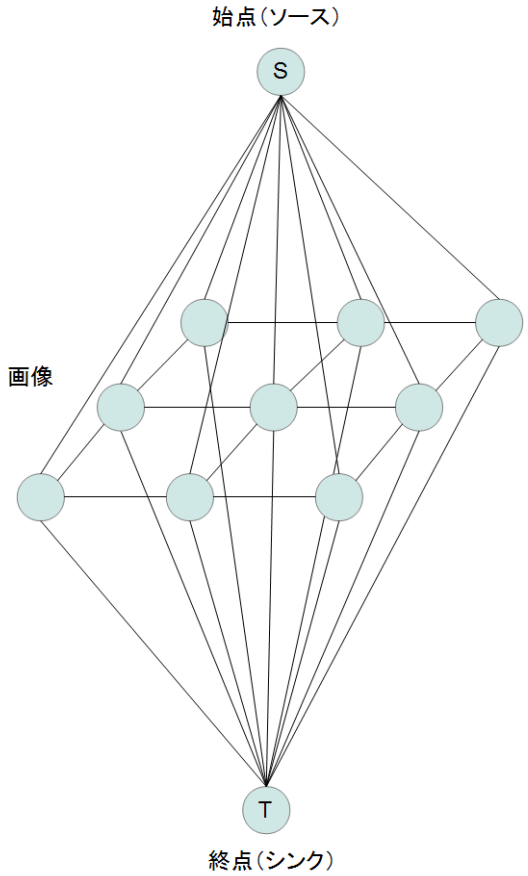

図 2.3: グラフカットによるセグメンテーションにおけるグラフ

2.5　自己組織化マップ

自己組織化マップとは、大脳皮質の視覚野をモデル化したニューラルネットワークの一種である。よく知られた Kohonen の自己組織化マップ [4] では、入力データの入力層と呼ばれる層と決められた数のノードを持つ競合層または出力層などと呼ばれる層で構成される。ノードの並び方は 1 次元、単純な 2 次元平面や球の表面にノードを並べたマップ、単純な 3 次元など様々あるが、本研究では、最もよく知られている単純な 2 次元平面の自己組織化マップを用いる。図 2.4 に自己組織化マップの模式図を示す。

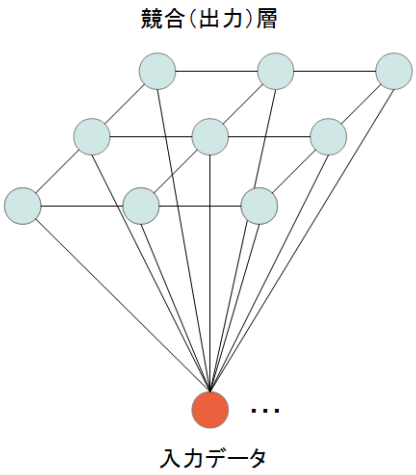

図 2.4: 自己組織化マップ

自己組織化マップは教師なし学習の一種で、主にクラスタリングや次元削減に用いられ分類や視覚化、要約などを行うのに使われる事が多い。例えば、高次元のデータをマップに写像することで高次元のデータをマップのノード数まで次元を小さくすることが出来る。

自己組織化マップの学習は、入力データに最も近い競合層のノードを勝者ノードとして、勝者ノードとその近傍を入力データに近づける処理を入力データごとに行う。これによって、近いデータが近い場所に集まってノードの変化が小さくなっていく。事前知識なしに、勝手にデータが集まっていくので自己組織化と呼ばれる。

以下に学習方法の詳細について示す。

学習方法

入力データの集合を X として、各入力データを $\mathbf{x} \in X$、i 番目の入力データを $\mathbf{x}_i \in X$ とする。入力データは N_{input} 次元のベクトル $\mathbf{x} = (x_0, x_1, ..., x_{N_{input}})$ である。また、自己組織化マップのノードの集合を M、ノードの数を $N_{\mathbf{M}}$ として、各ノードを $\mathbf{m} \in M$、j 番目のノードを $\mathbf{m}_j \in M$ とする。$\mathbf{m}$ は N_{input} 次元のベクトル $\mathbf{m} = (m_0, m_1, ..., m_{N_{input}})$ である。

$\mathbf{x}$ について、M から最も類似している $\mathbf{m}_c$ を探す。このとき、$\mathbf{x}$ と $\mathbf{m}$ の比較はユークリッド距離で行い、次の条件を満たす。

$$||\mathbf{x} - \mathbf{m}_c|| = \min_i ||\mathbf{x} - \mathbf{m}_i|| \tag{2.1}$$

$\mathbf{x}$ について最も類似しているノード $\mathbf{m}_c$ を決定したら、$\mathbf{m}_c$ とその近傍を更新する。図 2.5 に近傍の例を示す。この図では緑のノードを中心として黄色の範囲のノードを近傍としている。このとき、$\mathbf{m}_c$ の自身を含めた近傍を D_c、学習の大きさを決める関数 h_{ci} とする。以下に更新のための式を示す。

$$\mathbf{m}_i(t+1) = \begin{cases} \mathbf{m}_i(t) + h_{ci}||\mathbf{x} - \mathbf{m}_i(t)|| & i \in D_c(t) \\ \mathbf{m}_i(t) & i \notin D_c(t) \end{cases} \tag{2.2}$$

図 2.5 の例では、$D_c(t)$ は黄色の範囲にあるノードであり、$D_c(t)$ 以外はそれ以外のノードとなる。

これを全ての $\mathbf{x}$ について行うことで、自己組織化マップの学習となる。

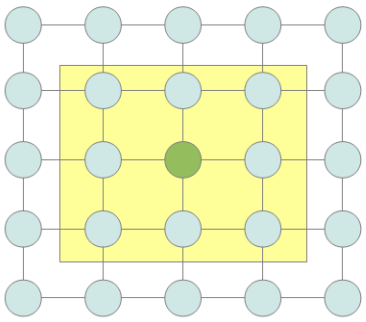

図 2.5: 自己組織化マップの学習時の近傍の例

2.6 OpenCV によるアニメ顔検出

画像分類の際に、anime.udp.jp の OpenCV によるアニメ顔検出 [5] を用いる。

アニメ顔検出はアニメや漫画、イラストなどの顔画像を学習させて検出できるようにしたもので、既に学習済みのデータを XML で提供されており、OpenCV のカスケード分類器 [6] を用いて検出を行う。

カスケードとは、直列に接続された単純な分類器から構成されることを意味している。候補が入力すると、候補はいずれかの分類器で棄却されるか全ての分類器を通過する。すべての分類器を通過すれば採用され、検出したい対象であるとする。

これをライブラリ化したものが OpenCV に存在し、学習済みデータを用いることで楽にカスケード分類器を使うことが出来る。

第3章　対象物抽出

3.1　既存手法：分裂する動的輪郭モデルを用いた手法

対象物抽出の既存手法の1つである分裂した分裂する動的輪郭モデルを用いた対象物抽出について示す。

イラスト画像が同じような色で固まっていると仮定して、色で画像を分割するために Hill-Climb を用いた領域分割と、対象物か背景かを推定するために分裂する動的輪郭モデルを用いた手法である。

輪郭の推定には分裂する動的輪郭モデルを用いる。分裂する動的輪郭モデル [9] とは、Kass らによる動的輪郭モデル [8] を改良したものである。Kass らによる動的輪郭モデルを改良したものである。Kass による動的輪郭モデルで必要であった分裂する動的輪郭モデルでは輪郭が交差するときに輪郭を分裂することによって、初期輪郭の数だけ対象物がいることと、初期輪郭を対象物の近傍に設置することがこのモデルでは必要なくなった。

領域分割は Hill-Climb を用いた手法を用いる。この手法は、事前に決めるパラメータ数が少なくて済み、クラスタ数を決める必要がなく、予備知識の必要もないので、自動化するのに最適であると思われる。また、単純なアルゴリズムなので高速と謳われている。領域が対象物かどうかの判断には、輪郭線内部に領域がどれだけの面積で入っているかを調べて領域全体の面積との比で判断すればよい。

以下にこの手法について示す。

3.1.1　分裂する動的輪郭モデル

Kass らによる動的輪郭モデルは、対象物の近傍においた閉曲線 $v(s) = (x(s), y(s))(0 \leq s \leq 1)$ を内部エネルギー $E_{int}(v(s))$、画像エネルギー $E_{image}(v(s))$、外部エネルギー $E_{con}(v(s))$ により定義されるエネルギー関数 E_{snakes} を最小化するように閉曲線の制御点を逐次的に動かして閉曲線を変形させ、エネルギー関数が極小状態のとき、その閉曲線を対象物の輪郭とするものである。

以下に E_{snakes} を示す。

$$E_{snakes} = \int_0^1 (E_{int}(v(s)) + E_{images}(v(s)) + E_{con}(v(s)))ds \tag{3.1}$$

Kass らによる動的輪郭モデルは、対象物の凹の形をした部分に入り込もうとする力が、隣接する制御点の内部エネルギーの引っ張る力によって入り込もうとする画像エネルギーの力と相殺され、凹の形の抽出が難しい。また、始めに対象物の数だけ閉曲線を作る必要があり、かつ、対象物の近傍に置く必要という問題がある。これらの問題を解決した分裂する動的輪郭モデルでは、エネルギー関数に面積項 E_{area} を導入で凹の形を抽出でき、閉曲線が分裂する事によって、始めから対象物の数だけ閉曲線を作ることと近傍に置く必要がない。

閉曲線の制御点を $v_i(x_i, y_i)(i = 1, 2, ..., n)$、$v_0 = v_n$、$v_{n+1} = v_1$ としたとき、分裂する動的輪郭モデルのエネルギー関数 E_{snakes} は以下のように定義される。

$$E_{snakes}(v_i) = E_{int}(v_i) + E_{image}(v_i) + E_{con}(v_i) + E_{area}(v_i) \tag{3.2}$$

面積項 E_{area} を導入すると閉曲線が交差するようになるので、この交差を切り離して閉曲線を分裂させる。交差の判定は次式によって行う。ただし、実数 p、q は $0 \leq p \leq 1$、$0 \leq q \leq 1$ を満たなければならない。

$$p(v_{i+1} - v_i) + v_i = q(v_{j+1}) - v_j) + v_j \tag{3.3}$$

交差している場合は v_i と v_{j+1}、v_{i+1} と v_j を連結する。また、分裂する動的輪郭モデルでは制御点の生成と消滅も行う。v_i、v_{i+1} 間の距離が閾値 D_{TH} 以上であれば、その間に制御点を生成し、$v_i v_{i-1}$ と $v_i v_i + 1$ のなす角 θ について、閾値 θ_{TH} を用いて $cos\theta > \theta_{TH}$ を満たす v_i を消滅させる。また、制御点の数が 5 未満の場合は輪郭モデル自体を消滅させる。

収束判定は制御点の移動した数が閾値 C_{TH} 以下であり、制御点の生成、消滅がない、または更新回数が t_max 以上になった場合、収束したとみなす。

3.1.2 Hill-Climb を用いた領域分割

[STEP1]

入力画像を HSV 色空間にする。H、S、V それぞれについて、あらかじめ決めた量子数に分割しヒストグラムを計算する。

[STEP2]

H、S、V それぞれのヒストグラムを、H、S、V の順で STEP3 と STEP4 の処理を行う。V の処理が完了したとき、STEP5 を行う。

[STEP3]

移動が行われていないヒストグラム、かつ、ヒストグラムのピクセル数が 0 でないものを始点とし、現在位置を p する。

[STEP4]

p と隣り合うヒストグラムのピクセル数を比較して、大きい方に p を移動させる。同じ場合は、隣り合うヒストグラムのさらに隣のヒストグラムと比較し、大きい方に p を移動させる。p の隣り合うヒストグラムが両方とも小さい場合、その位置記録して始点から p までのヒストグラムをそのピークに属させる。すべてのヒストグラムについて移動が行われたかどうかを調べる。行われたとき、STEP2 に戻る。それ以外の場合、STEP3 に戻る。

[STEP5]

入力画像のピクセルと対応するピークによって、画像を分割して終了する。

3.1.3 抽出手順

[STEP1]

入力画像について、分裂する動的輪郭モデルを用いて輪郭抽出を行う。輪郭抽出後の輪郭の数を N_{snakes} として、輪郭を $S_i(i = 1, 2, ..., N_{snakes})$ とする。

[STEP2]

入力画像について、Hill-Climbing を用いた領域分割を行う。分割された領域の数を N_{area} として、分割された領域を $a_j(j = 1, 2, ..., N_{area})$ とする。

[STEP3]

$i=0$、$j=0$ とする。

[STEP4]

S_i の内部に入っている a_j の面積を求めて、その面積が a_j の面積の閾値 P_{TH} 以上の場合、その領域を抽出して STEP6 を行う。それ以外の場合は STEP5 を行う。

[STEP5]

$i \leftarrow i+1$ とする。$i > N_{snakes}$ のとき、$j \leftarrow j+1$ とする。その後、STEP6 を行う。

[STEP6]

$j > N_{area}$ の場合、全ての a_j について判定したとなるので抽出完了とする。それ以外の場合、$i=0$ として STEP4 に戻る。

3.2 既存手法：変分ベイズ法を用いた手法

既存手法である動的輪郭モデルからアプローチを変えた領域分割を用いた対象物抽出の一種である手法について示す。

3.1 で示したように、領域分割だけでは分割された領域は、対象物の領域か背景の領域かがわからない。本手法では、分裂する動的輪郭モデルの代わりに変分ベイズ法による混合正規分布推定 [15) 18)] を用いる。混合正規分布の推定はパターン認識等で用いられるのが一般的である。ここでは、対象物を特定できそうな特徴量を使って分布の推定を行い、その分布をそのまま領域にできるのではないかと考える。

特徴量はエッジではなくコーナーを用いて対象物を推定する。コーナーとは、画像の特徴をよく表す点または角を表すものである。

動的輪郭モデルでは不必要な対象物ではない部分にエッジが多く含まれている場合、輪郭線が不必要な部分に沿ってしまいうまく抽出できない。この手法では、そのような不必要な部分について、エッジに頼らないので動的輪郭モデルよりも高い精度で抽出できると考えられる。

領域分割で用いる Hill-Climb を用いた領域分割については、3.1 で説明している。以下に変分ベイズ法の基礎とそれを用いた混合正規分布の説明を示し、それと領域分割を用いた抽出手順について示す。

3.2.1 ベイズ学習

ベイズ学習 [13)17)] とは、ほしい確率分布の未知パラメータをそれに関する確率分布を考え、未知パラメータの不確定性を表現してほしい確率分布を推定するものである。未知パラメータの確率分布は、データが観測される前を事前分布、その後を事後分布と呼ぶ。学習データ D、事前分布 $p(\theta)$、事後分布 $p(\theta|D)$ として、D が観測された後の事後分布 $p(\theta|D)$ はベイズの定理より以下のようになる。

$$p(\theta|D) = \frac{p(D|\theta)p(\theta)}{p(D)} \tag{3.4}$$

$$p(D) = \int p(D|\theta)p(\theta)d\theta \tag{3.5}$$

またベイズ学習では、事後分布 $p(\theta|D)$ を用いて分布を予測することが出来る。これを予測分布と呼び、未知データ x^* として次式で定義される。

$$p(\boldsymbol{x}^*|D) = \int p(\boldsymbol{x}^*|\theta)p(\theta|D)d\theta \tag{3.6}$$

式 (3.6) は、仮説 $p(\boldsymbol{x}^*|\theta)$ を事後分布 $p(\theta|D)$ で重み付き平均したものである。これにより、$\boldsymbol{x}^*$ が予測分布に入るか否かを確率的に予測することができる。

式 (3.5) について、右辺の分母は θ によらない、および、$-\log p$ が情報量であることから、次式のようにできる。

$$-\log p(\theta|D) \propto -\log p(D|\theta) - \log p(\theta) \tag{3.7}$$

式 (3.7) は、

事後の情報量$\propto$学習データの情報量 + 事前の情報量

といえ、事前の情報量に学習データの情報量が付加されて事後の情報量となっており、直感的に自然な情報論的解釈ができる。

3.2.2 変分ベイズ法

式 (3.6) は、一般に解析的に求めることが困難である。したがって、何らかの近似が必要となる。ここで変分法を用いて式 (3.6) の近似を行うのが、変分ベイズ法 [14] である。

変分法 [16] とは、関数を関数の関数 (汎関数) によって関数を近似する手法である。微分法と似ており、微分法では変数を変化させて極値を求めるのに対して、変分法では汎関数を変化させて極値を求める。

変分法を用いて導くための汎関数を導出を説明してアルゴリズムを示す。

潜在変数 Z、混合分布モデルにおける混合要素数 m として、すべての未知量を周辺化した周辺尤度 $L(D)$ は次式で表される。

$$L(D) = \log p(D) = \log \sum_m \sum_z \int p(D, Z, \theta, m) d\theta \tag{3.8}$$

新たな分布 q を導入して対数関数に対する Jensen の不等式を適用すると、$L(D)$ の下限値 $F[q]$ が得られる。

$$F[q] = \sum_m \sum_z \int q(Z, \theta, m) \log \frac{p(D, Z, \theta, m)}{q(Z, \theta, m)} d\theta \tag{3.9}$$

$L(D) - F[q]$ より、カルバック・ライブラダイバージェンス $\mathrm{KL}(q(Z, \theta, m|D), p(Z, \theta, m|D))$ を求めることができ、次式の関係となる。

$$L(D) = F[q] + \mathrm{KL}(q(Z, \theta, m|D), p(Z, \theta, m|D)) \tag{3.10}$$

式 (3.10) について、$F[q]$ を最大化することは KL を最小化する事と同義であり、KL を最小化することで真の事後分布の最良の近似となる。ここで $F[q]$ を汎関数と考えると、$F[q]$ の極値問題となって変分法を用いて $L(D)$ を近似することができる。

$p(D, Z, \theta, m)$ を分解した式を示し、$q(Z, \theta, m)$ を未知パラメータごとに分解した形を仮定する。

$$p(D, Z, \theta, m) = p(m)p(D, Z|m) \prod_i p(\theta_i|m) \tag{3.11}$$

$$q(Z, \theta, m) = q(m)q(Z|m) \prod_i q(\theta_i|m) \tag{3.12}$$

式 (3.12)、(3.12) を式 (3.9) に代入して、$q(Z|m)$、$q(\theta_i|m)$ を求めると、以下の式を得る。ただし、表記 $\langle f(x) \rangle_{p(x)}$ は $f(x)$ の $p(x)$ に関する期待値である。

$$q(Z|m) = C \exp \langle \log p(D, Z|\theta, m) \rangle_{q(\theta|m)} \tag{3.13}$$

$$q(\theta_i|m) = C' p(\theta_i|m) \exp \langle \log p(D, Z|\theta, m) \rangle_{q(Z|m), q(\theta_{-1}|m)} \tag{3.14}$$

C は $\sum_Z q(Z|m) = 1$ となるための定数、C' は $\int q(\theta_i|m) d\theta_i = 1$ となるための定数である。また、θ_{-1} は θ_i 以外のパラメータ集合を示す。式 (3.14)、(3.14) は相互に依存関係があるので、反復アルゴリズムによって逐次推定する。反復ステップ数を t として、アルゴリズムを以下に示す。

[STEP1]

分布 $q(\theta|m)^{(0)} = \prod_i p(\theta_i|m)^{(0)}$ を設定して、t←0 とする。

[STEP2] 以下を収束または更新回数が t_{max} になるまで繰り返す。

式 (3.14) より、

$$q(Z|m)^{(t+1)} = C \exp \langle \log p(D, Z|\theta, m) \rangle_{q(\theta|m)^{(t)}}$$

式 (3.14) より $i = 1, ..., I$ について、

$$q(\theta_i|m)^{(t+1)} = C' p(\theta_i|m) \exp \langle \log p(D, Z|\theta, m) \rangle_{q(Z|m)^{(t+1)}, q(\theta_{-1}|m)^{(t)}}$$

を計算して、$t \leftarrow t+1$ とする。

3.2.3 変分ベイズ法による混合正規分布推定

m 個の要素数を持つ d 次元の混合正規分布の確率密度関数は次式で表される。

$$p(\boldsymbol{x}; \theta) = \sum_{i=1}^{m} \alpha_i N(\boldsymbol{x}; \boldsymbol{\mu}_i, \boldsymbol{S}_i^{-1}) \tag{3.15}$$

$$N(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{S}_i^{-1}) = (2\pi)^{-\frac{d}{2}} |\boldsymbol{S}|^{\frac{1}{2}} \exp\left\{ -\frac{1}{2} (\boldsymbol{x} - \boldsymbol{\mu})^T \boldsymbol{S} (\boldsymbol{x} - \boldsymbol{\mu}) \right\} \tag{3.16}$$

ただし、入力ベクトル $\boldsymbol{x}$、平均ベクトル $\boldsymbol{\mu}$、共分散行列の逆行列である精度行列 $\boldsymbol{S}$ である。

$\boldsymbol{\alpha} = \{\alpha_i\}_{i=1}^m$、$\boldsymbol{\mu} = \{\boldsymbol{\mu}_i\}_{i=1}^m$、$\boldsymbol{S} = \{\boldsymbol{S}_i\}_{i=1}^m$ として、未知パラメータ θ の結合分布は以下にように分解できる。

$$p(\theta) = p(m) p(\boldsymbol{\alpha}|m) p(\boldsymbol{S}|m) p(\boldsymbol{\mu}|\boldsymbol{S}, m) \tag{3.17}$$

式 (3.14)、(3.14) を式 (3.17) に適用すると $p(\boldsymbol{\alpha}|\boldsymbol{m})$、$p(\boldsymbol{S}|m)$、$p(\boldsymbol{\mu}|\boldsymbol{S}, m)$ を求める事が出来る。

導出した結果を整理して、混合正規分布の推定のために変分ベイズ法を適用したアルゴリズムは、以下の通りとなる。

[STEP1] 初期化

事前分布の超パラメータ ϕ_0、ξ_0、η_0、$\boldsymbol{\nu}_0$、$\boldsymbol{B}_0$ と $\bar{N}_i^{(0)} \leftarrow \frac{N}{m}$ を設定する。その後、事後分布の超パラメータを以下のように設定する。

$i = 1, ..., m$ に対して、

$$\begin{aligned}
&\phi_i^{(0)} \leftarrow \phi_0 \\
&\bar{\boldsymbol{\mu}_i}^{(0)} \leftarrow \boldsymbol{\nu_0} \\
&\eta_i^{(0)} \leftarrow \eta_0, \boldsymbol{B}_i^{(0)} \leftarrow \boldsymbol{B}_0 \\
&f_{\boldsymbol{\mu}_i}^{(0)} \leftarrow \eta_0 + \bar{N}_i^{(0)} + 1 - d \\
&\Sigma_{\boldsymbol{\mu}_i}^{(0)} \leftarrow \frac{\boldsymbol{B}_i^{(0)}}{(\bar{N}_i^{(0)} + \xi_0) f_{\boldsymbol{\mu}_i}^{(0)}}
\end{aligned}$$

[STEP2] 潜在変数の事後分布の更新

$i = 1, ..., m$、$n = 1, ..., N$ に対して、次の計算を行う。

$$\begin{aligned}
\bar{z}_i^n &= \frac{\exp \gamma_i^n}{\sum_{j=1}^m \exp \gamma_j^n} \\
\gamma_i^n &\leftarrow \Psi(\phi_0 + \bar{N}_i^{(t)}) - \Psi(m\phi_0 + \sum_{i=1}^m \bar{N}_i^{(t)}) \\
&+ \frac{1}{2} \sum_{j=1}^d \Psi(\frac{\eta_0 + \bar{N}_i^{(t)} + 1 - j}{2}) - \frac{1}{2} \log |\boldsymbol{B}_i^{(t)}| \\
&- \frac{1}{2} \mathrm{Tr} \left\{ (\eta_0 + \bar{N}_i^{(t)}) (\boldsymbol{B}_i^{(t)})^{-1} \left(\frac{f_{\boldsymbol{\mu}_i}^{(t)}}{f_{\boldsymbol{\mu}_i}^{(t)} - 2} \Sigma_{\boldsymbol{\mu}_i}^{(t)} + (\boldsymbol{x}_n - \bar{\boldsymbol{\mu}_i}^{(t)}) ((\boldsymbol{x}_n - \bar{\boldsymbol{\mu}_i}^{(t)})^T \right) \right\}
\end{aligned}$$

[STEP3] パラメータの事後分布の更新
以下の計算を行う。

$$
\begin{aligned}
\bar{N}_i^{(t)} &\leftarrow \sum_{n=1}^{N} \bar{z}_i^n \\
\bar{\boldsymbol{x}}_i^{(t)} &\leftarrow \sum_{n=1}^{N} \boldsymbol{x}_n \\
\boldsymbol{C}_i^{(t)} &\leftarrow \sum_{n=1}^{N} \bar{z}_i^n (\boldsymbol{x}_n - \bar{\boldsymbol{x}}_i^{(t)})(\boldsymbol{x}_n - \bar{\boldsymbol{x}}_i^{(t)})^T \\
\phi_i^{(t)} &\leftarrow \phi_0 + \bar{N}_i^{(t)} \\
\eta_i^{(t)} &\leftarrow \eta_0 + \bar{N}_i^{(t)} \\
\bar{\boldsymbol{\mu}}_i^{(t)} &\leftarrow \frac{\bar{N}_i^{(t)} \bar{\boldsymbol{x}}_i^{(t)} + \xi_0 \boldsymbol{\nu}_0}{\bar{N}_i^{(t)} + \xi_0} \\
f_{\boldsymbol{\mu}_i}^{(t)} &\leftarrow \eta_i^{(t)} + 1 - d \\
\boldsymbol{B}_i^{(t)} &\leftarrow \boldsymbol{B}_0 + \bar{\boldsymbol{C}}_i^{(t)} + \frac{\bar{N}_i^{(t)} \xi_0}{\bar{N}_i^{(t)} + \xi_0} (\bar{\boldsymbol{x}}_i^{(t)} - \boldsymbol{\nu}_0)(\bar{\boldsymbol{x}}_i^{(t)} - \boldsymbol{\nu}_0)^T \\
\Sigma_{\boldsymbol{\mu}_i}^{(t)} &\leftarrow \frac{\boldsymbol{B}_i^{(t)}}{(\bar{N}_i^{(t)} + \xi_0) f_{\boldsymbol{\mu}_i}^{(t)}}
\end{aligned}
$$

[STEP4]
$t \leftarrow t+1$ とする。収束したとき終了し、それ以外の場合は STEP2 に戻る。
このアルゴリズムにより、事後分布を求めることが出来る。
事後分布を求めた後、予測分布を求める。式 (3.6) より、

$$
p(\boldsymbol{x}^*|D, m) = \sum_{i=1}^{m} \langle \alpha_i \rangle_{q(\boldsymbol{\alpha}|m)} \left\langle N(\boldsymbol{x}^*; \boldsymbol{\mu}_i, \boldsymbol{S}_i^{-1}) \right\rangle_{q(\boldsymbol{\mu}_i, \boldsymbol{S}_i|m)} \tag{3.18}
$$

となる。

3.2.4 抽出手順

変分ベイズ法による混合正規分布推定を用いた対象物抽出の手順について示す。
[STEP1]
混合正規分布の混合数 m を事前に決めておく。入力画像からコーナー点を検出して座標を 0.0 から 1.0 の範囲にする。
[STEP2]
コーナーの座標点をデータ点として、前述の変分ベイズ法による混合正規分布の手法で事後分布を求め、予測分布を計算できるようにする。入力画像の幅と高さをそれぞれ w、h として、座標 (0,0) から (w-1,h-1) まで未知データとして予測分布を計算して、そのときの値が閾値 P_{TH} 以上ならば対象物の領域候補とする。
[STEP3]
入力画像を Hill-Climbing を用いた領域分割をする。この領域分割手法のパラメータは事前に決めておく。

[STEP4]

STEP2 で分割した領域について、STEP1 で求めた分布領域に入っている面積を求める。このときの面積とその領域全体の面積の比が PA_{TH} 以上ならば対象物の領域として抽出する。

分割された領域 0 から 2 があり、青い線で示した推定した分布領域をそれに重ねる。PA_{TH} が 0.5 として、領域 1 以外は領域が分布領域の半分以上と重なっているため、重なった領域の面積と領域全体の面積の比は 0.5 以上であるといえ、領域 1 以外を抽出する。

抽出処理が完了したら終了する。

3.3　提案手法：グラフカットを用いた手法

本手法ではグラフカットでセグメンテーションするための領域指定を顕著性を示す画像を用いて行い、手動で領域指定するところを自動化している。また、捉えにくい対象物を抽出するために顕著性を示す画像を複数作成して、それぞれセグメンテーションをすることで複数の候補を作成している。

図 3.1 に処理の流れを示す。入力画像から輝度、赤、緑、青、黄の成分別に顕著性を示す画像を作成し、それらの画像を比較して似ているものを統合して 1 つ以上の画像にしてグラフカットでセグメンテーションを行う。

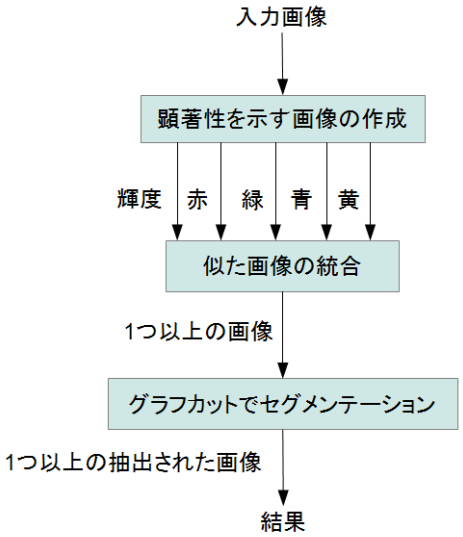

図 3.1: グラフカットを用いた対象物抽出

以下に提案手法の詳細を示す。入力画像は RGB 色空間とする。

3.3.1　顕著性を示す画像の作成

入力画像からレベル 0 から M まで原画像をダウンサンプリングした画像を作成する。レベル l のときの画像を I_l とし、画像は原画像の $1/2^l$ に縮小されている。レベル 0 を原画像として、レベル M までダウンサンプリングした場合におけるレベルを $l(0 \leq l \leq M)$ とする。全てのレベルの画像について輝度成分と赤、緑、青、黄の色成分に分解した画像を作成する。輝度成分 L、赤成分 R、緑成分 G、青成分 B で画像の画素の要素、赤 p_r、緑 p_g、青 p_r として、それぞれの

成分を以下の式で算出する。[23)]

$$L = \frac{p_r + p_g + p_b}{3} \tag{3.19}$$

$$R = p_r - \frac{p_g + p_b}{2} \tag{3.20}$$

$$G = p_g - \frac{p_r + p_b}{2} \tag{3.21}$$

$$B = p_b - \frac{p_r + p_g}{2} \tag{3.22}$$

$$Y = \frac{p_r + p_g}{2} - |p_r - p_g| - p_b \tag{3.23}$$

レベル l のときのそれぞれの成分を L_l、R_l、G_l、B_l、Y_l とする。

それぞれの成分に分けた画像を成分ごとに統合する。$resize(I, J)$ を最近傍法によって画像 I を画像 J の大きさにした画像を返す関数として、輝度成分 L_l を統合するための式を以下に示す。$l = 1, 2, ..., M-1$ として、

$$L'_{M-1} = |L_{M-1} - resize(L_M, L_{M-1})| \tag{3.24}$$

$$L'_{l-1} = L'_l + |L_{l-1} - resize(L_l, L_{l-1})| \tag{3.25}$$

式 (3.24)、式 (3.25) を用いて統合した L_l を L'_0 とする。

以下に R_l、G_l、B_l、Y_l を統合するための式を示す。$I_l = R_l, G_l, B_l, Y_l$、$l = 1, 2, ..., M-1$ として、

$$I'_{M-1} = I_{M-1} + resize(I_M, I_{M-1}) \tag{3.26}$$

$$I'_{l-1} = I_{l-1} + resize(I'_l, I_{l-1}) \tag{3.27}$$

このとき、統合した画像をそれぞれ R'_0、G'_0、B'_0、Y'_0 とする。

L'_0、 R'_0、 G'_0、 B'_0、 Y'_0 について正規化を行う。色成分 R'_0、G'_0、B'_0、Y'_0 について、それぞれの画像について互いの画素の値を比較する。次式に比較の式を示す。

$$\frac{count(I, J, t_1)}{area(I)} \geq t_2 \tag{3.28}$$

ここで、$count(I, J, t_1)$ は画像 I、J について同座標の画素を減算した値が t_1 以下の場合にカウントし、全ての画素におけるカウントの合計を返す関数である。$area(I)$ は画像 I の縦横のサイズの積を返す関数である。しきい値 t_2 以上の時、画像 I と J を $I + J$ とする。それ以外の場合はそのままとする。色成分同士で全て比較し、また生成された画像の集合を I'' とする。同様に L_0 と I'' の要素で比較を行う。ただし、I'' の要素同士では比較を行わない。このときのしきい値を t_3 とする。生成された画像の集合を C として C の要素全てを正規化する。また、C に L'_0、R'_0、 G'_0、 B'_0、 Y'_0 を全て加算して正規化した画像を追加する。この処理によって C が画像の顕著性を表す画像の集合となる。

3.3.2 セグメンテーション

C について、C の要素をそれぞれグラフカットを用いてセグメンテーションを行う。

以下にグラフカットによるセグメンテーション [21)] について述べる。画像 P でその画素 $p \in P$ としたとき、2 値のラベル $W = (W_0, ..., W_p, ..., W_{|P|})$ とする。このときのラベルについて、対象物を示す”obj”または背景を示す”bkg”が与えられる。また、p の近傍画素を $q \in N$ とする。

グラフカットによるセグメンテーションで作成するグラフに用いるモデルを以下に定義する。$U(W)$ は領域に対するペナルティ関数、$V(W)$ は境界に対するペナルティ関数である。$U(\cdot)$ は画素 p が対象物または背景のモデルにどれだけ一致するかを示し、$V_{p,q}$ は画素 p の近傍画素 q の輝度値が似ていると大きな値を出力する関数である。

$$E(W) = \lambda \cdot U(W) + V(W) \tag{3.29}$$

$$U(W) = \sum_{p \in P} U_p(W_p) \tag{3.30}$$

$$V(W) = \sum_{\{p,q\} \in N} V_{\{p,q\}} \cdot \delta(W_p, W_q) \tag{3.31}$$

$$\delta(W_p, W_q) = \begin{cases} 0 & \text{if} \quad W_p \neq W_q \\ 1 & \text{otherwise} \end{cases} \tag{3.32}$$

式 (3.29) が最小となるような W をグラフカットによって計算するためのグラフ G を構成する。画像 I、対象物である画素の集合 O、背景である画素の集合を β として、以下にエッジの重みを定義する。

表 3.1: エッジの重み

edge		cost	for
n-link	$\{p,q\}$	$V_{\{p,q\}}$	$\{p,q\} \in N$
t-link	$\{p,S\}$	$\lambda \cdot U_p(\text{"bkg"})$	$p \in P, p \notin O \cup \beta$
		K	$p \in O$
		0	$p \in \beta$
	$\{p,T\}$	$\lambda \cdot U_p(\text{"obj"})$	$p \in P, p \notin O \cup \beta$
		0	$p \in O$
		K	$p \in \beta$

$$U_p(\text{"obj"}) = -\ln Pr(I_p|O)$$

$$U_p(\text{"bkg"}) = -\ln Pr(I_p|\beta)$$

$$V_{\{p,q\}} \propto \frac{1}{dist(p,q)} \cdot \exp\left(-\frac{(I_p - Iq)^2}{2\sigma^2}\right)$$

$$K = 1 + \max_{p \in P} \sum_{q:\{p,q\} \in N} V_{\{p,q\}}$$

$dist(p,q)$ は pq 間のユーグリッド距離をあらわす関数である。グラフ G に対し最小カット最大フローアルゴリズム [24] を用いて対象物と背景を分割することでセグメンテーションできる。

$c \in C$ として c の画素を c_p、c_p のラベルを l_p とする。しきい値 tc_0、tc_1、tc_2 を導入して、$c_p \leq tc_0$ のとき $c_p \in \beta$、$tc_0 < c_p \leq tc_1$ のとき $c_p \notin O \cup \beta, l_p =$ "obj"、$c_p \leq tc_2$ のとき $c_p \in O$、それ以外を $c_p \notin O \cup \beta, l_p =$ "bkg" にする。これを全ての c について、それぞれグラフカットによるセグメンテーションを行う。それぞれのセグメンテーションを行った結果を出力して処理を終了する。

3.4 実験と結果

処理時間と精度について実験を行った。以下に実験方法とその結果を示す。

3.4.1 実験方法

処理時間の比較では、700 枚程度の画像を用意してプログラムを実行する。このときの処理時間を計測する。

精度の比較では、単純な背景と複雑な背景に分けたイラスト画像をそれぞれ 13 個、8 個用意してプログラムを実行する。手作業で抽出した画像とプログラムで抽出した画像を比較し、完全に一致した場合を 1、まったく一致しない場合を 0 として 0 から 1 の範囲の実数で正答率を求めた。ここで、単純な背景とは背景が 1 色だけのものや図形等が 1 個程度存在するようなもの、複雑な背景とは背景にエッジが多く含まれていたりする単純な背景以外のものとする。

ここでは提案手法と既存手法 [26] の分裂する動的輪郭モデルと領域分割を用いた手法、変分ベイズ法による混合正規分布推定と領域分割を用いた手法の比較を行う。ただし提案手法について、マルチスレッドを用いて同時にセグメンテーションを行っているために、処理時間は画像の顕著性を表す集合 C についてセグメンテーションをそれぞれ行った結果の処理時間の平均をとり、顕著性を示す画像を作成する処理の時間と合計したものである。また、精度に関して複数画像のうち最も正答率が高いものを選択している。

3.4.2 既存手法のパラメータ

既存手法のパラメータはイラスト画像における対象物抽出 [26] と同じにした。以下に実験における既存手法のパラメータを示す。

分裂する動的輪郭モデルのパラメータ

Hill-Climb を用いた領域分割のパラメータは H、S、V で 16、16、16 とした。

$$
\begin{aligned}
n &= 300 \\
t_{max} &= 1000 \\
C_{TH} &= 10 \\
D_{TH} &= 50.0 \\
\theta_{TH} &= 0.999 \\
w_{sp1} &= 1.0 \\
w_{sp2} &= 1.0 \\
w_{area} &= 3.0 \\
w_{dist} &= 1.0 \\
w_{edge} &= 2.0
\end{aligned}
$$

変分ベイズ法を用いた混合正規分布推定のパラメータ

Hill-Climb を用いた領域分割のパラメータは H、S、V で 16、16、16 とした。

- コーナー検出のパラメータ

$$\begin{aligned} \text{許容最低品質} &= 0.01 \\ \text{許容最低距離} &= 10 \\ \text{平均化ブロックのサイズ} &= 3 \end{aligned}$$

- 変分ベイズ法のパラメータ

$$\begin{aligned} m &= 7 \\ \phi_0 &= \frac{N}{m} \\ \xi_0 &= 1 \\ \eta_0 &= d+2 \\ \boldsymbol{\nu}_0 &= \hat{\boldsymbol{x}}_i \\ \boldsymbol{B}_0 &= \frac{1}{N}\sum_{n=1}^{N}(\boldsymbol{x}_i - \hat{\boldsymbol{x}}_i)^2\boldsymbol{I} \end{aligned}$$

3.4.3 提案手法のパラメータ

提案手法のパラメータはいくつかの画像を用いて提案手法で対象物抽出を行い、目視で最も良く対象物を抽出出来ていると思われるときのものを採用した。

$$\begin{aligned} M &= 8 \\ t_1 &= 0.05 \\ t_2 &= 0.3 \\ t_3 &= 0.1 \\ t_{c_0} &= 0.3 \\ t_{c_1} &= 0.22 \\ t_{c_2} &= 0.05 \end{aligned}$$

3.4.4 実験結果

それぞれの実験結果を示す図では提案手法は graphcut、動的輪郭モデルを用いた手法は csnake、変分ベイズ法を用いた手法は vb で示す。

処理時間

図 3.2 は、横軸が画素数、縦軸がミリ秒単位の時間のそれぞれの手法における処理時間を示したものである。提案手法は動的輪郭モデルを用いた手法に比べてやや時間がかかる。しかし動的輪郭モデルを用いた手法の場合に時間がかかることがあるが、提案手法は 10000ms 以下に収まっている。一方、提案手法と変分ベイズ法との比較では殆どの場合に提案手法のほうが早く処理できていることがわかる。

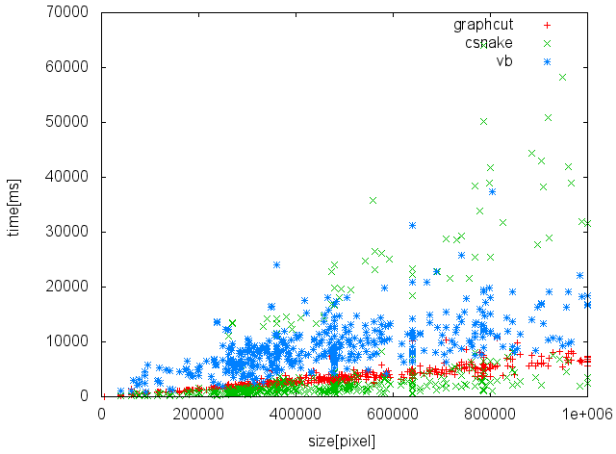

図 3.2: 処理時間

単純な背景における精度

図 3.3 は、横軸が画像を示す番号、縦軸が正答率の単純な背景における正答率を示している。既存手法が提案手法を上回ることがあるものの、提案手法はほぼ 0.9 以上であり既存手法に比べて安定した結果が得られたといえる。

複雑な背景における精度

図 3.4 は、単純な背景における精度と同様に横軸が画像を示す番号、縦軸が複雑な背景における正答率を示している。変分ベイズ法を用いた手法が提案手法を上回ることがあるものの、提案手法が既存手法よりも正答率が高いことが多い。画像 3 について赤、青、緑それぞれが大きく存在している画像なので、色成分で分解しているためにうまく抽出できなかったと思われる。

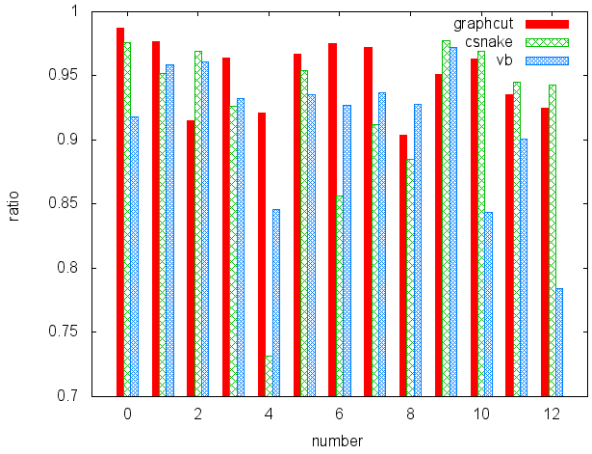

図 3.3: 単純な画像における正答率

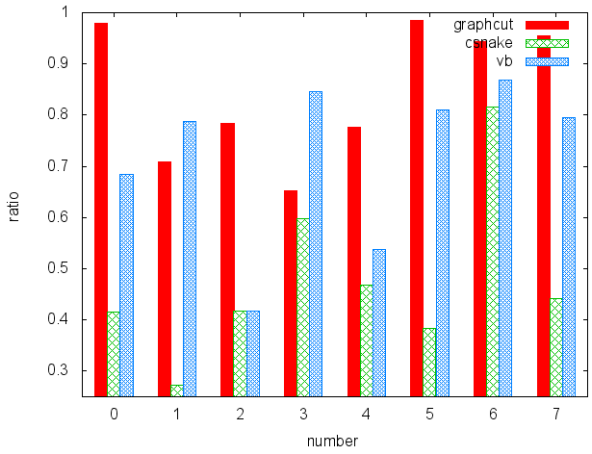

図 3.4: 複雑な画像における正答率

第4章　画像分類

4.1　提案手法

3 章で提案した対象物抽出と自己組織化マップ [4] を用いて分類器を作り、画像を分類する手法を提案する。

画像分類は既知の画像を用いて学習を行ってから分類を行う。提案手法では、対象物で分類してある画像を何枚ずつか用いて対象物抽出した結果を用いて自己組織化マップに学習させる。学習させたものを使って、分類したい画像を対象物抽出して自己組織化マップを使って分類する。

図 4.1 は学習時の処理、図 4.2 は分類時の処理を示す。学習時の処理、分類時の処理ともに対象物抽出、画像の大きさの正規化、入力データの生成を行う。これらは共通の前処理として 4.1.1 に示す。

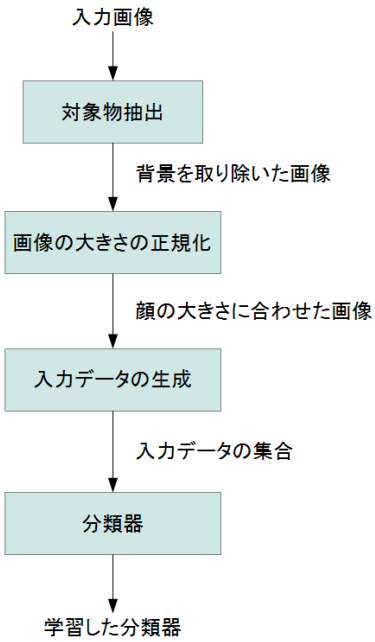

図 4.1: 学習時の処理

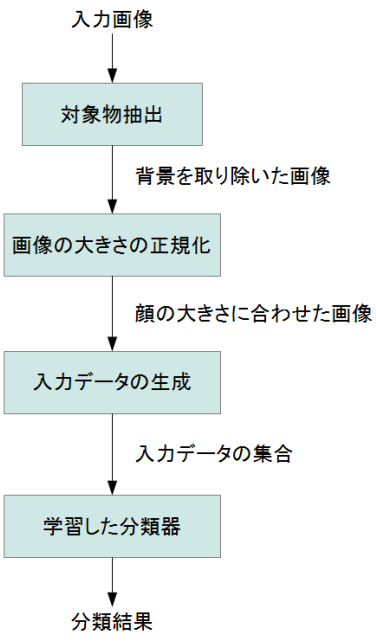

図 4.2: 分類時の処理

4.1.1 共通の前処理

対象物抽出とその結果の処理

学習や分類における自己組織化マップに対する入力データは、対象物抽出を行い、画像を正規化してから生成する。正規化では 2.6 の顔検出を用いて画像のサイズを変換している。これは、例えばキャラクタの全身だけではなく上半身のみの場合、画像全体に対する顔の大きさが全身のときに比べて上半身のみでは大きくなると考えて、顔のサイズを一定にすれば、画像に関わらず対象物の大きさが一定になるという仮定をしたためである。また、入力データは RGB 色空間ではなく L*a*b*色空間にしている。自己組織化マップでユーグリッド距離を使うことから、規格として色差をユーグリッド距離で表すことが出来る L*a*b*色空間が最適であると判断した。

対象物抽出の手法として 3.3 の手法を用いる。この手法では、抽出結果として複数の画像が出力される。複数の画像を絞り込むために 2.6 のアニメ顔検出を用いて検出できるかどうかを調べる。検出できればその画像を採用し、できなければ採用しない。

画像の大きさの正規化

2.6 の顔検出を用いると顔がある範囲を矩形で表される。この矩形のサイズが一定のサイズになるように画像全体を拡大縮小する。これを対象物抽出した結果の画像全てについて行う。このとき、変換後の矩形のサイズを以下に定義する。

$$\mathbf{r}_{face} = (\text{矩形の幅}, \text{矩形の高さ}) \tag{4.1}$$

入力データの生成

正規化した画像を用いて自己組織化マップに対する入力データを生成する。

画像を RGB 空間から L*a*b*空間に変換する。入力データは対象物抽出した結果の画像の一部分であり、$near \times near$ の窓である。このとき、$near$ は正の整数かつ奇数であり、中心座標は背景部分の座標は取らず、入力データ内の対象物であるピクセルの割合が閾値 t_{fg} 以上になるようにする。

これを対象物抽出の結果の各画像に対して、入力データの数が閾値 t_{smax} 個以下になるようにする。入力データをランダムにシャッフルした入力データの集合を X_{input} とする。

4.1.2　自己組織化マップと勝利ノードカウントマップによる分類器

図 4.3 に分類器の構成を示す。提案手法における分類器は、自己組織化マップと勝利ノードカウントマップを 1 つの分類グループと対応させたものであり、これが 2 つ以上の複数個で構成される。

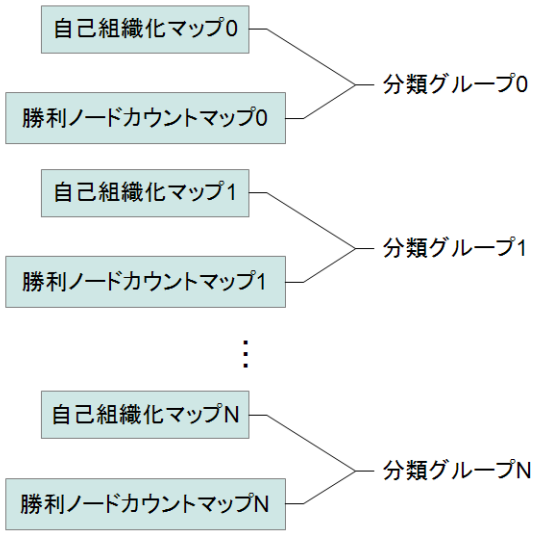

図 4.3: 分類器の構成

以下に提案手法の分類器における自己組織化マップと勝利ノードカウントマップについて示す。

自己組織化マップ

本提案手法では、最も一般的に思われる Kohonen の自己組織化マップ [4] を用いる。

自己組織化マップを用いた分類器は複数の自己組織化マップで構成されており、それぞれの自己組織化マップは各対象物と対応している。

分類グループの集合を T、T の要素に対応する自己組織化マップを M とする。このとき、n 番目の分類グループ T_n は M_n と対応する。ノードの数は c_{node} とする。

勝利ノードカウントマップ

勝利ノードカウントマップとは、全ての入力データについて式 (2.1) を用いて勝利ノードを判定し、それぞれのノードがどれだけ勝利したかをカウントしたものである。

入力データ集合の要素に対して式 (2.1) で勝利ノードを判定する。ノードには 0 から始まるカウンタがあり、カウンタに 1 加算する。これを入力データ集合の全てについて行う。カウンタを入力データ集合の要素数で正規化する。この自己組織化マップに対するカウンタをマップとみなして勝利ノードカウントマップとする。

図 4.4 に勝利ノードカウントマップと自己組織化マップとの関係の図を示す。

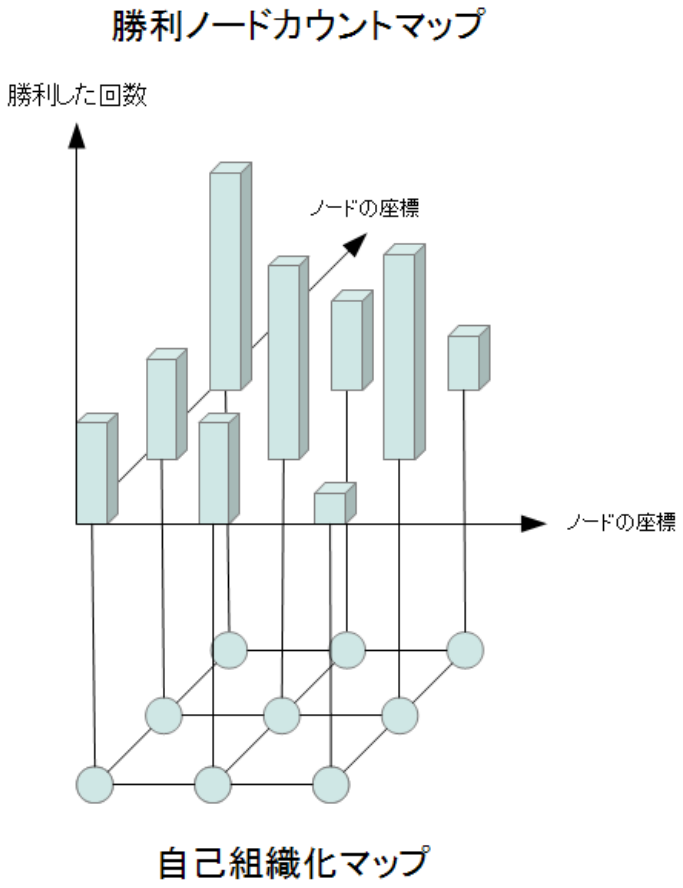

図 4.4: 勝利ノードカウントマップ

4.1.3 学習

図 4.5 に学習の処理を示す。

図 4.5 では、入力データ集合 X_{input} を自己組織化マップの学習と勝利ノードカウントマップの生成で用いて、学習済みの自己組織化マップとその勝利ノードカウントマップを結果として返すことを表している。学習済み自己組織化マップと学習時勝利ノードカウントマップを保持しておき、分類時に使用する。

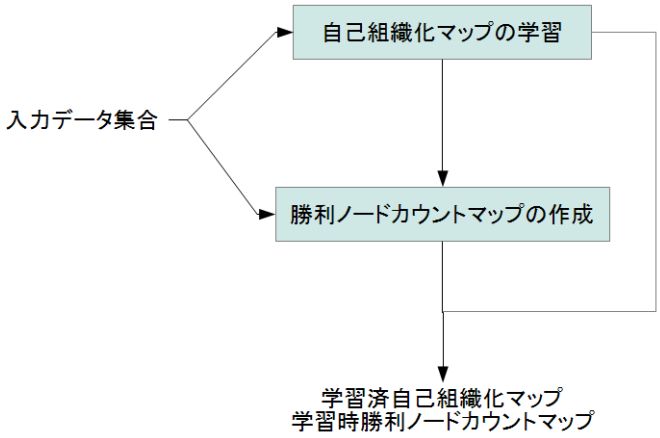

図 4.5: 学習の処理

以下に学習処理の詳細を示す。

M に対応する学習用入力画像を用いて、M を学習させる。

4.1.1 で示した共通の前処理を行い、入力データ集合 X_{input} を生成する。

自己組織化マップの初期化はノードの数だけ X_{input} を順番に用いて行う。

式 (2.1) と式 (2.2) を用いて勝利ノードの判定と更新を行う。このとき、勝利ノードを中心として近傍関数 h_{ci} を以下のように定義する。

$$
\begin{aligned}
h_{ci} &= \begin{cases} t_{center} & \text{勝利ノード} \\ t_{near} & \text{勝利ノード以外の近傍 1 のノード} \end{cases} \\
t_{center} &= \text{勝利ノードに対する重み} \\
t_{near} &= \text{勝利ノード以外の近傍 1 のノードに対する重み}
\end{aligned}
\tag{4.2}
$$

これを全ての M について、それぞれ学習させる。

学習させた後、X_{input} を M のそれぞれに入力して学習時勝利ノードカウントマップを生成する。それらを M_{win} とする。それぞれの M と M_{win} を学習結果とする。

4.1.4 分類

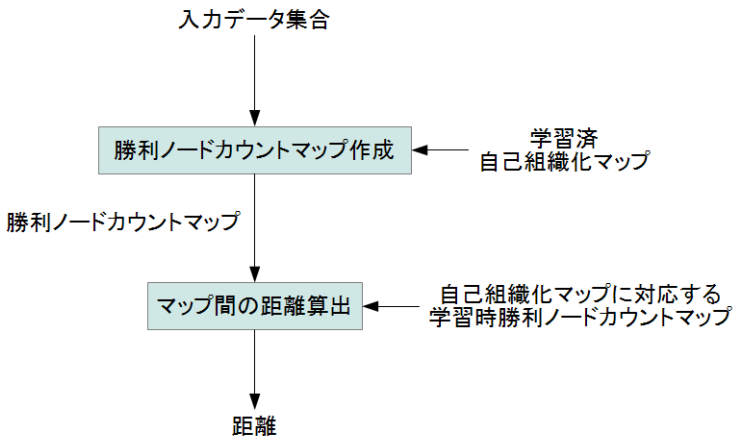

図 4.6: 分類に使う距離を求める

図 4.6 に分類に使う距離を求める処理を示す。分類したい画像から生成した入力データ集合と学習済みの自己組織化マップ M を使って勝利ノードカウントマップ M'_{win} を生成する。M'_{win} を M と対応する学習時勝利ノードカウントマップ M_{win} と比較して距離を算出する。

全ての M と M_{win} について、M'_{win} と比較して距離を求めて最も距離が短いものを採用し、それに対応する対象物に分類する。

以下に分類処理の詳細を示す。

学習済みの M を用いて画像の分類を行う。

分類したい画像から 4.1.1 で示した共通の前処理で学習と同様に入力データ集合 X_{input} を生成する。

それぞれの学習済み M に X_{input} を入力したとき、勝利ノードカウントマップ M'_{win} を生成する。勝利ノードの判定には式 (2.1) を用いる。

M'_{win} それぞれについて、対応する M_{win} と比較する。このとき、M'_{win} と対応する M_{win} で同じ座標に存在するノードでユーグリッド距離を求め、全てのノードのユーグリッド距離の合計が最も小さいものに対応する対象物へ分類を行う。

4.2　実験と結果

4.2.1　実験内容

以下に提案手法に対する実験の内容について述べる。

実験では、学習用の画像を提案手法で全て学習させ、分類用の画像を提案手法を用いて分類を行った。個々の画像には対象物が 1 つ含まれている。

まず、学習用の画像を提案手法の分類器に全て学習させる。学習した分類器によって分類用の画像を行い、その結果から対象物ごとのディレクトリに画像をコピーする。分類を分類用の画像全てについて行い、分類グループごとのディレクトリにある画像を結果とする。

分類グループは 3 つあり、それぞれ T_0, T_1、T_2 とする。それぞれの例を図 4.7 [1]、図 4.8 [2]、図 4.9 [3]に示す。学習用の画像はそれぞれ 7、7、6 個、分類用の画像はそれぞれ 35、49、39 個である。学習用の画像、分類用の画像ともに顔画像検出される画像に限定している。

また、分類器のパラメータを変えて実験を行った。対象物抽出のパラメータは 3.4.3 を用いる。

図 4.7: T_0の例

図 4.8: T_1 の例

図 4.9: T_2の例

[1]『チェックポイント』ー関口みぃる web site ー http://check-p.com/cg/color/2007_html/2007_17.htm
[2]ARANCIO TELA http://homepage2.nifty.com/aranciotela/log_11.htm
[3]りとほり http://litholi.net/pictures/th03

4.2.2 実験結果

評価方法

提案手法を用いて分類した画像が入ったディレクトリにある画像の数を C_i、予め正しく分類した画像が入ってるディレクトリにある画像の数を A_i として、提案手法を用いて分類した画像が入ったディレクトリと予め正しく分類した画像が入ったディレクトリを比較して同じ画像の数を p_i とする。

個々のディレクトリに関して、正しく分類された率 $positive_i$ を以下の式で計算する。

$$positive_i = \frac{p_i}{A_i} \tag{4.3}$$

一方、誤った画像が入った率 $negative_i$ を以下の式で計算する。

$$negative_i = \frac{C_i - p_i}{C_i} \tag{4.4}$$

$positive_i$ は $0 \leq positive_i \leq 1$ で、値が大きいほど良い。例えば、0 ならば対象物の入った画像がなく、1 ならば対象物の入った画像があるといえる。

$negative_i$ は $0 \leq negative_i \leq 1$ で、値が小さいほど良い。例えば、0 ならば対象物と違う画像が入っていない状態、1 ならば全て対象物ではない画像しかない状態を示す。

全体の正答率 $positive_{all}$ は、分類する全ての画像の数 A_{all}、正しく分類された画像の数 p_{all} として、以下の式で求める。

$$positive_{all} = \frac{p_{all}}{A_{all}} \tag{4.5}$$

$positive_{all}$ は $0 \leq positive_{all} \leq 1$ で、値が大きいほど良い。例えば、0 ならばすべての分類用画像と対象物の分類が一致していない状態、1 ならば全ての分類用画像と対象物の分類が一致している状態を示す。

全体の結果

図 A.1 から図 A.12 に対象物を分類した結果を示す。その一部を図 4.10 から図 4.12 に示す。横軸はウィンドウサイズ（$n \times n$ の n のみ表示）、縦軸は $positive_{all}$ である。

顔のサイズが 16×16、32×32、64×64 のいずれもウィンドウサイズが大きくなるほど $positive_{all}$ の値が小さくなっていく。ただし、顔のサイズが大きくなるほど差は小さくなっているといえる。また、ウィンドウサイズが 1×1 や 3×3 も低くなっている事が多い。$positive_{all}$ が最も高くなるピークは、5×5 から 9×9 付近に多い事がわかる。

個々の対象物の結果

図 A.13 から図 A.48 に対象物を分類した結果を示す。その一部を図 4.13 から図 4.24

T_1、T_2 に関して、$positive_1$、$positive_2$ は 0.8 を超えていることが多く、$negative_1$、$negative_2$ は 0.2 前後かそれ以下に抑えられていることが多い。

T_0 に関して、$positive_0$ は 0.8 以上担っているものが多いものの、$negative_0$ は 0.4 以上と他の対象物に比べて高くなっていることがわかる。

対象物抽出で背景が残ってしまい、他の対象物の特徴となる色と区別ができなくなって他の対象物に分類されてしまうと考えれられる。特に T_0 は、黒色と白色が特徴となっているので、背景が黒色では T_0 に分類されてしまうことが多くなったと考えられ、$negative_0$ の値が高くなってしまったと思われる。

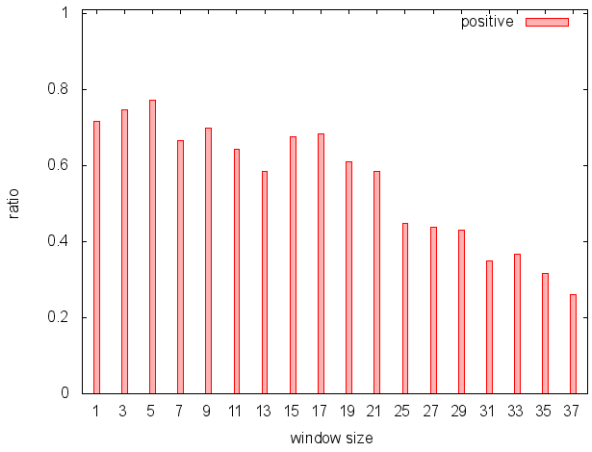

図 4.10: 顔サイズ 16×16　中心の重み 0.5　近傍の重み 0.1

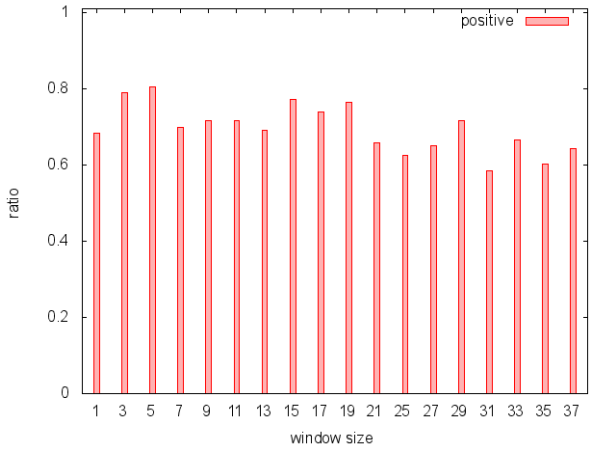

図 4.11: 顔サイズ 32×32　中心の重み 0.5　近傍の重み 0.1

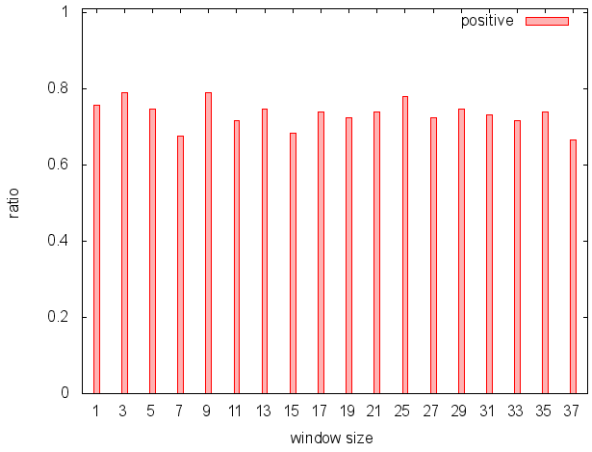

図 4.12: 顔サイズ 64×64　中心の重み 0.5　近傍の重み 0.1

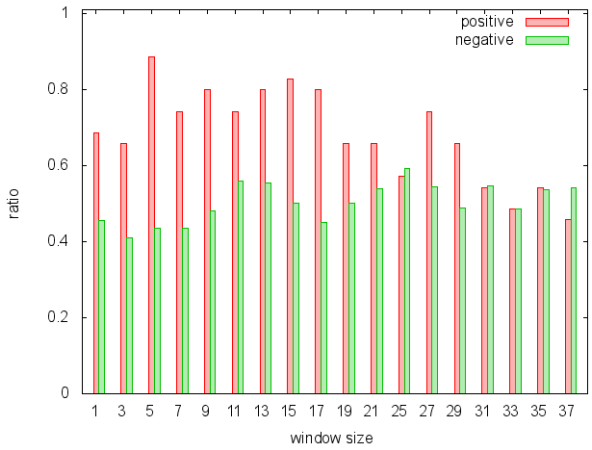

図 4.13: 対象物 1　顔サイズ 16 × 16　中心の重み 0.5　近傍の重み 0.1

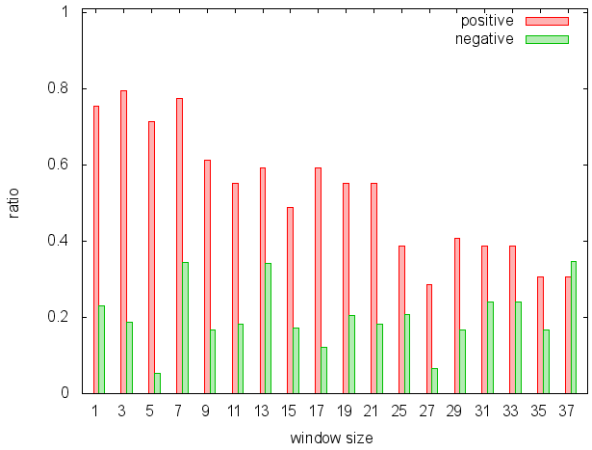

図 4.14: 対象物 2　顔サイズ 16 × 16　中心の重み 0.5　近傍の重み 0.1

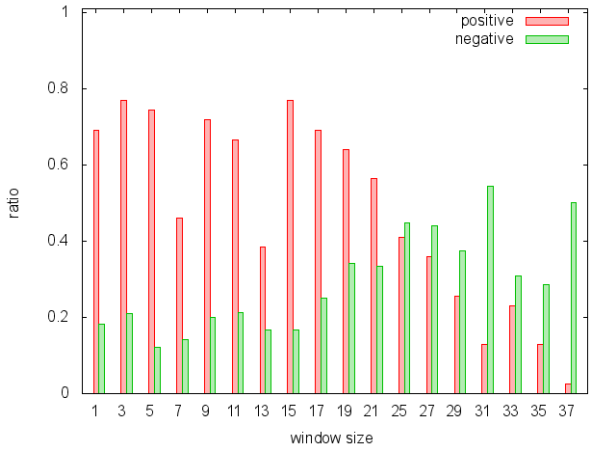

図 4.15: 対象物 3　顔サイズ 16 × 16　中心の重み 0.5　近傍の重み 0.1

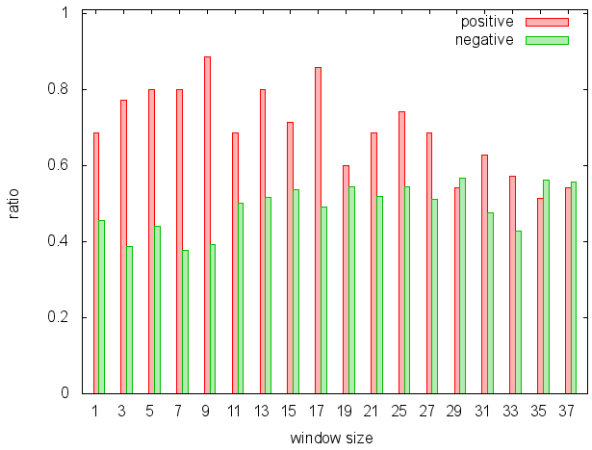

図 4.16: 対象物 1　顔サイズ 16 × 16　中心の重み 0.5　近傍の重み 0.3

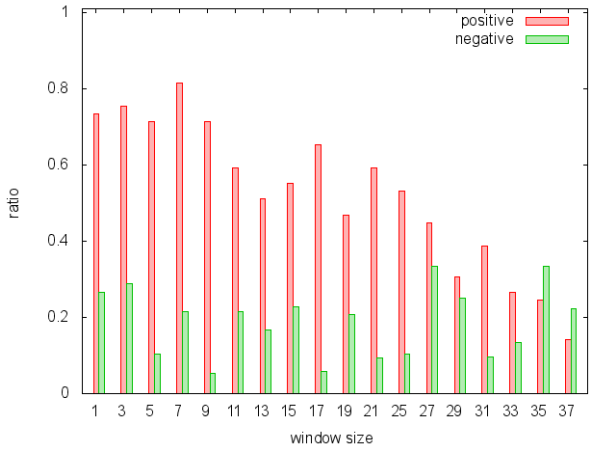

図 4.17: 対象物 2　顔サイズ 16 × 16　中心の重み 0.5　近傍の重み 0.3

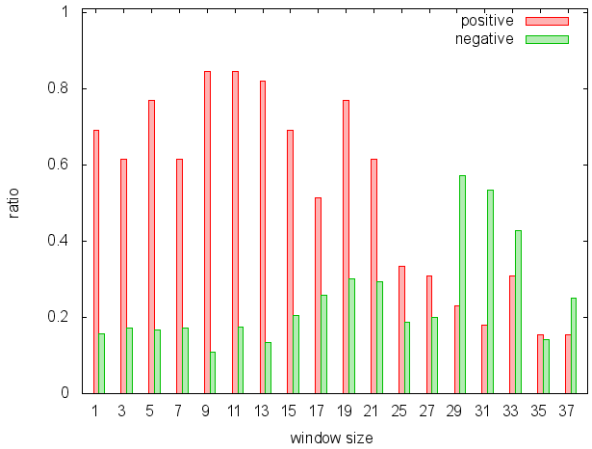

図 4.18: 対象物 3　顔サイズ 16 × 16　中心の重み 0.5　近傍の重み 0.3

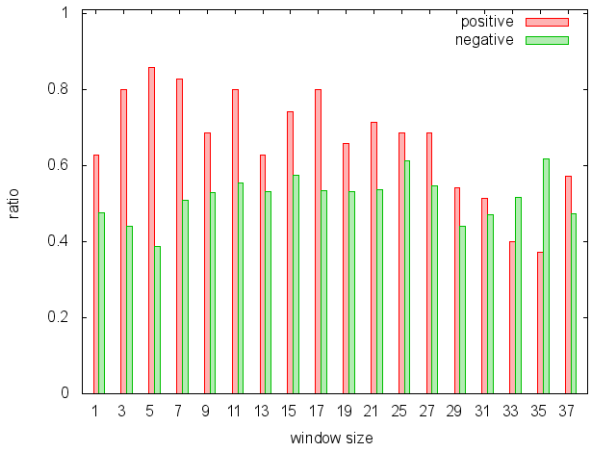

図 4.19: 対象物 1　顔サイズ 16 × 16　中心の重み 0.7　近傍の重み 0.3

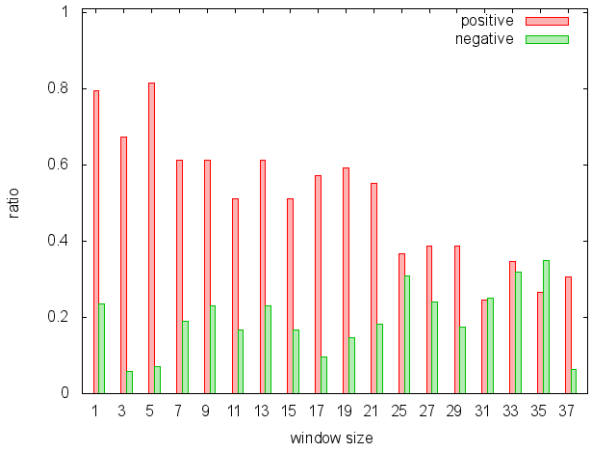

図 4.20: 対象物 2　顔サイズ 16 × 16　中心の重み 0.7　近傍の重み 0.3

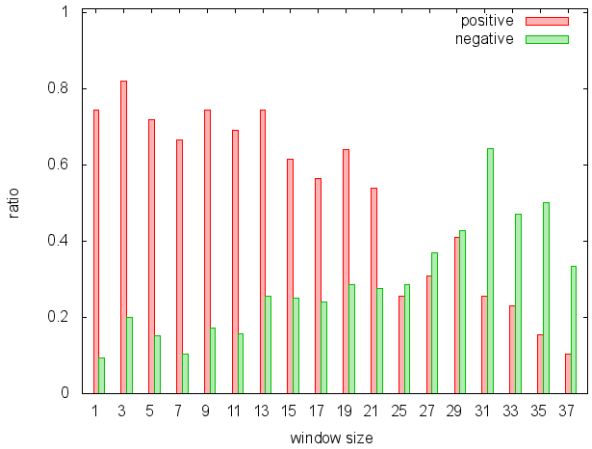

図 4.21: 対象物 3　顔サイズ 16 × 16　中心の重み 0.7　近傍の重み 0.3

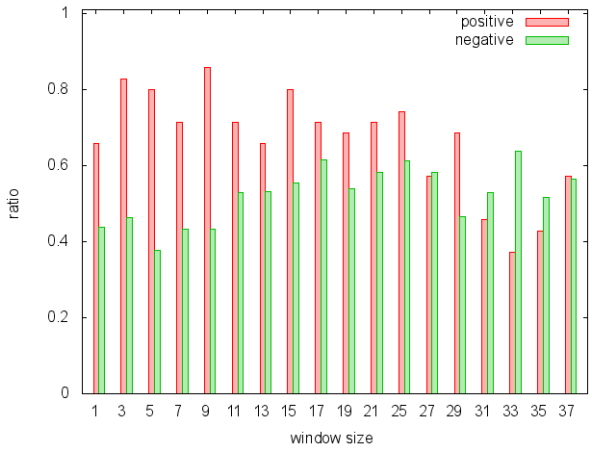

図 4.22: 対象物 1　顔サイズ 16 × 16　中心の重み 0.7　近傍の重み 0.5

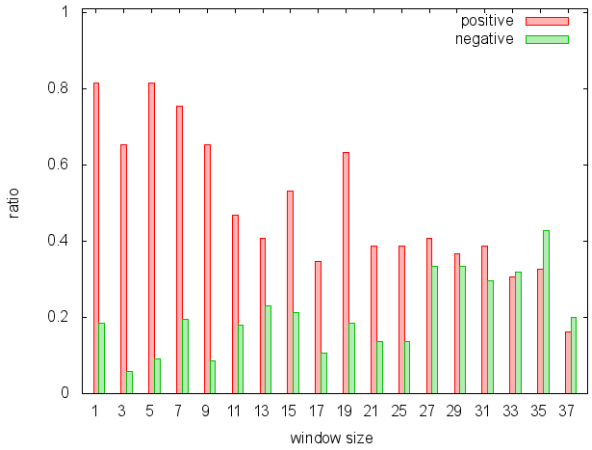

図 4.23: 対象物 2　顔サイズ 16 × 16　中心の重み 0.7　近傍の重み 0.5

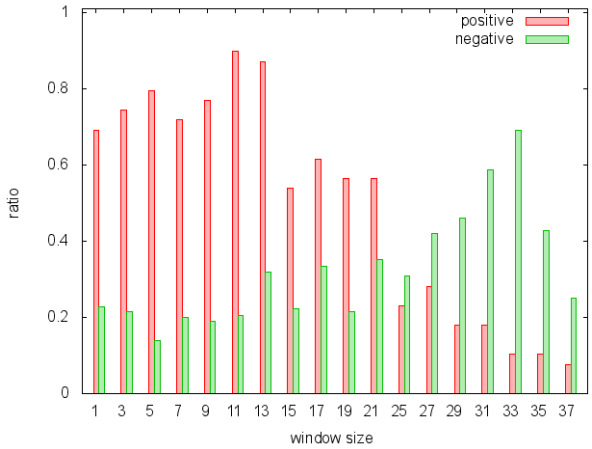

図 4.24: 対象物 3　顔サイズ 16 × 16　中心の重み 0.7　近傍の重み 0.5

第5章　おわりに

本研究ではグラフカットを用いた対象物抽出と自己組織化マップを用いた画像分類の手法を提案した。以下にそれぞれの手法についてまとめる。

5.1　グラフカットを用いた対象物抽出

顕著性を表す画像を作成して、統合し、グラフカットでセグメンテーションを行う手法を提案した。

3.4 の実験結果より、多くの場合に既存手法に比べて提案手法が有効であると思われる。問題点として、赤、緑、青、黄において別々の場所に顕著性が現れた場合にそれぞれ独立した対象物とみなしてしまい、うまく抽出できないことがある。画像にグラデーションが使われている場合、対象物に被さると抽出しにくいことがある。対象物と背景で使われている色が似ていると抽出しにくいことがある。といったような色成分で分解して顕著性を調べる手法における問題点もあることがわかった。今後は、しきい値の決定を自動化してしきい値の数を減らすことや手法をさらに追加したり修正したりすることによって精度の向上を図ることが考えられる。

5.2　自己組織化マップを用いた画像分類

グラフカットを用いた対象物抽出を行った画像から生成した自己組織化マップと勝利ノードカウントマップを用いた手法を提案した。

4.2.2 および 4.2.2 の実験結果より、全体の結果としてウィンドウサイズが 5×5 から 9×9 にかけて最も良い結果になっていることが多い。処理時間は顔サイズやウィンドウサイズに依存するため、それぞれが小さいほど短い処理時間となる。したがって、結果が同程度の正答率ならば、顔サイズとウィンドウサイズが小さくするほうが良いといえ、顔サイズ 16×16 のウィンドウサイズ 5×5 が本実験において最も良いパラメータと考えられる。

できるだけ学習させる画像の数を少なくなるようにした画像分類の手法を提案した。実験結果より、限定した条件下であるものの正答率が 8 割程度である。より広い範囲の画像を識別するには、対象物抽出や分類器の精度の向上させなければならない。分類器の精度向上について、提案手法で行なっている画像の比較はパターンマッチングと同等なので、パターンマッチングの改良手法を使うことで精度向上ができるのではないかと思われる。

謝辞

本研究にあたり、多くの助言、指導をしてくださった三好力教授に心からお礼申し上げます。
また、様々なコメントをしてくださった三好研究室の諸氏に厚くお礼申し上げます。

これまでの発表履歴

- 奥村亮仁, 三好力,
  イラスト画像に対する対象物抽出,
  (情報処理学会創立 50 周年記念大会 (第 72 回) 全国大会講演論文集 2010) p.2.647-2.648, 1Y-7
- 奥村亮仁, 三好力, グラフカットを用いたイラスト画像に対する対象物抽出,
  第 27 回ファジィシステムシンポジウム, TF3-1

参考文献

1) R. ディーステル著, 根上生也/太田克弘訳,
グラフ理論,
(シュプリンガー・ジャパン, 2000).

2) Corinna Cortes, Vladimir Vapnik,
Support-Vector Networks
(Machine Learning, 20, 1995) p.273-297.

3) Yoav Freund, Robert E. Schapire,
A decision-theoretic generalization of on-line learning and an application to boosting
(1996).

4) Teuvo Kohonen,
The Self-Organizing Map
(PROCEEDINGS OF THE IEEE, VOL. 78, No. 9, SEPTEMBER 1990) p.1464-1480.

5) anime.udp.jp,
http://anime.udp.jp/

6) OpenCV 2.2 C++ リファレンス – カスケード型分類器,
http://opencv.jp/opencv-2svn/cpp/objdetect_cascade_classification.html

7) 玉木 徹,
画像中の物体および人物領域の抽出手法に関する研究 (2000).

8) Michael Kass, Andraw Witkin, and Demetri Terzopoulos,
Snakes: Active Contour Models (1988).

9) 荒木 昭一, 横矢 直和, 岩佐 英彦, 竹村 治雄,
複数物体の抽出を目的とした交差判定により分裂する動的輪郭モデル (1996).

10) Donna J. Williams and Mubark Shah,
Fast Algorithm for Active Contours and Curvature Estimation (1991).

11) 大橋 巧, Zaher AGHBARI, 牧之内 顕文,
Hill-Climbing を用いたイメージセグメンテーション (2003).

12) 上田 修功,
EM アルゴリズムの新展開:変分ベイズ法 (2002).

13) 上田 修功,
ベイズ学習 [II] -ベイズ学習の基礎- (2002).

14) 上田 修功,
ベイズ学習 [III] -変分ベイズ学習の基礎- (2002).

15) 上田 修功,
ベイズ学習 [IV・完] -変分ベイズ学習の応用例- (2002).

16) 物理のかぎしっぽ,
変分法 1
http://www12.plala.or.jp/ksp/mathInPhys/variations1/ (2005).

17) C・M・ビショップ著, 元田 浩/栗田 多喜夫/樋口 知之/松本 裕治/村田 昇監訳,
パターン認識と機械学習 上 ベイズ理論による統計的予測
(シュプリンガー・ジャパン, 2007).

18) C・M・ビショップ著, 元田 浩/栗田 多喜夫/樋口 知之/松本 裕治/村田 昇監訳,
パターン認識と機械学習 下 ベイズ理論による統計的予測
(シュプリンガー・ジャパン, 2007) p.175-200.

19) 荒木佑季,
変分ベイズ学習による混合正規分布推定 -検証と改善- (2008).

20) 荒木佑季,
変分ベイズ学習の具体的使用法 (2008) p.1-4.

21) Yuri Y. Boykov, Marie-Pierre Jolly,
Interactive Graph Cuts
(Proceedings of "Internation Conference on Computer Vision",
Vancouver, Canada, vol.I 2001) p.105-112.

22) 石川博,
グラフカット,
(情報処理学会研究報告 2007-CVIM-158-26) p.193-204.

23) L.Itti, C.Koch and E.Niebur,
A Model of Saliency-based Visual Attention for Rapid Scene Analysis
(IEEE Trans. Pattern Analysisand Machine Intelligence, Vol.20, No.11, 1998)
p.1254-1259.

24) Yuri Boykov, Vladimir Kolmogorov,
An Experimental Comparison of Min-Cut/Max-Flow Algorithm for Energy Minimization
in Vision
(In IEEE Transactions on PAMI, Vol. 26, No. 9, 2004) p.1124-11137.

25) 福田恵太, 滝口哲也, 有木康雄,
AdaBoost と Saliency Map を用いた Graph Cuts による花弁領域の自動抽出法
(画像の認識・理解シンポジウム (MIRU2008), 2008) p.796-801.

26) 奥村亮仁, 三好力,
イラスト画像に対する対象物抽出,
(情報処理学会創立 50 周年記念大会 (第 72 回) 全国大会講演論文集 2010) p.2.647-2.648,
1Y-7

付 録 A　実験結果のグラフ

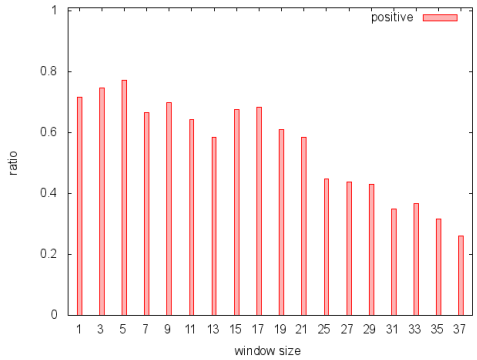

図 A.1: 顔サイズ 16 × 16　中心の重み 0.5　近傍の重み 0.1

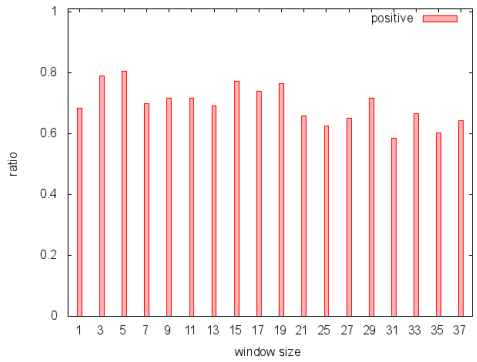

図 A.2: 顔サイズ 32 × 32　中心の重み 0.5　近傍の重み 0.1

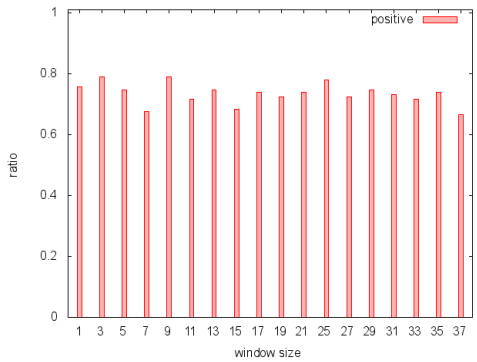

図 A.3: 顔サイズ 64 × 64　中心の重み 0.5　近傍の重み 0.1

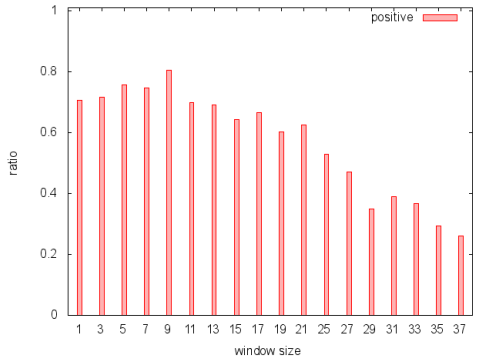

図 A.4: 顔サイズ 16×16　中心の重み 0.5　近傍の重み 0.3

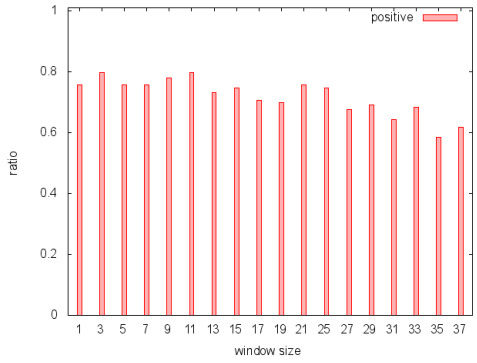

図 A.5: 顔サイズ 32×32　中心の重み 0.5　近傍の重み 0.3

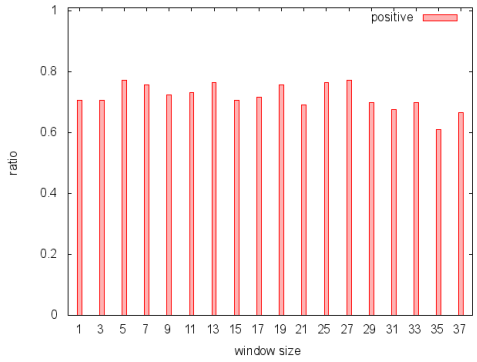

図 A.6: 顔サイズ 64×64　中心の重み 0.5　近傍の重み 0.3

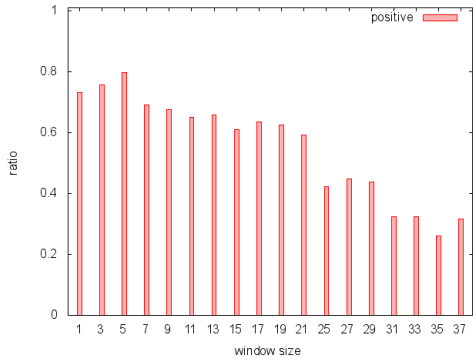

図 A.7: 顔サイズ 16×16　中心の重み 0.7　近傍の重み 0.3

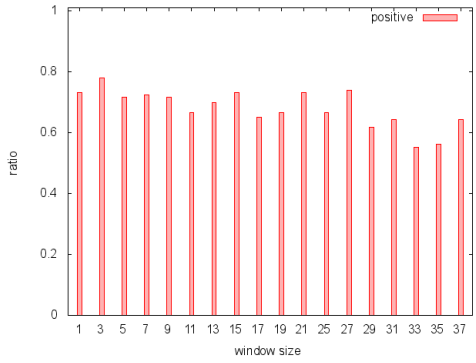

図 A.8: 顔サイズ 32×32　中心の重み 0.7　近傍の重み 0.3

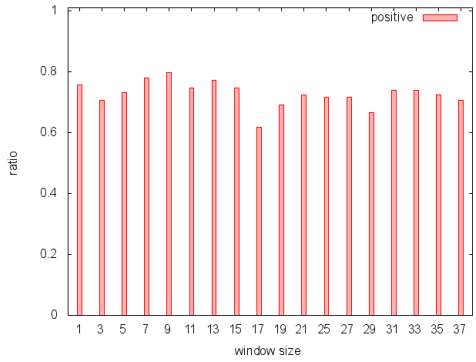

図 A.9: 顔サイズ 64×64　中心の重み 0.7　近傍の重み 0.3

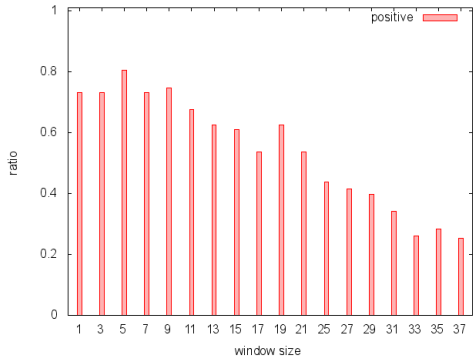

図 A.10: 顔サイズ 16 × 16　中心の重み 0.7　近傍の重み 0.5

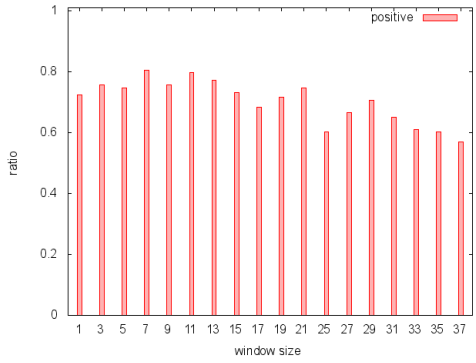

図 A.11: 顔サイズ 32 × 32　中心の重み 0.7　近傍の重み 0.5

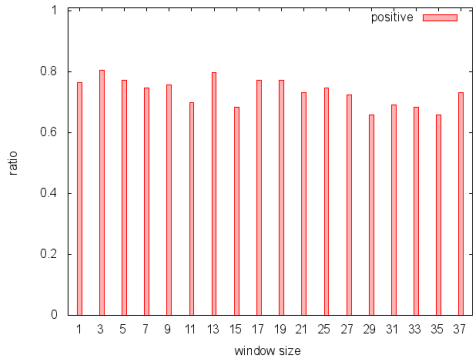

図 A.12: 顔サイズ 64 × 64　中心の重み 0.7　近傍の重み 0.5

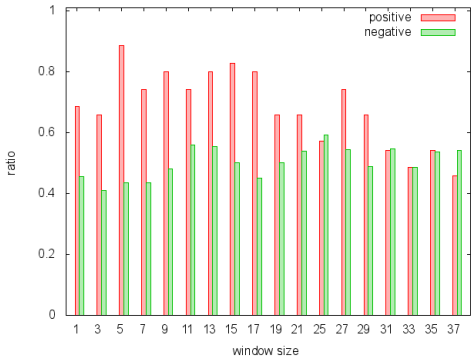

図 A.13: 対象物 1　顔サイズ 16 × 16　中心の重み 0.5　近傍の重み 0.1

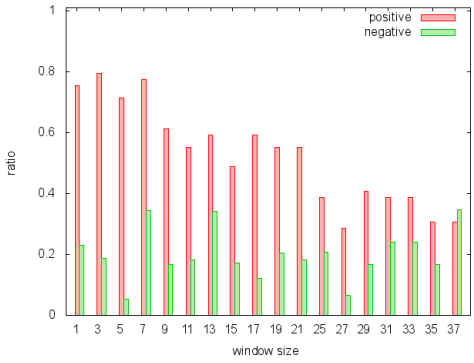

図 A.14: 対象物 2　顔サイズ 16 × 16　中心の重み 0.5　近傍の重み 0.1

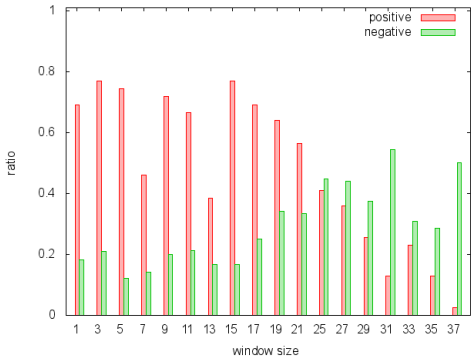

図 A.15: 対象物 3　顔サイズ 16 × 16　中心の重み 0.5　近傍の重み 0.1

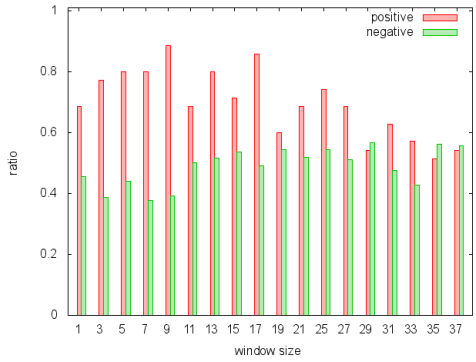

図 A.16: 対象物 1　顔サイズ 16 × 16　中心の重み 0.5　近傍の重み 0.3

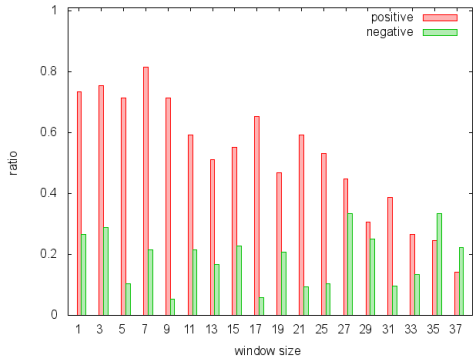

図 A.17: 対象物 2　顔サイズ 16 × 16　中心の重み 0.5　近傍の重み 0.3

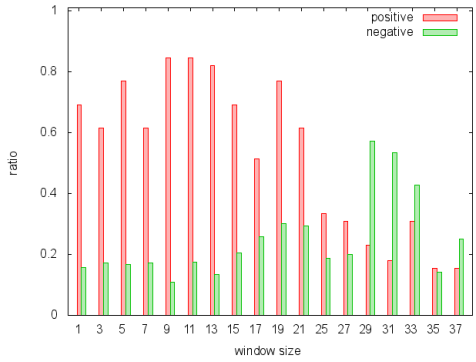

図 A.18: 対象物 3　顔サイズ 16 × 16　中心の重み 0.5　近傍の重み 0.3

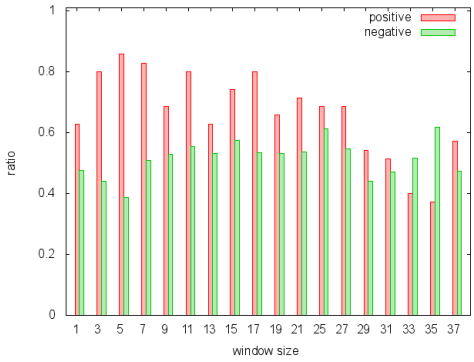

図 A.19: 対象物 1　顔サイズ 16 × 16　中心の重み 0.7　近傍の重み 0.3

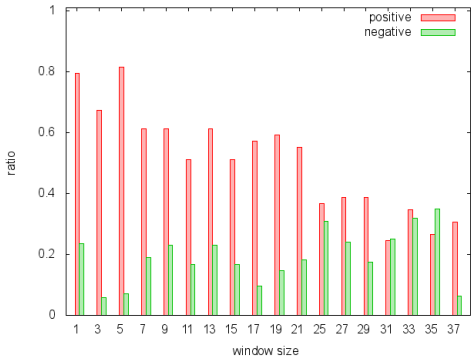

図 A.20: 対象物 2　顔サイズ 16 × 16　中心の重み 0.7　近傍の重み 0.3

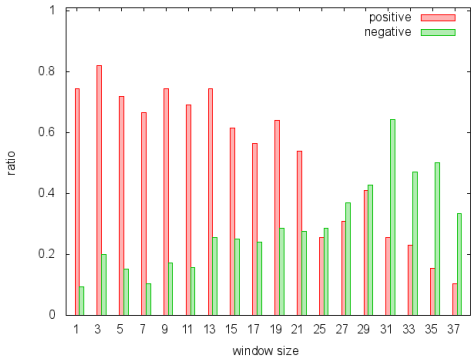

図 A.21: 対象物 3　顔サイズ 16 × 16　中心の重み 0.7　近傍の重み 0.3

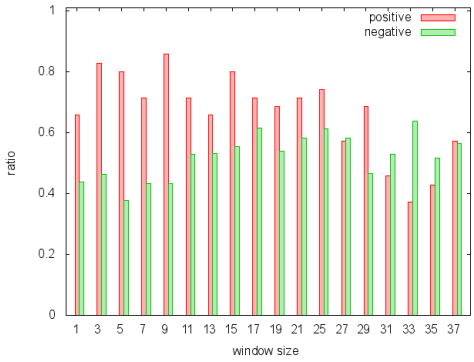

図 A.22: 対象物 1　顔サイズ 16 × 16　中心の重み 0.7　近傍の重み 0.5

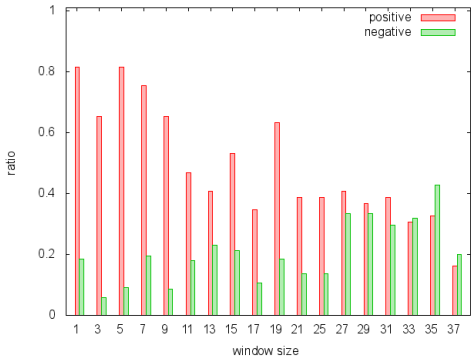

図 A.23: 対象物 2　顔サイズ 16 × 16　中心の重み 0.7　近傍の重み 0.5

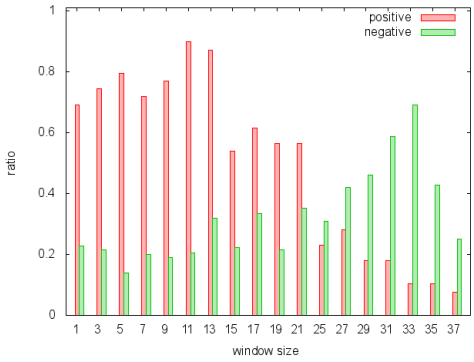

図 A.24: 対象物 3　顔サイズ 16 × 16　中心の重み 0.7　近傍の重み 0.5

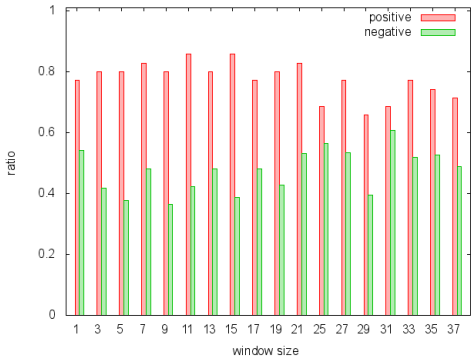

図 A.25: 対象物 1　顔サイズ 32 × 32　中心の重み 0.5　近傍の重み 0.1

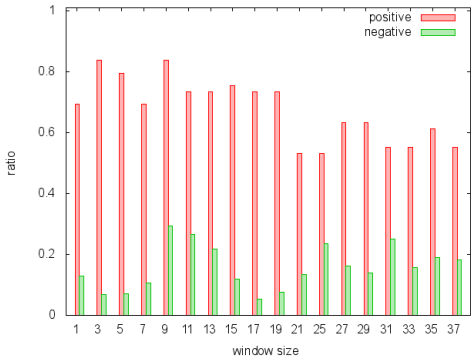

図 A.26: 対象物 2　顔サイズ 32 × 32　中心の重み 0.5　近傍の重み 0.1

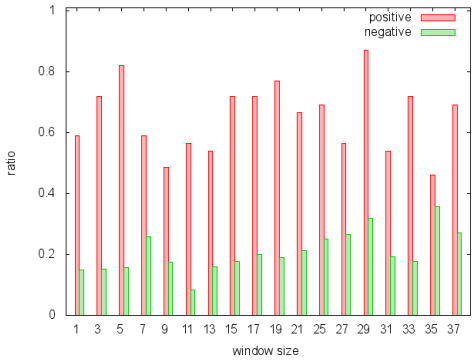

図 A.27: 対象物 3　顔サイズ 32 × 32　中心の重み 0.5　近傍の重み 0.1

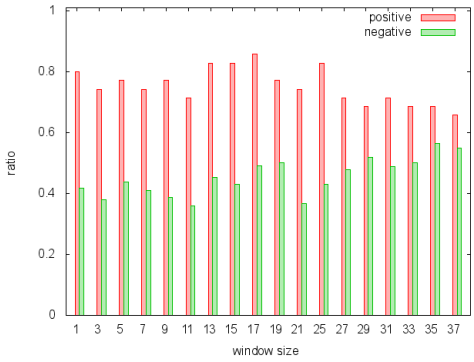

図 A.28: 対象物 1　顔サイズ 32 × 32　中心の重み 0.5　近傍の重み 0.3

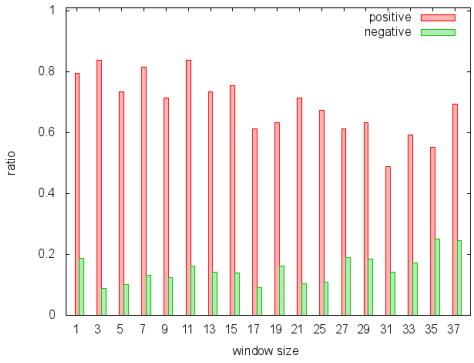

図 A.29: 対象物 2　顔サイズ 32 × 32　中心の重み 0.5　近傍の重み 0.3

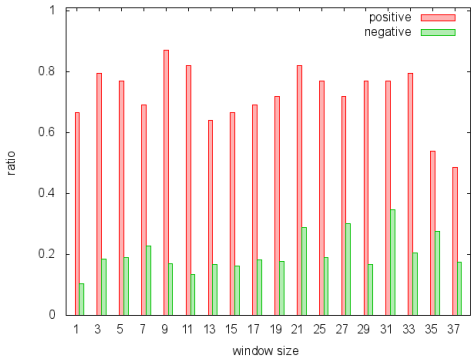

図 A.30: 対象物 3　顔サイズ 32 × 32　中心の重み 0.5　近傍の重み 0.3

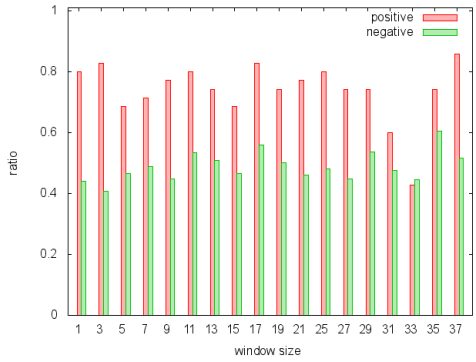

図 A.31: 対象物 1　顔サイズ 32 × 32　中心の重み 0.7　近傍の重み 0.3

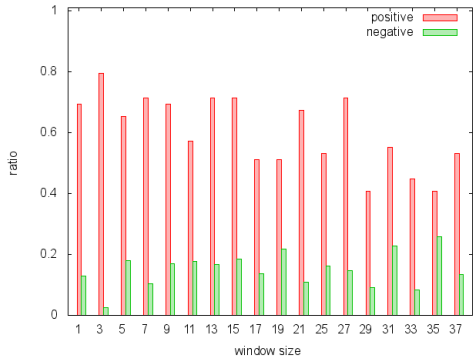

図 A.32: 対象物 2　顔サイズ 32 × 32　中心の重み 0.7　近傍の重み 0.3

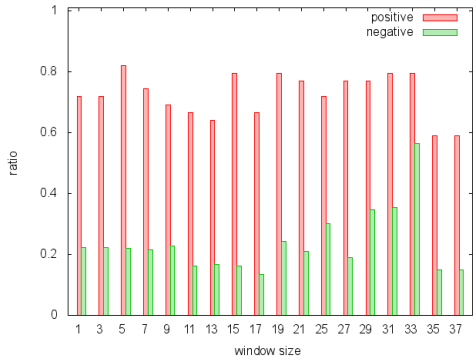

図 A.33: 対象物 3　顔サイズ 32 × 32　中心の重み 0.7　近傍の重み 0.3

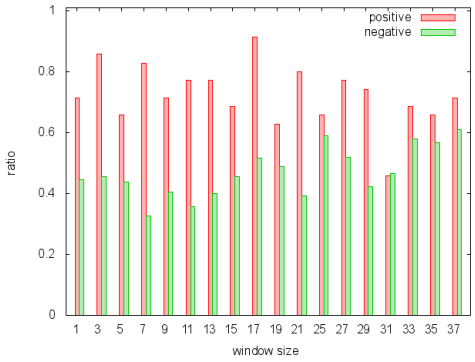

図 A.34: 対象物 1　顔サイズ 32 × 32　中心の重み 0.7　近傍の重み 0.5

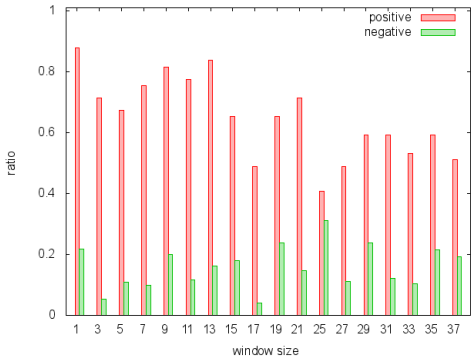

図 A.35: 対象物 2　顔サイズ 32 × 32　中心の重み 0.7　近傍の重み 0.5

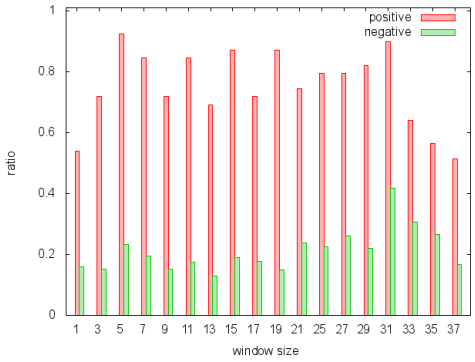

図 A.36: 対象物 3　顔サイズ 32 × 32　中心の重み 0.7　近傍の重み 0.5

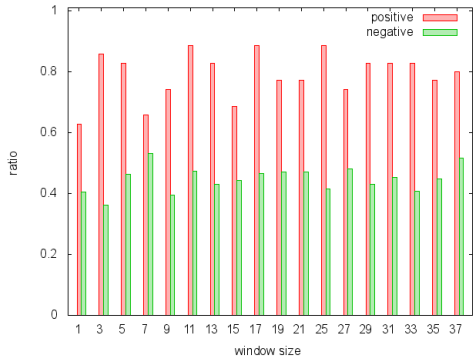

図 A.37: 対象物 1　顔サイズ 64 × 64　中心の重み 0.5　近傍の重み 0.1

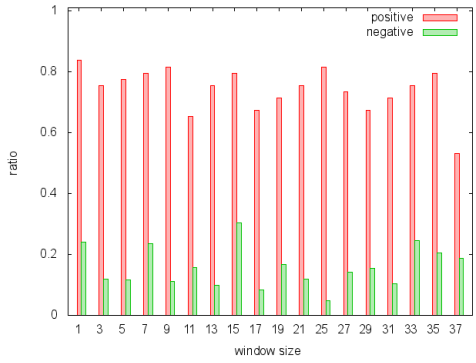

図 A.38: 対象物 2　顔サイズ 64 × 64　中心の重み 0.5　近傍の重み 0.1

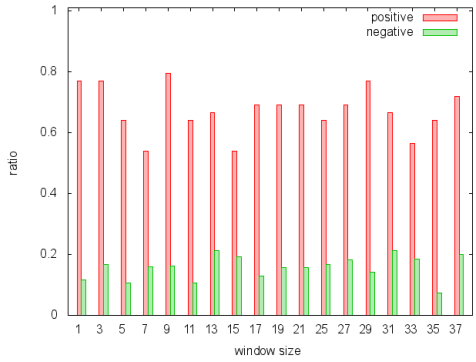

図 A.39: 対象物 3　顔サイズ 64 × 64　中心の重み 0.5　近傍の重み 0.1

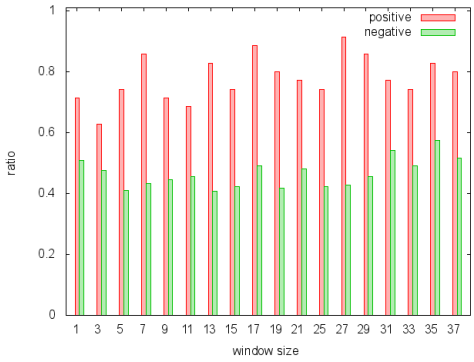

図 A.40: 対象物 1　顔サイズ 64 × 64　中心の重み 0.5　近傍の重み 0.3

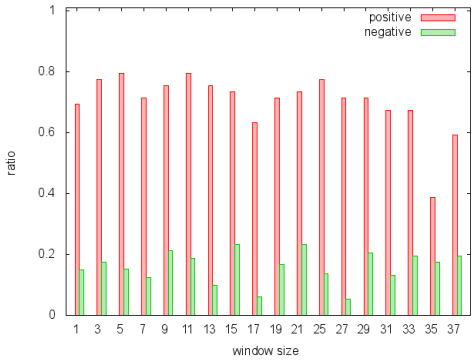

図 A.41: 対象物 2　顔サイズ 64 × 64　中心の重み 0.5　近傍の重み 0.3

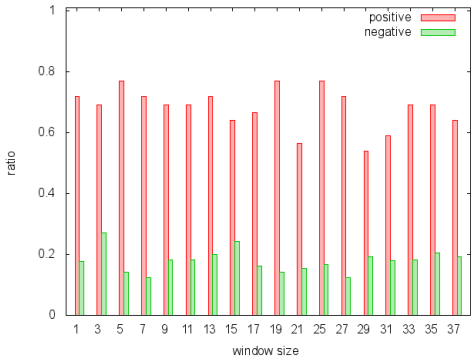

図 A.42: 対象物 3　顔サイズ 64 × 64　中心の重み 0.5　近傍の重み 0.3

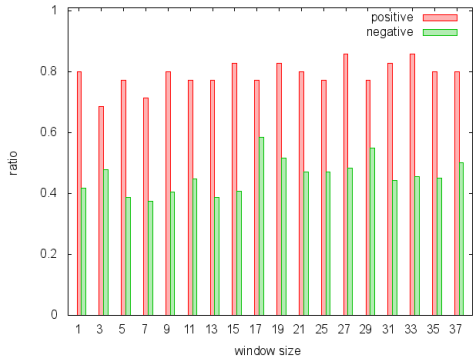

図 A.43: 対象物 1　顔サイズ 64 × 64　中心の重み 0.7　近傍の重み 0.3

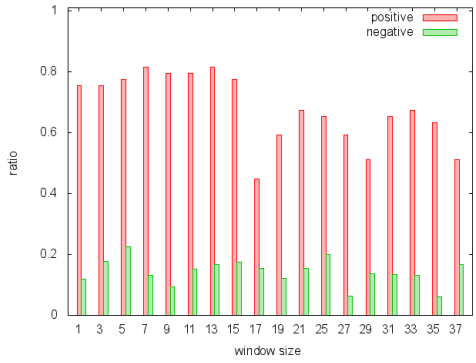

図 A.44: 対象物 2　顔サイズ 64 × 64　中心の重み 0.7　近傍の重み 0.3

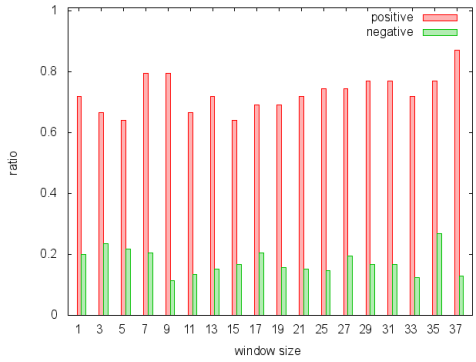

図 A.45: 対象物 3　顔サイズ 64 × 64　中心の重み 0.7　近傍の重み 0.3

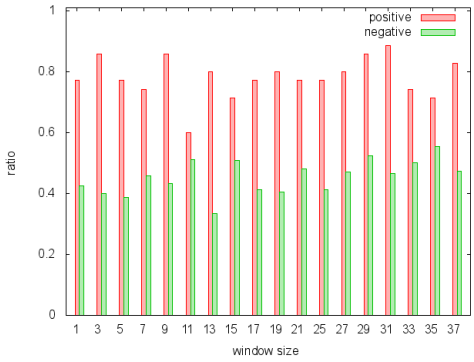

図 A.46: 対象物 1　顔サイズ 64 × 64　中心の重み 0.7　近傍の重み 0.5

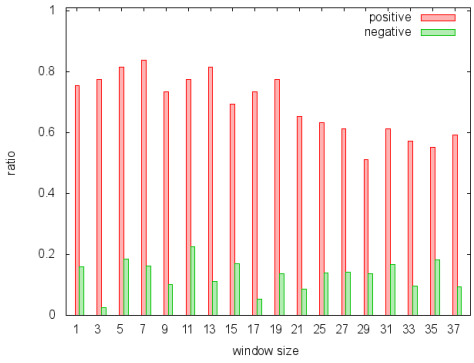

図 A.47: 対象物 2　顔サイズ 64 × 64　中心の重み 0.7　近傍の重み 0.5

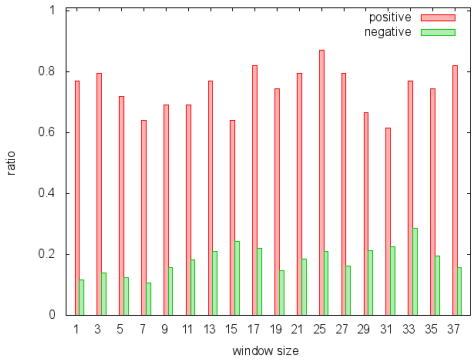

図 A.48: 対象物 3　顔サイズ 64 × 64　中心の重み 0.7　近傍の重み 0.5

付 録B　ソースコード

ソースコード B.1: cut.hpp

```
1  #ifndef CUT_HPP_
2  #define CUT_HPP_
3
4  #include <cstdint>
5  #include <limits>
6  #include <algorithm>
7  #include <boost/range.hpp>
8  #include <boost/range/irange.hpp>
9  #include <boost/range/algorithm.hpp>
10 #include <boost/foreach.hpp>
11 #define BOOST_THREAD_USE_LIB
12 #include <boost/thread.hpp>
13 #include <boost/thread/mutex.hpp>
14 #include <opencv2/opencv.hpp>
15 #include "cv_ex.hpp"
16
17 namespace cv_ex {
18
19 namespace {
20
21     inline void normalize(cv::Mat_<float>& img, float min_v, float max_v)
22     {
23         BOOST_FOREACH( auto& v, img ) {
24             v = ( v - min_v ) / ( max_v - min_v );
25         }
26     }
27
28     inline void normalize(cv::Mat_<float>& img)
29     {
30         float min_v = std::numeric_limits<float>::max();
31         float max_v = std::numeric_limits<float>::min();
32
33         BOOST_FOREACH( auto v, img ) {
34             min_v = std::min( min_v, v );
35             max_v = std::max( max_v, v );
36         }
37
38         normalize( img, min_v, max_v );
39     }
40
41     template <class F>
42     inline std::vector<cv::Mat_<float> > pyr_conv(const cv::Mat_<cv::Vec3f>& src, int level, F f)
43     {
44         std::vector<cv::Mat_<float> > dst;
45
46         const auto pyr = create_pyr( src, level );
47         BOOST_FOREACH( const auto& p, pyr ) {
48             dst.push_back( f( p ) );
49         }
50
51         return dst;
52     }
53
54     template <class F>
55     inline cv::Mat_<float> create_map(const cv::Mat_<cv::Vec3f>& src, int level, F f)
56     {
57         const auto pyr = pyr_conv( src, level, f );
58
59         cv::Mat_<float> dst( pyr.back().size(), 0.0f );
60         BOOST_FOREACH( int i, boost::irange( static_cast<int>( pyr.size() ) - 1, 0, -1 ) ) {
```

```
61              auto t = cv_ex::resize( pyr[i], pyr[i − 1].size() );
62              auto d = cv_ex::resize( dst, pyr[i − 1].size() );
63
64              BOOST_FOREACH( int y, boost::irange( 0, t.size().height ) ) {
65                  BOOST_FOREACH( int x, boost::irange( 0, t.size().width ) ) {
66                      d( y, x ) = d( y, x ) + std::fabs( pyr[i − 1]( y, x ) − t( y, x ) );
67                  }
68              }
69
70              dst = d;
71          }
72
73          return dst;
74      }
75
76      template <class F>
77      inline cv::Mat_<float> add_map(const cv::Mat_<cv::Vec3f>& src, int level, F f)
78      {
79          const auto pyr = pyr_conv( src, level, f );
80
81          cv::Mat_<float> dst( pyr.back().size(), 0.0f );
82          BOOST_FOREACH( int i, boost::irange( static_cast<int>( pyr.size() ) − 1, 0, −1 ) ) {
83              auto t = cv_ex::resize( pyr[i], pyr[i − 1].size() );
84              auto d = cv_ex::resize( dst, pyr[i − 1].size() );
85
86              BOOST_FOREACH( int y, boost::irange( 0, t.size().height ) ) {
87                  BOOST_FOREACH( int x, boost::irange( 0, t.size().width ) ) {
88                      d( y, x ) = d( y, x ) + t( y, x );
89                  }
90              }
91
92              dst = d;
93          }
94
95          return dst;
96      }
97
98      inline cv::Mat_<float> lumi_map(const cv::Mat_<cv::Vec3f>& src, int level)
99      {
100         const auto to_gray = [](const cv::Mat_<cv::Vec3f>& src) −> cv::Mat_<float> {
101             cv::Mat_<float> dst( src.size(), 0.0f );
102
103             auto src_itr = src.begin();
104             auto dst_itr = dst.begin();
105             for( ; src_itr != src.end(); ++src_itr, ++dst_itr ) {
106                 BOOST_FOREACH( int c, boost::irange( 0, 3 ) ) {
107                     ∗dst_itr += (∗src_itr)[c];
108                 }
109                 ∗dst_itr /= 3.0f;
110             }
111
112             return dst;
113         };
114
115         return create_map( src, level, to_gray );
116     }
117
118     template <class F>
119     inline cv::Mat_<float> color_(const cv::Mat_<cv::Vec3f>& src, F f)
120     {
121         cv::Mat_<float> dst( src.size(), 0.0f );
122
123         auto src_itr = src.begin();
124         auto dst_itr = dst.begin();
125         for( ; src_itr != src.end(); ++src_itr, ++dst_itr ) {
126             const float v = f( ∗src_itr );
127             ∗dst_itr = v < 0.0f ? 0.0f : v;
128         }
129
130         return dst;
131     }
132
133     inline std::vector<cv::Mat_<float> > color_map(const cv::Mat_<cv::Vec3f>& src, int level)
134     {
```

```
135         std::vector<cv::Mat_<float> > dst;
136
137         typedef std::function<cv::Mat_<float> (const cv::Mat_<cv::Vec3f>&)> function_t;
138         const std::vector<function_t> func = {
139             [](const cv::Mat_<cv::Vec3f>& src) -> cv::Mat_<float> {
140                 return color_( src, [](const cv::Vec3f& v) -> float{
141                         return v[2] - ( v[1] + v[0] ) / 2.0f; } );
142             },
143             [](const cv::Mat_<cv::Vec3f>& src) -> cv::Mat_<float> {
144                 return color_( src, [](const cv::Vec3f& v) -> float {
145                         return v[1] - ( v[2] + v[0] ) / 2.0f; } );
146             },
147             [](const cv::Mat_<cv::Vec3f>& src) -> cv::Mat_<float> {
148                 return color_( src, [](const cv::Vec3f& v) -> float {
149                         return v[0] - ( v[2] + v[1] ) / 2.0f; } );
150             },
151             [](const cv::Mat_<cv::Vec3f>& src) -> cv::Mat_<float> {
152                 return color_( src, [](const cv::Vec3f& v) -> float {
153                         return ( v[2] + v[1] ) / 2.0f - std::fabs( v[2] - v[1] ) / 2.0f - v[0]; } );
154             }
155         };
156
157         BOOST_FOREACH( auto f, func ) {
158             dst.push_back( add_map( src, level, f ) );
159         }
160
161         return dst;
162     }
163
164     inline std::vector<cv::Mat_<float> > make_maps(const cv::Mat_<cv::Vec3f>& src, int level)
165     {
166         std::vector<cv::Mat_<float> > dst;
167
168         dst.push_back( lumi_map( src, level ) );
169         const auto colors = color_map( src, level );
170         BOOST_FOREACH( const auto& c, colors ) {
171             dst.push_back( c );
172         }
173
174         return dst;
175     }
176
177     std::pair<float, float> image_min_max(const cv::Mat_<float>& src)
178     {
179         std::pair<float, float> dst( std::numeric_limits<float>::max(), std::numeric_limits<float>::
                min() );
180
181         for( auto itr = src.begin(); itr != src.end(); ++itr ) {
182             dst.first = std::min( dst.first, *itr );
183             dst.second = std::max( dst.second, *itr );
184         }
185
186         return dst;
187     }
188
189     double image_loc_max_avg(const cv::Mat_<float>& src)
190     {
191         cv::Mat_<double> dx1( src.size() ), dy1( src.size() );
192         cv::Mat_<double> dx2( src.size() ), dy2( src.size() );
193
194         double max_v = std::numeric_limits<double>::min();
195         cv::Point max_pos;
196         std::vector<double> loc_max;
197
198         for( int j = 0; j < src.size().height; ++j ) {
199             const int my = j == 0 ? j : j - 1;
200             const int py = j == src.size().height - 1 ? j : j + 1;
201
202             for( int i = 0; i < src.size().width; ++i ) {
203                 const int mx = i == 0 ? i : i - 1;
204                 const int px = i == src.size().width - 1 ? i : i + 1;
205
206                 if( max_v < src( j, i ) ) {
```

```
207                     max_v = src( j, i );
208                     max_pos = cv::Point( i, j );
209                 }
210
211                 dx1( j, i ) = src( j, px ) - src( j, mx );
212                 dy1( j, i ) = src( py, i ) - src( my, i );
213             }
214         }
215
216         for( int y = 0; y < src.size().height; ++y ) {
217             const int my = y == 0 ? y : y - 1;
218             const int py = y == src.size().height - 1 ? y : y + 1;
219             for( int x = 0; x < src.size().width; ++x ) {
220                 const int mx = x == 0 ? x : x - 1;
221                 const int px = x == src.size().height - 1 ? x : x + 1;
222
223                 dx2( y, x ) = dx1( y, px ) - dx1( y, mx );
224                 dy2( y, x ) = dy1( py, x ) - dy1( my, x );
225             }
226         }
227
228         for( int y = 0; y < src.size().height; ++y ) {
229             for( int x = 0; x < src.size().width; ++x ) {
230                 if( x == max_pos.x && y == max_pos.y ) {
231                     continue;
232                 }
233
234                 if( std::fabs( dx1( y, x ) ) < 0.01 && std::fabs( dy1( y, x ) ) < 0.01 ) {
235                     if( dx2( y, x ) < 0 && dy2( y, x ) < 0 ) {
236                         loc_max.push_back( src( y, x ) );
237                     }
238                 }
239             }
240         }
241
242         double dst = 0;
243         for( std::size_t i = 0; i < loc_max.size(); ++i ) {
244             dst += loc_max[i];
245         }
246         dst /= loc_max.size();
247
248         return dst;
249     }
250
251     inline std::uint64_t equal_count(const cv::Mat_<float>& x, const cv::Mat_<float>& y,
252                                      float th = std::numeric_limits<float>::epsilon())
253     {
254         std::uint64_t cnt = 0;
255
256         auto x_itr = x.begin();
257         auto y_itr = y.begin();
258         for( ; x_itr != x.end(); ++x_itr, ++y_itr ) {
259             if( std::fabs( *x_itr - *y_itr ) < th ) {
260                 ++cnt;
261             }
262         }
263
264         return cnt;
265     }
266
267     template <class F>
268     inline void for_each(cv::Mat_<float>& lhs, const cv::Mat_<float>& rhs, F f)
269     {
270         auto l_itr = lhs.begin();
271         auto r_itr = rhs.begin();
272         for( ; l_itr != lhs.end(); ++l_itr, ++r_itr ) {
273             *l_itr = f( *l_itr, *r_itr );
274         }
275     }
276
277     cv::Mat_<cv::Vec3b> mask_image(const cv::Mat_<cv::Vec3b>& src, const cv::Mat_<unsigned
          char>& mask,
278                                    cv::Vec3b ignore = cv::Vec3b( 255, 0, 255 ))
```

```
279     {
280         cv::Mat_<cv::Vec3b> dst( src.size() );
281         BOOST_FOREACH( int y, boost::irange( 0, src.size().height ) ) {
282             BOOST_FOREACH( int x, boost::irange( 0, src.size().width ) ) {
283                 dst( y, x ) = mask( y, x ) == cv::GC_BGD || mask( y, x ) == cv::GC_PR_BGD ?
284                     ignore : src( y, x );
285             }
286         }
287
288         return dst;
289     }
290
291     cv::Mat_<cv::Vec3b> draw_mask(const cv::Mat_<cv::Vec3b>& src, const cv::Mat_<unsigned char
          >& mask)
292     {
293         cv::Mat_<cv::Vec3b> dst = src.clone();
294
295         for( int y = 0; y < src.size().height; ++y ) {
296             for( int x = 0; x < src.size().width; ++x ) {
297                 if( mask( y, x ) == cv::GC_BGD ) {
298                     dst( y, x ) = cv::Vec3b( 0, 0, 255 );
299                 }
300                 else if( mask( y, x ) == cv::GC_FGD ) {
301                     dst( y, x ) = cv::Vec3b( 255, 0, 0 );
302                 }
303                 else if( mask( y, x ) == cv::GC_PR_FGD ) {
304                     dst( y, x ) = cv::Vec3b( 128, 128, 0 );
305                 }
306             }
307         }
308
309         return dst;
310     }
311
312     void graph_cut(const cv::Mat_<cv::Vec3b>& src, const cv::Mat_<float>& i,
313                    double fgd_comp, double pr_fgd_comp, double bgd_comp, int cnt,
314                    std::vector<cv::Mat_<cv::Vec3b> >& dst, boost::mutex& mutex,
315                    cv::Vec3b ignore)
316     {
317         cv::Mat_<unsigned char> mask( src.size(), cv::GC_PR_BGD );
318         bool err = true;
319
320         BOOST_FOREACH( int y, boost::irange( 0, i.size().height ) ) {
321             BOOST_FOREACH( int x, boost::irange( 0, i.size().width ) ) {
322                 if( i( y, x ) >= fgd_comp ) {
323                     mask( y, x ) = cv::GC_FGD;
324                     err = false;
325                 }
326                 else if( i( y, x ) >= pr_fgd_comp ) {
327                     mask( y, x ) = cv::GC_PR_FGD;
328                     err = false;
329                 }
330                 else if( i( y, x ) <= bgd_comp ) {
331                     mask( y, x ) = cv::GC_BGD;
332                 }
333             }
334         }
335         if( err ) {
336             return;
337         }
338
339         cv::Mat bgd, fgd;
340         cv::grabCut( src, mask, cv::Rect(), bgd, fgd, cnt, cv::GC_INIT_WITH_MASK );
341
342         boost::mutex::scoped_lock lock( mutex );
343         dst.push_back( mask_image( src, mask, ignore ) );
344     }
345
346 } // namespace
347
348     inline std::vector<cv::Mat_<cv::Vec3b> > cut(
349         const cv::Mat_<cv::Vec3b>& src, int level,
350         double fgd_comp, double pr_fgd_comp, double bgd_comp,
```

```
351         double ec_color, double ec_lumi, int cnt, cv::Vec3b ignore = cv::Vec3b( 255, 0, 255 ))
352     {
353         const cv::Mat_<cv::Vec3f> img = cv_ex::byte_to_float( src );
354 
355         auto maps = make_maps( img, level );
356         BOOST_FOREACH( auto& i, maps ) {
357             const auto minmax = image_min_max( i );
358             normalize( i, minmax.first, minmax.second );
359             const double loc = image_loc_max_avg( i );
360             const double diff = std::pow( minmax.second - loc, 2 );
361             BOOST_FOREACH( auto& n, i ) {
362                 n *= diff;
363             }
364             normalize( i );
365         }
366 
367         for( auto i = maps.begin() + 1; i < maps.begin() + ( maps.size() - 1 ); ++i ) {
368             for( auto j = i + 1; j < maps.end(); ) {
369                 const double ratio = equal_count( *i, *j ) / static_cast<double>( src.size().area() );
370                 if( ratio >= ec_color ) {
371                     for_each( *i, *j, [](float lhs, float rhs) -> float { return lhs + rhs; } );
372                     j = maps.erase( j );
373                 }
374                 else {
375                     ++j;
376                 }
377             }
378         }
379         for( auto i = maps.begin() + 1; i < maps.end(); ++i ) {
380             const double ratio =
381                 equal_count( *i, maps[0], 0.05f ) / static_cast<double>( src.size().area() );
382 
383             if( ratio >= ec_lumi ) {
384                 for_each( *i, maps[0], [](float lhs, float rhs) -> float { return lhs + rhs; } );
385             }
386             normalize( *i );
387         }
388         maps.erase( maps.begin() );
389 
390         std::vector<cv::Mat_<cv::Vec3b> > dst;
391         boost::mutex mutex;
392 
393 #if defined( CUT_NO_THREAD )
394         BOOST_FOREACH( auto& i, maps ) {
395             graph_cut( src, i, fgd_comp, pr_fgd_comp, bgd_comp, cnt, dst, mutex, ignore );
396         }
397 
398 #else
399         std::vector<boost::thread> th;
400 
401         BOOST_FOREACH( auto& i, maps ) {
402             th.push_back( boost::thread( [
403                                 &src, &i, fgd_comp, pr_fgd_comp, bgd_comp, cnt, &dst, &mutex, ignore
                                        ]() {
404                             graph_cut( src, i, fgd_comp, pr_fgd_comp, bgd_comp, cnt, dst, mutex, ignore
                                    );
405                         } ) );
406         }
407         BOOST_FOREACH( auto& t, th ) {
408             t.join();
409         }
410 #endif
411 
412         return dst;
413     }
414 
415 } // namespace cv_ex
416 
417 #endif // CUT_HPP_
```

ソースコード B.2: adjustment.hpp

```
1  #ifndef NCLR_CV_EX_ADJUSTMENT_HPP_
2  #define NCLR_CV_EX_ADJUSTMENT_HPP_
3
4  #include <opencv2/opencv.hpp>
5  #include <vector>
6  #include <utility>
7
8  namespace cv_ex
9  {
10
11 namespace detail
12 {
13     std::vector<std::vector<int> > histogram(const cv::Mat_<cv::Vec3b>& src)
14     {
15         std::vector<std::vector<int> > result;
16
17         for( int c = 0; c < src.channels(); ++c ) {
18             std::vector<int> v( 256, 0 );
19             result.push_back( v );
20         }
21         for( int y = 0; y < src.size().height; ++y ) {
22             for( int x = 0; x < src.size().width; ++x ) {
23                 for( int c = 0; c < src.channels(); ++c ) {
24                     ++result[c][src( y, x )[c]];
25                 }
26             }
27         }
28
29         return result;
30     }
31
32     std::pair<int, int> clipping(const std::vector<int>& hist, int ignore_black, int ignore_white)
33     {
34         assert( ignore_black >= 0 );
35         assert( ignore_white >= 0 );
36
37         int cnt = 0;
38
39         int l;
40         for( l = 0; l < 256; ++l ) {
41             cnt += hist[l];
42             if( cnt > ignore_black ) {
43                 break;
44             }
45         }
46
47         cnt = 0;
48
49         int r;
50         for( r = 255; r > l; --r ) {
51             cnt += hist[r];
52             if( cnt > ignore_white ) {
53                 break;
54             }
55         }
56         if( r <= l ) {
57             r = l + 1;
58         }
59
60         return std::make_pair( static_cast<int>( l ), static_cast<int>( r ) );
61     }
62
63     template <class T>
64     cv::Mat_<T> histogram_extension(const cv::Mat_<T>& src, const std::vector<std::pair<int, int
           > >& ranges)
65     {
66         assert( static_cast<int>( ranges.size() ) == src.channels() );
67
68         cv::Mat_<T> dst( src.size(), 0 );
69
70         for( int y = 0; y < src.size().height; ++y ) {
71             for( int x = 0; x < src.size().width; ++x ) {
```

```
72                      for( int c = 0; c < src.channels(); ++c ) {
73                          dst( y, x )[c] = cv::saturate_cast<unsigned char>(
74                              ( ( src( y, x )[c] − ranges[c].first ) ∗ 255 ) / ( ranges[c].second − ranges[c].
                                    first ) );
75                      }
76                  }
77              }
78
79              return dst;
80          }
81
82      } // namesapce detail
83
84          template <class T>
85          cv::Mat_<T> color_adjustment(const cv::Mat_<T>& src, double clip_black, double clip_white)
86          {
87              assert( clip_black >= 0.0 && clip_black <= 1.0 );
88              assert( clip_white >= 0.0 && clip_white <= 1.0 );
89
90              const int ignore_black = static_cast<int>( src.size().area() ∗ clip_black );
91              const int ignore_white = static_cast<int>( src.size().area() ∗ clip_white );
92
93              const auto hists = detail::histogram( src );
94
95              std::vector<std::pair<int, int> > ranges;
96              for( int c = 0; c < src.channels(); ++c ) {
97                  ranges.push_back( detail::clipping( hists[c], ignore_black, ignore_white ) );
98              }
99
100             return detail::histogram_extension( src, ranges );
101         }
102
103     } // namespace cv_ex
104
105     #endif // NCLR_CV_EX_ADJUSTMENT_HPP_
```

ソースコード B.3: cascade_classifier.hpp

```
1  #ifndef CASCADE_CLASSIFIER_HPP_
2  #define CASCADE_CLASSIFIER_HPP_
3
4  #include <boost/range/irange.hpp>
5  #include <opencv2/opencv.hpp>
6  #include "cv_ex.hpp"
7
8  inline std::vector<cv::Mat_<cv::Vec3b>> centering_faces(
9          const cv::Mat_<cv::Vec3b>& src, cv::CascadeClassifier& cc)
10 {
11     const auto rcs = cv_ex::detect( src, cc );
12
13     std::vector<cv::Mat_<cv::Vec3b>> dst;
14     for( const auto& rc : rcs ) {
15         const cv::Point2i cent = cv_ex::center( rc );
16         const int dx = cent.x - ( src.size().width / 2 );
17
18         cv::Mat_<cv::Vec3b> tmp( src.size().height, src.size().width, cv::Vec3b( 255, 0, 255 ) );
19         for( int y : boost::irange( 0, src.size().height ) ) {
20             for( int x : boost::irange( 0, src.size().width ) ) {
21                 const int px = x + dx;
22                 if( px < 0 ) {
23                     tmp( y, x ) = src( y, 0 );
24                 }
25                 else if( px >= src.size().width ) {
26                     tmp( y, x ) = src( y, src.size().width - 1 );
27                 }
28                 else {
29                     tmp( y, x ) = src( y, px );
30                 }
31             }
32         }
33
34         const int width = src.size().width - std::abs( dx );
35         cv::Mat_<cv::Vec3b> img( src.size().height, width );
36
37         const int offset = dx >= 0 ?
38             std::abs( src.size().width / 2 - width / 2 ) :
39             -dx -::abs( src.size().width / 2 - width / 2 );
40
41         for( int y : boost::irange( 0, img.size().height ) ) {
42             for( int x : boost::irange( 0, img.size().width ) ) {
43                 const int ox = x + offset;
44                 if( ox >= 0 && ox < src.size().width ) {
45                     img( y, x ) = tmp( y, ox );
46                 }
47             }
48         }
49
50         dst.push_back( img );
51     }
52
53     return dst;
54 }
55
56 #endif // CASCADE_CLASSIFIER_HPP_
```

ソースコード B.4: classifier.hpp

```
1  #ifndef CLASSIFIER_HPP_
2  #define CLASSIFIER_HPP_
3
4  #include <iostream>
5  #include <fstream>
6  #include <stdexcept>
7  #include <cmath>
8  #include <tuple>
9  #include <algorithm>
10 #include <numeric>
11 #define BOOST_FILESYSTEM_VERSION 3
12 #include <boost/filesystem.hpp>
13 #include <boost/iterator/iterator_facade.hpp>
14 #include <boost/range/iterator_range.hpp>
15 #include <boost/range.hpp>
16 #include <boost/range/algorithm.hpp>
17 #include <boost/range/numeric.hpp>
18 #include <nclr/command_line.hpp>
19 #include <nclr/printf.hpp>
20 #include <opencv2/opencv.hpp>
21 #include "cv_ex.hpp"
22 #include "adjustment.hpp"
23 #include "cascade_classifier.hpp"
24 #include "cut.hpp"
25 #include "som.hpp"
26
27 class point_iterator :
28     public boost::iterator_facade<point_iterator, cv::Point2i, boost::forward_traversal_tag, cv::Point2i
          >
29 {
30     cv::Point2i pt_;
31     const cv::Size sz_;
32
33 public:
34     point_iterator(cv::Point pt, cv::Size sz) :
35         pt_( pt ), sz_( sz )
36     { }
37
38 private:
39     friend class boost::iterator_core_access;
40
41     void increment()
42     {
43         if( pt_.y < sz_.height ) {
44             ++pt_.x;
45             if( pt_.x >= sz_.width ) {
46                 ++pt_.y;
47                 pt_.x = pt_.y >= sz_.height ? sz_.width : 0;
48             }
49         }
50     }
51
52     bool equal(const point_iterator& other) const
53     {
54         return pt_ == other.pt_ && sz_ == other.sz_;
55     }
56
57     cv::Point2i dereference() const
58     {
59         return pt_;
60     }
61 };
62
63 struct point_iterator_range :
64     public boost::iterator_range<point_iterator>
65 {
66     point_iterator_range(cv::Point2i pt, cv::Size sz) :
67         boost::iterator_range<point_iterator>(
68                 point_iterator{ pt, sz },
69                 point_iterator{ { sz.width, sz.height }, sz } )
70     { }
71 };
72
```

```
73  inline point_iterator_range point_range(cv::Point pt, cv::Size sz)
74  {
75      return { pt, sz };
76  }
77
78  inline point_iterator_range point_range(cv::Size sz)
79  {
80      return { cv::Point2i{ 0, 0 }, sz };
81  }
82
83  inline double distance(const std::vector<double>& lhs, const std::vector<double>& rhs)
84  {
85      double d = 0;
86
87      auto lhs_itr = lhs.begin();
88      auto rhs_itr = rhs.begin();
89      for( ; lhs_itr != lhs.end(); ++lhs_itr, ++rhs_itr ) {
90          const double elem = *lhs_itr - *rhs_itr;
91          d += elem * elem;
92      }
93
94      return std::sqrt( d );
95  }
96
97  inline double cross_sum(const std::vector<double>& lhs, const std::vector<double>& rhs)
98  {
99      double v = 0;
100     auto l_itr = lhs.begin();
101     auto r_itr = rhs.begin();
102     for( ; l_itr != lhs.end(); ++l_itr, ++r_itr ) {
103         v += (*l_itr) * (*r_itr);
104     }
105
106     return v;
107 }
108
109 inline double cross_correlation(const std::vector<double>& lhs, const std::vector<double>& rhs)
110 {
111     const double MN = lhs.size() * rhs.size();
112     const double lr = cross_sum( lhs, rhs );
113     const double l = boost::accumulate( lhs, 0 );
114     const double r = boost::accumulate( rhs, 0 );
115     const double l2 = cross_sum( lhs, lhs );
116     const double r2 = cross_sum( rhs, rhs );
117
118     return ( -( ( MN * lr - l * r ) / std::sqrt( ( MN * l2 - l * l ) * ( MN * r2 - r * r ) ) ) + 1.0 ) /
            2.0;
119 }
120
121 template <class F>
122 inline void for_each_file(const std::string& dir, F f)
123 {
124     boost::filesystem3::directory_iterator itr{ dir };
125     boost::filesystem3::directory_iterator end;
126
127     for( ; itr != end; itr = std::next( itr ) ) {
128         f( itr->path() );
129     }
130 }
131
132 inline cv::Mat_<unsigned char> create_mask(
133         const cv::Mat_<cv::Vec3b>& src, cv::Vec3b ignore = cv::Vec3b( 255, 0, 255 ))
134 {
135     cv::Mat_<unsigned char> mask( src.size(), 1 );
136
137     for( cv::Point2i pt : point_range( src.size() ) ) {
138         mask( pt.y, pt.x ) = src( pt.y, pt.x ) == ignore ? 0 : 1;
139     }
140
141     return mask;
142 }
143
144 inline double rate(const std::vector<int>& src)
```

```
145 {
146     double ratio = 0;
147     for( int v : src ) {
148         ratio += v;
149     }
150     ratio /= src.size();
151 
152     return ratio;
153 }
154 
155 inline bool within(cv::Point pt, cv::Size sz)
156 {
157     return pt.x >= 0 && pt.x < sz.width && pt.y >= 0 && pt.y < sz.height;
158 }
159 
160 template <class Generator>
161 inline std::vector<std::vector<double>> create_window_samples(
162         const cv::Mat_<cv::Vec3b>& src, const cv::Mat_<int>& mask,
163         cv::Size wnd_sz, int max_point, double ratio, Generator& gen)
164 {
165     const auto tmp = cv_ex::for_each<cv::Vec3f>( src,
166             [](cv::Vec3b p) -> cv::Vec3f { return { p[0] / 255.0f, p[1] / 255.0f, p[2] / 255.0f }; } );
167     cv::Mat_<cv::Vec3f> fimg;
168     cv::cvtColor( tmp, fimg, CV_BGR2Lab );
169 
170     //const auto fimg = cv_ex::for_each<cv::Vec3d>( src, [](cv::Vec3b p) -> cv::Vec3d {
171     // return { p[0] / 255.0, p[1] / 255.0, p[2] / 255.0 };
172     //} );
173 
174     std::vector<cv::Point2i> pts;
175     for( cv::Point2i pt : point_range( src.size() ) ) {
176         std::vector<int> mask_wnd( wnd_sz.area(), 0 );
177         for( cv::Point2i wnd_pt : point_range( wnd_sz ) ) {
178             const auto a = pt.x + ( wnd_sz.width / 2 ) + wnd_pt.x;
179             const auto b = pt.y + ( wnd_sz.height / 2 ) + wnd_pt.y;
180             if( within( { a, b }, src.size() ) ) {
181                 mask_wnd[wnd_pt.x + wnd_pt.y * wnd_sz.width] = mask( b, a ) != 0 ? 1 : 0;
182             }
183         }
184         if( rate( mask_wnd ) >= ratio ) {
185             pts.push_back( pt );
186         }
187     }
188 
189     boost::uniform_int<std::size_t> dist( 0, pts.size() - 1 );
190     boost::variate_generator<Generator&, decltype( dist )> random = { gen, dist };
191     std::random_shuffle( pts.begin(), pts.end(), random );
192 
193     std::vector<std::vector<double>> dst;
194     for( int cnt : boost::irange( 0,
195             max_point < static_cast<int>( pts.size() ) ? max_point : static_cast<int>( pts.size()
                ) ) ) {
196         const cv::Point2i pt = pts[cnt];
197 
198         std::vector<double> wnd( wnd_sz.area() * 3, 0.0 );
199         for( cv::Point2i p : point_range( wnd_sz ) ) {
200             const auto a = pt.x + ( wnd_sz.width / 2 ) + p.x;
201             const auto b = pt.y + ( wnd_sz.height / 2 ) + p.y;
202             if( within( { a, b }, src.size() ) ) {
203                 const auto i = p.x + p.y * wnd_sz.width;
204                 for( int c : boost::irange( 0, 3 ) ) {
205                     wnd[i * 3 + c] = fimg( b, a )[c];
206                 }
207             }
208         }
209 
210         dst.push_back( wnd );
211     }
212 
213     return dst;
214 }
215 
216 inline std::vector<std::tuple<cv::Mat_<cv::Vec3b>, cv::Mat_<unsigned char>>> cut(
```

```
217         const cv::Mat_<cv::Vec3b>& src, cv::CascadeClassifier& cc)
218 {
219     std::vector<std::tuple<cv::Mat_<cv::Vec3b>, cv::Mat_<unsigned char>>> result;
220
221     const auto imgs = cv_ex::cut( cv_ex::color_adjustment( src, 0.05, 0.05 ),
222             8, 0.3, 0.22, 0.05, 0.3, 0.1, 1, { 255, 0, 255 } );
223
224     for( const auto& i : imgs ) {
225         if( !cv_ex::detect( i, cc ).empty() ) {
226             result.push_back( std::make_tuple( src, create_mask( i ) ) );
227         }
228     }
229
230     return result;
231 }
232
233 inline std::vector<std::tuple<cv::Mat_<cv::Vec3b>, cv::Mat_<unsigned char>>> adjust_faces(
234         const cv::Mat_<cv::Vec3b>& src, const cv::Mat_<unsigned char>& mask,
235         cv::CascadeClassifier& cc, cv::Size face_sz)
236 {
237     std::vector<std::tuple<cv::Mat_<cv::Vec3b>, cv::Mat_<unsigned char>>> dst;
238
239     const auto rcs = cv_ex::detect( src, cc );
240     for( const auto& rc : rcs ) {
241         const double w_ratio = face_sz.width / static_cast<double>( rc.width );
242         const double h_ratio = face_sz.height / static_cast<double>( rc.height );
243         const cv::Size sz = {
244             static_cast<int>( w_ratio * src.size().width ),
245             static_cast<int>( h_ratio * src.size().height ) };
246         dst.push_back( std::make_tuple(
247                 cv_ex::resize( src, sz, cv::INTER_NEAREST ),
248                 cv_ex::resize( mask, sz, cv::INTER_NEAREST ) ) );
249     }
250
251     return dst;
252 }
253
254 inline std::vector<double> counting(const som& m, const std::vector<std::vector<double>>&
      samples)
255 {
256     std::vector<double> result( m.width() * m.height(), 0 );
257
258     for( const auto& s : samples ) {
259         double min_d = std::numeric_limits<double>::max();
260         cv::Point2i min_pt{ 0, 0 };
261         for( cv::Point2i pt : point_range( { m.width(), m.height() } ) ) {
262             const double d = distance( m( pt.x, pt.y ), s );
263             //const double d = cross_correlation( m( pt.x, pt.y ), s );
264             if( d < min_d ) {
265                 min_d = d;
266                 min_pt = pt;
267             }
268         }
269
270         result[min_pt.x + min_pt.y * m.width()] += 1.0;
271     }
272     for( double& v : result ) {
273         v /= samples.size();
274     }
275
276     return result;
277 }
278
279 template <class Generator>
280 inline std::vector<std::vector<double>> create_samples(
281         const cv::Mat_<cv::Vec3b>& src, cv::CascadeClassifier& cc,
282         cv::Size face_sz, cv::Size wnd_sz, int max_point, double valid_ratio, Generator& gen)
283 {
284     nclr::printf( std::clog, "cutting\r" );
285
286     const auto imgs = cut( src, cc );
287     if( imgs.empty() ) {
288         return {};
```

```
289     }
290
291     nclr::printf( std::clog, "add_samples\r" );
292
293     std::vector<std::vector<double>> samples;
294     for( const auto& i : imgs ) {
295         const auto v = adjust_faces( std::get<0>( i ), std::get<1>( i ), cc, face_sz );
296         for( const auto& j : v ) {
297             const std::vector<std::vector<double>> ss( create_window_samples(
298                     std::get<0>( j ), std::get<1>( j ), wnd_sz, max_point, valid_ratio, gen ) );
299             for( const auto& s : ss ) {
300                 samples.push_back( s );
301             }
302         }
303     }
304
305     std::random_shuffle( samples.begin(), samples.end() );
306     if( samples.size() > static_cast<std::size_t>( max_point ) ) {
307         samples.erase( samples.begin() + max_point, samples.end() );
308     }
309
310     return samples;
311 }
312
313 inline void save(const std::string& path, const som& m, const std::vector<double>& count)
314 {
315     std::ofstream ofs( path, std::ios::binary );
316
317     const int width = m.width();
318     const int height = m.height();
319     const int node_sz = m.data()[0].size();
320     ofs.write( (char *)( &width ), sizeof( int ) );
321     ofs.write( (char *)( &height ), sizeof( int ) );
322     ofs.write( (char *)( &node_sz ), sizeof( int ) );
323
324     for( const auto& v : m.data() ) {
325         ofs.write( (char *)( &v[0] ), sizeof( decltype( v[0] ) ) * v.size() );
326     }
327     ofs.write( (char *)( &count[0] ), sizeof( decltype( count[0] ) ) * count.size() );
328 }
329
330 inline std::tuple<som, std::vector<double>> load(const std::string& path)
331 {
332     std::ifstream ifs( path, std::ios::binary );
333
334     int width, height, node_sz;
335     ifs.read( (char *)( &width ), sizeof( int ) );
336     ifs.read( (char *)( &height ), sizeof( int ) );
337     ifs.read( (char *)( &node_sz ), sizeof( int ) );
338
339     som m = { make_som( width, height, node_sz ) };
340
341     for( const auto& v: m.data() ) {
342         ifs.read( (char *)( &v[0] ), sizeof( decltype( v[0] ) ) * v.size() );
343     }
344
345     std::vector<double> count( width * height, 0.0 );
346     ifs.read( (char *)( &count[0] ), sizeof( decltype( count[0] ) ) * count.size() );
347
348     return std::make_tuple( m, count );
349 }
350
351 #endif // CLASSIFIER_HPP_
```

ソースコード B.5: cut.hpp

```
1  #ifndef CUT_HPP_
2  #define CUT_HPP_
3
4  #include <cstdint>
5  #include <limits>
6  #include <algorithm>
7  #include <boost/range.hpp>
8  #include <boost/range/irange.hpp>
9  #include <boost/range/algorithm.hpp>
10 #include <boost/foreach.hpp>
11 #define BOOST_THREAD_USE_LIB
12 #include <boost/thread.hpp>
13 #include <boost/thread/mutex.hpp>
14 #include <opencv2/opencv.hpp>
15 #include "cv_ex.hpp"
16
17 namespace cv_ex {
18
19 namespace {
20
21     inline void normalize(cv::Mat_<float>& img, float min_v, float max_v)
22     {
23         BOOST_FOREACH( auto& v, img ) {
24             v = ( v - min_v ) / ( max_v - min_v );
25         }
26     }
27
28     inline void normalize(cv::Mat_<float>& img)
29     {
30         float min_v = std::numeric_limits<float>::max();
31         float max_v = std::numeric_limits<float>::min();
32
33         BOOST_FOREACH( auto v, img ) {
34             min_v = std::min( min_v, v );
35             max_v = std::max( max_v, v );
36         }
37
38         normalize( img, min_v, max_v );
39     }
40
41     template <class F>
42     inline std::vector<cv::Mat_<float> > pyr_conv(const cv::Mat_<cv::Vec3f>& src, int level, F f)
43     {
44         std::vector<cv::Mat_<float> > dst;
45
46         const auto pyr = create_pyr( src, level );
47         BOOST_FOREACH( const auto& p, pyr ) {
48             dst.push_back( f( p ) );
49         }
50
51         return dst;
52     }
53
54     template <class F>
55     inline cv::Mat_<float> create_map(const cv::Mat_<cv::Vec3f>& src, int level, F f)
56     {
57         const auto pyr = pyr_conv( src, level, f );
58
59         cv::Mat_<float> dst( pyr.back().size(), 0.0f );
60         BOOST_FOREACH( int i, boost::irange( static_cast<int>( pyr.size() ) - 1, 0, -1 ) ) {
61             auto t = cv_ex::resize( pyr[i], pyr[i - 1].size() );
62             auto d = cv_ex::resize( dst, pyr[i - 1].size() );
63
64             BOOST_FOREACH( int y, boost::irange( 0, t.size().height ) ) {
65                 BOOST_FOREACH( int x, boost::irange( 0, t.size().width ) ) {
66                     d( y, x ) = d( y, x ) + std::fabs( pyr[i - 1]( y, x ) - t( y, x ) );
67                 }
68             }
69
70             dst = d;
71         }
72
```

```
73          return dst;
74      }
75
76      template <class F>
77      inline cv::Mat_<float> add_map(const cv::Mat_<cv::Vec3f>& src, int level, F f)
78      {
79          const auto pyr = pyr_conv( src, level, f );
80
81          cv::Mat_<float> dst( pyr.back().size(), 0.0f );
82          BOOST_FOREACH( int i, boost::irange( static_cast<int>( pyr.size() ) - 1, 0, -1 ) ) {
83              auto t = cv_ex::resize( pyr[i], pyr[i - 1].size() );
84              auto d = cv_ex::resize( dst, pyr[i - 1].size() );
85
86              BOOST_FOREACH( int y, boost::irange( 0, t.size().height ) ) {
87                  BOOST_FOREACH( int x, boost::irange( 0, t.size().width ) ) {
88                      d( y, x ) = d( y, x ) + t( y, x );
89                  }
90              }
91
92              dst = d;
93          }
94
95          return dst;
96      }
97
98      inline cv::Mat_<float> lumi_map(const cv::Mat_<cv::Vec3f>& src, int level)
99      {
100         const auto to_gray = [](const cv::Mat_<cv::Vec3f>& src) -> cv::Mat_<float> {
101             cv::Mat_<float> dst( src.size(), 0.0f );
102
103             auto src_itr = src.begin();
104             auto dst_itr = dst.begin();
105             for( ; src_itr != src.end(); ++src_itr, ++dst_itr ) {
106                 BOOST_FOREACH( int c, boost::irange( 0, 3 ) ) {
107                     *dst_itr += (*src_itr)[c];
108                 }
109                 *dst_itr /= 3.0f;
110             }
111
112             return dst;
113         };
114
115         return create_map( src, level, to_gray );
116     }
117
118     template <class F>
119     inline cv::Mat_<float> color_(const cv::Mat_<cv::Vec3f>& src, F f)
120     {
121         cv::Mat_<float> dst( src.size(), 0.0f );
122
123         auto src_itr = src.begin();
124         auto dst_itr = dst.begin();
125         for( ; src_itr != src.end(); ++src_itr, ++dst_itr ) {
126             const float v = f( *src_itr );
127             *dst_itr = v < 0.0f ? 0.0f : v;
128         }
129
130         return dst;
131     }
132
133     inline std::vector<cv::Mat_<float> > color_map(const cv::Mat_<cv::Vec3f>& src, int level)
134     {
135         std::vector<cv::Mat_<float> > dst;
136
137         typedef std::function<cv::Mat_<float> (const cv::Mat_<cv::Vec3f>&)> function_t;
138         const std::vector<function_t> func = {
139             [](const cv::Mat_<cv::Vec3f>& src) -> cv::Mat_<float> {
140                 return color_( src, [](const cv::Vec3f& v) -> float{
141                         return v[2] - ( v[1] + v[0] ) / 2.0f; } );
142             },
143             [](const cv::Mat_<cv::Vec3f>& src) -> cv::Mat_<float> {
144                 return color_( src, [](const cv::Vec3f& v) -> float {
145                         return v[1] - ( v[2] + v[0] ) / 2.0f; } );
```

```
146                 },
147                 [](const cv::Mat_<cv::Vec3f>& src) -> cv::Mat_<float> {
148                     return color_( src, [](const cv::Vec3f& v) -> float {
149                             return v[0] - ( v[2] + v[1] ) / 2.0f; } );
150                 },
151                 [](const cv::Mat_<cv::Vec3f>& src) -> cv::Mat_<float> {
152                     return color_( src, [](const cv::Vec3f& v) -> float {
153                             return ( v[2] + v[1] ) / 2.0f - std::fabs( v[2] - v[1] ) / 2.0f - v[0]; } );
154                 }
155             };
156 
157             BOOST_FOREACH( auto f, func ) {
158                 dst.push_back( add_map( src, level, f ) );
159             }
160 
161             return dst;
162         }
163 
164         inline std::vector<cv::Mat_<float> > make_maps(const cv::Mat_<cv::Vec3f>& src, int level)
165         {
166             std::vector<cv::Mat_<float> > dst;
167 
168             dst.push_back( lumi_map( src, level ) );
169             const auto colors = color_map( src, level );
170             BOOST_FOREACH( const auto& c, colors ) {
171                 dst.push_back( c );
172             }
173 
174             return dst;
175         }
176 
177         std::pair<float, float> image_min_max(const cv::Mat_<float>& src)
178         {
179             std::pair<float, float> dst( std::numeric_limits<float>::max(), std::numeric_limits<float>::
                    min() );
180 
181             for( auto itr = src.begin(); itr != src.end(); ++itr ) {
182                 dst.first = std::min( dst.first, *itr );
183                 dst.second = std::max( dst.second, *itr );
184             }
185 
186             return dst;
187         }
188 
189         double image_loc_max_avg(const cv::Mat_<float>& src)
190         {
191             cv::Mat_<double> dx1( src.size() ), dy1( src.size() );
192             cv::Mat_<double> dx2( src.size() ), dy2( src.size() );
193 
194             double max_v = std::numeric_limits<double>::min();
195             cv::Point max_pos;
196             std::vector<double> loc_max;
197 
198             for( int j = 0; j < src.size().height; ++j ) {
199                 const int my = j == 0 ? j : j - 1;
200                 const int py = j == src.size().height - 1 ? j : j + 1;
201 
202                 for( int i = 0; i < src.size().width; ++i ) {
203                     const int mx = i == 0 ? i : i - 1;
204                     const int px = i == src.size().width - 1 ? i : i + 1;
205 
206                     if( max_v < src( j, i ) ) {
207                         max_v = src( j, i );
208                         max_pos = cv::Point( i, j );
209                     }
210 
211                     dx1( j, i ) = src( j, px ) - src( j, mx );
212                     dy1( j, i ) = src( py, i ) - src( my, i );
213                 }
214             }
215 
216             for( int y = 0; y < src.size().height; ++y ) {
217                 const int my = y == 0 ? y : y - 1;
218                 const int py = y == src.size().height - 1 ? y : y + 1;
```

```
219             for( int x = 0; x < src.size().width; ++x ) {
220                 const int mx = x == 0 ? x : x - 1;
221                 const int px = x == src.size().height - 1 ? x : x + 1;
222 
223                 dx2( y, x ) = dx1( y, px ) - dx1( y, mx );
224                 dy2( y, x ) = dy1( py, x ) - dy1( my, x );
225             }
226         }
227 
228         for( int y = 0; y < src.size().height; ++y ) {
229             for( int x = 0; x < src.size().width; ++x ) {
230                 if( x == max_pos.x && y == max_pos.y ) {
231                     continue;
232                 }
233 
234                 if( std::fabs( dx1( y, x ) ) < 0.01 && std::fabs( dy1( y, x ) ) < 0.01 ) {
235                     if( dx2( y, x ) < 0 && dy2( y, x ) < 0 ) {
236                         loc_max.push_back( src( y, x ) );
237                     }
238                 }
239             }
240         }
241 
242         double dst = 0;
243         for( std::size_t i = 0; i < loc_max.size(); ++i ) {
244             dst += loc_max[i];
245         }
246         dst /= loc_max.size();
247 
248         return dst;
249     }
250 
251     inline std::uint64_t equal_count(const cv::Mat_<float>& x, const cv::Mat_<float>& y,
252                                      float th = std::numeric_limits<float>::epsilon())
253     {
254         std::uint64_t cnt = 0;
255 
256         auto x_itr = x.begin();
257         auto y_itr = y.begin();
258         for( ; x_itr != x.end(); ++x_itr, ++y_itr ) {
259             if( std::fabs( *x_itr - *y_itr ) < th ) {
260                 ++cnt;
261             }
262         }
263 
264         return cnt;
265     }
266 
267     template <class F>
268     inline void for_each(cv::Mat_<float>& lhs, const cv::Mat_<float>& rhs, F f)
269     {
270         auto l_itr = lhs.begin();
271         auto r_itr = rhs.begin();
272         for( ; l_itr != lhs.end(); ++l_itr, ++r_itr ) {
273             *l_itr = f( *l_itr, *r_itr );
274         }
275     }
276 
277     cv::Mat_<cv::Vec3b> mask_image(const cv::Mat_<cv::Vec3b>& src, const cv::Mat_<unsigned
          char>& mask,
278                                    cv::Vec3b ignore = cv::Vec3b( 255, 0, 255 ))
279     {
280         cv::Mat_<cv::Vec3b> dst( src.size() );
281         BOOST_FOREACH( int y, boost::irange( 0, src.size().height ) ) {
282             BOOST_FOREACH( int x, boost::irange( 0, src.size().width ) ) {
283                 dst( y, x ) = mask( y, x ) == cv::GC_BGD || mask( y, x ) == cv::GC_PR_BGD ?
284                     ignore : src( y, x );
285             }
286         }
287 
288         return dst;
289     }
290 
```

```
291     cv::Mat_<cv::Vec3b> draw_mask(const cv::Mat_<cv::Vec3b>& src, const cv::Mat_<unsigned char
            >& mask)
292     {
293         cv::Mat_<cv::Vec3b> dst = src.clone();
294
295         for( int y = 0; y < src.size().height; ++y ) {
296             for( int x = 0; x < src.size().width; ++x ) {
297                 if( mask( y, x ) == cv::GC_BGD ) {
298                     dst( y, x ) = cv::Vec3b( 0, 0, 255 );
299                 }
300                 else if( mask( y, x ) == cv::GC_FGD ) {
301                     dst( y, x ) = cv::Vec3b( 255, 0, 0 );
302                 }
303                 else if( mask( y, x ) == cv::GC_PR_FGD ) {
304                     dst( y, x ) = cv::Vec3b( 128, 128, 0 );
305                 }
306             }
307         }
308
309         return dst;
310     }
311
312     void graph_cut(const cv::Mat_<cv::Vec3b>& src, const cv::Mat_<float>& i,
313                    double fgd_comp, double pr_fgd_comp, double bgd_comp, int cnt,
314                    std::vector<cv::Mat_<cv::Vec3b> >& dst, boost::mutex& mutex,
315                    cv::Vec3b ignore)
316     {
317         cv::Mat_<unsigned char> mask( src.size(), cv::GC_PR_BGD );
318         bool err = true;
319
320         BOOST_FOREACH( int y, boost::irange( 0, i.size().height ) ) {
321             BOOST_FOREACH( int x, boost::irange( 0, i.size().width ) ) {
322                 if( i( y, x ) >= fgd_comp ) {
323                     mask( y, x ) = cv::GC_FGD;
324                     err = false;
325                 }
326                 else if( i( y, x ) >= pr_fgd_comp ) {
327                     mask( y, x ) = cv::GC_PR_FGD;
328                     err = false;
329                 }
330                 else if( i( y, x ) <= bgd_comp ) {
331                     mask( y, x ) = cv::GC_BGD;
332                 }
333             }
334         }
335         if( err ) {
336             return;
337         }
338
339         cv::Mat bgd, fgd;
340         cv::grabCut( src, mask, cv::Rect(), bgd, fgd, cnt, cv::GC_INIT_WITH_MASK );
341
342         boost::mutex::scoped_lock lock( mutex );
343         dst.push_back( mask_image( src, mask, ignore ) );
344     }
345
346 } // namespace
347
348     inline std::vector<cv::Mat_<cv::Vec3b> > cut(
349         const cv::Mat_<cv::Vec3b>& src, int level,
350         double fgd_comp, double pr_fgd_comp, double bgd_comp,
351         double ec_color, double ec_lumi, int cnt, cv::Vec3b ignore = cv::Vec3b( 255, 0, 255 ))
352     {
353         const cv::Mat_<cv::Vec3f> img = cv_ex::byte_to_float( src );
354
355         auto maps = make_maps( img, level );
356         BOOST_FOREACH( auto& i, maps ) {
357             const auto minmax = image_min_max( i );
358             normalize( i, minmax.first, minmax.second );
359             const double loc = image_loc_max_avg( i );
360             const double diff = std::pow( minmax.second - loc, 2 );
361             BOOST_FOREACH( auto& n, i ) {
362                 n *= diff;
```

```
363             }
364             normalize( i );
365         }
366
367         for( auto i = maps.begin() + 1; i < maps.begin() + ( maps.size() - 1 ); ++i ) {
368             for( auto j = i + 1; j < maps.end(); ) {
369                 const double ratio = equal_count( *i, *j ) / static_cast<double>( src.size().area() );
370                 if( ratio >= ec_color ) {
371                     for_each( *i, *j, [](float lhs, float rhs) -> float { return lhs + rhs; } );
372                     j = maps.erase( j );
373                 }
374                 else {
375                     ++j;
376                 }
377             }
378         }
379         for( auto i = maps.begin() + 1; i < maps.end(); ++i ) {
380             const double ratio =
381                 equal_count( *i, maps[0], 0.05f ) / static_cast<double>( src.size().area() );
382
383             if( ratio >= ec_lumi ) {
384                 for_each( *i, maps[0], [](float lhs, float rhs) -> float { return lhs + rhs; } );
385             }
386             normalize( *i );
387         }
388         maps.erase( maps.begin() );
389
390         std::vector<cv::Mat_<cv::Vec3b> > dst;
391         boost::mutex mutex;
392
393 #if defined( CUT_NO_THREAD )
394         BOOST_FOREACH( auto& i, maps ) {
395             graph_cut( src, i, fgd_comp, pr_fgd_comp, bgd_comp, cnt, dst, mutex, ignore );
396         }
397
398 #else
399         std::vector<boost::thread> th;
400
401         BOOST_FOREACH( auto& i, maps ) {
402             th.push_back( boost::thread( [
403                                 &src, &i, fgd_comp, pr_fgd_comp, bgd_comp, cnt, &dst, &mutex, ignore
                                        ]() {
404                             graph_cut( src, i, fgd_comp, pr_fgd_comp, bgd_comp, cnt, dst, mutex, ignore
                                    );
405                         } ) );
406         }
407         BOOST_FOREACH( auto& t, th ) {
408             t.join();
409         }
410 #endif
411
412         return dst;
413     }
414
415 } // namespace cv_ex
416
417 #endif // CUT_HPP_
```

ソースコード B.6: cv_ex.hpp

```
1  #ifndef CV_EX_HPP_
2  #define CV_EX_HPP_
3
4  #include <boost/range/irange.hpp>
5  #include <opencv2/opencv.hpp>
6
7  namespace cv_ex {
8
9      template <class T>
10     inline cv::Mat_<T> resize(const cv::Mat_<T>& src, cv::Size sz, int type = cv::INTER_LINEAR)
11     {
12         cv::Mat_<T> dst;
13         cv::resize( src, dst, sz, type );
14         return dst;
15     }
16     template <class T>
17     inline std::vector<cv::Mat_<T> > create_pyr(const cv::Mat_<T>& src, int level)
18     {
19         std::vector<cv::Mat_<T> > pyr( level + 1 );
20
21         pyr[0] = src;
22         for( int i : boost::irange( 1, level + 1 ) ) {
23             cv::Mat_<T> dst( pyr[i - 1].rows / 2, pyr[i - 1].cols / 2 );
24             cv::pyrDown( pyr[i - 1], dst );
25             pyr[i] = dst;
26         }
27
28         return pyr;
29     }
30
31     template <class Dest, class T>
32     inline cv::Mat_<Dest> cvt_color(const cv::Mat_<cv::Vec<T, 3>>& src)
33     {
34         cv::Mat_<Dest> dst;
35         cv::cvtColor( src, dst, CV_BGR2GRAY );
36         return dst;
37     }
38
39     template <class Dest>
40     inline cv::Mat_<Dest> cvt_color(const cv::Mat_<unsigned char>& src)
41     {
42         cv::Mat_<Dest> dst;
43         cv::cvtColor( src, dst, CV_GRAY2BGR );
44         return dst;
45     }
46
47     template <class Dest, class Src, class F>
48     inline cv::Mat_<Dest> for_each(const cv::Mat_<Src>& src, F f)
49     {
50         cv::Mat_<Dest> dst( src.size() );
51
52         auto src_itr = src.begin();
53         auto dst_itr = dst.begin();
54         for( ; src_itr != src.end(); ++src_itr, ++dst_itr ) {
55             (*dst_itr) = f( (*src_itr) );
56         }
57
58         return dst;
59     }
60
61     template <class T>
62     inline cv::Mat_<T> read_image(const std::string& filename)
63     {
64         try {
65             return cv::imread( filename );
66         }
67         catch( ... ) {
68             return {};
69         }
70     }
71
72     template <>
```

```
73      inline cv::Mat_<cv::Vec3f> read_image<cv::Vec3f>(const std::string& filename)
74      {
75          const auto f = [](const cv::Vec3b& p) -> cv::Vec3f
76          {
77              cv::Vec3f dst;
78              for( int c = 0; c < 3; ++c ) {
79                  dst[c] = p[c] / 255.0f;
80              }
81              return dst;
82          };
83
84          try {
85              return for_each<cv::Vec3f>( cv::Mat_<cv::Vec3b>( cv::imread( filename ) ), f );
86          }
87          catch( ... ) {
88              return {};
89          }
90      }
91
92      inline cv::Mat_<cv::Vec3f> byte_to_float(const cv::Mat_<cv::Vec3b>& src)
93      {
94          cv::Mat_<cv::Vec3f> dst( src.size() );
95
96          auto src_itr = src.begin();
97          auto dst_itr = dst.begin();
98          for( ; src_itr != src.end(); ++src_itr, ++dst_itr ) {
99              for( int c = 0; c < 3; ++c ) {
100                 (*dst_itr)[c] = (*src_itr)[c] / 255.0f;
101             }
102         }
103
104         return dst;
105     }
106
107     inline cv::Mat_<cv::Vec3b> float_to_byte(const cv::Mat_<cv::Vec3f>& src)
108     {
109         cv::Mat_<cv::Vec3b> dst( src.size() );
110
111         auto src_itr = src.begin();
112         auto dst_itr = dst.begin();
113         for( ; src_itr != src.end(); ++src_itr, ++dst_itr ) {
114             for( int c = 0; c < 3; ++c ) {
115                 (*dst_itr)[c] = cv::saturate_cast<unsigned char>( (*src_itr)[c] * 255 );
116             }
117         }
118
119         return dst;
120     }
121
122     inline std::vector<cv::Rect> detect(const cv::Mat_<unsigned char>& img, cv::CascadeClassifier&
          cc)
123     {
124         std::vector<cv::Rect> objs;
125         cc.detectMultiScale( img, objs );
126
127         return objs;
128     }
129
130     inline std::vector<cv::Rect> detect(const cv::Mat_<cv::Vec3b>& img, cv::CascadeClassifier& cc)
131     {
132         std::vector<cv::Rect> objs;
133         cv::Mat_<unsigned char> gray;
134
135         cv::cvtColor( img, gray, CV_BGR2GRAY );
136         cc.detectMultiScale( gray, objs );
137
138         return objs;
139     }
140
141     template <class T>
142     inline cv::Point_<T> center(const cv::Rect_<T>& rc)
143     {
144         return { rc.x + rc.width / 2, rc.y + rc.height / 2 };
```

```
145     }
146
147     template <class T>
148     inline cv::Mat_<T> convert_color(const cv::Mat_<T>& src, int type)
149     {
150         cv::Mat_<T> dst;
151         cv::cvtColor( src, dst, type );
152
153         return dst;
154     }
155
156 } // namespace cv_ex
157
158 #endif // CV_EX_HPP_
```

ソースコード B.7: som.hpp

```
1  #ifndef SOM_HPP_
2  #define SOM_HPP_
3
4  #include <vector>
5  #include <algorithm>
6  #include <limits>
7  #include <cmath>
8  #include <fstream>
9  #include <boost/random.hpp>
10 #include <boost/range/irange.hpp>
11
12 struct som
13 {
14     typedef std::vector<double> node_type;
15     typedef std::vector<node_type> map_type;
16
17 private:
18     map_type map_;
19     int width_, height_;
20
21 public:
22     som() = default;
23
24     som(map_type& map, int width, int height) :
25         map_( map ), width_( width ), height_( height )
26     { }
27
28     inline int width() const
29     {
30         return width_;
31     }
32
33     inline int height() const
34     {
35         return height_;
36     }
37
38     inline node_type& operator()(int x, int y)
39     {
40         return map_[x + y * width_];
41     }
42
43     inline const node_type& operator()(int x, int y) const
44     {
45         return map_[x + y * width_];
46     }
47
48     inline map_type& data()
49     {
50         return map_;
51     }
52
53     inline const map_type& data() const
54     {
55         return map_;
56     }
57 };
58
59 inline som make_som(int width, int height, int element_size)
60 {
61     som::map_type map( width * height );
62     for( auto& i : map ) {
63         som::node_type node( element_size );
64         i = node;
65     }
66
67     return som( map, width, height );
68 }
69
70 namespace
71 {
72     template <class Generator>
73     inline void learing_initialize(som& m, Generator& gen)
```

```
74      {
75          boost::uniform_real<double> dist( 0.0, 1.0 );
76          boost::variate_generator<Generator&, decltype( dist )> random( gen, dist );
77
78          for( auto& n : m.data() ) {
79              for( auto& i : n ) {
80                  i = random();
81              }
82          }
83      }
84
85      inline double cross_sum_(const std::vector<double>& lhs, const std::vector<double>& rhs)
86      {
87          double v = 0;
88          auto l_itr = lhs.begin();
89          auto r_itr = rhs.begin();
90          for( ; l_itr != lhs.end(); ++l_itr, ++r_itr ) {
91              v += (*l_itr) * (*r_itr);
92          }
93
94          return v;
95      }
96
97      inline double cross_correlation_(const std::vector<double>& lhs, const std::vector<double>& rhs)
98      {
99          const double MN = lhs.size() * rhs.size();
100         const double lr = cross_sum_( lhs, rhs );
101         const double l = boost::accumulate( lhs, 0 );
102         const double r = boost::accumulate( rhs, 0 );
103         const double l2 = cross_sum_( lhs, lhs );
104         const double r2 = cross_sum_( rhs, rhs );
105
106         return ( -( ( MN * lr - l * r ) / std::sqrt( ( MN * l2 - l * l ) * ( MN * r2 - r * r ) ) ) + 1.0
                ) / 2.0;
107     }
108
109     template <class F>
110     inline void learing_impl(som& m, const std::vector<std::vector<double>>& samples, F f, int cnt)
111     {
112         for( auto& v : samples ) {
113             int mx = 0, my = 0;
114             double min_d = std::numeric_limits<double>::max();
115
116             for( int j : boost::irange( 0, m.height() ) ) {
117                 for( int i : boost::irange( 0, m.width() ) ) {
118                     //const double d = cross_correlation_( m( i, j ), v );
119                     double d = 0;
120
121                     auto m_itr = m( i, j ).begin();
122                     auto v_itr = v.begin();
123                     for( ; v_itr != v.end(); ++m_itr, ++v_itr ) {
124                         d += std::pow( *v_itr - *m_itr, 2.0 );
125                     }
126                     if( d < min_d ) {
127                         min_d = d;
128                         mx = i;
129                         my = j;
130                     }
131                 }
132             }
133
134             for( int j : boost::irange( 0, m.height() ) ) {
135                 for( int i : boost::irange( 0, m.width() ) ) {
136                     const double p = f( i, j, mx, my, cnt );
137
138                     auto m_itr = m( i, j ).begin();
139                     auto v_itr = v.begin();
140                     for( ; v_itr != v.end(); ++m_itr, ++v_itr ) {
141                         *m_itr += p * ( *v_itr - *m_itr );
142                     }
143                 }
144             }
145         }
```

```
146     }
147
148 }
149
150 template <class Generator, class F>
151 inline void learning(
152         som& m, const std::vector<std::vector<double>>& samples, F f, Generator& gen,
153         int max_count = 1, double eps = std::numeric_limits<double>::epsilon())
154 {
155     learing_initialize( m, gen );
156
157     for( int cnt : boost::irange( 0, max_count ) ) {
158         learing_impl( m, samples, f, cnt );
159     }
160 }
161
162 inline void save_som(const std::string& filename, const som& m)
163 {
164     std::ofstream ofs( filename.c_str(), std::ios::binary );
165
166     const int width = m.width();
167     const int height = m.height();
168     const int node_sz = m.data()[0].size();
169     ofs.write( (char *)( &width ), sizeof( int ) );
170     ofs.write( (char *)( &height ), sizeof( int ) );
171     ofs.write( (char *)( &node_sz ), sizeof( int ) );
172
173     for( const auto& v : m.data() ) {
174         ofs.write( (char *)( &v[0] ), sizeof( decltype( v[0] ) ) * v.size() );
175     }
176 }
177
178 inline som load_som(const std::string& filename)
179 {
180     std::ifstream ifs( filename.c_str(), std::ios::binary );
181
182     int width, height, node_sz;
183     ifs.read( (char *)( &width ), sizeof( int ) );
184     ifs.read( (char *)( &height ), sizeof( int ) );
185     ifs.read( (char *)( &node_sz ), sizeof( int ) );
186
187     som m( make_som( width, height, node_sz ) );
188
189     for( const auto& v : m.data() ) {
190         ifs.read( (char *)( &v[0] ), sizeof( decltype( v[0] ) ) * v.size() );
191     }
192
193     return m;
194 }
195
196 #endif // SOM_HPP_
```

ソースコード B.8: learn.cpp

```
1  #include "classifier.hpp"
2
3  int main(int argc, char *argv[])
4  {
5      try {
6          const auto cl = nclr::command_line( argc, argv );
7          if( cl.size() != 13 ) {
8              throw std::runtime_error{ "invalid argument" };
9          }
10
11         const cv::Size face_sz = { std::atoi( cl[1].c_str() ), std::atoi( cl[2].c_str() ) };
12         const cv::Size wnd_sz = { std::atoi( cl[3].c_str() ), std::atoi( cl[4].c_str() ) };
13         const cv::Size node_sz = { std::atoi( cl[5].c_str() ), std::atoi( cl[6].c_str() ) };
14         const int max_point = std::atoi( cl[7].c_str() );
15         const double valid_ratio = std::atof( cl[8].c_str() );
16         const double center_ratio = std::atof( cl[9].c_str() );
17         const double neighbor_ratio = std::atof( cl[10].c_str() );
18
19         boost::mt19937 gen;
20
21         cv::CascadeClassifier cc{ "lbpcascade_animeface.xml" };
22         if( cc.empty() ) {
23             throw std::runtime_error{ "cannot load the cascade file" };
24         }
25
26         nclr::printf( std::clog, "create samples\n" );
27
28         std::vector<std::vector<double>> samples;
29         for_each_file( cl[11], [&](const boost::filesystem3::path& p) {
30             if( boost::filesystem3::is_directory( p ) ) {
31                 return;
32             }
33             nclr::printf( std::clog, "% \n", p.generic_string() );
34
35             const auto src = cv_ex::read_image<cv::Vec3b>( p.generic_string() );
36             if( src.empty() ) {
37                 return;
38             }
39
40             const auto ss = create_samples( src, cc, face_sz, wnd_sz, max_point, valid_ratio, gen );
41             for( const auto& s : ss ) {
42                 samples.push_back( s );
43             }
44         } );
45         if( samples.empty() ) {
46             throw std::runtime_error{ "not create samples" };
47         }
48
49         nclr::printf( std::clog, "som learning\n" );
50
51         som m = { make_som( node_sz.width, node_sz.height, wnd_sz.area() * 3 ) };
52         const auto neighbor = [center_ratio, neighbor_ratio](int x, int y, int mx, int my, int) ->
               double {
53             const int i = x - mx;
54             const int j = y - my;
55             return i == 0 && j == 0 ? center_ratio :
56                 std::abs( i ) <= 1 && std::abs( j ) <= 1 ? neighbor_ratio : 0.0;
57         };
58         for( int i : boost::irange( 0, node_sz.area() ) ) {
59             m.data()[i] = samples[i];
60         }
61
62         for( int cnt : boost::irange( 0, 1 ) ) {
63             learning( m, samples, neighbor, gen );
64             nclr::printf( std::clog, "% \r", cnt );
65         }
66
67         nclr::printf( std::clog, "counting\n" );
68
69         const auto count = counting( m, samples );
70
71         save( cl[12], m, count );
```

```
72
73          nclr::printf( std::clog, "finish\n" );
74      }
75      catch ( const std::exception&& e ) {
76          nclr::printf( std::cerr, "%s\n", e.what() );
77      }
78      catch ( ... ) {
79          nclr::printf( std::cerr, "other exception\n" );
80      }
81  }
```

ソースコード B.9: predicate.cpp

```
1  #include "classifier.hpp"
2
3  int main(int argc, char *argv[])
4  {
5      try {
6          const auto cl = nclr::command_line( argc, argv );
7          if( cl.size() != 10 ) {
8              throw std::runtime_error{ nclr::make_string( "invalid argument", cl.size() ) };
9          }
10
11         const cv::Size face_sz = { std::atoi( cl[1].c_str() ), std::atoi( cl[2].c_str() ) };
12         const cv::Size wnd_sz = { std::atoi( cl[3].c_str() ), std::atoi( cl[4].c_str() ) };
13         const int max_point = std::atoi( cl[5].c_str() );
14         const double valid_ratio = std::atof( cl[6].c_str() );
15
16         boost::mt19937 gen;
17
18         cv::CascadeClassifier cc{ "lbpcascade_animeface.xml" };
19         if( cc.empty() ) {
20             throw std::runtime_error{ "cannot load the cascade file" };
21         }
22
23         nclr::printf( std::clog, "load maps\n" );
24
25         std::vector<std::tuple<som, std::vector<double>>> maps;
26         for_each_file( cl[8], [&maps](const boost::filesystem3::path& p) {
27             if( boost::filesystem3::is_directory( p ) ) {
28                 return;
29             }
30             nclr::printf( std::clog, "% \n", p.generic_string() );
31
32             maps.push_back( load( p.generic_string() ) );
33         } );
34         if( maps.empty() ) {
35             throw std::runtime_error{ "no maps" };
36         }
37
38         nclr::printf( std::clog, "create samples\n" );
39
40         for_each_file( cl[7], [&](const boost::filesystem3::path& p) {
41             if( boost::filesystem3::is_directory( p ) ) {
42                 return;
43             }
44             nclr::printf( std::clog, "% \n", p.generic_string() );
45
46             const auto src = cv_ex::read_image<cv::Vec3b>( p.generic_string() );
47             if( src.empty() ){
48                 return;
49             }
50
51             const auto samples = create_samples( src, cc, face_sz, wnd_sz, max_point, valid_ratio, gen
                    );
52             if( samples.empty() ) {
53                 return;
54             }
55
56             nclr::printf( std::clog, "counting\n" );
57             std::vector<std::vector<double>> counts;
58             for( const auto& m : maps ){
59                 counts.push_back( counting( std::get<0>( m ), samples ) );
60             }
61
62             nclr::printf( std::clog, "predicate\n" );
63             double min_d = std::numeric_limits<double>::max();
64             int min_i = 0;
65             for( int i : boost::irange( 0, static_cast<int>( maps.size() ) ) ) {
66                 const double d = distance( std::get<1>( maps[i] ), counts[i] );
67                 //const double d = cross_correlation( std::get<1>( maps[i] ), counts[i] );
68                 if( d < min_d ) {
69                     min_d = d;
70                     min_i = i;
71                 }
```

```
72                  }
73                  cv::imwrite( nclr::make_string( "%_/%_/%_", cl[9], min_i, p.filename().generic_string() ),
                        src );
74              } );
75
76              nclr::printf( std::clog, "finish\n" );
77          }
78          catch ( const std::exception&& e ) {
79              nclr::printf( std::cerr, "%_\n", e.what() );
80          }
81          catch ( ... ) {
82              nclr::printf( std::cerr, "other_exception\n" );
83          }
84      }
```