Реализация нейронной сети Кохонена для анализа и классификации геометрических фигур

Условие

Написать программу моделирования нейронных сетей (НС) заданного типа и показать их работоспособность на практических примерах использования НС для указанной задачи. Параметры НС представлены в таблице 1.

Таблица 1. Параметры модели

Вход	Изображения на некотором фоне одной из 5 геометрических фигур
Выход	Какая фигура
Тип НС	сеть Кохонена

1 Постановка задачи, связанной с практическим применением НС

Компьютерное зрение на основе методов распознавания геометрической формы получило широкое распространие на производстве в таких областях как промышленный осмотр, идентификация, и автоматический контроль качества продукции. В задаче автоматической сборки, значительный объем информации о детали необходимо распознавать и классифицировать, а ее ориентация должна быть автоматически определена, прежде чем робот (манипулятор) сможет ухватиться за изделие или его часть. Также применяются методы распознавания формы для оптического распознавания символов, рукописного текста, а также медицинских изображений, и.т.д. [4].

Базовой задачей, предваряющей перечисленные выше является задача распознавания и классификации плоских геометрических фигур таких как треугольник, квадрат, пентагон, шестиугольник и круг (многоугольник с количеством сторон ~ 100).

2 Описание теоретической базы рассматриваемой модели НС

Нейронные сети Кохонена – класс нейронных сетей, основным элементом которых является слой Кохонена. В «базовой» версии слой Кохонена (рис.1) представляет собой некоторое число N параллельно действующих линейных элементов, имеющих одинаковое число входов M и получающих на свои входы один и тот же вектор входных сигналов $x = (x_1, ..., x_M)$. На выходе j-го линейного элемента, в таком случае, получаем сигнал:

$$D_j = w_{j0} + \sum_{i=1}^{M} w_{ji} x_i$$

где w_{ji} - весовой коэффициент i-го входа j-го линейного элемента (нейрона), i – номер входа, j – номер нейрона, $w_j 0$ – пороговый коэффициент.

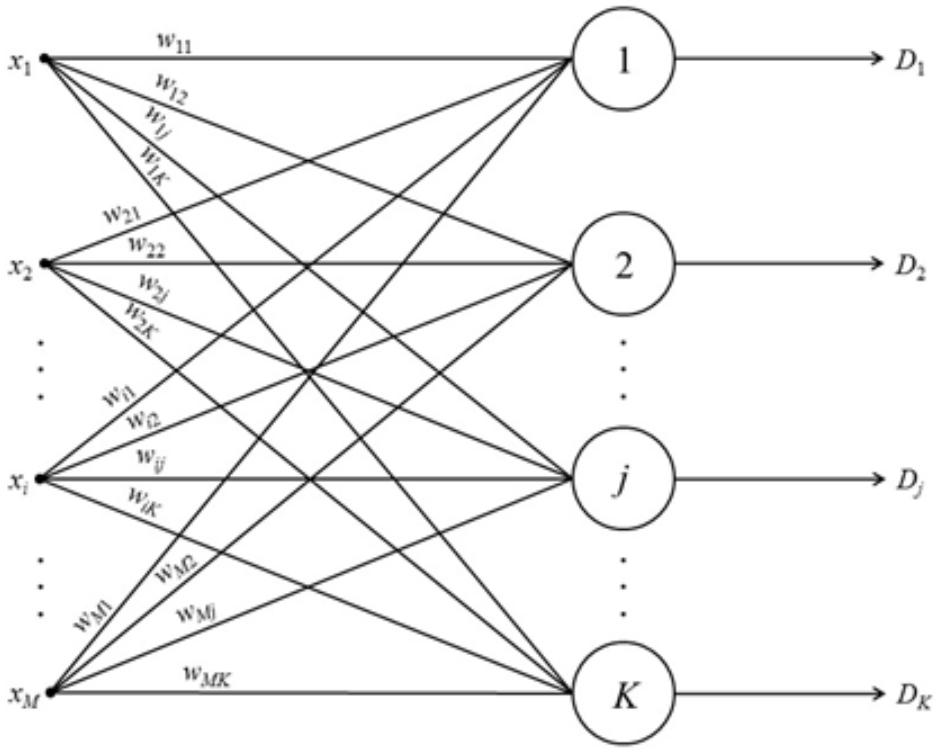

Рис. 1. Слой Кохонена ($w_j 0 = 0$)

После прохождения слоя линейных элементов сигналы посылаются на обработку по правилу «победитель забирает всё»: среди выходных сигналов выполняется поиск максимального (либо минимального) D_j:

$$j_{ok} = arg \max_j (D_j)$$

или

$$j_{ok} = arg \min_j (D_j)$$

Окончательно, на выходе сигнал с номером j_{ok} равен единице, остальные – нулю. Если же максимум одновременно достигается на нескольких j_{ok}, то все соответствующие сигналы принимают равными единице, либо равным единице по соглашению принимается только первый из сигналов в списке.

Нейронная сеть Кохонена представляет собой типичный пример нейросетевой архитектуры, обучающейся без учителя. Отсюда и перечень решаемых ими задач: кластеризация данных или прогнозирование свойств.

Кроме того, сети Кохонена могут использоваться с целью уменьшения размерности данных с минимальной потерей информации.

В рассматриваемой архитектуре сигнал распространяется от входов к выходам в прямом направлении, как, например, в FFNN-сети. Однако FFNN, как и многие другие архитектуры, обучается с учителем на выборках данных, включающих множество примеров, состоящих из соответствующих друг другу пар входных и выходных векторов. При этом выходные значения принимают самое непосредственное участие в настройке весовых коэффициентов. Так, при выполнении лабораторной работы №1 в системе MATLAB был реализован программный модуль обучения стандартной FFNN-сети (персептрона). Обучение предполагало тренировку сети с помощью метода обратного распространения ошибки

на примерах вида

$$(I_j; v_j)$$

где I_j — j-е изображение геометрической фигуры, v_j — соответствующий ему вектор-метка. Вектор $v - j$: *треугольник* – [1, 0, 0, 0, 0]; *квадрат* – [0, 1, 0, 0, 0]; *круг* – [0, 0, 1, 0, 0]; *пятиугольник* – [0, 0, 0, 1, 0]; *шестиугольник* – [0, 0, 0, 0, 1].

В нейронных сетях Кохонена выходные вектора в обучающей выборке могут быть, но могут и отсутствовать, и, в любом случае, они не принимают участия в процессе обучения. То есть выходы не используются в качестве ориентиров при коррекции синапсов. Именно поэтому данный принцип настройки нейронной сети называется самообучением (обучением без учителя).

Количество нейронов N делают равным количеству кластеров, среди которых происходит начальное распределение и последующее перераспределение обучающих примеров. Количество входных переменных нейронной сети равно числу признаков, характеризующих объект исследования и на основе которых происходит отнесение его к одному из кластеров.

Рассмотрим геометрическую интерпретацию слоя Кохонена. Пусть каждому j-му нейрону поставлена в соответствие точка

$$W_j = (w_{j1}, ..., w_{jM})$$

в M-мерном пространстве. Будем для каждого входного вектора

$$x = (x_1, ..., x_M)$$

вычислять его расстояния $\rho_j(x)$ до точек W_j для каждого нейрона), например, в евклидовой метрике. Примем $\|v\|$ за обозначение евклидовой длины вектора v, тогда:

$$\rho_j(x) = \|x - W_j\|^2 = \|W_j\|^2 - 2\sum_{i=1}^{M} w_{ji}x_i + \|x\|^2$$

Заметим, что последнее слагаемое $\|x\|^2$ одинаково для каждого j.

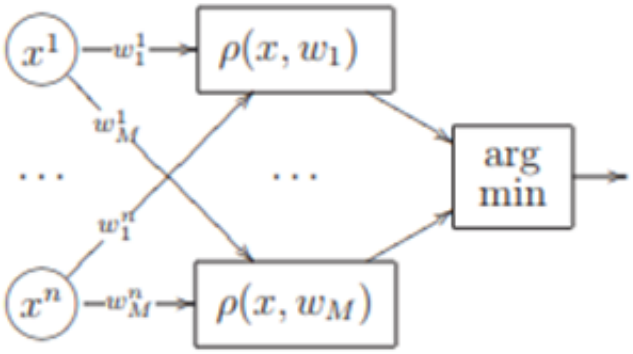

Рис. 2. Argmin ($w_{j0} = 0$)

По принципу «победитель забирает всё» найдём тот нейрон, для которого расстояние до вектора x минимально (argmin) (рис. 2).

$$j_o k = arg\min_j \rho_j(x) = arg\min_j \left(\|W_j\|^2 - 2\sum_{i=1}^{M} w_{ji}x_i \right) =$$

$$arg\min_j \left(\frac{1}{2}\|W_j\|^2 - \sum_{i=1}^{M} w_{ji}x_i \right) = arg\min_j \left(-\frac{1}{2}\|W_j\|^2 + \sum_{i=1}^{M} w_{ji}x_i \right)$$

$$= arg\min_j \left(w_{j0} + \sum_{i=1}^{M} w_{ji}x_i \right) = arg\min_j (D_j)$$

где $w_{j0} = -\frac{1}{2}\|W_j\|^2$

В случае если точки W_j изначально заданы, имеющееся M-мерное пространство разобьется на т.н. многогранники Вороного-Дирихле V_j (рис. 2) Каждый многогранник V_j состоит из точек, которые ближе к W_j, чем к другим W_k, где $k \neq j$.

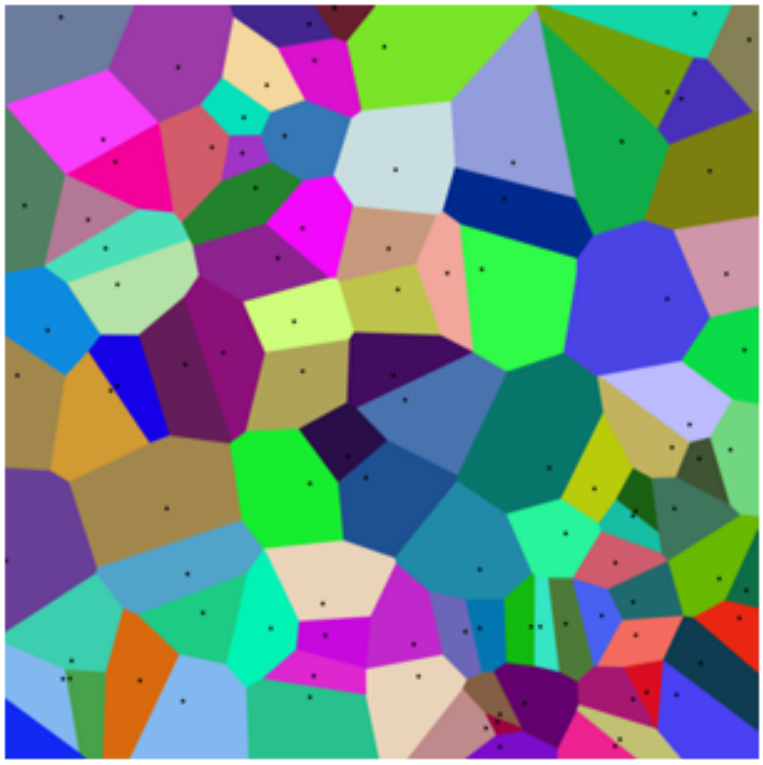

Рис. 3. Argmin ($w_j 0 = 0$)

Следует различать собственно самообучение и самоорганизацию нейронной сети Кохонена. При обычном *самообучении* сеть имеет строго фиксированную структуру, т. е. количество нейронов, не изменяющееся на протяжении всего жизненного цикла.
При *самоорганизации* сеть, напротив, не имеет постоянной структуры. В зависимости от найденного расстояния до нейрона-победителя либо этот нейрон используется для кластеризации примера, либо для поданного на входы примера создается новый кластер с соответствующими ему весовыми коэффициентами. Кроме того, в процессе самоорганизации структуры сети Кохонена отдельные нейроны могут исключаться из нее.

3 Описание разработанных ПМ и решение задачи

Для решения поставленной задачи распознавания геометрических фигур на фоне различных цветов с помощью нейронной сети Кохонена реализуем обучающую и тестовую выборки изображений. В качестве геометрических фигур рассмотрим правильные n–угольники. Для $n = 3$ — треугольник, $n = 4$ — квадрат, $n = 5$ — пятиугольник, $n = 6$ — шестиугольник и $n = 100$ — круг. (рис. 4)

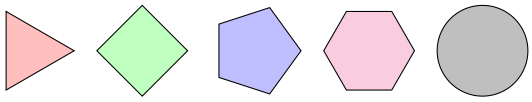

Рис. 4. N-угольники

Был реализован алгоритм, позволяющий получить изображения указанных геометрических фигур различных размеров и цветов. Предусмотрена возможность поворота фигуры на некоторый случайный угол.

Файлы `Matlab`**:**

```
gen_set_of_ngons.m
gen_ngon_image.m
run_gen_ngon.m
```

Как и в лабораторной работе №1 реализуем 2 выборки: «сложную» и «упрощённую».

Выборка № 1 ”Сложная”

Изображения размером 28×28 пикселей. Фигуры расположены произвольно относительно центра изображения, произведён поворот на случайный угол. Цвет фигуры и цвет фона также случайные.

Рис. 5. Примеры сгенерированных фигур

Выборка № 2 ”Упрощённая”

Содержит изображения фигур с фиксированными положением, размером и цветом. От изображения к изображению меняется только цвет фона. Данная выборка представляет собой упрощённый вариант выборки-1.

Выборка № 2 ”Упрощённая”

Вариация выборки-2 «упрощённой»: к существующим изображениям добавляются те же, но зашумленные гауссовским шумом до максимального уровня 0.5.

Все изображения переведём в оттенки серого и преобразуем в одномерный массив длиной 784 элемента.

Обучение

Перед применением к сгенерированным выборкам сети Кохонена, необходимо произвести нормализацию входных переменных, которая может быть выполнена в пределах [–1, 1] или [0, 1]. Выберем первый вариант, так как такая нормализация уже использовалась в лабораторной работе №2 перед применением сети Хопфилда.

Алгоритм обучения сети Кохонена включает этапы, состав которых зависит от типа структуры: постоянной (самообучающаяся сеть) или переменной (самоорганизующаяся сеть). В данной работе будем рассматривать только самообучающуюся сеть.

Для самообучения последовательно выполняются:

1. Задание структуры сети (кол-ва нейронов N в слое Кохонена).

2. Случайная инициализация весовых коэффициентов значениями, удовлетворяющими одному из следующих ограничений:

a. При нормализации исходной выборки в пределах [-1, 1]:

$$|w_{ij}| \leq \frac{1}{\sqrt{M}}$$

b. При нормализации исходной выборки в пределах [0, 1]:

$$0.5 - \frac{1}{\sqrt{M}} \leq w_{ij} \leq 0.5 + \frac{1}{\sqrt{M}}$$

M – количество входных переменных сети.

3. Подача на входы сети случайного обучающего примера текущей эпохи обучения и расчет евклидовых расстояний от входного вектора до центров всех кластеров:

$$R_j = \sqrt{\sum_{i=1} M\,(X_i - w_{ij})}.$$

4. По наименьшему из значений R_j выбирается нейрон-победитель j, в наибольшей степени близкий по значениям с входным вектором. Для выбранного нейрона (и только для него) выполняется коррекция весовых коэффициентов.

$$w_{ij}^{q+1} = w_{ij}^{q} + v \cdot \left(X_i - w_{ij}^{q}\right)$$

где v - скорость обучения.

5. Цикл повторяется с шага 3 до выполнения одного или нескольких условий окончания:

– исчерпано заданное предельное количество эпох обучения;
– не произошло значимого изменения весовых коэффициентов в пределах заданной точности на протяжении последней эпохи обучения;
– исчерпано заданное предельное физическое время обучения.

В MATLAB слой Кохонена создается следующим образом:

```
net = newc(PR,S,KLR)
```

здесь PR –[N (кол-во нейронов) x 2] матрица минимальных и максимальных значений входных переменных. В нашем случае это [-1, 1]. S – число нейронов (= N). KLR – Kohonen learning rate – скорость обучения по Кохонену (стандарт: 0.01).

Обучение производится с помощью:

```
net = train(net, P);
```

где P – подготовленная выборка данных.

Файлы `Matlab`:

`load_dataset.m`
`create_n_train_koh.m`
`sim.m`

В предыдущих лабораторных работах была рассмотрена возможность в качестве дополнительной предобработки изображений, перед подачей данных на вход нейронной сети для классификации, применять к имеющимся изображениям геометрических фигур т.н. Фурье-преобразование с целью получения 2D-спектра фигуры (далее совершая сдвиг по нулевой частоте для центрирования). Как показало исследование, при использовании FFNN-сети, процент распознанных фигур, имеющих произвольные площадь, поворот и цвет на изображении («сложные»), резко возрастает, если предварительно рассчитать по данному изображению его 2D-спектр. Однако воссоздать похожие результаты при использовании сети Кохонена не удалось. Процент распознавания на «сложной» выборке всего 3-4%. Сеть не способна по тем или иным причинам отнести изображения различных (разных) геометрических фигур из данной выборки (№1) к различным (также – разным) кластерам (даже после Фурье-обработки). На выборке №2, сеть Кохонена, как и предполагалось (как и FFNN, и сеть Хопфилда), выдает оличный процент распознавания, близкий к 100%.

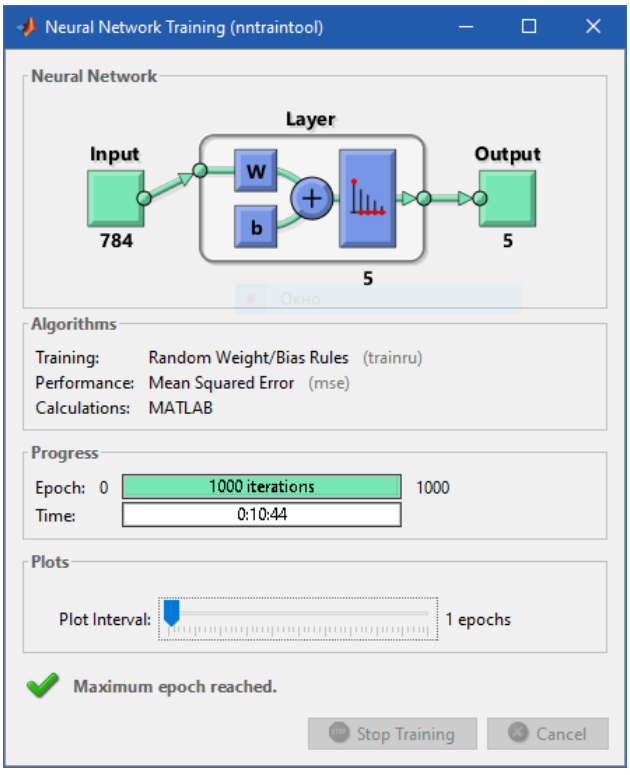

Рис. 6. Средства MATLAB. NNtrainTool

В данный момент автор обучает сеть Кохонена на примерах выборки №3.

4 Альтернативные способы решения

Альтернативными являются два подхода классификации фигур, которые во многом противоположны друг другу.

Первым подходом является подход на основе спектрального анализа [5]. Представление формы объекта на основе *Фурье дескрипторов* легко организовать в плане вычислений, а результаты будут устойчивы к внешнему шуму. Фурье дескрипторы получаются при помощи преобразования Фурье, применённого к *сигнатуре формы объекта*, границе объекта как одномерной функции.

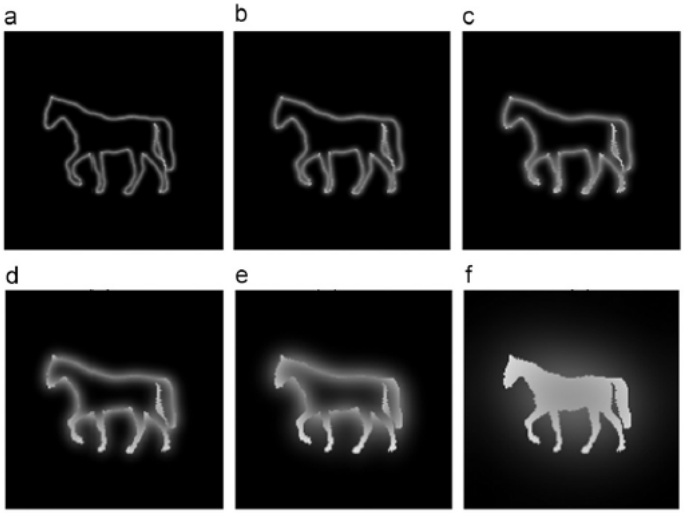

Рис. 7. Пример обработки изображения лошади [5]

Существуют и другие сигнатуры формы, например, расстояние от центра объекта (в пикселях) до остальных пикселей, кривизна границы и кумулятивный угол. Геометрические характеристики (инварианты) объекта определяются на этапе определения сигнатуры формы объекта, который может происходить как до Фурье преобразования, так и после. При этом, нижние частоты дескрипторов содержат информацию о форме объекта, а верхние частоты о деталях.

Вторым подходом является геометрический подход, который предполагает выделение контура [6]. Он основывается на разработке дескрипторов формы малого разрешения, которые могут быть устойчивы к поворотам, масштабированию и деформации объекта.

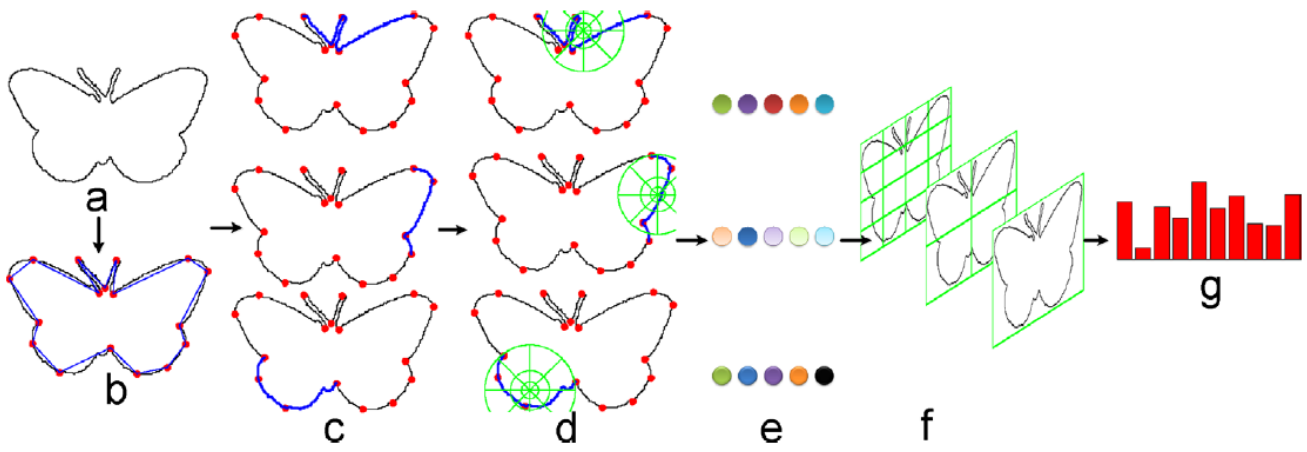

Рис. 8. Пример обработки изображения бабочки [6]

В таком подходе граница объекта разбивается на составные части (сегменты), каждая из которых далее описывается при помощи того или иного дескриптора. После чего для решения задачи классификации применяются методы машинного обучения, например, метод опорных векторов [7].

5 Области применения НС заданного типа

Сети финского ученого Кохонена применяются для решения самых разнообразных задач моделирования и прогнозирования, выявления наборов независимых признаков, поиска закономерностей в больших массивах данных, разработке компьютерных игр, квантизации цветов к их ограниченному числу индексов в цветовой палитре: при печати на принтере и ранее на ПК или же на приставках с дисплеем с пониженным числом цветов, для архиваторов или видео-кодеков, и т. д.

Список литературы

[1] Kohonen, T. (1989/1997/2001), Self-Organizing Maps, Berlin — New York: Springer-Verlag. First edition 1989, second edition 1997, third extended edition 2001, ISBN 0-387-51387-6, ISBN 3-540-67921-9

[2] Hecht-Nielsen, R. (1990), Neurocomputing, Reading, MA: Addison-Wesley, ISBN 0-201-09355-3.

[3] Cohen M.A., Grossberg S.G., Absolute stability of global pattern formation and parallel memory storage by compatitive neural networks. IEEE Transactions on Systems, Man and Cybernetics, Vol. 13, 1983, pp. 815–26. **DOI: :10.1109/TSMC.1983.6313075**

[4] Tung-Hsu Hou and Ming-Der Pern, A Computer Vision-Based Shape-Classification System Using Image Projection and a Neural Network. Int. J. Adv. Manuf. Technol., 15, 1999, pp. 843–850. **DOI: 10.1007/s001700050141**

[5] Cem Direkoğlu, Mark S. Nixon, Shape classification via image-based multiscale description. Pattern Recognition, 44, 2011, pp. 2134–2146. **DOI: 10.1016/j.patcog.2011.02.016**

[6] Xinggang Wang, Bin Feng, et.al., Bag of contour fragments for robust shape classification. Pattern Recognition, 47, 2014, pp. 2116–2125. **DOI: 10.1016/j.patcog.2013.12.008**

[7] К.В. Воронцов. Лекции по методу опорных векторов. **URL:** http://www.ccas.ru/voron/download/SVM.pdf

Тексты программ

`gen_ngon_image.m`

```
function out = gen_ngon_image(n)

%% Suppose that domain is [0 100]x[0 100] square
dom = [0 100 0 100];

% Center of n-polygon
xC = randi([30 70],1,1);
yC = randi([30 70],1,1);
center = [xC yC];
nS = n; % number of sides of n-gon
th = linspace(0, 2*pi, nS + 1);
% Rotate the shape by subtracting an offset.
rot = randi([-10 10],1,1);
th = th - pi/rot;
R = randi([15 35],1,1);
x = R * cos(th) + center(1);
y = R * sin(th) + center(1);

%% Show image
figure_color      = 0.5 + 0.5*rand(1,3);
background_color  = 0.2*rand(1,3);

h = fill(x, y, figure_color);
set(h,'edgecolor',figure_color);
ax = gca;
set(ax,'xtick',[]); set(ax,'ytick',[]);

ax.XColor = background_color; ax.YColor = background_color ;
axis square; axis(dom);
set(ax,'Color',background_color )

img = getframe(gca);
out = img;
```

`gen_set_of_ngons.m`

```
function gen_set_of_ngons(number_of_images, nS, folder, baseFileName, type, ...
    speedup)

if strcmp(type,'simplified')
if strcmp(speedup,'yes')
parfor k = 1:number_of_images
img = gen_ngon_image_simplified(nS);
fullFileName = [folder  baseFileName '-' int2str(k)  '.png'];
save_image(folder,baseFileName,fullFileName,img.cdata);
disp([int2str(k) '/' int2str(number_of_images)])
end %for k
else
for k = 1:number_of_images
img = gen_ngon_image_simplified(nS);
fullFileName = [folder  baseFileName '-' int2str(k)  '.png'];
save_image(folder,baseFileName,fullFileName,img.cdata);
disp([int2str(k) '/' int2str(number_of_images)])
```

```
end %for k
end %if
else
if strcmp(speedup,'yes')
parfor k = 1:number_of_images
img = gen_ngon_image(nS);
fullFileName = [folder  baseFileName '-' int2str(k)  '.png'];
save_image(folder,baseFileName,fullFileName,img.cdata);
disp([int2str(k) '/' int2str(number_of_images)])
end %for k
else
for k = 1:number_of_images
img = gen_ngon_image(nS);
fullFileName = [folder  baseFileName '-' int2str(k)  '.png'];
save_image(folder,baseFileName,fullFileName,img.cdata);
disp([int2str(k) '/' int2str(number_of_images)])
end %for k
end %if
end
```

`run_gen_ngons.m`

```
clear all; close all; clc;

% Get path
CurrentFolder = pwd;


% Generate ngon image
number_of_images = 20000;

% Generate set of 3-gons
nS = 100;
type = 'simplified';
speedup = 'yes';
folder = [CurrentFolder '\_img_\' int2str(nS) '-gone-' type '\'];
mkdir_if_not_exist(folder);
baseFileName = [int2str(nS) '-gone'];
gen_set_of_ngons(number_of_images, nS, folder, baseFileName, type, speedup);
% Compress folder with images
zippedfiles = zip([CurrentFolder '\_img_\' int2str(nS) '-gone-' type ...
    '.zip'], folder);
% Remove folder with images
rmdir(['_img_/' int2str(nS) '-gone-' type],'s')
```

`load_dataset.m`

```
triangleFolder = '_img_/3-gone/';       % label: 1
rectangleFolder = '_img_/4-gone/';      % label: 2
circleFolder = '_img_/100-gone/';       % label: 3
pentagonFolder = '_img_/5-gone/';       % label: 4
hexagonFolder = '_img_/6-gone/';        % label: 5

% getting filelists
dirData = dir(triangleFolder);
dirIndex = [dirData.isdir];
fileListTriangles = {dirData(~dirIndex).name}';
dirData = dir(rectangleFolder);
```

```
dirIndex = [dirData.isdir];
fileListRectangles = {dirData(¬dirIndex).name}';
dirData = dir(circleFolder);
dirIndex = [dirData.isdir];
fileListCircles = {dirData(¬dirIndex).name}';
dirData = dir(pentagonFolder);
dirIndex = [dirData.isdir];
fileListPentagones = {dirData(¬dirIndex).name}';
dirData = dir(hexagonFolder);
dirIndex = [dirData.isdir];
fileListHexagones = {dirData(¬dirIndex).name}';

N = 1;
hvP_size = 28;
P_size = hvP_size * hvP_size;

trianglesI = zeros(P_size, N);
rectanglesI = zeros(P_size, N);
circlesI = zeros(P_size, N);
pentagonesI = zeros(P_size, N);
hexagonesI = zeros(P_size, N);
trianglesFFT = zeros(P_size, N);
rectanglesFFT = zeros(P_size, N);
circlesFFT = zeros(P_size, N);
pentagonesFFT = zeros(P_size, N);
hexagonesFFT = zeros(P_size, N);

for i = 1:N
fpath = strcat(triangleFolder, fileListTriangles{i});
disp(fpath);
I = imread(fpath);
I = imresize(I, [hvP_size hvP_size]);
II = reshape(I(:,:,1), [P_size, 1]);
trianglesI(:, i) = II;
I = fft2(I);
I = reshape(abs(fftshift(I(:,:,1))), [P_size,1]);
trianglesFFT(:, i) = I;

fpath = strcat(rectangleFolder, fileListRectangles{i});
disp(fpath);
I = imread(fpath);
I = imresize(I, [hvP_size hvP_size]);
II = reshape(I(:,:,1), [P_size, 1]);
rectanglesI(:, i) = II;
I = fft2(I);
I = reshape(abs(fftshift(I(:,:,1))), [P_size,1]);
rectanglesFFT(:, i) = I;

fpath = strcat(circleFolder, fileListCircles{i});
disp(fpath);
I = imread(fpath);
I = imresize(I, [hvP_size hvP_size]);
II = reshape(I(:,:,1), [P_size, 1]);
circlesI(:, i) = II;
I = fft2(I);
I = reshape(abs(fftshift(I(:,:,1))), [P_size,1]);
circlesFFT(:, i) = I;

fpath = strcat(pentagonFolder, fileListPentagones{i});
```

```
71  disp(fpath);
72  I = imread(fpath);
73  I = imresize(I, [hvP_size hvP_size]);
74  II = reshape(I(:,:,1), [P_size, 1]);
75  pentagonesI(:, i) = II;
76  I = fft2(I);
77  I = reshape(abs(fftshift(I(:,:,1))), [P_size,1]);
78  pentagonesFFT(:, i) = I;
79
80  fpath = strcat(hexagonFolder, fileListHexagones{i});
81  disp(fpath);
82  I = imread(fpath);
83  I = imresize(I, [hvP_size hvP_size]);
84  II = reshape(I(:,:,1), [P_size, 1]);
85  hexagonesI(:, i) = II;
86  I = fft2(I);
87  I = reshape(abs(fftshift(I(:,:,1))), [P_size,1]);
88  hexagonesFFT(:, i) = I;
89  end
90
91  N5 = 5 * N;
92  DataSet_Images_Total = zeros(1, P_size, N5);
93  DataSet_FFTs_Total = zeros(1, P_size, N5);
94
95  % Total
96  k = 1;
97  for i = 1:N
98  DataSet_Images_Total(1,:,k) = trianglesI(:, i);
99  DataSet_FFTs_Total(1,:,k) = trianglesFFT(:, i);
100 k = k + 1;
101
102 DataSet_Images_Total(1,:,k) = rectanglesI(:, i);
103 DataSet_FFTs_Total(1,:,k) = rectanglesFFT(:, i);
104 k = k + 1;
105
106 DataSet_Images_Total(1,:,k) = circlesI(:, i);
107 DataSet_FFTs_Total(1,:,k) = circlesFFT(:, i);
108 k = k + 1;
109
110 DataSet_Images_Total(1,:,k) = pentagonesI(:, i);
111 DataSet_FFTs_Total(1,:,k) = pentagonesFFT(:, i);
112 k = k + 1;
113
114 DataSet_Images_Total(1,:,k) = hexagonesI(:, i);
115 DataSet_FFTs_Total(1,:,k) = hexagonesFFT(:, i);
116 k = k + 1;
117 end
118
119 Images = squeeze(DataSet_Images_Total(1, :, :));
120 FFTs = squeeze(DataSet_FFTs_Total(1, :, :));
121 for i = 1:P_size
122 for j = 1:N5
123 Images(i, j) = round(Images(i, j)/256);
124 if (Images(i, j) == 0)
125 Images(i, j) = -1;
126 end
127 if (Images(i, j) > 1)
128 Images(i, j) = 1;
129 end
```

```
130 FFTs(i, j) = round(FFTs(i, j)/256);
131 if (FFTs(i, j) == 0)
132 FFTs(i, j) = -1;
133 end
134 if (FFTs(i, j) > 1)
135 FFTs(i, j) = 1;
136 end
137 end
138 end
139
140 I = reshape(Images(:, 5), [hvP_size, hvP_size]);
141 imagesc(I);
142 % I = reshape(FFTs(:, 5), [hvP_size, hvP_size]);
143 % imagesc(I);
```

`create_n_train_koh.m`

```
1  L = zeros(784, 2);
2  L(:, 2) = ones(784, 1);
3  %
4  net = newc(L, 5);
5
6  P = squeeze(DataSet_Images_Total(1, :, :));
7  % P = squeeze(DataSet_FFTs_Total(1, :, :));
8
9  % L = zeros(112, 2);
10 % L(:, 2) = ones(112, 1);
11
12 % net = newc(L, 5);
13 %
14 % P = squeeze(DataSet_Features_Total(1, :, :));
15
16
17 net = train(net, P);
```

`sim.m`

```
1 Y = sim(net, P);
2 Yc = vec2ind(Y);
3
4 disp(Yc);
```

Содержание