Реализация нейронной сети Хопфилда 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 Хопфилда
`Ns`	Число элементов в скрытом слое

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. [3].

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

Сеть Хопфилда представляет собой однослойную полносвязную нейронную сеть с симметричной матрицей связей [1]. Каждый нейрон может принимать как на входе, так и на выходе лишь одно из двух состояний $\{1, -1\}$. Рассмотрим схему сети Хопфилда с тремя нейронами, представленную на рис. 1. Выход каждого из нейронов замыкается (обратная связь) на вход всех остальных нейронов сети.

Пусть W есть матрица связей сети Хопфилда с `Ns` нейронами, тогда

$$W = \begin{pmatrix} 0 & \dots & w_{1\mathtt{Ns}} \\ \vdots & \ddots & \vdots \\ w_{\mathtt{Ns}1} & \dots & 0 \end{pmatrix}, \quad w_{1\mathtt{Ns}} = w_{\mathtt{Ns}1}.$$

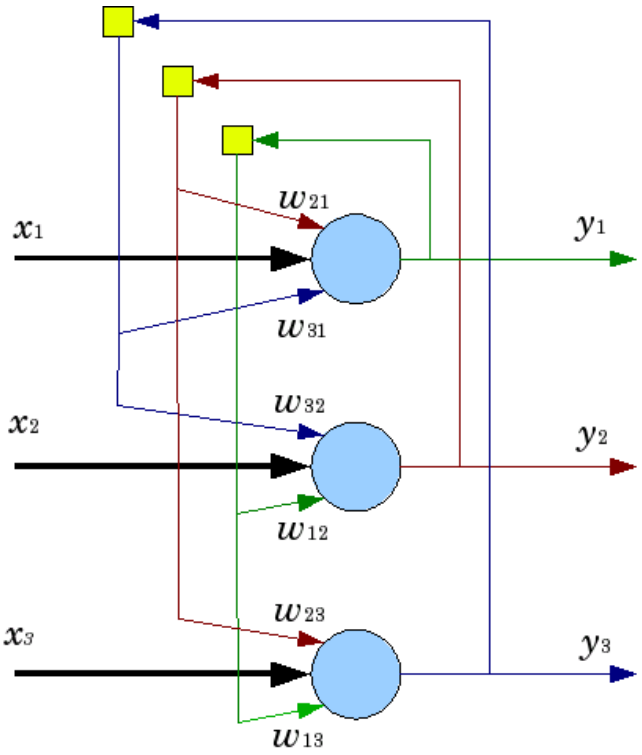


Рис. 1. Схема сети Хопфилда

Построенная таким образом нейронная сеть обладает свойством конвергенции, т.е. конечным (мощностью M) множеством положений равновесия, к которым сеть будет сходиться в своей динамике. Под динамикой в данном случае будем иметь в виду итеративный (например, $I : 1 \to 500$) процесс преобразования вектора входных значений X_ξ размерности `Ns` (`Ns` равно общему числу нейронов) к некоторому выходному вектору Y_ξ той же размерности. Причём, если X_i – положение равновесия, то

$$X_i = WX_i. \tag{1}$$

В случае, когда $X_\xi = X_i - \varepsilon$ имеем

$$WX_\xi \to Y_\xi = X_i.$$

Для сетей с обратными связями симметричность и нули на главной диагонали матрицы связей представляют собой необходимое условие [1] существования устойчивых положений равновесия (некое положение равновесия называют устойчивым, если рассматриваемая система, оказавшись в этом положении, стремится в нём и остаться). В случае сетей Хопфилда достаточным условием устойчивости сети будет т.н. асинхронный (в отличии от синхронного) режим работы. При соблюдении необходимого, но не достаточного условий устойчивости, возможен переход к динамическому аттрактору: ситуации, при которой сеть бесконечно (т.е. с каждой новой итерацией) переключается между двумя своими разными состояниями.

Отличие синхронного и асинхронного режимов работы заключается в способе обсчета новых состояний нейронов сети. При синхронном режиме нейроны просматриваются последовательно, однако их текущие состояния запоминаются отдельно и не меняются до тех пор, пока не будут пройдены все нейроны сети. В асинхронном режиме текущее состояние заменяется новым ещё до прохода по всем нейронам.

Процесс обучения сети Хопфилда заключается в построении такой матрицы связей W, чтобы все запоминаемые вектора–образы X_i оказались устойчивыми положениями равновесия (1). Если все запоминаемые векторы имеют бинарный вид, то матрица W может быть найдена вычислением внешнего произведения каждого запоминаемого вектора с самим собой и суммированием матриц, полученных таким образом в виде

$$W = \frac{1}{\mathtt{Ns}} \sum_i X_i X_i^T.$$

Алгоритм обучения сети Хопфилда существенно отличается от таких классических алгоритмов обучения перцептронов, как метод коррекции ошибки или метод обратного распространения ошибки. Отличие заключается в том, что вместо последовательного приближения к нужному состоянию с вычислением ошибок, все коэффициенты матрицы рассчитываются по одной формуле, за один цикл, после чего сеть сразу готова к работе.

3 Описание разработанных ПМ и решение задачи

Были рассмотрены четыре геометрические фигуры, являющиеся правильными выпуклыми $n-$угольниками: треугольник, квадрат, пятиугольник и шестиугольник. В качестве пятой фигуры был рассмотрен правильный 100−угольник (аналог круга). См. рис. 2.

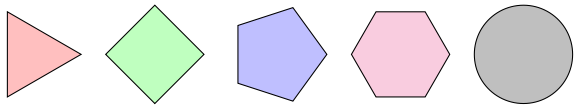


Рис. 2. Геометрические фигуры, на которых обучалась НС

Сформированы обучающие и соответствующие им тестовые выборки.

Рис. 3. Примеры сгенерированных фигур

Выборка №1 содержит изображения пяти указанных геометрических фигур, с фиксированными положением, размером и цветом. От изображения к изображению меняется только цвет фона.

Выборка №2 содержит изображения с фигурами разной площади и цвета заливки. Введён случайный угол поворота фигуры. Цвет фона изображения также меняется.

Рис. 4. Зашумленный пятиугольник

Выборка №3 представляет собой выборку №1 с той разницей, что на некоторые изображения наложен гауссовский шум с максимальным уровнем 0.5. Пример изображения представлен на рис. 4.

Файлы `Matlab`:

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

Поскольку сеть Хопфилда может работать лишь с массивами бинарных данных, имеющиеся изображения необходимо приводить к соответствующему виду. Для этого переведём изображение (изначально 3 канала)

$$R \in [0,\, 255],\ G \in [0,\, 255],\ B \in [0,\, 255]$$

в оттенки серого

$$R = G = B = x,$$

преобразуем его в одномерный 1D – массив. Далее приведём полученные значения к диапазону от -1 до 1. Все имеющиеся значения округлим по следующему правилу:
значения от -1 до 0 приравняем к -1,
значения от 0 до 1 приравняем к 1.

На рис. 5 показан результат перечисленных действий в применении к изображению ромба (квадрата) (без преобразования к одномерному массиву).

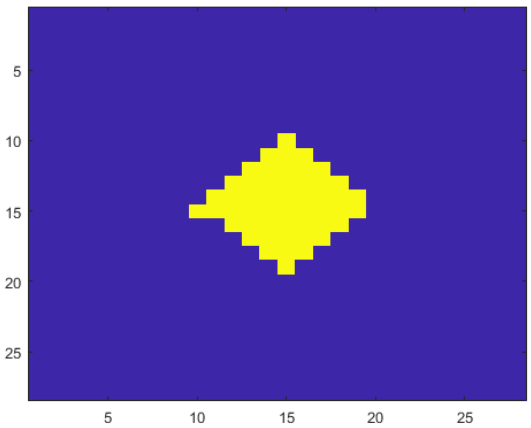


Рис. 5. Ромб (квадрат)

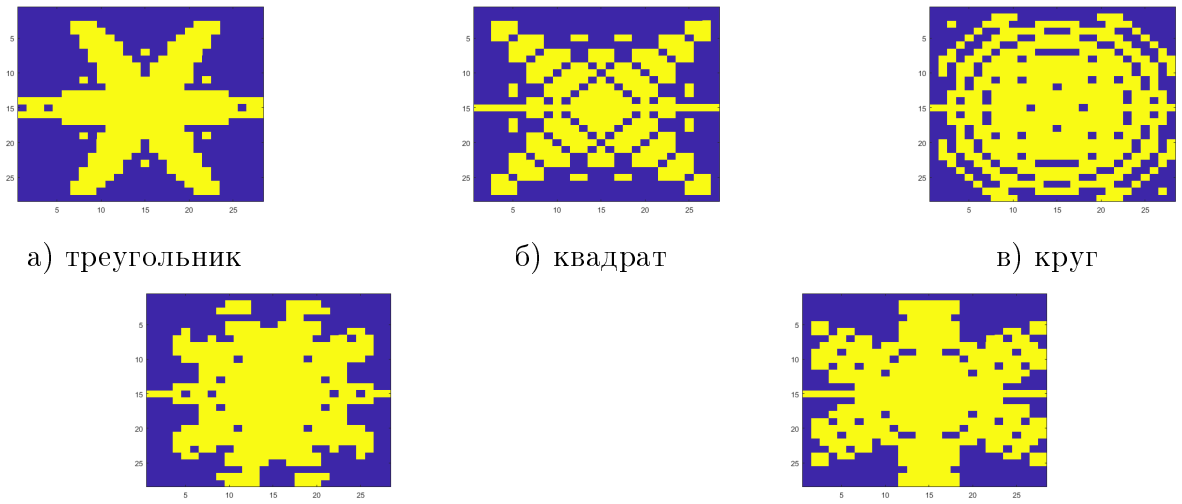
Указанным выше способом сформируем пять примеров–эталонов для всех имеющихся фигур и обучим на таких примерах сеть Хопфилда. Количество нейронов в сети будет варьироваться в зависимости от размера изображения (высота × ширина попиксельно). В данной работе использовались изображения 28×28 и 50×50. Вход–выход: 784—784 и 2500—2500 соответственно.

Сконфигурированная и обученная таким образом сеть показывает близкие к 100% результаты на выборках № 1 и № 3. Каждое изображение из тестового набора изображений (500 примеров) было верно проассоциировано с эталоном. Действительно, сети Хопфилда удачно применяют наряду со множеством существующих шумоподавляющих фильтров, рис.6.

Рис. 6. Пример эталонного изображения и изображения с шумом

На "сложной" выборке № 2 распознавание едва превысило $5 \sim 12\%$.

В рамках эксперимента и с целью улучшения качества распознавания фигур, была предпринята попытка обучить сеть Хопфилда не на самих изображениях эталонных фигур, а на их спектрах Фурье. Приведение исходных изображений к их 2D–спектрам решает проблему произвольного положения фигуры относительно центра изображения в выборке № 2: достаточно сместить полученный Фурье–спектр по нулевой частоте. Форма 2D–спектра определённой фигуры меняется в зависимости от поворота этой фигуры: некоторые линии утолщаются и немного сдвигаются, другие – наоборот. Однако формы спектров одних и тех же фигур продолжают оставаться визуально схожими, тогда как для различных фигур различия формы сохраняются.

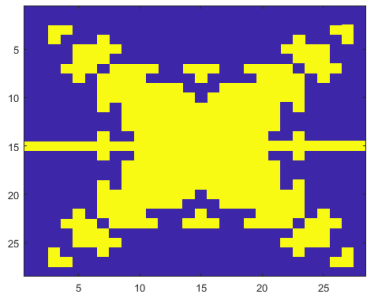


а) треугольник б) квадрат в) круг

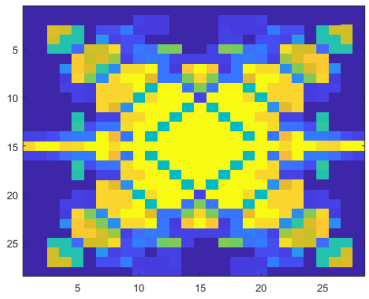
г) пятиугольник д) шестиугольник

Рис. 7. Спектры фигур из множества эталонных примеров

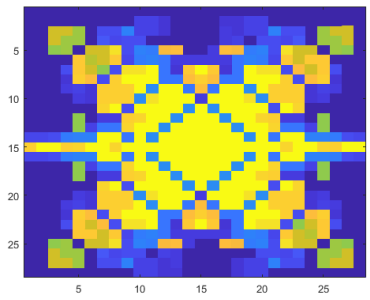
После обучения сети и её проверки на тестовых наборах изображений процент распознавания вырос всего до 30%. Сеть Хопфилда в динамике представлена на рис. 8.



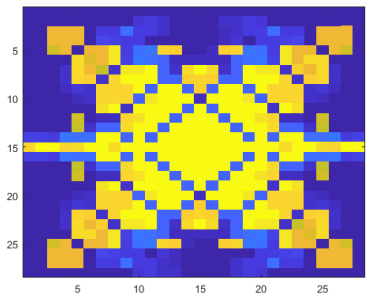
а) тестовый пример



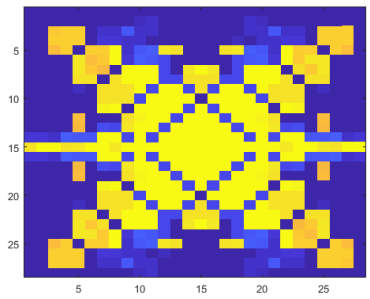
б) итерация 1



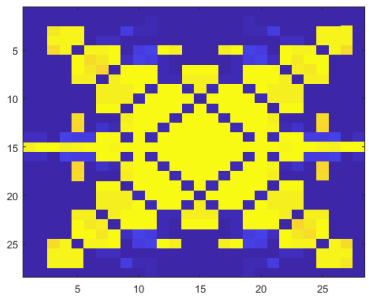
в) итерация 2



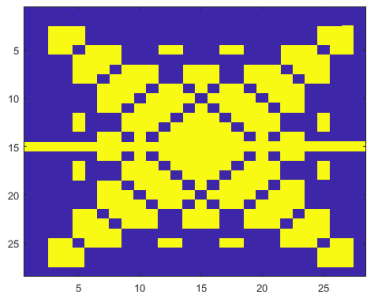
г) итерация 3



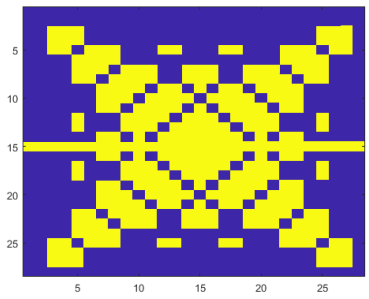
д) итерация 4



е) итерация 5



ё) итерация 6



ё) итерация 7

Рис. 8. Итерационный процесс для сети Хопфилда

Файлы `Matlab`**:**

`load_dataset.m`
`load_testdataset.m`
`create_net.m`
`test_net.m`

4 Альтернативные способы решения

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

Первым подходом является подход на основе спектрального анализа [4]. 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.

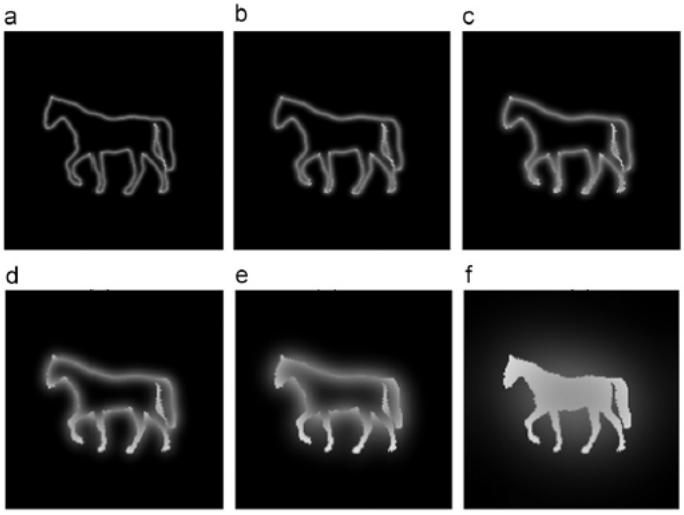


Рис. 9. Пример обработки изображения лошади [4]

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 [5]. Он основывается на разработке дескрипторов формы малого разрешения, которые могут быть устойчивы к поворотам, масштабированию и деформации объекта.

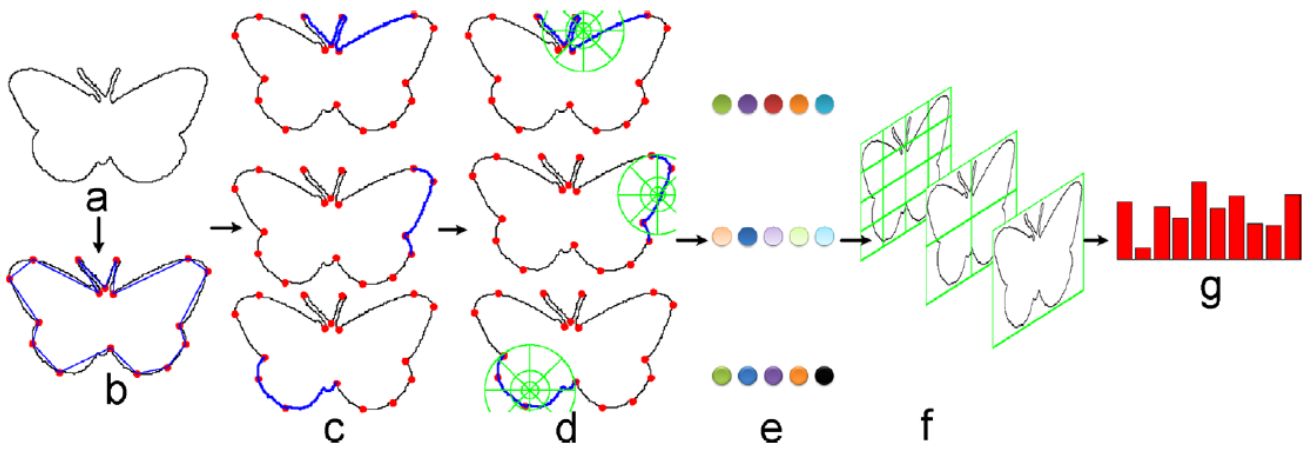


Рис. 10. Пример обработки изображения бабочки [5]

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 [6].

5 Области применения НС заданного типа

Ассоциативность памяти нейронной сети Хопфилда является не единственным ее прктическим свойством. Другим важным свойством этой архитектуры является уменьшение ее функции Ляпунова в процессе нейродинамики. Отсюда, сеть Хопфилда можно рассматривать как алгоритм оптимизации целевой функции в форме энергии сети.

Класс целевых функций, которые могут быть минимизированы нейронной сетью достаточно широк: в него попадают все билинейные и квадратичные формы с симметричными матрицами.

Сюда относятся такие традиционные задачи, как дифференциальные уравнения в вариационной постановке; задачи линейной алгебры и системы нелинейных алгебраических уравнений, где решение ищется в форме минимизации невязки, и другие.

Помимо того, что сеть Хопфилда может быть использована как автоассоциативная память, и для решения некоторых задач оптимизации, она также может быть использована как фильтр.

Список литературы

[1] J.J. Hopfield, Neural Networks and Physical Systems with Emergent Collective Computational Abilities. Proc. NatL Acad. Sci. USA, Vol. 79, 1982, pp. 2554–2558. **DOI: :10.1073/pnas.79.8.2554**

[2] 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**

[3] 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**

[4] 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**

[5] 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**

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

Тексты программ

`test_net.m`

```
steps = 500;
[Y_img, Pf1, Af1, E1, perf1] = sim(net_img, {steps}, [], TestImages(:, 5));
% [Y_fft, Pf2, Af2, E2, perf2] = sim(net_fft, {steps}, [], TestFFTs(:, 2));

imagesc(reshape(Y_img{1, steps}, [hvP_size, hvP_size]));
% imagesc(reshape(Y_fft{1, steps}, [hvP_size, hvP_size]));
```

`create_net.m`

```
% newhop - Hopfield network create
net_img = newhop(Images);
net_fft = newhop(FFTs);
```

`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}';
...

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

trianglesI = zeros(P_size, N);
...
trianglesFFT = 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;

  ...
end

N5 = 5 * N;
DataSet_Images_Total = zeros(1, P_size, N5);
DataSet_FFTs_Total = zeros(1, P_size, N5);

%% Total
k = 1;
for i = 1:N
  DataSet_Images_Total(1,:,k) = trianglesI(:, i);
  DataSet_FFTs_Total(1,:,k) = trianglesFFT(:, i);
  k = k + 1;

  DataSet_Images_Total(1,:,k) = rectanglesI(:, i);
  DataSet_FFTs_Total(1,:,k) = rectanglesFFT(:, i);
  k = k + 1;

  ...
end

Images = squeeze(DataSet_Images_Total(1, :, :));
FFTs = squeeze(DataSet_FFTs_Total(1, :, :));
for i = 1:P_size
  for j = 1:N5
    Images(i, j) = round(Images(i, j)/256);
    if (Images(i, j) == 0)
      Images(i, j) = -1;
    end
    if (Images(i, j) > 1)
      Images(i, j) = 1;
    end
    FFTs(i, j) = round(FFTs(i, j)/256);
    if (FFTs(i, j) == 0)
      FFTs(i, j) = -1;
    end
    if (FFTs(i, j) > 1)
      FFTs(i, j) = 1;
    end
  end
end

I = reshape(Images(:, 5), [hvP_size, hvP_size]);
imagesc(I);
%% I = reshape(FFTs(:, 5), [hvP_size, hvP_size]);
%% imagesc(I);
%
```

`load_testdataset.m`

```
triangleFolder = '_img_/TEST/3-gone/';      % label: 1
rectangleFolder = '_img_/TEST/4-gone/';     % label: 2
```

```
circleFolder = '_img_/TEST/100-gone/';        % label: 3
pentagonFolder = '_img_/TEST/5-gone/';        % label: 4
hexagonFolder = '_img_/TEST/6-gone/';         % label: 5

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

N = 20;

trianglesI = zeros(P_size, N);
...
trianglesFFT = 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;

...
end

N5 = 5 * N;
TestDataSet_Images_Total = zeros(1, P_size, N5);
TestDataSet_FFTs_Total = zeros(1, P_size, N5);

% Total
k = 1;
for i = 1:N
TestDataSet_Images_Total(1,:,k) = trianglesI(:, i);
TestDataSet_FFTs_Total(1,:,k) = trianglesFFT(:, i);
k = k + 1;

TestDataSet_Images_Total(1,:,k) = rectanglesI(:, i);
TestDataSet_FFTs_Total(1,:,k) = rectanglesFFT(:, i);
k = k + 1;

...
end

```

```
62 TestImages = squeeze(TestDataSet_Images_Total(1, :, :));
63 TestFFTs = squeeze(TestDataSet_FFTs_Total(1, :, :));
64
65 for i = 1:P_size
66 for j = 1:N5
67 TestImages(i, j) = round(TestImages(i, j)/256);
68 if (TestImages(i, j) == 0)
69 TestImages(i, j) = -1;
70 end
71 if (TestImages(i, j) > 1)
72 TestImages(i, j) = 1;
73 end
74 TestFFTs(i, j) = round(TestFFTs(i, j)/256);
75 if (TestFFTs(i, j) == 0)
76 TestFFTs(i, j) = -1;
77 end
78 if (TestFFTs(i, j) > 1)
79 TestFFTs(i, j) = 1;
80 end
81 end
82 end
83
84 I = reshape(TestImages(:, 1), [hvP_size, hvP_size]);
85 imagesc(I);
86 % I = reshape(TestFFTs(:, 2), [hvP_size, hvP_size]);
87 % imagesc(I);
```

Содержание