САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И СИСТЕМ

Иванов Дмитрий Алексеевич

Выпускная квалификационная работа бакалавра

Алгоритм текстурной сегментации с использованием вейвлетов Габора

Направление 01003

Фундаментальная информатика и информационные технологии

Научный руководитель,
кандидат физ-мат. наук,
доцент
Мисенов Б.А.

Санкт-Петербург
2016

Содержание

Введение

Современное развитие вычислительной техники, сопровождающееся непрерывным ростом производительности, доступной по потребительским ценам, неуклонно ведет к ускорению процессов консолидации мирового информационного пространства. В рамках этих процессов неизбежно возникают задачи извлечения информации, из источников, традиционно считавшихся непригодными для автоматизированного анализа – изображения, видео- и аудио записи. В рамках задач, связанных с анализом изображений и видеозаписей естественным образом возникают задачи, носящих общее название задач текстурной сегментации.

Задача текстурной сегментации, в общей постановке, это задача разбиения заданного изображения, на участки, однородные по структуре. На данный момент различают два общих подхода к решению этой задачи: структурный и статистический. Структурный подход применяется к сегментированию изображений, сгенерированных искусственно. В нем, текстура рассматривается как шаблон, организованный из повторяющихся текселей – элементов текстуры. Статистический подход применятся к изображениям, имеющим природное происхождение – он менее точен, в нем применяются численные оценки расположения интенсивностей пикселов на участке изображения.

На сегодняшний день, можно различить две группы методов, активно развивающихся, и относящихся к методам группы статистического подхода. Это методы основанные на нейросетевых алгоритмах – группа методов Глубокого Обучения и методы, основанные на различных вейвлет-разложениях изображения. Среди методов, основанных на вейвлет-преобразованиях, особо перспективными считаются методы, основанные на применении вейвлетов Габора. Вейвлеты Габора, обладают двумя замечательными свойствами – имеют наименьшую неопределенность при локализации свойств сигнала в нескольких доменах (областях), а также, ведут себя как некоторые нейронные структуры

зрительной коры.

Алгоритмы, основанные на вейвлет-преобразованиях, использующих фильтры Габора, хорошо показывают себя в задачах, связанных с распознаванием лиц (и лицевых выражений), отпечатков пальцев, анализе медицинских изображений. Также в последнее время развиваются методы, связанные с применением вейвлет-преобразования Габора для анализа спутниковых снимков, и изображений аэрофотосъемки. Однако их применение на более широких классах изображений сопряжено с вычислительными трудностями – для вейвлет преобразований Габора в общем случае нельзя задать параметры, которые приведут к наиболее адекватному разложению.

Постановка задачи

В рамках данной работы ставится задача синтеза и реализации алгоритма текстурной сегментации с использованием вейвлет-преобразования на основе фильтров Габора. В данной работе ставится задача реализовать алгоритм, дающий приемлемые результаты на изображениях в градациях серого, взятых из открытого репозитория конкурса алгоритмов сегментации Беркли.

Обзор литературы

Основные идеи, использовавшиеся в данной работе, были высказаны Sochen et. al. в [1]. Они были также изложены в работах Sagiv et. al. в работе [2], когда автор применяла модификацию алгоритма активных контуров, поверх полученного с помощью вейвлетов Габора представления. За время, прошедшее с момента публикаций Sochen и Sagiv было разработано множество применений вевйвлетов Габора в анализе изображений. Большинство из них, такие как работы [3], [4], [5] – ограничивались узкими классами изображений, к примеру, отпечатками ладоней, снимками томографий, изображениями аэрофотосъемки. Во многих из них также упоминается работа [6] T. S. Lee, 1996 года – работа в которой автор обращается к проблеме подбора оптимальных параметров для фильтров Габора. Относительно проблемы подбора параметров фильтров так же существует работа [7] Dunn et. al. 1995 года, в которой автор представляет метод подбора параметров, при условии, что известны образцы текстур.

В большинстве работ, в которых строятся алгоритмы анализа изображений с использованием вейвлетов Габора, поверх представления изображения используются различные алгоритмы машинного обучения – нечеткий алгоритм KNN,машины поддерживающих векторов, нейронные сети. Для минимизации количества необходимых вычислений в многомерном пространстве представлений Габора используются различные вариации алгоритма PCA (modular 2DPCA, [8]).

Во всех приведенных работах, в силу крайне узкоспециализированных классов изображений, мало внимания отводится вопросам практического построения представления изображений. Однако, как только класс изображений становится чуть менее однородным – авторы сталкиваются с резким ухудшением результатов работы [4].

Глава 1. Теоретические основы.

1.1 Построение представления.

Базис вейвлет-преобразования Габора в общем виде можно определить как:

$$g(x', y', \lambda, \sigma, \xi, \nu, \theta) = \frac{1}{2\pi\sigma^2} e^{\left(\frac{-x'^2}{2\sigma^2\lambda^2} - \frac{y'^2}{2\sigma^2}\right)} e^{2\pi i\sqrt{\xi^2+\nu^2}x'}$$

(1)

Здесь, λ – соотношение сторон эллиптического Гауссова окна, σ – параметр масштаба, ξ, ν – модулирующие частоты преобразования Фурье, θ – ориентация фильтра. x', y' пространственные координаты, повернутые на угол θ:

$$(x', y') = (x\cos\theta + y\sin\theta, -x\sin\theta + y\cos\theta)$$

(2)

Вейвлеты Габора можно получить из представления (1) путем его поворота и растяжения или сжатия для известного числа масштабов S и ориентаций K:

$$g_{mn} = \frac{1}{c^m} g\left(\frac{x'}{c^m}, \frac{y'}{c^m}\right)$$

(3)

где x', y' – пространственные координаты, повернутые на углы $\frac{\pi n}{K}$

, и масштабированные по степеням $m = 0 \ldots S-1.$

Для того чтобы выбранные фильтры адекватно представляли информацию текстуры изображения необходимо выбирать параметры для каждого класса изображений индивидуально.

Тем не менее, в общем, случае, при заданных наборах возможных параметров (частот, ориентаций, масштабов), представление изображения в пространстве вейвлетов Габора можно построить следующим образом: сначала изображение сворачивается с каждым возможным фильтром (4), затем, для

каждого расположения пиксела отбирается только фильтр, имеющий максимальную вещественную часть в данном расположении.

$$I(x,y)*g(x,y)_{mn}=\Re_{mn}(x,y)+\Im_{mn}(x,y) \quad (4)$$

Таким образом, изображение переводится из 3-х мерного представления по координатам $(x,y,I(x,y))$, т.е. расположение пиксела и его интенсивность в 7-ми мерное:

$$(x,y,\theta(x,y),\Re(x,y),\Im(x,y),f(x,y),\sigma(x,y)) \quad (5)$$

В (5) каждому расположению пиксела ставится в соответствие вещественная и мнимая части отклика фильтра, а так же частота, масштаб и ориентация данного фильтра.

Параметр λ, характеризующий отношение сторон эллиптического Гауссова окна, как утверждается в [6] постоянен и равен 2:1.

1.2 Регуляризация представления.

Построенное 7-ми мерное представление не является достаточно гладким для проведения дальнейшего анализа. Для того, чтобы его сгладить можно воспользоваться следующими соображениями:

1. Вначале необходимо задать на нем метрику используя процедуру pullback
2. Используя заданную метрику провести минимизацию функционала представляющего элемент площади поверхности.

В результате этих действий получится многообразие, на котором, с приемлемой точностью, можно будет определить детектор границы.

Процедура pullback является общим соображением, известным из дифференциальной геометрии. Формально, pullback можно определить следующим образом:

пусть $\varphi:\Sigma\to M$

гладкое отображение Σ

на M и существует $f:M\to R,$ так же гладкое, тогда $(f*\varphi):\Sigma\to R$,

определяется как

$$(f*\varphi)=f(\varphi)$$

(6)

В данном случае Σ – 7-ми мерное многообразие, в которое встроено трехмерное пространство изображения (M). Т.е. карта встройки, исходя из (5):

$$\varphi:(x,y,I(x,y))\to(x,y,\theta(x,y),\Re(x,y),\Im(x,y),f(x,y),\sigma(x,y))(7)$$ Отображение вида $f:M\to R$ в данном случае это:

$$f:(x,y,I(x,y))\to\begin{bmatrix}\frac{\partial^2 I}{\partial x^2} & \frac{\partial I}{\partial x}\frac{\partial I}{\partial y}\\ \frac{\partial I}{\partial y}\frac{\partial I}{\partial x} & \frac{\partial^2 I}{\partial y^2}\end{bmatrix}(8)$$

И, последняя часть процедуры pullback, определение метрики встраивающего многообразия, исходя из метрики встроенного, и формулы (7):

$$(f*\varphi)=\begin{bmatrix}\sum\frac{\partial^2 G^i}{\partial x^2} & \sum\frac{\partial G^i\partial G^i}{\partial x\partial y}\\ \sum\frac{\partial G^i\partial G^i}{\partial x\partial y} & \sum\frac{\partial^2 G^i}{\partial x\partial y}\end{bmatrix}(9)$$ Здесь,

$G^i\in\{\theta(x,y),\Re(x,y),\Im(x,y),f(x,y),\sigma(x,y)\}$.

Определив, таким образом, метрику в пространстве представления вейвлетов Габора, легко задать функционал, определяющий площадь поверхности элемента представления:

$S=\iint\sqrt{det\,((f*\varphi))}\,dxdy(10)$ Имея в виду тот факт, что каждый из элементов представления– частота, ориентация, масштаб, вещественный и мнимый отклики – не зависят друг от друга (это допущение данной модели), для данного функционала можно построить итеративный метод приближения, минимизируя каждый из элементов представления по отдельности. Пусть r некоторый элемент представления, тогда, используя соображения вариационного счисления, можно записать уравнение в частных производных,

минимизирующее данный элемент, в следующем виде:

$$\frac{\delta S}{\delta r} = -¿\left(\frac{\nabla_r\left(det\left(\left(f * \varphi\right)\right)\right)}{2\sqrt{det\left(\left(f * \varphi\right)\right)}}\right)(11)$$ где

$$\nabla_r\left(det\left(\left(f * \varphi\right)\right)\right) = \left(\frac{\partial\, det\left(\left(f * \varphi\right)\right)}{\frac{\partial r}{\partial x}}, \frac{\partial\, det\left(\left(f * \varphi\right)\right)}{\frac{\partial r}{\partial y}}\right)¿ \quad 12)$$

В соответствии с методом наискорейшего спуска, можно задать итеративный процесс минимизации (более подробно в [1]), со следующим шагом:

$$\frac{\partial r}{\partial t} = \frac{-1}{\sqrt{det\left(\left(f * \varphi\right)\right)}} \frac{\delta S}{\delta r}(13)$$

1.3 Детектор края

В полученном представлении удобно определить детектор края, как:

$D(x, y) = \frac{1}{1 + det\left(\left(f * \varphi\right)(x, y)\right)}(14)$ Таким образом, значения функции (14), будут лежать в интервале $[0,1]$, причем так как детерминант по сути отражает величину градиента в данной точке, значения функции будут близки к нулю в точках границы.

Глава 2. Реализация алгоритма

2.1 Общее описание.

Получившийся алгоритм состоит из следующих основных шагов:

1. Сгенерировать представление изображения с помощью вейвлетов Габора.
2. Сгладить представление используя минимизацию элемента площади поверхности (11).
3. Используя детектор края (14) определить границы текстур.

Ниже данные шаги будут рассмотрены более подробно.

2.2 Генерация представления изображения с помощью вейвлетов Габора.

Алгоритм генерации выглядит следующим образом:

Алгоритм I

Вход: I (матрица $n \times m$ пикселей изображения), F (множество частот), S (множество масштабов), O (множество ориентаций)

Выделить матрицу P $n \times m$, в ячейках которой будет храниться информация о вейвлете, давшем наибольший отклик в данной точке.

Присвоить изначально всем откликам значение 0.

Для каждой частоты $f \in F$

Для каждого масштаба $s \in S$

Для каждой ориентации $o \in O$

Вычислить ядро фильтра с данными (f, s, o)

Выполнить свертку изображения с данным фильтром, получить $\hat{I}$

Для каждого $i \in n$

Для каждого $j \in m$

Если $\left|\hat{I}(i,j).\Re\right| > \left|P(i,j).\Re\right|$

То $P(i,j) = \{f, s, o, \hat{I}(i,j).\Re, \hat{I}(i,j).\Im\}$

Конец

Результатом работы алгоритма является матрица представления, P в ячейках которой содержится информация о вейвлете давшем максимальный отклик в данной точке. Данная матрица в дальнейшем будет рассматриваться как 5 независимых, двумерных функций, все из которых в совокупности дают представление изображения в данном пространстве вейвлетов Габора. Можно также заметить, что каждое конкретное пространство в данном случае будет определяться множествами F, S, O . В следующем разделе будет дано описание алгоритма, входными данными которого будет являться построенное представление.

2.3 Регуляризация представления.

Получившееся в результате работы алгоритма I представление необходимо сгладить, перед тем как к нему можно будет применять детектор края (14). Однако важно отметить, что в рамках практической реализации целесообразно перед сглаживанием с помощью итеративного процесса с шагом (13), каждый из каналов представления необходимо пропустить через фильтр Гаусса.

Также, в качестве ограничения по количеству шагов итеративного процесса удобно использовать следующую величину:

$$\Delta = \frac{\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{5}\left|\frac{\partial r_k}{\partial t}\right|}{5 \cdot n \cdot m} \quad (15)$$

На каждом шаге величина (15) будет отражать величину, на которую изменилось представление (см. (13)).

Таким образом, получившийся алгоритм можно суммировать в следующем виде:

Алгоритм II

Вход: матрица $n \times m$ **представления** P

GaussianFilter (P.Re, kernel = 9)

GaussianFilter (P.Im, kernel = 9)

GaussianFilter (P.S, kernel = 9)

GaussianFilter (P.F, kernel = 9)

GaussianFilter (P.O, kernel = 9)

$\Delta = 1$

Пока $\Delta > \varepsilon$

$\Delta = 0$

Для каждого $i \in n$

Для каждого $j \in m$

Для каждого канала представления

$c \in \{P(i,j).\Re, P(i,j).\Im, P(i,j).O, P(i,j).S, P(i,j).F\}$

Вычислить величину шага Δ_c **по формуле (13)**

Прибавить Δ_c **к** c

Прибавить Δ_c **к** Δ

Конец цикла

Конец цикла

Конец цикла

Конец

Над получившимся в данном алгоритме сглаженном представлении уже можно применять детектор края (14).

2.4 Детектор края

Алгоритм детектора края с одной стороны достаточно прямолинеен – в простейшем случае достаточно просто обойти все точки получившегося сглаженного представления и отметить те из них, значения детектора в которых

«близки к нулю». Определение близости к нулю по сути своей задача эмпирическая – для данного пространства вейвлетов близость к нулю будет своя, необходимо изучить несколько образцов получившихся представлений, что бы оценить близость к нулю. С учетом этого замечания, можно описать алгоритм детектора края следующим образом:

Алгоритм III

Вход: матрица $n \times m$ **представления** P

Для каждого $i \in n$

Для каждого $j \in m$

Вычислить величину D **по формуле (14)**

Если $D(i,j) < \varepsilon$

Добавить (i,j) **к списку точек границы.**

Конец цикла

Конец цикла

Глава 3. Результаты выполнения алгоритма

В результате проделанной работы по формализации была написана реализация алгоритма текстурной сегментации на языке C#. Для операций с изображениями, в частности свертки и фильтра Гаусса была использована библиотека EmguCV представляющая собой адаптацию известной библиотеки алгоритмов OpenCV.

Алгоритм был протестирован на изображениях Berkley Segmentation Dataset and Benchmark. Были взяты изображения в градациях серого размером 481x321 пиксель (или 321x481). Для несложных изображений – с относительно простым фоном были получены приемлемые результаты (рис. 1,2 и 3, 4).

Рис. 1 Ворон

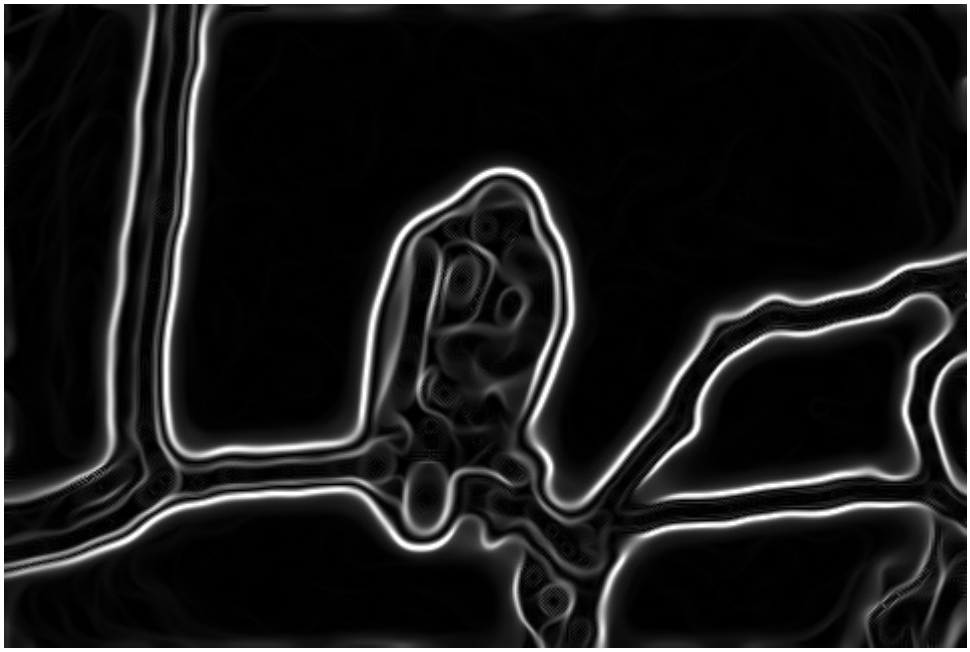

Рис. 2 Результат сегментации

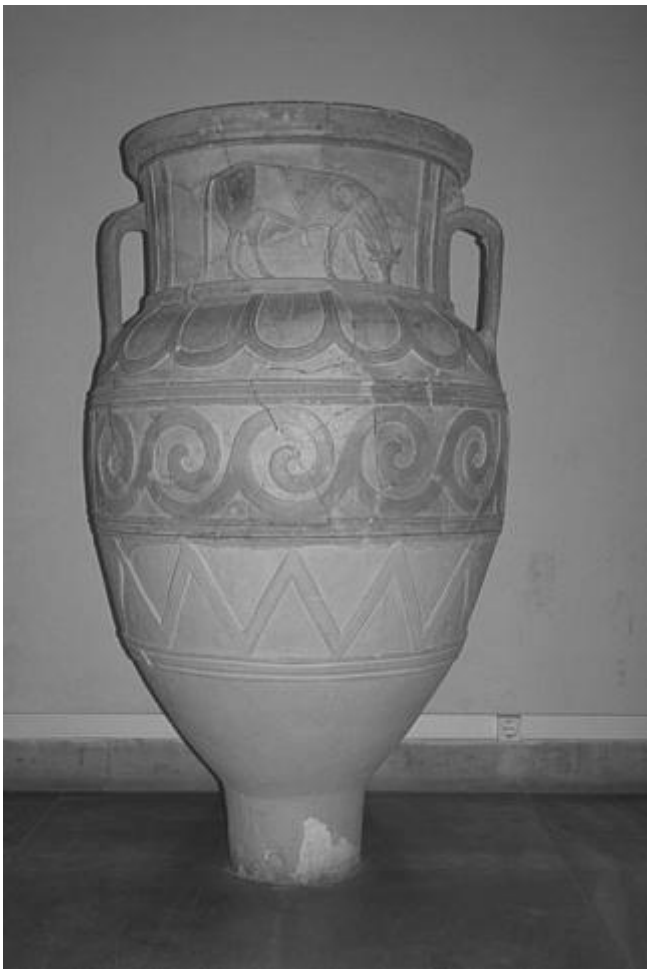

Рис. 3 Амфора

Рис. 4 Результат сегментации

Для изображений обладающих более сложной структурой результат работы будет уже существенно хуже (рис. 5 и 6).

Рис. 5 Руины

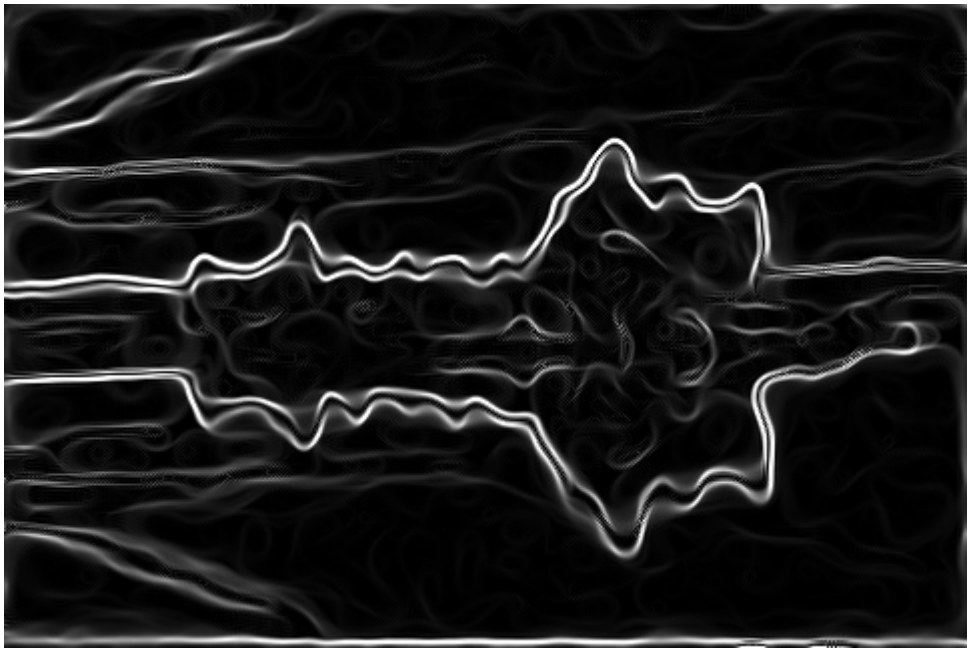

Рис. 6 Результат сегментации

Как можно видеть на изображениях, где объекты расположены однородном фоне результат работы приемлем – следствие выбора пространства вейвлетов.

Отдельно следует сказать о времени работы алгоритма – в среднем от 3 m 45 s – 3m 55s, значительно больше чем у детектора Кэнни. Однако и результат

значительно лучше (рис. 7, 8, 9).

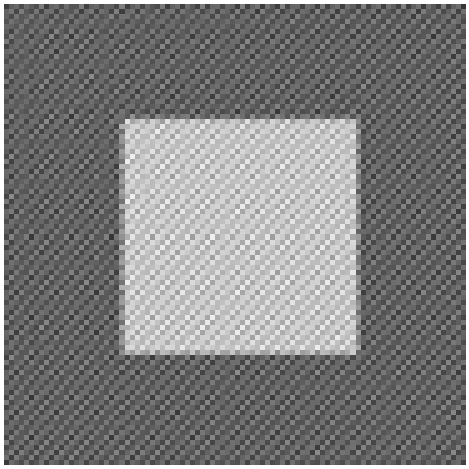

Рис. 7 Контрольное изображение для сравнения

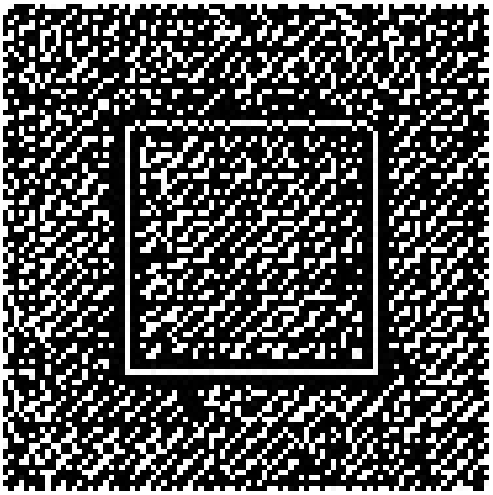

Рис. 8 Результат работы детектора Кэнни

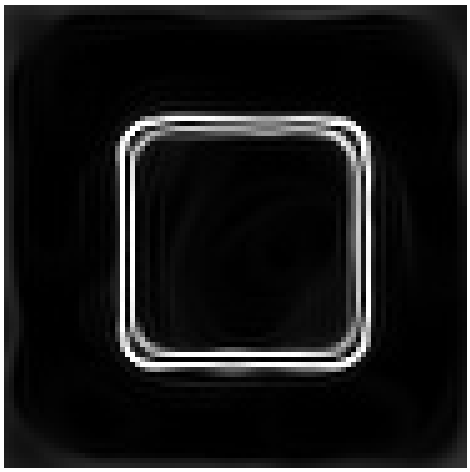

Рис. 9 Результат работы предложенного алгоритма.

Выводы

Как можно видеть из времени работы (~4 минуты на изображении ~150 000 пикселей), алгоритм является достаточно ресурсоемким. Однако, в силу своей природы (большое количество операций, которые могут выполняться независимо), он также допускает весьма значительную оптимизацию, при условии большого объема доступной оперативной памяти. Выполнение данной оптимизации, теме не менее, осталось за рамками данной работы.

Резюмируя, можно сказать, что синтезированный алгоритм подходит для сегментации изображений, обладающих относительно несложной структурой - небольшое число объектов на изображении, относительно равномерный фон. При времени работы, в наивной имплементации, очевидно, слишком высоком для использования алгоритма в системах реального времени – алгоритм, при его успешной распределенной реализации может являться эффективным дополнением различных систем распознавания.

Заключение

В данной работе была рассмотрена одна из возможностей синтеза алгоритма текстурной сегментации (нахождения границы) с использованием вейвлетов Габора. На сегодняшний день, вейвлеты Габора переживают бурное развитие – число публикующихся работ, в которых они используются, растет. Возникают новые применения, решаются новые и существующие задачи.

Тем не менее, как стало ясно в процессе выполнения данной работы, вейвлет-разложение Габора, помимо преимуществ, несет в себе также глубокие теоретические вопросы, ответы на которые являются критическими, для успешного их применения. Безусловно, самый главный из них – как выбрать правильное представление изображения. В данной работе был взят набор параметров, исходя из рекомендаций, данных в работе [6]. Получившееся представление дало приемлемый результат сегментации, однако вопрос о том, является ли это представление наилучшим, остается открытым.

Список литературы

1. N. Sochen, R. Kimmel, R. Malladi A general framework for low level vision // IEEE Transactions On Image Processing, 1998. Vol. 7, P. 310-318.
2. C. Sagiv, Sochen N. A., Y. Y. Zeevi Integrated Active Contours for Texture Segmentation // IEEE Transactions On Image Processing, 2004. Vol. 1
3. EKİNCİ, Murat, AYKUT, Murat Kernel Fisher discriminant analysis of Gabor-Features for online palmprint verification // Turkish Journal of Electrical Engineering & Computer Sciences, 2016. Vol. 24, No. 2, P. 355- 369.
4. Z. Wanga, R. Boescha, C. Ginzler Forest delineation of aerial images with Gabor wavelets // International Journal of Remote Sensing, 2012. Vol. 33, No. 7, P. 2196-2213.
5. X. Zhu, X. He, P. Wang, Q. He, D. Gao, J. Cheng, B. Wu A method of localization and segmentation of intervertebral discs in spine MRI based on Gabor filter bank // Bio-Medical Engineering On-Line, 2016. Vol. 15, No. 32, P. 245-261.
6. 6. T.S. Lee Image Representation using 2D Gabor-Wavelets // IEEE Transactions on PAMI, 1996. Vol. 18, No. 10, P. 959-971.
7. D. Dunn, W. Higgins Optimal Gabor Filters for Texture Segmentation // IEEE Transactions On Image Processing, 1995. Vol. 4, P. 947-964.
8. H. Yan, P. Wang, W.D. Chen , J. Liu Face Recognition Based on Gabor Wavelet Transform and Modular 2DPCA // International Conference on Power Electronics and Energy Engineering, 2016. Vol. 1, P. 245-248.

Приложение

Функция генерации представления

```
public                                    GaborFeatureSpaceElement[,]
GenerateFutureSpaceWithMaximalResponseCoefficients(
                  Image<Gray, float> image, double[] orientations, double[] scales,
double[] frequencies)
        {

            int szOrientations = orientations.Length;
            int szScales = scales.Length;
            int szFrequencies = frequencies.Length;

            Size imageSize = image.Size;

            int width = imageSize.Width;
            int height = imageSize.Height;

              //TODO find maximum coefficient at each point. 3D ARRAY (X,Y, Z = [0;
NUMBER_OF_DIFFERENT_WAVELETS])
                               GaborFeatureSpaceElement[] featureSpace =
AllocateIntermediateFeatureSpace(imageSize);

                          Image<Gray, float> imageConvolutionReal = new Image<Gray,
float>(imageSize);
                           Image<Gray, float> imageConvolutionImg = new Image<Gray,
float>(imageSize);



            for (int i = 0; i < szOrientations; i++)
            {
                for (int j = 0; j < szScales; j++)
                {
                    for (int k = 0; k < szFrequencies; k++)
                    {

                                                                GaborWavelet wavelet =
GaborWaveletGenerator.GenerateGaborWavelet(GaborKernelSize, scales[j],
                             orientations[i], frequencies[k]);

                              CvInvoke.cvFilter2D(image.Ptr, imageConvolutionReal.Ptr,
wavelet.ConvolutionKernelReal, new Point(-1, -1));
                               CvInvoke.cvFilter2D(image.Ptr, imageConvolutionImg.Ptr,
wavelet.ConvolutionKernelImg, new Point(-1, -1));

                           GaborCoefficient coefficient = new GaborCoefficient{RealPart
= imageConvolutionReal.Data, ImgPart = imageConvolutionImg.Data};

                          for (int l = 0; l < height; l++)
                          {
```

```
                            for (int p = 0; p < width; p++)
                            {
                                Vector2D coeff = new Vector2D
                                {
                                    X = coefficient.RealPart[l, p, 0],
                                    Y = 0 //coefficient.ImgPart[l, p, 0]
                                };
                                                    float responseMagnitude = (float)
_analysis.GetEuclideanMetricLength(ref coeff);

                                   if (featureSpace[l*width + p].ResponseMagnitude <
responseMagnitude)
                                {

                                     featureSpace[l*width + p].AssociatedGaborWavelet
= wavelet;
                                        featureSpace[l*width + p].RealResponsePart =
(float) coeff.X;
                                     coeff.Y = coefficient.ImgPart[l, p, 0];
                                          featureSpace[l*width + p].ImgResponsePart =
(float) coeff.Y;
                                    featureSpace[l*width + p].ResponseMagnitude =
responseMagnitude;
                                }
                            }
                        }
                    }
                }

                return ConvertFromIntermediate(ref featureSpace, imageSize);
        }
```

Функция сглаживания представления

```
void CoupledBeltramiFlowDiffusion()
        {
            double[,] determinantValues = new double[_mapHeight,_mapWidth];
                    double[,] squareRootOfDeterminantValues = new double[_mapHeight,
_mapWidth];

               Vector2D[,] realVectorField = Util.Util.AllocateVectorField(_mapWidth,
_mapHeight);
                Vector2D[,] imgVectorField = Util.Util.AllocateVectorField(_mapWidth,
_mapHeight);
                                    Vector2D[,] frequencyVectorField =
Util.Util.AllocateVectorField(_mapWidth, _mapHeight);
             Vector2D[,] scaleVectorField = Util.Util.AllocateVectorField(_mapWidth,
_mapHeight);
                                Vector2D[,] orientationVectorField =
Util.Util.AllocateVectorField(_mapWidth, _mapHeight);

            double simDelta = 1.0f;

            for (; simDelta > 0.01; )
            {
                for (int y = 0; y < _mapHeight; y++)
                {
                    for (int x = 0; x < _mapWidth; x++)
                    {
```

```
                        determinantValues[y, x] =
                            1/_metricsCalculator.GetGaborFeatureMetricsDeterminantAt
Point(ref _embeddingMap, x, y);
                                                squareRootOfDeterminantValues[y, x] =
Math.Sqrt(1+determinantValues[y, x]);
                    }
                }


               for (int y = 0; y < _mapHeight; y++)
                {
                    for (int x = 0; x < _mapWidth; x++)
                    {
                             Vector2D real = _analysis.CalculateGradientOnDomain(ref
determinantValues,
                            ref _embeddingMap.RealResponseFunction, x, y);
                              Vector2D img = _analysis.CalculateGradientOnDomain(ref
determinantValues,
                            ref _embeddingMap.ImgResponseFunction, x, y);
                            Vector2D scale = _analysis.CalculateGradientOnDomain(ref
determinantValues,
                            ref _embeddingMap.ScaleFunction, x, y);
                         Vector2D frequency = _analysis.CalculateGradientOnDomain(ref
determinantValues,
                            ref _embeddingMap.FrequencyFunction, x, y);
                                                            Vector2D orientation =
_analysis.CalculateGradientOnDomain(ref determinantValues,
                            ref _embeddingMap.OrientationFunction, x, y);

                          double denominator = 1/(2*squareRootOfDeterminantValues[y,
x]);

                        realVectorField[y, x].X = real.X / denominator;
                        realVectorField[y, x].Y = real.Y / denominator;

                        imgVectorField[y, x].X = img.X/ denominator;
                        imgVectorField[y, x].Y = img.Y / denominator;

                                    orientationVectorField[y, x].X = orientation.X /
denominator;
                                    orientationVectorField[y, x].Y = orientation.Y /
denominator;

                        frequencyVectorField[y, x].X = frequency.X / denominator;
                        frequencyVectorField[y, x].Y = frequency.Y / denominator;

                        scaleVectorField[y, x].X = scale.X / denominator;
                        scaleVectorField[y, x].Y = scale.Y / denominator;
                    }
                }
                   CoupledBeltramiFlowUpdate(ref realVectorField, ref imgVectorField,
ref orientationVectorField, ref scaleVectorField, ref frequencyVectorField, ref
squareRootOfDeterminantValues, out simDelta);
            }
            SmoothOutFetures();
                        Util.Util.SaveMatrix(ref _embeddingMap.RealResponseFunction,
"realResponse");
                         Util.Util.SaveMatrix(ref _embeddingMap.ImgResponseFunction,
"imgResponse");
            Util.Util.SaveMatrix(ref _embeddingMap.ScaleFunction, "scaleResponse");
                         Util.Util.SaveMatrix(ref _embeddingMap.OrientationFunction,
```

```
"orientationResponse");
                        Util.Util.SaveMatrix(ref  _embeddingMap.FrequencyFunction,
"frequencyResponse");

        }


              private void CoupledBeltramiFlowUpdate(ref Vector2D[,] realField, ref
Vector2D[,] imgField, ref Vector2D[,] orientationField, ref Vector2D[,] scaleField,
ref Vector2D[,] frequencyField, ref double[,] squareRootsOfDet, out double delta)
        {

            delta = 0;
            for (int i = 0; i < _mapHeight; i++)
            {
                for (int j = 0; j < _mapWidth; j++)
                {
                                    double realDelta = -1 / squareRootsOfDet[i, j] *
_analysis.CalculateDivergence(ref realField, j, i);
                    double imgDelta = -1 / squareRootsOfDet[i, j] *
                                                                    _analysis.CalculateD
ivergence(ref imgField, j, i);
                    double frequencyDelta = -1 / squareRootsOfDet[i, j] *
                                                                    _analysis.CalculateD
ivergence(ref frequencyField, j, i);
                    double orientationDelta = -1 / squareRootsOfDet[i, j] *
                                                                    _analysis.CalculateD
ivergence(ref orientationField, j, i);
                    double scaleDelta = -1 / squareRootsOfDet[i, j]*
                                                        _analysis.CalculateDivergence(ref
scaleField, j, i);

                    _embeddingMap.RealResponseFunction[i, j] += realDelta;

                    _embeddingMap.ImgResponseFunction[i, j] += imgDelta;

                    _embeddingMap.FrequencyFunction[i, j] += frequencyDelta;

                    _embeddingMap.OrientationFunction[i, j] += orientationDelta;

                    _embeddingMap.ScaleFunction[i, j] += scaleDelta;

                                delta += (Math.Abs(realDelta) + Math.Abs(imgDelta) +
Math.Abs(frequencyDelta) + Math.Abs(orientationDelta) + Math.Abs(scaleDelta));
                }
            }
```

Функция получения границы

```
public double[,] GetStoppingTermFunction()
        {
            double[,] stoppingTermFunction = new double[_mapHeight, _mapWidth];

            for (int y = 0; y < _mapHeight; y++)
            {
                for (int x = 0; x < _mapWidth; x++)
                {
                    stoppingTermFunction[y, x] = 1/
                                                 (1 +
```

```
                                                    _metricsCalculator.GetGaborFeature
MetricsDeterminantAtPoint(
                                                        ref _embeddingMap, x, y));
                }
            }


            SmoothOuFeature(ref stoppingTermFunction);
            Util.Util.SaveMatrix(ref stoppingTermFunction, "stoppingTerm");

            return stoppingTermFunction;
        }


        public LinkedList<Point> GetBorderPoints()
        {
            CoupledBeltramiFlowDiffusion();

            _levelSetFunctionValues = GetStoppingTermFunction();
            LinkedList<Point> points = new LinkedList<Point>();

           double[,] gradientMagnitudes = new double[_mapHeight,_mapWidth];
            //GaussKernelSize = 5;
            SmoothOuFeature(ref _levelSetFunctionValues);
            double expected;
                   double min = Util.Util.MinValue(ref _levelSetFunctionValues, out
expected);

            for (int i = 0; i < _mapHeight; i++)
            {
                for (int j = 0; j < _mapWidth; j++)
                {
                                        Vector2D t = _analysis.CalculateGradient(ref
_levelSetFunctionValues, j, i);
                                                gradientMagnitudes[i, j] =
_analysis.GetEuclideanMetricLength(ref t);
                }
            }

            Util.Util.SaveMatrix(ref _levelSetFunctionValues, "levelSet");
            Util.Util.SaveMatrix(ref gradientMagnitudes, "levelSetGradient");
                  using (Image<Gray, float> img = new Image<Gray, float>(_mapWidth,
_mapHeight))
            {
                for (int i = 0; i < _mapHeight; i++)
                {
                    for (int j = 0; j < _mapWidth; j++)
                    {
                        img.Data[i, j, 0] = (float) (gradientMagnitudes[i, j]);
                    }
                }
                img.Save(@"~/../../../dataset/segmented_gradient.jpg");
                Image<Gray, byte> img1 = img.Convert<Gray, byte>();

            }

            return points;
        }
```