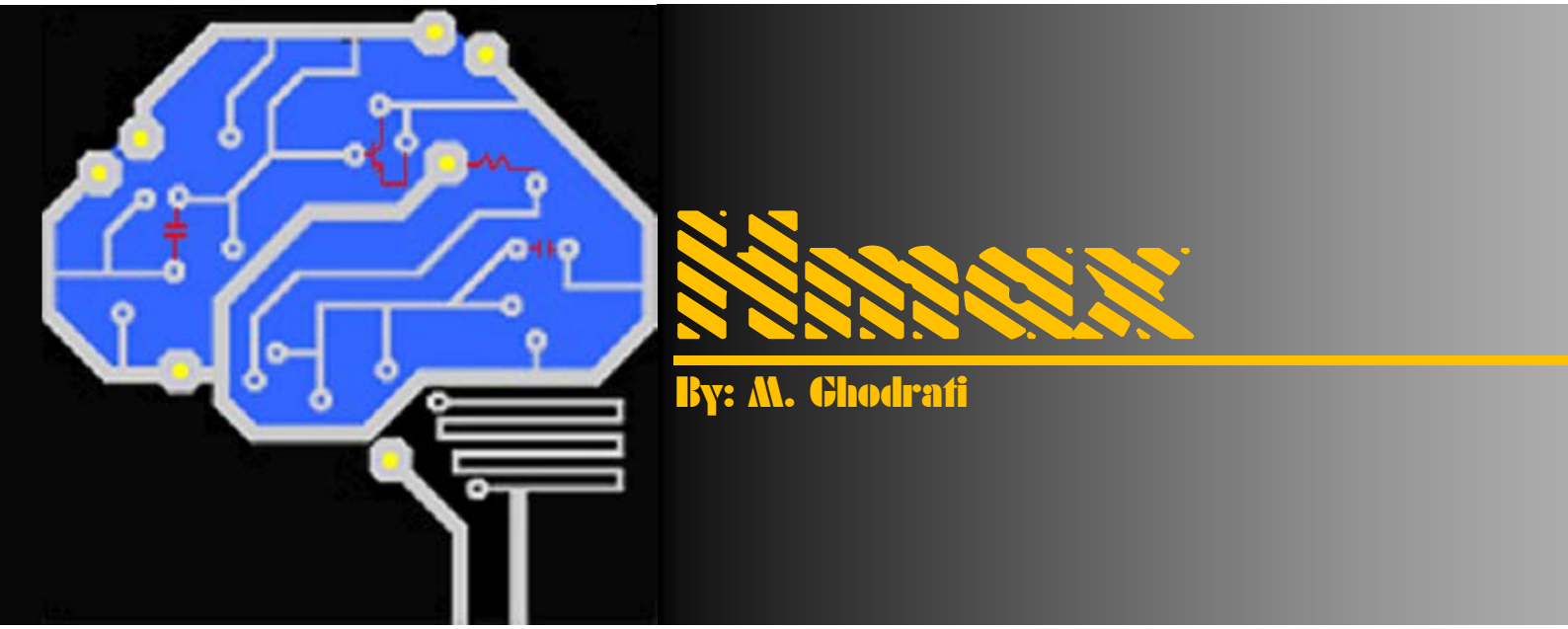


Masoud Ghodrati

Hmax Model

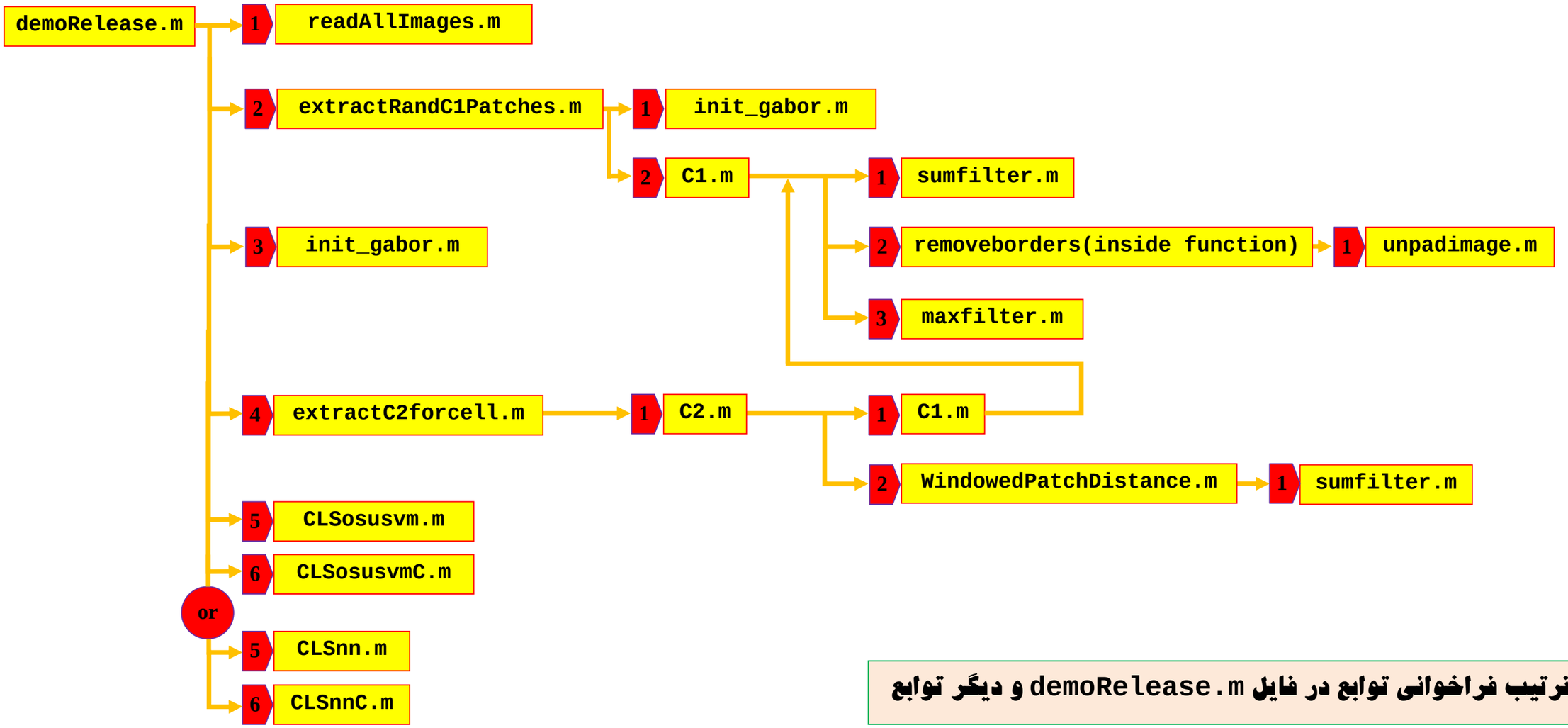
گزارش دوم مطالعه و بررسی نرم‌افزار

استاد: جناب آقای دکتر ابراهیم‌پور

Software Studying Report

2010-12-16

Masoud Ghodrati | ***مسعود قدرتی***



ترتیب فراخوانی توابع در فایل demoRelease.m و دیگر توابع

فهرست

مقدمه

برنامه شبیه‌سازی سیستم شناسایی شئ شامل ۱۵ تابع و یک فایل اصلی `demoRelease.m` می‌باشد. در این فایل تمامی توابع فراخوانی و اجرا می‌شوند.

توابع استفاده شده در پیاده‌سازی مدل

۱. فایل اصلی یا `demoRelease.m`
۲. توابع پیاده‌سازی لایه‌های مدل
- `C1.m` این تابع لایه‌های S1 و C1 مدل استاندارد را پیاده‌سازی می‌نماید.
- `C2.m` این تابع لایه های S2 و C2 مدل استاندارد را پیاده‌سازی می‌نماید.

۳. توابع طبقه‌بندی کننده
- `CLSnn.m` تابع طبقه‌بندی کننده با استفاده از روش نزدیک‌ترین همسایه (Nearest Neighbor) برای داده‌های آموزش.
- `CLSnnC.m` تابع طبقه‌بندی کننده با استفاده از روش نزدیک‌ترین همسایه (Nearest Neighbor) برای داده‌های تست.
- `CLSosusvm.m` تابع طبقه‌بندی کننده با استفاده از روش SVM برای داده‌های آموزش.
- `CLSosusvm.C` تابع طبقه‌بندی کننده با استفاده از روش SVM برای داده‌های تست.

۴. توابع استفاده شده دیگر
- `extractC2forcell.m` : استخراج C2 برای تمامی تصاویر در یک سلول با به‌کارگیری تمامی Patchها.
- `extractRandC1Patches.m` : استخراج Patchهای تصاویر برای داده‌های `train_set.pos` .
- `init_gabor.m` : فیلتر گابور.
- `maxfilter.m` : اعمال max محلی در یک تصویر (local maximum).
- `padimage.m` : اضافه فریم صفر حول تصویر(یک قاب صفر).
- `readAllImages.m` : خواندن تمامی تصاویر و ذخیره آنها در یک سلول.
- `sumfilter.m` : اعمال local sum برای یک تصویر.
- `unpadimage.m` : برش اطراف تصویر.
- `WindowedPatchDistance.m` : اسکن تصویر و جستجو برای یافتن بهترین تطابق برای یک Patch.

ذکر چند نکته قبل از اجرای برنامه دارای اهمیت می‌باشد.

۱. قبل از اجراء برنامه، مسیر یا directory تصاویر را در صورت لزوم تغییر دهید و آدرس دقیق محل ذخیره شدن تصاویر را وارد نماید (در مورد این قسمت در بخش‌های بعد توضیح داده می‌شود).
۲. در صورت تمایل به استفاده از طبقه‌بندی کننده SVM باید فایل‌ها و کدهای مربوط به آن را از `http://www.ece.osu.edu/~maj/osu_svm` دانلود نموده و سپس نصب نمایید (در نرم‌افزارMATLAB) این کار با copy کردن فایل دانلود شده در یک مسیر و با استفاده از دستور addpath انجام می‌گردد.
۳. تصاویر ورودی باید به‌صورت gray scale باشند و همچنین بهتر است اندازه طول و عرض تمامی تصاویر یکسان باشد.
۴. اگر تمایل دارید که الگوریتم از ویژگی‌های object-specific استخراج شده در طول برنامه استفاده کند مقدار متغیر `READPATCHESFROMFILE` را در خط ۹ فایل `demoRelease.m` برابر "۰" قرار داده. در غیر این‌صورت شما می‌توانید از ویژگی‌های universal استفاده نمایید در این حالت باید مقدار این متغیر را برابر "۱" قرار دهید.

فایل demoRelease.m

جهت بررسی نحوه عملکرد مدل شبیه‌سازی شده بهتر است از فایل یا کد demoRelease.m شروع کنیم. در این فایل در خط ۵ از شما خواسته شده که با دستور addpath فایل‌ها و توابع مربوط به طبقه بندی کننده SVM را نصب نمایید. این کار با paste کردن folder مربوط به SVM در مسیر مشخص شده و تایپ این مسیر در مقابل addpath انجام می‌شود، به‌صورت زیر :

```
addpath 'E:\Program Files\MATLAB\R2006a\toolbox\osu-svm' %put your own path
```

در خطوط بعدی مطابق با توضیحات قبل مقادیر متغیرهای زیر را مشخص می‌کنیم:

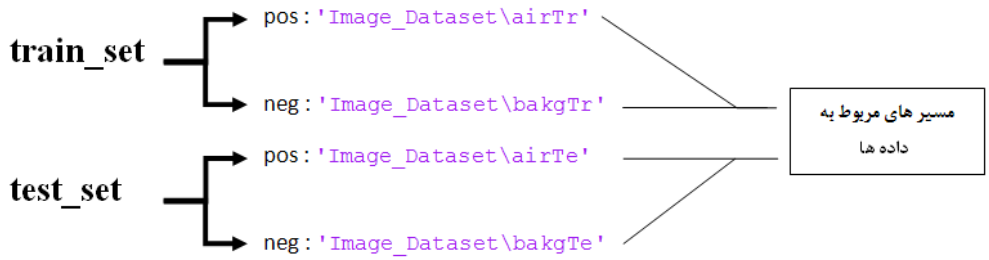
useSVM =	۱. در صورت تمایل به استفاده از طبقه‌بندی کننده SVM مقدار این متغیر برابر "۱" است. ۲. در صورت تمایل به استفاده طبقه‌بندی کننده با استفاده از روش نزدیک‌ترین همسایه (Nearest Neighbor) مقدار این متغیر برابر "۰" است.
READPATCHESFROMFILE =	۱. اگر تمایل دارید که الگوریتم از ویژگی‌های object-specific استخراج شده در طول برنامه استفاده کند مقدار متغیر READPATCHESFROMFILE را "۰" قرار داده. ۲. اگر تمایل دارید که از ویژگی‌های universal استفاده نمایید در این حالت باید مقدار این متغیر را برابر "۱" قرار دهید.
patchSizes =	بردار patch ها .patchSizes = [4 8 12 16]

جدول(۱)

خطوط ۱۸ تا ۲۱ فایل demoRelease.m برای تعریف و ذخیره‌سازی مسیرهای مربوط به داده‌ها می‌باشد. در این مرحله به ایجاد یک ساختار (structure) برای مشخص‌شدن و تعریف مسیر داده‌های آموزش و تست می‌پردازیم. این داده‌ها به دو دسته positive و negative تقسیم می‌گردند. با این تفاسیر داده‌ها با استفاده از خطوط ۱۸ تا ۲۱ به چهار گروه تعریف می‌شوند (توجه شود که در این مرحله تنها تعاریف مسیرها انجام می‌پذیرد و هیچ‌گونه خواندن یا load تصویر انجام نمی‌گردد) این چهار خط به‌صورت زیر است.

```
%specify directories for training and testing images
train_set.pos    = 'Image_Dataset\airTr';
train_set.neg    = 'Image_Dataset\bakgTr';
test_set.pos     = 'Image_Dataset\airTe';
test_set.neg     = 'Image_Dataset\bakgTe';
```

همان‌طور که بیان شد داده‌ها به چهار گروه مثبت آموزش، منفی آموزش، مثبت تست و منفی تست تعریف شده‌اند. در سمت راست عبارات فوق مسیر قرارگیری داده‌ها تعریف می‌شود، که در صورت لزوم با توجه به مسیر تصاویر می‌تواند توسط کاربر تغییر یابد. نتایج اجرای این خطوط توسط نمایش گرافیکی شکل(۱) واضح‌تر بیان شده است.



شکل(۱)،مسیر قرارگیری داده‌ها.

۱. تابع readAllImages.m

cI = readAllImages(train_set,test_set)		
Input arguments	train_set:	مسیرهای داده‌های آموزش از نوع Posetive وNegative.
	test_set:	مسیرهای داده‌های تست از نوع Posetive وNegative.
output arguments	cI:	سلول ذخیره‌سازی تصاویر.

جدول(۱-۱)

اولین تابعی که در فایل demoRelease.m فراخوانی می‌گردد تابع readAllImages.m است. به‌طور کلی این تابع برای خواندن یا load کردن تمامی تصاویر و ذخیره آنها در یک سلول با طول ۴ مورد استفاده قرار می‌گیرد. این تابع دارای دو آرگومان ورودی است که همان مسیرهای تصاویر است. خروجی این تابع cI{} می‌باشد که یک سلول با طول ۴ است که تمامی تصاویر پایگاه داده را در خود دارد.

همان‌طور که گفته شد نقش این تابع خواندن تصاویر است. با این وجود، جهت روشن شدن نحوه خواندن تصاویر در این تابع بهتر است به جزئیات آن بپردازیم. اگر به تابع readAllImages.m مراجعه کنیم، خواهیم دید که یک آرگومان ورودی دیگر نیز برای این تابع وجود دارد نام آرگومان maximagesperdir می‌باشد. خطوط زیر تعداد آرگومان‌های ورودی تابع را بررسی می‌کند.

```
if nargin<3
  maximagesperdir = inf;
end
```

بعد از این مرحله سلولی به‌نام denames{} ساخته می‌شود که درایه‌های آن همان آدرس‌های تصاویر می‌باشند. شکل(۱-۱).

$$\text{dnames}=\begin{bmatrix} \text{'Image\_Dataset\textbackslash airTr'} \\ \text{'Image\_Dataset\textbackslash bakgTr'} \\ \text{'Image\_Dataset\textbackslash airTe'} \\ \text{'Image\_Dataset\textbackslash airTe'} \end{bmatrix}$$

شکل(۱-۱).

همان‌طور که قبلا گفته شد خروجی تابع readAllImages.m یک سلول به‌طول ۴ است که تمام تصاویر را در خود جای داده. پس در ابتدا باید یک سلول به‌طول ۴ ساخت که در ابتدا خالی می‌باشد. شکل(۲-۱).

```
cI = cell(4,1);
```

$$\text{cI}=\begin{bmatrix} [\,] \\ [\,] \\ [\,] \\ [\,] \end{bmatrix}$$

شکل(۲-۱). سلولی به‌طول ۴ که تمام تصاویر را در خود جای خواهد داد.

بعد از انجام مراحل فوق وارد فاز فراخوانی تصاویر از پایگاه داده می‌شویم، که به‌ترتیب به‌صورت زیر است:

در ابتدا وارد یک حلقه for می‌شویم که چهار بار تکرار می‌گردد(زیرا ۴ گروه تصویر داریم). در ابتـدای ایـن حلقـه دسـتور زیـر را داریم.

```
c{i} = dir(dnames{i});
```

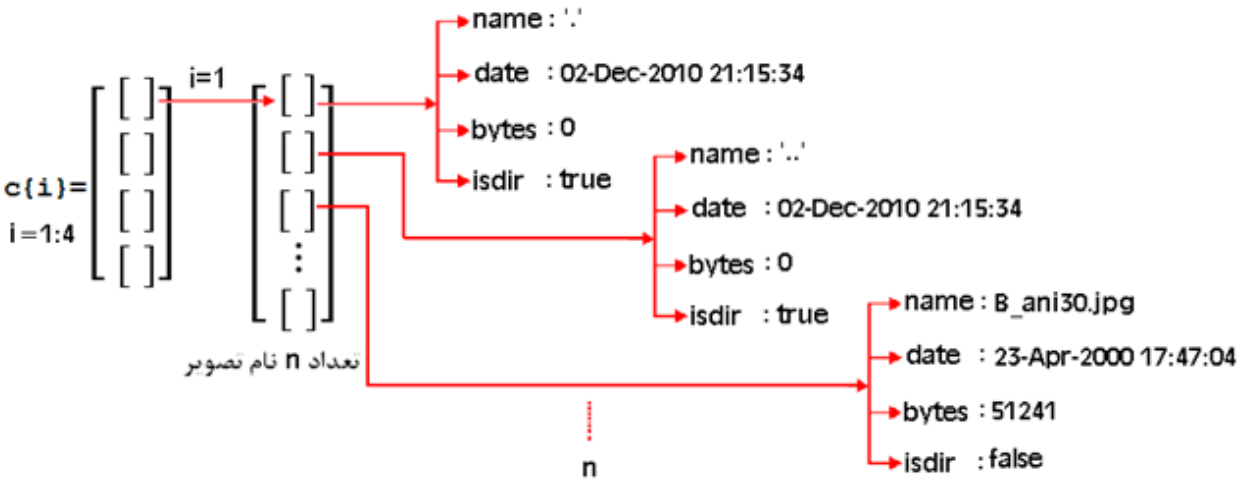
تابع `dir(dnames{i})` مسیر و نام تمامی تصاویر موجود در آرگومان خود را به‌عنوان خروجی ارائه می‌دهد (البته به‌اضـافه ۲ عنصر "." و ".." ، به‌عنوان مثال اگر ۴ تصویر داشته باشیم تعداد خروجی ۶ است. در آخر دستور `dir(dnames{i})` یک آرایه بـه طول تعداد تصاویر مسیر `i` ام می‌سازد که هر عضو این آرایه شامل اطلاعات زیر می‌باشد.

name
date
bytes
isdir

برای فراخوانی هر یک از اطلاعات فوق می‌توان به‌صورت زیر عمل کرد.(مثلا نام تصویر اول در سلول iام).

```
c{i}(1).name
```

در نهایت `c{i}` چهار عنصر خواهد داشت. جهت روشن‌تر شدن مطلب شکل(۳-۱) را برای چند مسیر تصویر مثال زده‌ایم.



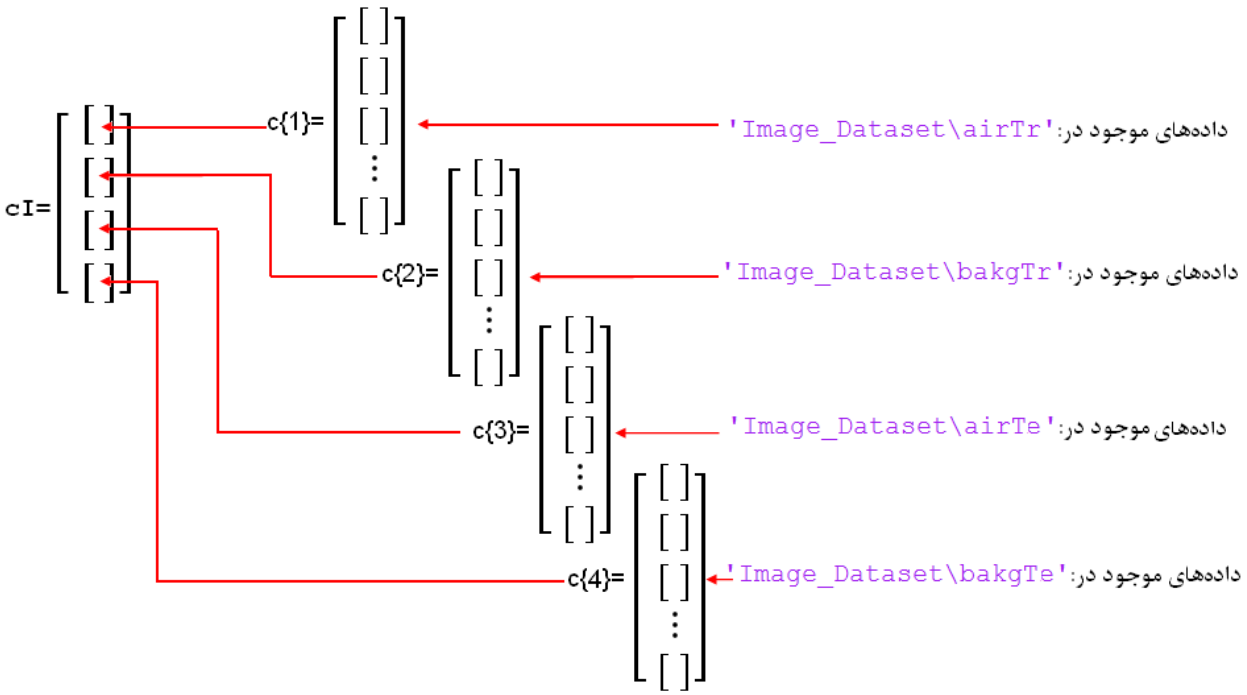
شکل(۳-۱). نمایش سلول `c{i}`.

همان‌طور که در بالا می‌بینید، برای مثال دو عنصر ابتدایی سلول اول `c{}` دارای نام "." و ".." می‌باشند، که نباید به‌عنوان تصویر ذخیره شوند و `c{}` فقط شامل نام و مسیر تصویر باشد. این کار را خطوط زیر انجام می‌دهند.

```
for i = 1:4,
  c{i} = dir(dnames{i});
  if length(c{i})>0,
    if c{i}(1).name == '.',
      c{i} = c{i}(3:end);
    end
  end
  if length(c{i})>maximagesperdir,
    c{i} = c{i}(1:maximagesperdir);
  end
  cI{i} = cell(length(c{i}),1);
  for j = 1:length(c{i}),
    cI{i}{j} = double(rgb2gray(imread([dnames{i} '/' c{i}(j).name]))./255;
  end
end
```

در این مرحله باید با استفاده از مسیرها و نام‌های موجود در c{} تمامی تصاویر را در cI{} فراخوانی یـا بارگـذاری کـرده بـرای انجام این کار در هر سلول cI{} یک بردار با طول تعداد نام‌های تصویر موجود در سلول iام c{} می‌سازیم. این عمل با دستور زیـر انجام می‌گردد. شکل(۴-۱).

```
cI{i} = cell(length(c{i}),1);
```



شکل(۴-۱). سلول‌ها و واحدهای cI{} (به جهت فلش‌ها دقت شود).

تا اینجا تعدادی بردار ساخته شده و در یک سلول به‌طول ۴ قرار داده شده است (که شامل مسیرها و نام‌های تصاویر است). حـال باید تصاویر load شوند و در محل خود قرار گیرند. این کار با سه خط زیر انجام می‌شود.

```
for j = 1:length(c{i}),
  cI{i}{j} = double(rgb2gray(imread([dnames{i} '/' c{i}(j).name]))./255;
end
```

که i بیان‌گر شماره سلول در cI{} است که از ۱ تا ۴ می‌باشد و j شماره تصویر j ام در c{} می‌باشد کـه بـه تعـداد تصـاویر موجود در یک مسیر خاص در c{} است. شکل(۵-۱).

```
cI{i}{j} = double(rgb2gray(imread([dnames{i} '/' c{i}(j).name]))./255
```

نام مسیر

نام یکی از تصاویر در این مسیر

شکل(۵-۱).

بعد از مرحله بارگذاری تصاویر درون cI{} برنامه به فایل اصلی demoRelease.m بازخواهد گشت. سه خط زیر تنها برای اطمینان از پر یا خالی بودن cI{} استفاده شده‌اند.

```
if isempty(cI{1}) | isempty(cI{2})
    error(['No training images were loaded -- did you remember to' ...
        ' change the path names?']);
end
```

با اطمینان از وجود تصاویر در cI{} وارد مرحله استخراج Prototype ها یا Patch های c1 می‌شویم. خطوط زیر این مرحله را پوشش می‌دهند.

```
%below the c1 prototypes are extracted from the images/ read from file
if ~READPATCHESFROMFILE
    tic
    numPatchesPerSize = 250; %more will give better results, but will
    %take more time to compute
    cPatches = extractRandC1Patches(cI{1}, numPatchSizes, ...
        numPatchesPerSize, patchSizes); %fix: extracting from positive only

    totaltimespectextractingPatches = toc;
else
    fprintf('reading patches');
    cPatches = load('PatchesFromNaturalImages250per4sizes','cPatches');
    cPatches = cPatches.cPatches;
end
```

در ابتدا "۰" یا "۱" بودن متغیر READPATCHESFROMFILE بررسی می‌گردد تا طبق توضیحاتی که قبلا داده شد، نوع استخراج Prototype ها یا Patch ها انتخاب گردد.

در صورت "۰" بودن : تابع extractRandC1Patches.m فراخوانی می‌گردد و از روی تصاویر ورودی(البته تنها تصاویر آموزش از نوع positive یا train_set.pos) Prototype ها استخراج می‌گردند.

در صورت"۱"بودن : برنامه، از ویژگی‌های از قبل استخراج شده universal features استفاده می‌کند. در این صورت باید فایل PatchesFromNaturalImages250per4sizes.mat را بارگذاری(load)کرد.

از آنجا که ما گزینه اول را انتخاب کردیم، لذا لازم است که در ابتدا به بررسی نحوه عملکرد تابع extractRandC1Patches.m بپردازیم.

۲. تابع extractRandC1Patches.m

cPatches = extractRandC1Patches(cI{1}, numPatchSizes, numPatchesPerSize, patchSizes);		
Input arguments	cI{1}:	تصاویر موجود در سلول اول cI{1} که همان تصاویر آموزش از نوع positive هستند.
	numPatchSizes:	طول بردار اندازه patch ها که ۴ می‌باشد زیرا patchSizes = [4 8 12 16].
	numPatchesPerSize:	تعداد patch استخراج شده برای هر اندازه که برابر ۲۵۰ است.
	patchSizes:	بردار patch ها patchSizes = [4 8 12 16].
output arguments	cPatches:	سلول با طول ۴ که حاوی تمام patch ها می‌باشد.

جدول(۱-۲)

همان‌طور که در جدول (۱-۲) می‌بینید این تابع از ۴ آرگومان ورودی تشکیل شده است. آرگومان ورودی `cI{}` همان داده‌ها یـا مثال‌های آموزش از نوع `train_set.pos` هستند که برای استخراج Prototype ها استفاده می‌شوند. آرگومان دوم وسوم در ابتدای فایل `demoRelease.m` مقداردهی شده‌اند. به‌منظور بررسی نحوه عملکرد این تابع به کد تابع مراجعه می‌کنیم. به‌طـور کلـی نقـش این تابع استخراج Prototype ها یا Patch ها به‌صورت تصادفی، به‌عنوان قسمتی از فاز آموزش برای طبقه‌بندی در مرحله C2 می‌باشد.

باید به این موضوع توجه کرد که در این تابع بر اساس جدول پارامترهای ارائه شده در مقاله Object Recognition with Features Inspired by Visual Cortex که در زیر هم آمده تنها از باند دوم (باند زرد رنگ) جهت استخراج Prototype ها استفاده شده.

Band Σ	1	2	3	4	5	6	7	8
filt. sizes s	7 & 9	11 & 13	15 & 17	19 & 21	23 & 25	27 & 29	31 & 33	35 & 37
σ	2.8 & 3.6	4.5 & 5.4	6.3 & 7.3	8.2 & 9.2	10.2 & 11.3	12.3 & 13.4	14.6 & 15.8	17.0 & 18.2
λ	3.5 & 4.6	5.6 & 6.8	7.9 & 9.1	10.3 & 11.5	12.7 & 14.1	15.4 & 16.8	18.2 & 19.7	21.2 & 22.8
grid size N^{Σ}	8	10	12	14	16	18	20	22
orient. θ	$0; \frac{\pi}{4}; \frac{\pi}{2}; \frac{3\pi}{4}$							
patch sizes n_i	$4 \times 4; 8 \times 8; 12 \times 12; 16 \times 16$ (×4 orientations)							

جدول(۲-۲)

خطوط زیر که در ابتدا تابع آمده، بیشتر برای جلوگیری از خطای احتمالی می‌باشد، زیرا تعداد آرگومان‌ها دوباره بررسی می‌گردد و در صورت عدم وجود، برخی از آنها دوباره مقداردهی می‌گردند.

```
if nargin<2
    numPatchSizes = 4;
    numPatchesPerSize = 250;
    patchSizes = 4:4:16;
end
```

مرحله بعدی تعیین برخی از پارامترها برای ساخت فیلترهای گابور و استخراج Patch ها با توجه به بانـد دوم جـدول(۲-۲) مـی-باشد.

```
rot = [90 -45 0 45];
c1ScaleSS = [1 3];
RF_siz    = [11 13];
c1SpaceSS = [10];
minFS     = 11;
maxFS     = 13;
div = [4:-.05:3.2];
Div       = div(3:4);
```

۲.۱. فیلتر گابور (`init_gabor.m`)

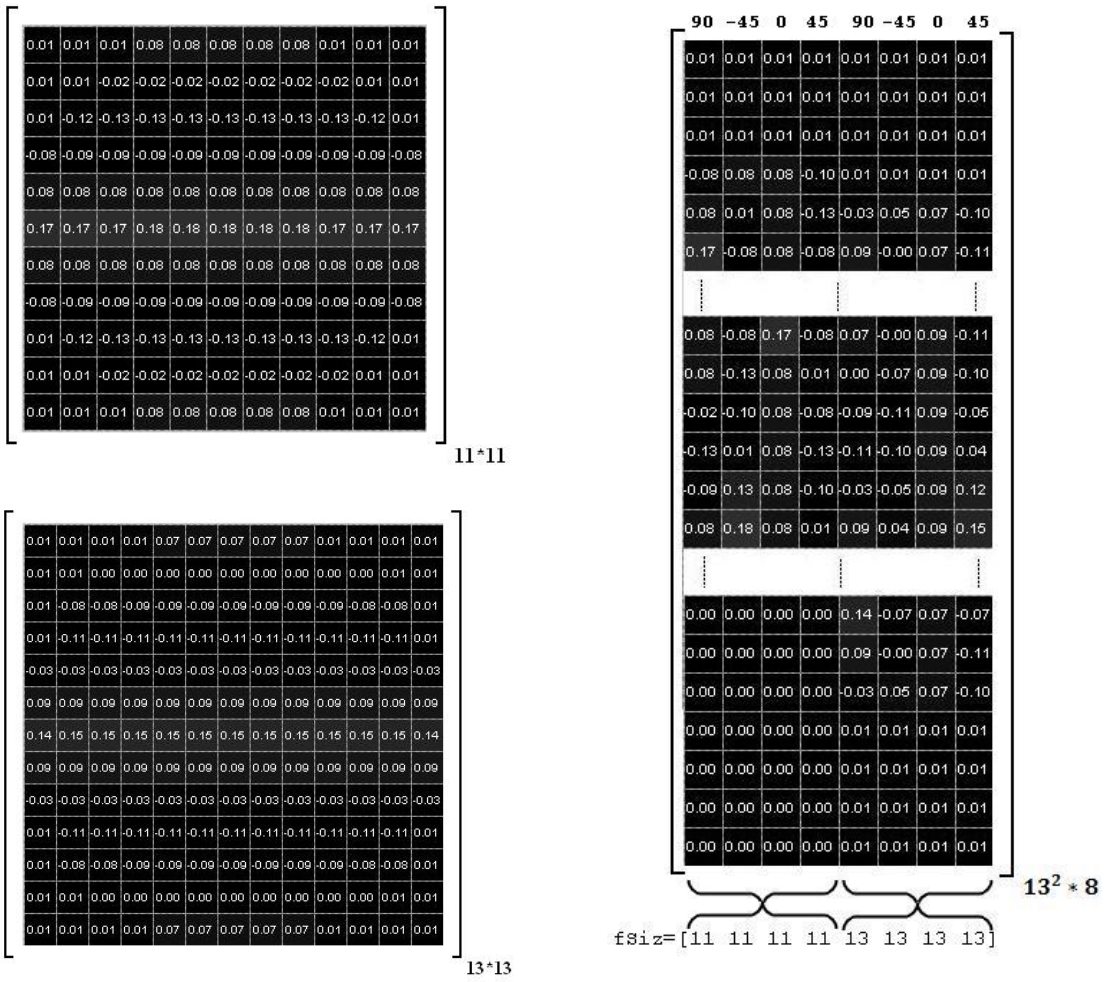
`[fSiz,filters,c1OL,numSimpleFilters] = init_gabor(rot, RF_siz, Div);`		
Input arguments	`rot:`	بردار جهت‌ها `rot = [90 -45 0 45]`.
	`RF_siz:`	بردار اندازه فیلترها که در این مرحله `RF_siz = [11 13]` است.
	`Div:`	`Div = [3.9 3.85]`.
output arguments	`fSiz:`	برداری که تعداد داراییهای آن برابر تعداد فیلترهاست که در آخر مقدار `[11 11 11 11 13 13 13 13]` را خواهد داشت.
	`Filters:`	یک ماتریس که تمامی فیلترها را بعد از reshape کردن در خود جای می‌دهد(اندازه آن در این مرحله $۸^{*}۱۳^{۲}$ است).
	`c1OL:`	متغیر با مقدار ۲.
	`numSimpleFilters:`	تعداد جهت‌های استفاده شده برای ساخت فیلترها که ۴ جهت می‌باشد.

جدول(۱-۱-۲)

در سه خط بعدی مقادیر λ و σ و γ تعیین و محاسبه می‌گردد.

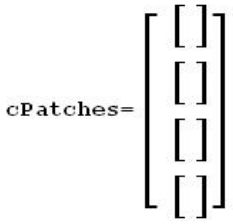
```
lambda = RF_siz*2./Div;
sigma  = lambda.*0.8;
G      = 0.3;   % spatial aspect ratio: 0.23 < gamma < 0.92
```

مرحله بعدی، فاز ساختن فیلترها و ذخیره آنها در یک ماتریس می‌باشد. در این مرحله فیلترها بعد از ساخت و reshape شدن درون یک ماتریس $۱۳^{۲}$*۸ قرار می‌گیرند که این ماتریس و دوفیلتر در اندازه‌های ۱۱ و۱۳به‌صورت گرافیکی در شکل (۱-۱-۲) نمایش داده شده‌اند. اندازه فیلترها هم درون بردار fSiz ذخیره می‌شود.



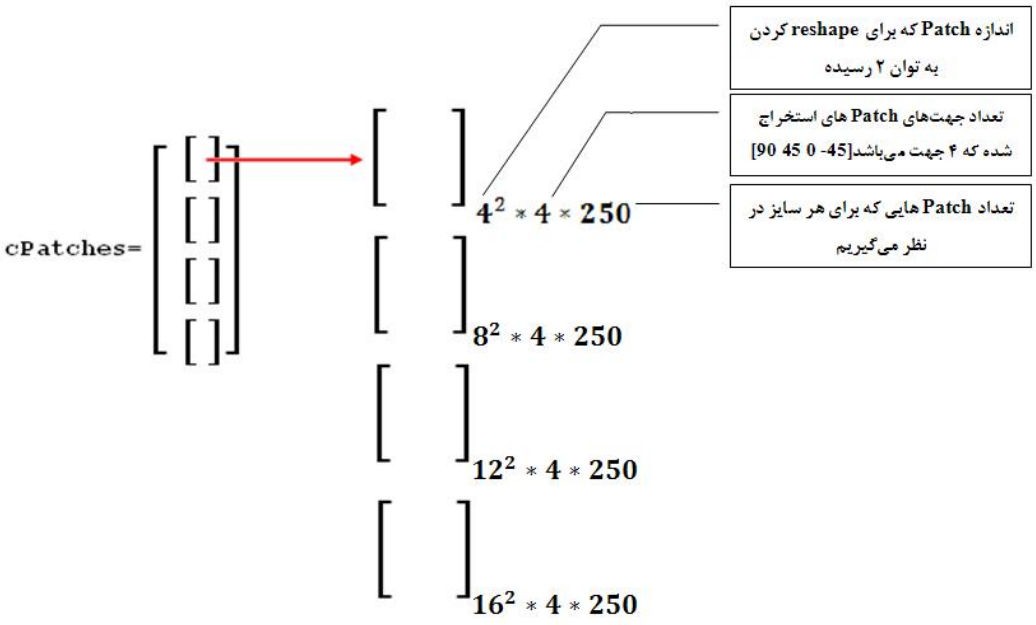
شکل(۱-۱-۲). نمایش گرافیکی ماتریس filters و دو فیلتر با اندازه ۱۱ و ۱۳.

بعد از ساخت فیلترها به تابع extractRandC1Patches.m برگشته. همان‌طور که اشاره شده ۴ اندازه برای patch ها داریم ۴و۸و۱۲و۱۶، به‌همین منظور یک سلول با طول ۴ برای ذخیره‌سازی patch ها می‌سازیم. شکل(۲-۱-۲).



شکل(۲-۱-۲). سلول با طول ۴ برای ذخیره‌سازی patch ها.

در هر سلول یک ماتریس صفر با تعداد ستون ۲۵۰ و تعداد سطر($4 * [4,8,12,16]^2$) می‌سازیم، این ماتریس‌ها برای ذخیره‌سازی patch ها ساخته می‌شوند. شکل(۳-۱-۲).



شکل(۳-۱-۲). ساخت ماتریس صفر برای ذخیره‌سازی patch ها در هر سلول.

مرحله بعدی انتخاب تصادفی یک تصویر و فراخوانی تابع `C1.m` می‌باشد.

۲.۲. تابع `C1.m`

`[c1source,s1source] = C1(stim, filters, fSiz, c1SpaceSS,c1ScaleSS, c1OL)`		
Input arguments	`stim:`	تصویر ورودی.
	`filters:`	ماتریس فیلترها که در مرحله قبل ساخته شد.
	`fSiz:`	برداری که تعداد دارایه‌های آن برابر تعداد فیلتر هاست که در این مرحله `[11 11 11 11 13 13 13 13]` است.
	`c1SpaceSS:`	برداری که محدوده pooling را مشخص می‌نماید.
	`c1ScaleSS:`	.Vector defining the scale bands
	`c1OL:`	متغیر با مقدار ۲.
output arguments	`c1source:`	ویژگی‌های C1 ، نتیجه پس از max گیری و sub sampling.
	`s1source:`	نتیجه تصاویر فیلتر شده .

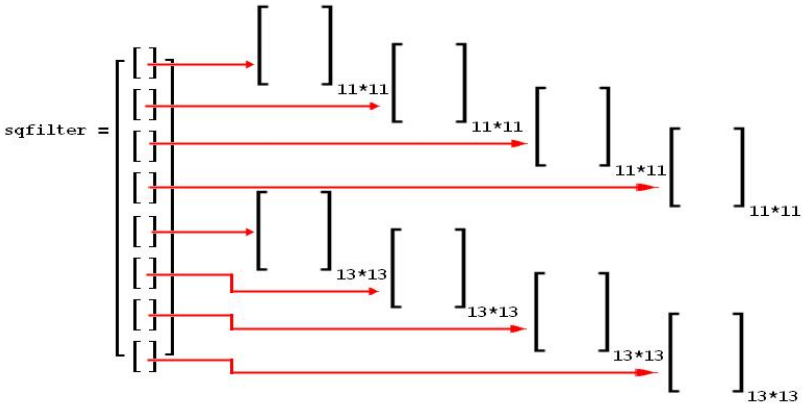
جدول(۱-۲-۲)

آرگومان‌های ورودی و خروجی در جدول(۱-۲-۲) توضیح داده شده‌اند. به‌طور کلی این تابع دو لایه S1 و C1 در مدل استاندارد را پیاده‌سازی می‌کند. آرگومان‌های `stim, filters, fSiz` برای پیاده‌سازی لایه S1 و اعمال فیلترهای ساخته شده مورد استفاده قرار می‌گیرد و ارگومان‌های `c1SpaceSS,c1ScaleSS, c1OL` برای پیاده‌سازی لایه C1 مورد استفاده قرار می‌گیرند.

در ابتدای کار بعد از تعریف برخی از متغیرهای درون تابعی به بازسازی فیلترهای گابور ساخته شده در تابع `init_gabor.m` می‌پردازیم. همان‌طور که قبلا توضیح داده شد، فیلترها بعد از reshape شدن درون یک ماتریس ۸*$۱۳^۲$ قرار می‌گیرند. حال در این‌جا با توجه به دو اندازه فیلتر که در باند دوم جدول (۲-۲) آمده باید آنها را به اندازه اولیه ۱۳*۱۳و ۱۱*۱۱ جهت اعمال به تصاویر بازگرداند. این کار با خطوط زیر در تابع `C1.m` انجام می‌شود.

```
% Rebuild all filters (of all sizes)
nFilts = length(fSiz);
for i = 1:nFilts
    sqfilter{i} = reshape(filters(1:(fSiz(i)^2),i),fSiz(i),fSiz(i));
    if USECONV2
        sqfilter{i} = sqfilter{i}(end:-1:1,end:-1:1); %flip in order to use
                                                      conv2 instead of imfilter2
    end
end
```

که در نهایت فیلترهای بازسازی‌شده درون یک ساختار به‌صورت زیر قرار می‌گیرند. شکل(۱-۲-۲).



شکل(۱-۲-۲). فیلترهای بازسازی‌شده که درون `sqfilter` قرار می‌گیرند.

توجه : دستور زیر(که با رنگ زرد مشخص شده) بدین منظور استفاده شده است که در مراحل بعدی بجای استفاده از دستور `imfiltre2` در MATLAB از `conv2` استفاده کنیم، این کار سرعت اجرای برنامه را افزایش می‌دهد. البته این خط در صورت انجام می شود که مقدار متغیر`USECONV2` یک باشد.

```
% Rebuild all filters (of all sizes)
%%%%%%%%
nFilts = length(fSiz);
for i = 1:nFilts
    sqfilter{i} = reshape(filters(1:(fSiz(i)^2),i),fSiz(i),fSiz(i));
    if USECONV2
        sqfilter{i} = sqfilter{i}(end:-1:1,end:-1:1); %flip in order to use
                                                      conv2 instead of imfilter2
    end
end
```

بعد از بازسازی فیلترها وارد محاسبات مربوط به S1 خواهیم شد. مرحله اول اعمال فیلترها به تصویر است. (از لغت تصویر به‌جای تصاویر استفاده شده زیرا تابع `C1.m` روی تک تک تصاویر به‌صورت جداگانه اعمال می‌گردد نه گروهی از تصاویر به‌صورت همزمان). قبل از اعمال فیلترها به تصاویر یک مرحل پیش محاسبه یا پیش فیلتر توسط تابع `sumfilter.m` صورت می‌گیرد.

```
sqim = stim.^2;
iUFilterIndex = 0;
% precalculate the normalizations for the usable filter sizes
uFiltSizes = unique(fSiz);
for i = 1:length(uFiltSizes)
    s1Norm{uFiltSizes(i)} = (sumfilter(sqim,(uFiltSizes(i)-1)/2)).^0.5;
    %avoid divide by zero
    s1Norm{uFiltSizes(i)} = s1Norm{uFiltSizes(i)} + ~s1Norm{uFiltSizes(i)};
end
```

توضیحات مربوط به این تابع در زیر آمده است.

۲.۲.۱. تابع sumfilter.m

I3 = sumfilter(I,radius)		
Input arguments	I:	تصویر ورودی.
	radius:	بردار شعاع پنجره برای عمل conv2 (که در این مرحله ۵ یا ۶ است).
output arguments	I3:	تصویر کانوال شده.

جدول(۱-۱-۲-۲)

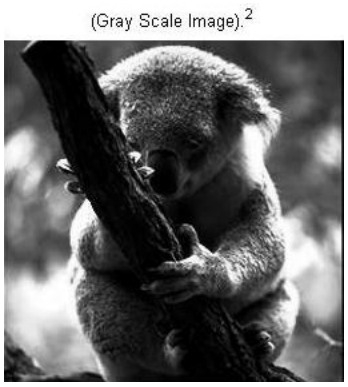
عملکرد این تابع بسیار ساده می‌باشد. این تابع با توجه به طول متغیر یا آرگومان ورودی دو انتخاب دارد(۱-طول بردار شعاع ۴ باشد. ۲-طول بردار شعاع ۱ باشد.) که در این‌جا مورد دوم انتخاب می‌شود. که خطوط مربوط به انتخاب دوم در زیر آمده.

```
switch length(radius)
    case 4,
        I2 = conv2(ones(1,radius(2)+radius(4)+1), ones(radius(1)+radius(3)+1,1), I);
        I3 = I2((radius(4)+1:radius(4)+size(I,1)),
(radius(3)+1:radius(3)+size(I,2)));
    case 1,
        mask = ones(2*radius+1,1);
        I2 = conv2(mask, mask, I);
        I3 = I2((radius+1:radius+size(I,1)), (radius+1:radius+size(I,2)));
end
```

در این مرحله به ساخت یک بردار "یک" به‌صورت شکل(۱-۱-۲-۲) و استفاده از این بردار برای اعمال کانولوشن بـه تصـویر مـی-پردازیم. نتیجه هر مرحله در شکل(۲-۱-۲-۲) آمده است.

$$mask=\begin{bmatrix}1\\1\\1\\1\\1\\1\\1\\1\\1\\1\\1\end{bmatrix}_{11*1} \quad or \quad mask=\begin{bmatrix}1\\1\\1\\1\\1\\1\\1\\1\\1\\1\\1\\1\\1\end{bmatrix}_{13*1}$$

شکل(۱-۱-۲-۲). بردار "یک".




```
I2 = conv2(mask, mask, I);
```

کانوال برای شعاع ۵ با mask با اندازه ۱*۱۱

کانوال برای شعاع ۶ با mask با اندازه ۱*۱۳

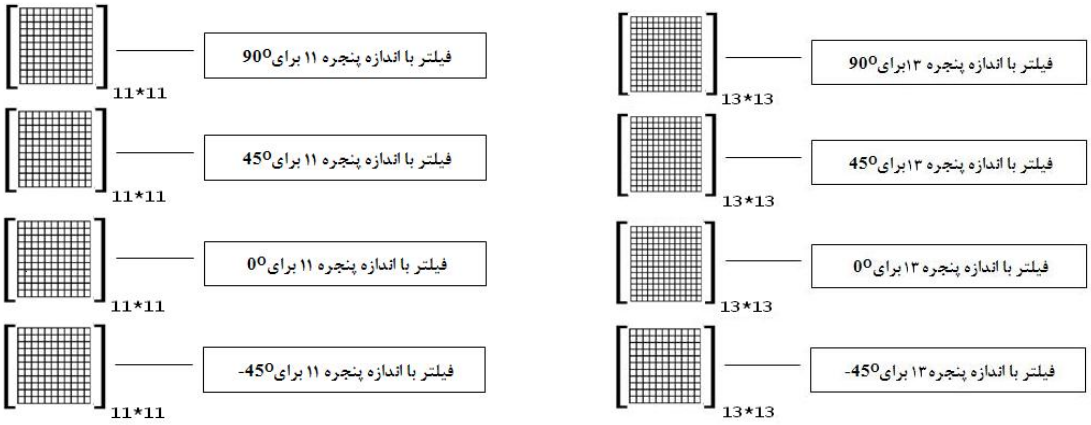
```
I3 = I2((radius+1:radius+size(I,1)), (radius+1:radius+size(I,2)));
```

خروجی برای شعاع ۵ با mask با اندازه ۱*۱۱

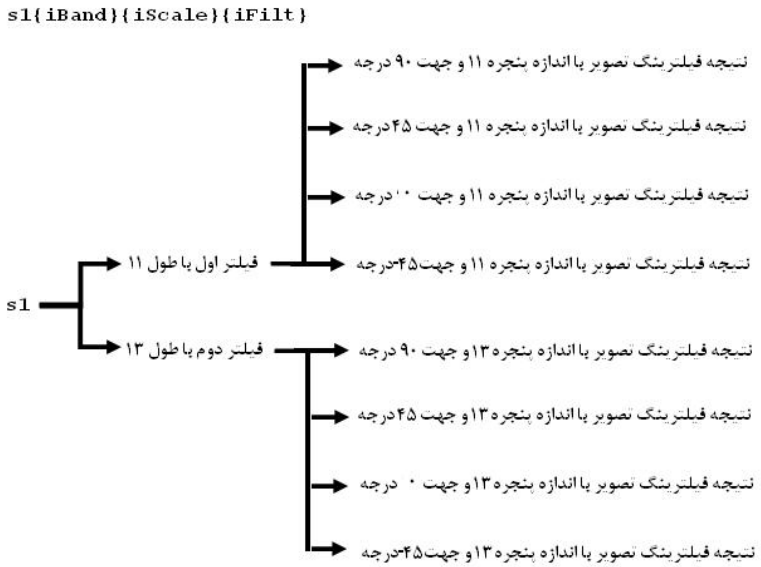
خروجی برای شعاع ۶ با mask با اندازه ۱*۱۳

شکل(۲-۱-۲-۲). نتایج حاصل از تابع `sumfilter.m`.

نتایج درون `s1Norm{}` قرار می‌گیرند که در مراحل بعدی برای نرمالیزه کردن تصویر فیلتر شده مورد استفاده قرار می‌گیرند. بعد از خروج از تابع `sumfilter.m` باید فیلترهای گابور را به تصویر اعمال کرد. توجه داشته باشید که تعداد فیلترها ۸ می‌باشد (در دو اندازه ۱۱و۱۳ و در ۴ جهت [90,-45,0,45]).شکل(۳-۱-۲-۲).



شکل(۳-۱-۲-۲). فیلترها در دو اندازه ۱۱و۱۳ و در ۴ جهت [90,-45,0,45].



شکل(۴-۱-۲-۲). نحوه ذخیره نتایج بعد از فیلترکردن در سلول S1.

یکی از این ۸ فیلتر به تصویر اعمال می‌گردد و با توجه به خطوط زیر، نتیجه درون یک ساختار به شکل(۴-۱-۲-۲) قرار می‌گیرند.

```
for iBand = 1:numScaleBands
    for iScale = 1:length(ScalesInThisBand{iBand})
        for iFilt = 1:numSimpleFilters
            iUFilterIndex = iUFilterIndex+1;
            if ~USECONV2
               s1{iBand}{iScale}{iFilt}=
               abs(imfilter(stim,sqfilter{iUFilterIndex},'symmetric','same','corr'));
                if(~INCLUDEBORDERS)
                   s1{iBand}{iScale}{iFilt}=
                   removeborders(s1{iBand}{iScale}{iFilt},fSiz(iUFilterIndex));
                end
               s1{iBand}{iScale}{iFilt}=
               im2double(s1{iBand}{iScale}{iFilt})./s1Norm{fSiz(iUFilterIndex)};
            else %not 100% compatible but 20% faster at least
                s1{iBand}{iScale}{iFilt} =
                abs(conv2(stim,sqfilter{iUFilterIndex},'same'));
                if(~INCLUDEBORDERS)
                  s1{iBand}{iScale}{iFilt}=
                  removeborders(s1{iBand}{iScale}{iFilt},fSiz(iUFilterIndex));
```

```
                end
                s1{iBand}{iScale}{iFilt}=
                im2double(s1{iBand}{iScale}{iFilt})./s1Norm{fSiz(iUFilterIndex)};
            end
        end
    end
end
```

بعد از اعمال یکی از این فیلترهای ۸ گانه به تصویر یک تابع درون تابعی در C1.m به نـام removeborders فراخـوانی مـی-شود. عملکرد این تابع در زیر آمده است.

۲.۲.۲. تابع removeborders

sout = removeborders(sin,siz)		
Input arguments	sin:	تصویر ورودی(بعد از اعمال یک از فیلترهای گابور) .
	siz:	اندازه فیلتری که تصویر با آن فیلتر شده.
output arguments	sout:	تصویر خروجی بعد از برش یک فریم از اطراف تصویر و اضافه کردن صفر بجای آن.

جدول(۱-۲-۲-۲)

به‌طور کلی نقش این تابع حذف یک فریم (frame) از اطراف تصویر و جایگزینی این فریم حذف‌شده با مقادیر صـفر مـی‌باشـد. در این تابع که چهار خط است و در زیر آمده یک تابع دیگر به نام unpadimage.m فراخوانی می‌گردد و پـس از آن توسـط دو دسـتور مشخص شده در زیر فریم حذف شده با صفر جایگذاری می‌شود. تابع unpadimage.m و مراحل دیگر در ادامه توضیح داده می‌شود.

```
function sout = removeborders(sin,siz)
sin = unpadimage(sin, [(siz+1)/2,(siz+1)/2,(siz-1)/2,(siz-1)/2]);
sin = padarray(sin, [(siz+1)/2,(siz+1)/2],0,'pre');
sout = padarray(sin, [(siz-1)/2,(siz-1)/2],0,'post');
```

۲.۲.۲.۱. تابع unpadimage.m

o = unpadimage(i,amnt)		
Input arguments	i:	تصویر ورودی(بعد از اعمال یک از فیلترهای گابور).
	amnt:	بردار تعداد سطر و سطون برای برش از اطراف تصویر.
output arguments	o:	تصویر خروجی بعد از برش یک فریم از اطراف تصویر .

جدول(۱-۱-۲-۲-۲)

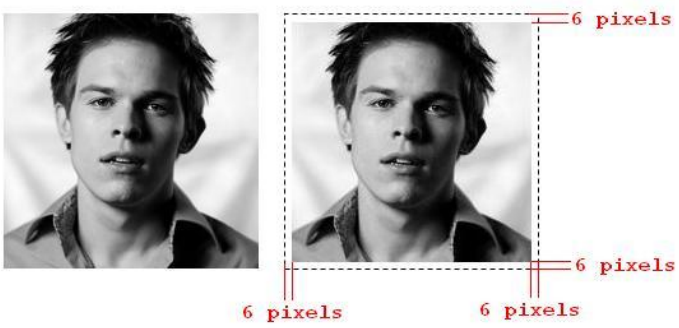
برای فهم بهتر از این تابع کد آن به‌طور کامل در زیر آمده.

```
function o = unpadimage(i,amnt)
%function o = unpadimage(i,amnt)
%
%un does padimage
%if length(amnt == 1), unpad equal on each side
%if length(amnt == 2), first amnt is left right, second up down
%if length(amnt == 4), then [left top right bottom];
switch(length(amnt))
case 1
  sx = size(i,2) - 2 * amnt;
```

```
  sy = size(i,1) - 2 * amnt;
  l = amnt + 1;
  r = size(i,2) - amnt;
  t = amnt + 1;
  b = size(i,1) - amnt;
case 2
  sx = size(i,2) - 2 * amnt(1);
  sy = size(i,1) - 2 * amnt(2);
  l = amnt(1) + 1;
  r = size(i,2) - amnt(1);
  t = amnt(2) + 1;
  b = size(i,1) - amnt(2);
case 4
  sx = size(i,2) - (amnt(1) + amnt(3));
  sy = size(i,1) - (amnt(2) + amnt(4));
  l = amnt(1) + 1;
  r = size(i,2) - amnt(3);
  t = amnt(2) + 1;
  b = size(i,1) - amnt(4);
otherwise
  error('illegal unpad amount\n');
end
if(any([sx,sy] < 1))
    fprintf('unpadimage newsize < 0, returning []\n');
    o = [];
    return;
end
o = i(t:b,l:r,:);
```

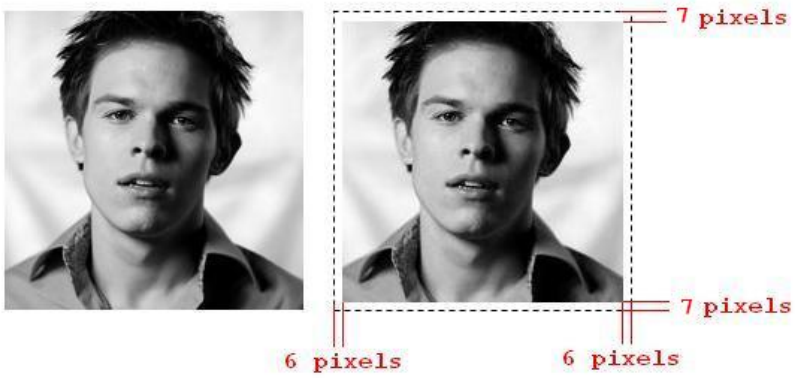
در این تابع ۳ انتخاب وجود دارد که با توجه به طول بردار amnt یکی از آنها انتخاب می‌گردد. در صورتی که طـول بـردار amnt یک باشد، اندازه فریم برای برش از اطراف تصویر در چهار ضلع تصویر یکسان می‌باشد. مثلا اگر amnt=6 باشد داریـم. شـکل(۱-۱-۲-۲).

تصویر ورودی



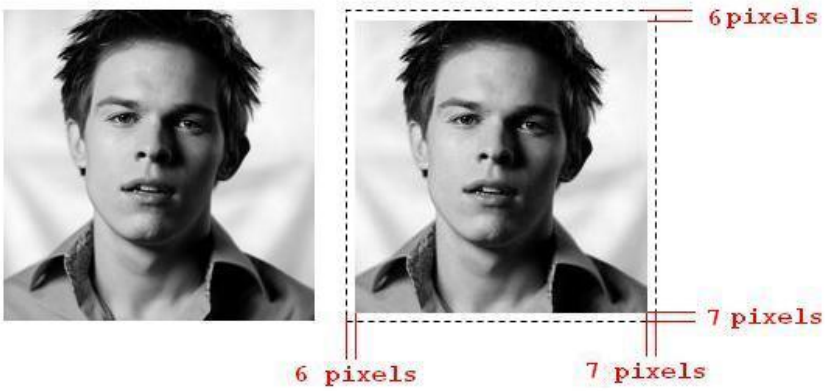
شکل(۱-۱-۲-۲). درصورتی که amnt=6 باشد.

در صورتی که طول بردار amnt دو باشد درایه اول آن میزان برش تصویر از چپ و راست و درایه دوم آن میـزان بـرش تصـویر از بالا و پایین است. مثلا اگر amnt=[6 7] باشد داریم. شکل(۲-۱-۲-۲-۲).



شکل(۲-۱-۲-۲-۲). درصورتی که amnt=[6 7] باشد.

در صورتی که طول بردار amnt برابر با ۴ باشد درایه اول میزان برش تصویر از چپ ، درایه دوم از بالا، درایه سوم از راست و درایه چهارم از پایین می‌باشد. مثلا اگر amnt=[6 6 7 7] باشد داریم. شکل(۳-۱-۲-۲-۲).



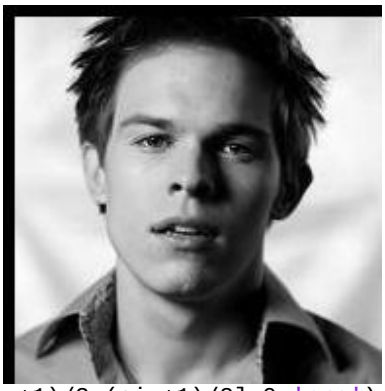
شکل(۳-۱-۲-۲-۲). درصورتی که amnt=[6 6 7 7] باشد.

پس از انجام مراحل فوق در تابع unpadimage.m به تابع C1.m بازگشته و با اضافه کـردن صـفر بـه چهـار طـرف تصـویر بـا دستورات زیر نتایج شکل(۴-۱-۲-۲-۲) را خواهیم داشت. توجه داشته باشید که مثال‌های فوق برای یک تصویر فیلتر نشده آورده شده و تنها به‌جهت فهم بهتر عملکرد تابع unpadimage.m می‌باشد.

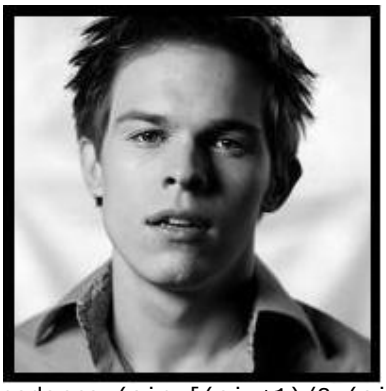
```
sin = padarray(sin, [(siz+1)/2,(siz+1)/2],0,'pre');
sout = padarray(sin, [(siz-1)/2,(siz-1)/2],0,'post');
```

تصویر ورودی

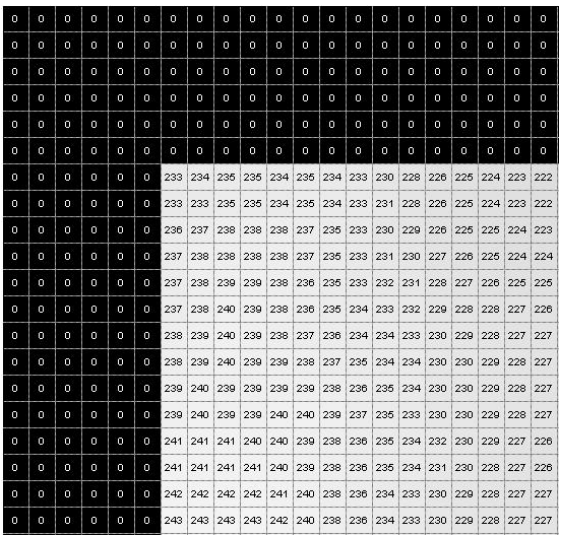
نتیجه برش یک فریم از اطراف تصویر



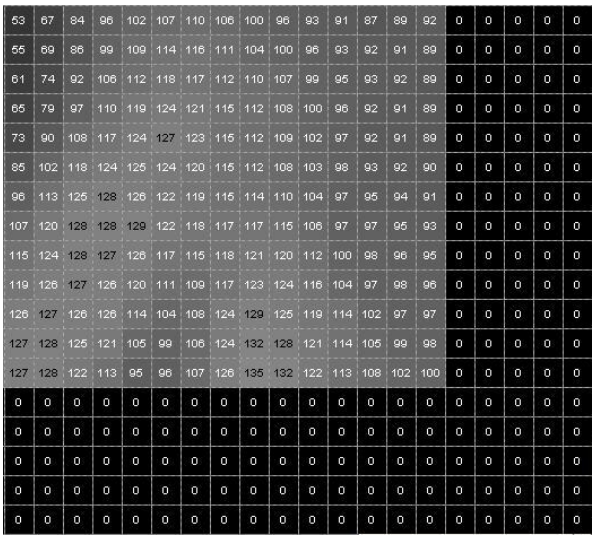
`padarray(sin,[(siz+1)/2,(siz+1)/2],0,'pre')`



`padarray(sin,[(siz+1)/2,(siz+1)/2],0,'post')`



گوشه تصویر بزرگ شده بالا



گوشه تصویر بزرگ شده بالا

شکل(۴-۱-۲-۲-۲). نتیجه مراحل تابع `removeborders` و `unpadimage.m`.

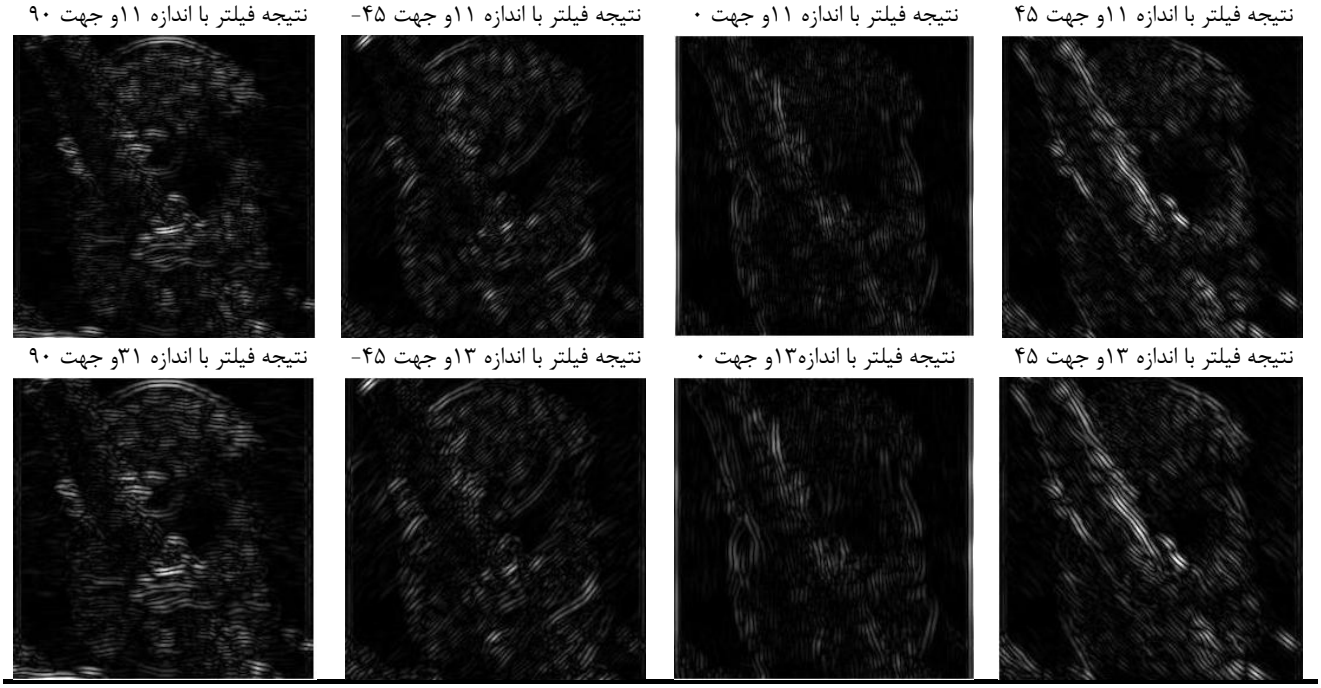
محاسبات مربوط به S1 در مراحل فوق انجام شد. این مراحل به‌صورت خلاصه در زیر آمده:

۱. اعمال یکی از فیلترهای ۸ گانه به تصویر.
۲. فراخوانی تابع `removeborders` و اضافه کردن یک فریم یا قاب صفر با توجه به اندازه فیلترها (۱۱یا ۱۳) بـه چهـار طرف تصویر.
۳. به مرحله ۱ رفته.

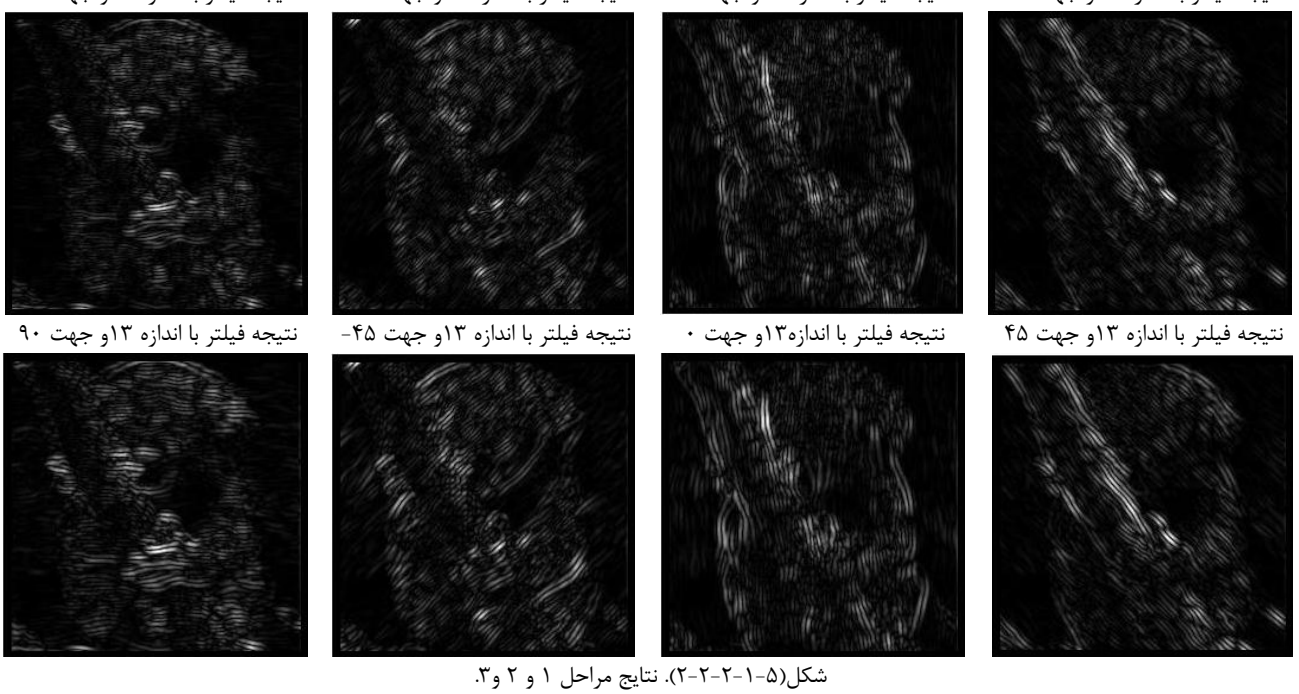
نتایج مراحل ۱ و ۲ و۳ برای یک تصویر در شکل (۵-۱-۲-۲-۲) آمده است.

تصویر ورودی

نتایج فیلترکردن یک تصور با اعمال فیلترهای ۸ گانه قبل از فراخوانی تابع removeborders



نتایج فیلترکردن یک تصور با اعمال فیلترهای ۸ گانه بعد از فراخوانی تابع removeborders



شکل(۵-۱-۲-۲-۲). نتایج مراحل ۱ و ۲ و۳.

پس از اتمام مراحل محاسباتی برای S1 در این فاز به محاسبات مربوط به C1 پرداخته می‌شود. این قسمت شامل ۳ مرحلـه مـی-باشد که در زیر آمده :

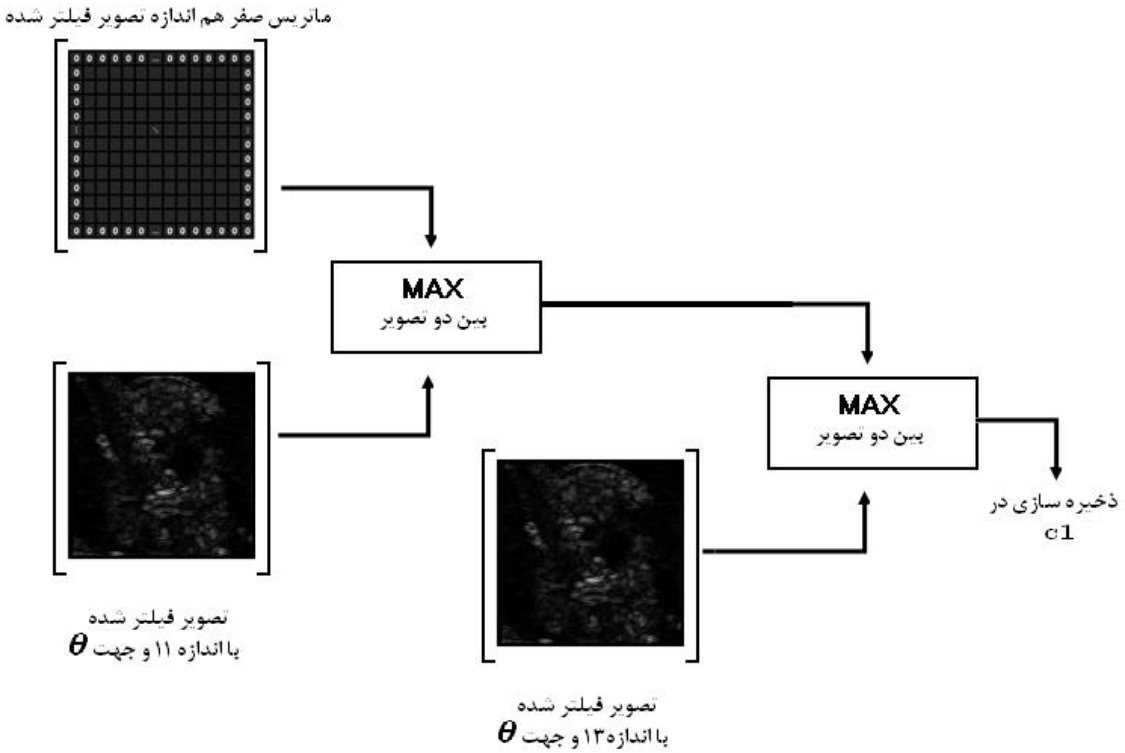
```
(1) pool over scales within band
(2) pool over local neighborhood
(3) subsample
```

مرحله اول عمل max گیری بین دو مقیاس یا scale متفاوت با جهت یا درجه یکسان می‌باشد. خطوط زیر این کار را انجـام مـی-دهند.

```
%   (1) pool over scales within band
for iBand = 1:numScaleBands
    for iFilt = 1:numSimpleFilters
        c1{iBand}(:,:,iFilt) = zeros(size(s1{iBand}{1}{iFilt}));
        for iScale = 1:length(ScalesInThisBand{iBand});
            c1{iBand}(:,:,iFilt) = max(c1{iBand}(:,:,iFilt),s1{iBand}{iScale}{iFilt});
        end
        size(c1{iBand}(:,:,iFilt)),pause()
    end
end
```

در ابتدا یک ماتریس صفر به اندازه هر تصویر ذخیره شده در S1 ساخته می‌شود، سپس با دستور زیر مقدار max بـین ایـن و یـک تصویر فیلتر شده با اندازه ۱۱ محاسبه می‌گردد (که قطعا مقدار max همان تصویر فیلتـر شـده اسـت) و در `c1{iBand}(:,:,iFilt)` ذخیره می‌شود. تصویر ذخیره شده در تکرار بعدی حلقه for با یک تصویر فیلتر شده با اندازه ۱۳ و جهت یکسـان مقایسـه مـی‌شـود و مقدار max در `c1{iBand}(:,:,iFilt)` ذخیره می‌شود. این مرحله با استفاده از حلقه for که در بالا مشخص شده انجام می‌شود.

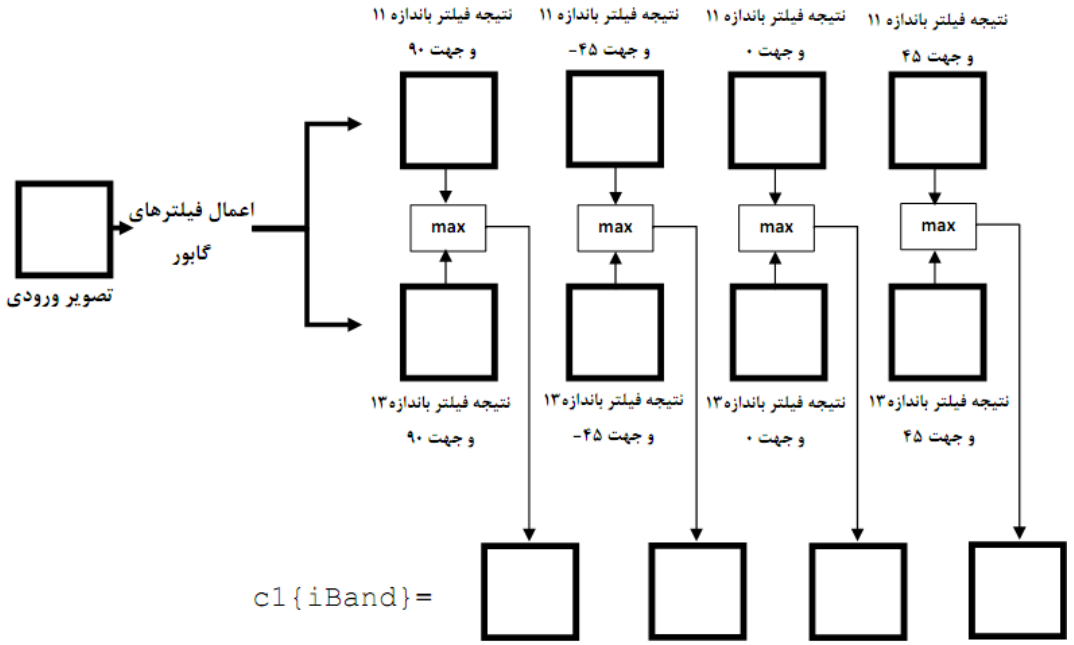
توجه شود که مقایسه یا عمل max گیری روی کل تصویر انجام می‌شود و بـین دو مقیـاس متفـاوت ۱۱ و ۱۳ بـا جهـت یکسـان صورت می‌گیرد برای روشن شدن این مفهوم شکل(۲-۲-۲) را در نظر بگیرید.



شکل(۲-۲-۲). عمل max گیری بین دو مقیاس متفاوت ۱۱ و ۱۳.

دقت شود که max گیری بین ماتریس صفر و تصویر فیلتر شده در مرحله اول نتیجه‌ای جز تصویر فیلتر شده نخواهد داشت.

مراحل فوق برای چهار جهت [90,-45,0,45] و دو مقیاس ۱۱ و ۱۳ انجام می‌گردد حاصل در c1 ذخیره می‌شود. خلاصه مراحل برای یک تصویر و اعمال فیلترها تا max گیری به‌صورت گرافیکی در شکل(۳-۲-۲) نشان داده شده است.



شکل(۳-۲-۲). خلاصه مراحل C1.m برای یک تصویر و اعمال فیلترها تا max گیری.

پس از محاسبه مقدار max بین دو مقیاس، ۴ تصویر خروجی برای یک تصویر ورودی خواهیم داشت. مرحله بعدی محاسبه max محلی یا pool over local neighborhood است، که مراحل کار با استفاده از خطوط زیر انجام می‌شود.

```
%   (2) pool over local neighborhood
for iBand = 1:numScaleBands
    poolRange = (c1SpaceSS(iBand));
    for iFilt = 1:numSimpleFilters
        c1{iBand}(:,:,iFilt) = maxfilter(c1{iBand}(:,:,iFilt),[0 0 poolRange-1
        poolRange-1]);
    end
end
```

همان‌طور که ملاحظه می‌شود این کار با استفاده از خطوط فوق و تابع maxfilter.m انجام می‌شود که نحوه عملکرد تابع maxfilter.m در زیر آمده است.

۲.۲.۳. تابع `maxfilter.m`

`I = maxfilter(I,radius)`		
Input arguments	`I:`	یکی از تصاویر مرحله قبل (بعد از اعمال max گیری در مرحله قبل).
	`radius:`	بردار شعاع پنجره برای عمل `local pooling`(که در این‌جا [9 9 0 0]).
output arguments	`I:`	تصویر خروجی بعد از `local pooling`.

جدول(۱-۳-۲-۲)

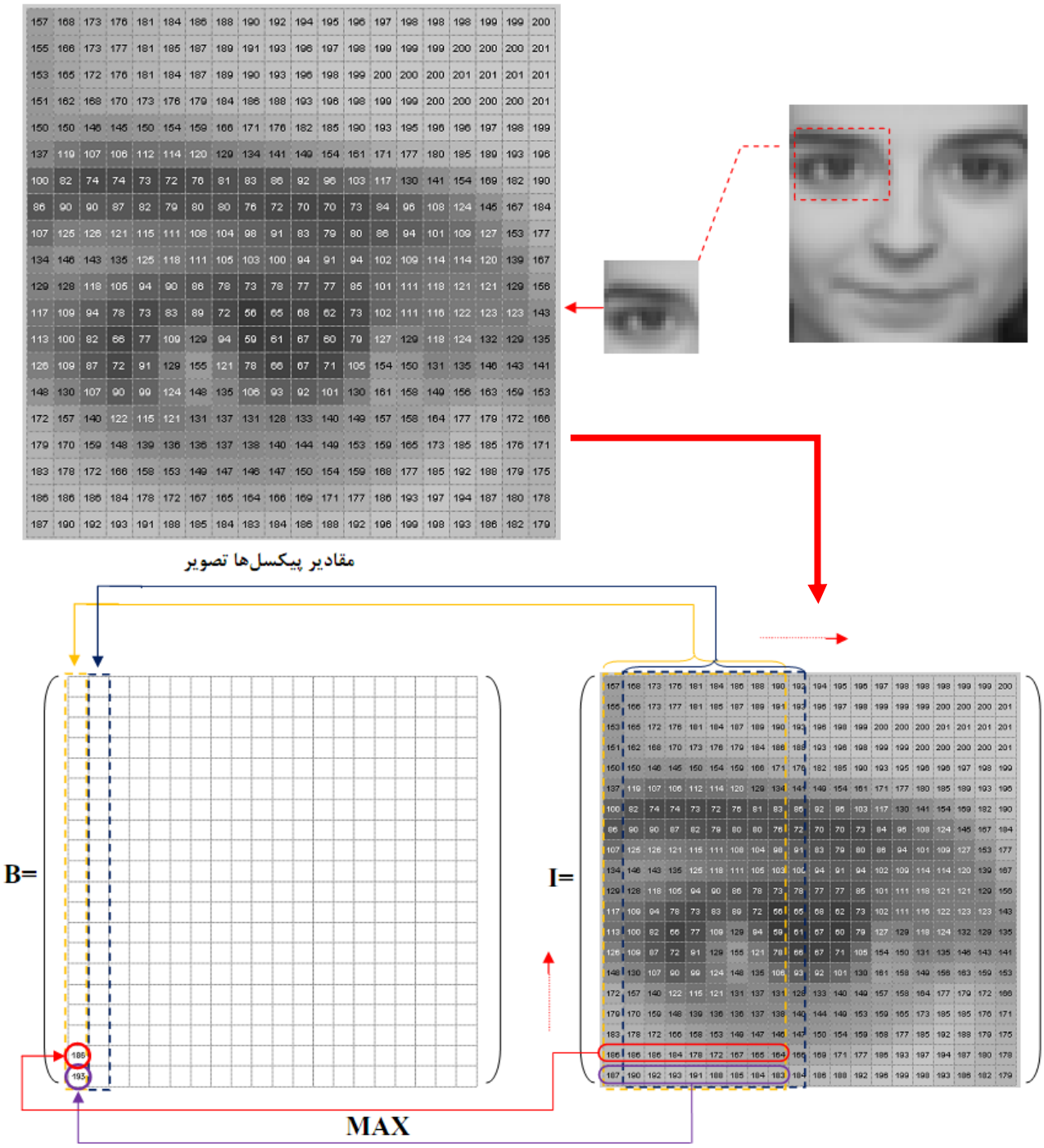
به‌طور کلی نقش این تابع انجام عمل morphological dilation روی تصاویر حاصل از مرحله قبل می‌باشد. با توجه به مقدار آرگومان ورودی `radius` در تابع `maxfilter.m` دو انتخاب وجود دارد.

۱. طول بردار `radius` یک باشد.
۲. طول بردار `radius` ۴ باشد.

ما در این‌جا با توجه به مقدار ورودی `radius` مورد دوم را مورد استفاده قرار دادیم. در این صورت max گیری در این مرحله به‌صورت زیر است :

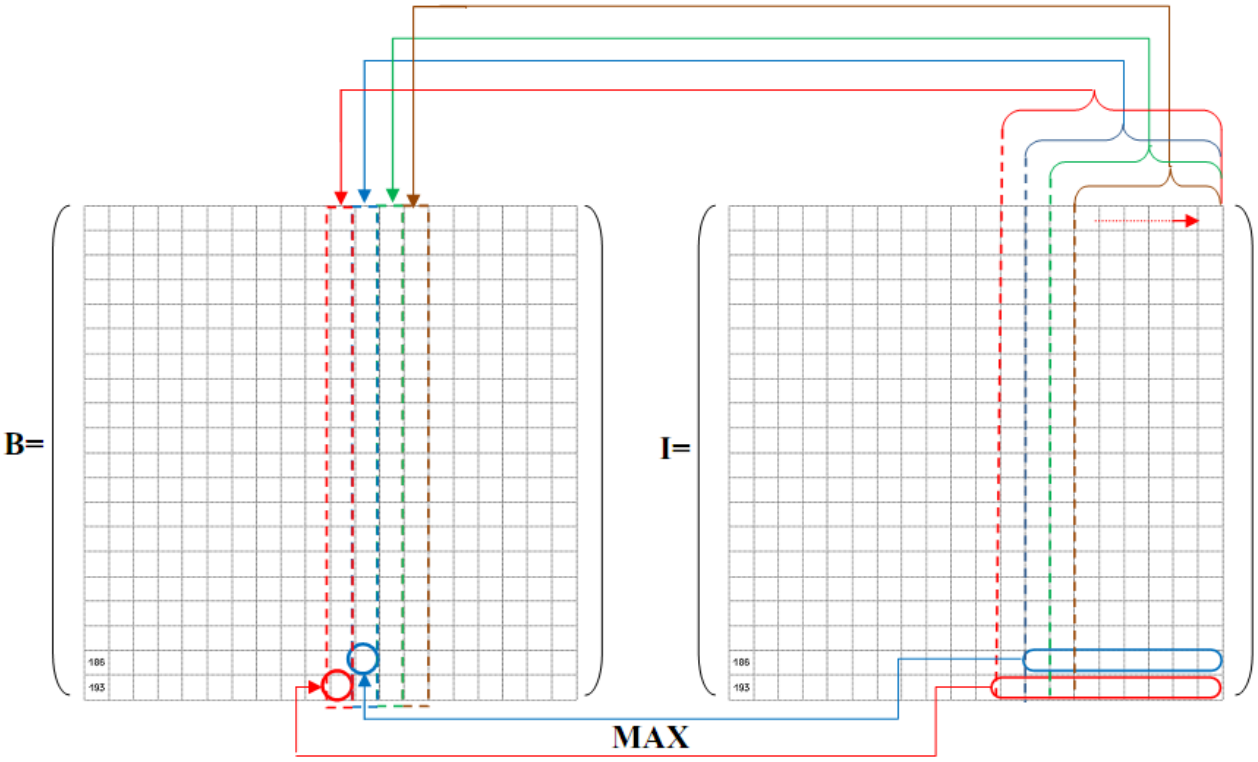
مرحله اول : این مرحله با یک مثال از برشی کوچک از یک تصویر توضیح داده می‌شود. همان‌طور که در شکل(۱-۳-۲-۲) مشاهده می‌شود یک پنجره مستطیلی به تعداد ستون ۹ پیکسل و تعداد سطور کل تصویر در نظر گرفته می‌شود، این پنجره روی تصویر می‌لغزد و مقدار max خروجی حاصل از این پنجره در ماتریس B جایگذاری می‌شود. همان‌طور که از شکل(۱-۳-۲-۲) مشخص است با هر بار لغزش پنجره به جلو یک ستون کامل در ماتریس B جایگذاری می‌شود.

برای فهم نحوه عملکرد دستور max در MATLAB به Help آن مراجع گردد.



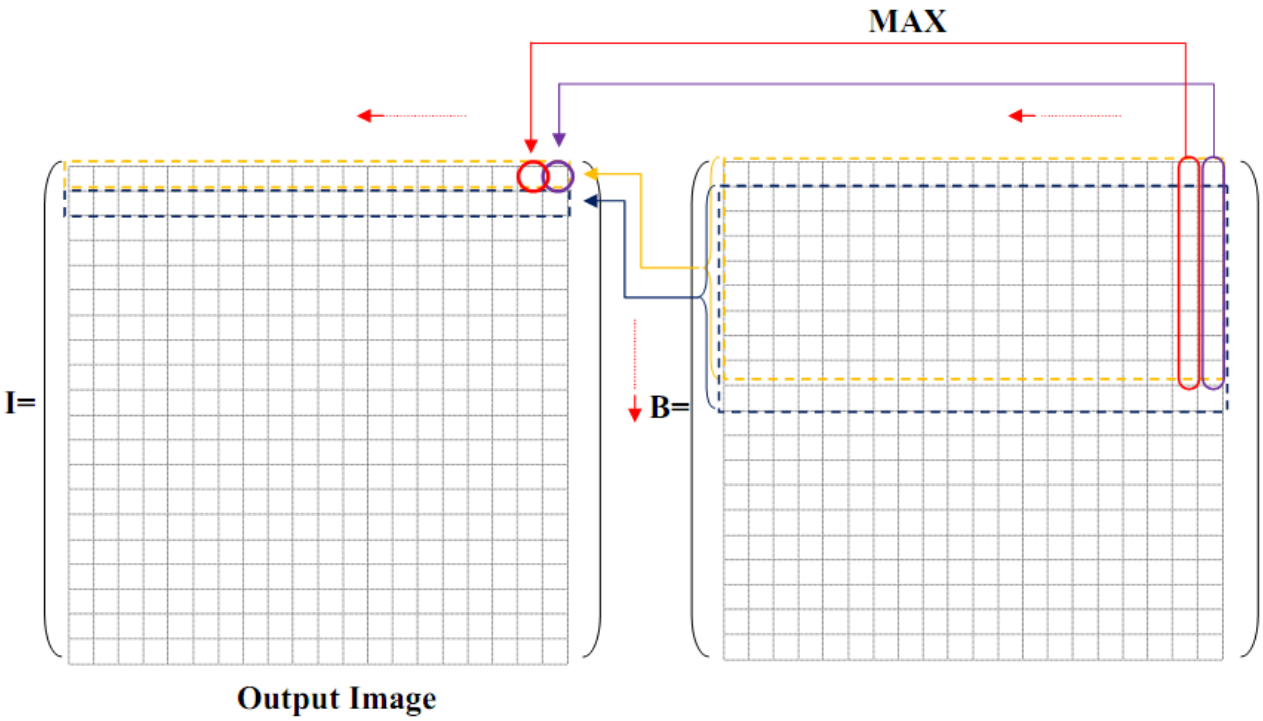
شکل(۱-۳-۲-۲). انجام عمل `local pooling` در مرحله اول تابع `maxfilter.m`.

در صورتی که لبه راست پنجره در شکل (۱-۳-۲-۲) به انتهای تصویر برسد، نحوه محاسبه max کمی تغییر کرده و به‌صورت شکل (۲-۳-۲-۲) خواهد بود.



شکل(۲-۳-۲-۲). محاسبه max در صورتی که لبه راست پنجره به انتهای تصویر برسد.

مرحله ۲ : بعد از محاسبات فوق مقدار max به‌صورت افقی با همان کیفیت مرحله اول انجام می‌گیرد، با این تفاوت که max روی تصویر حاصل شده از مرحله ۱ محاسبه می‌گردد و در نهایت در ماتریس I که خروجی تابع max است ذخیره می‌شود. شکل (۳-۳-۲-۲) مراحل انجام این بخش را نشان می‌دهد.

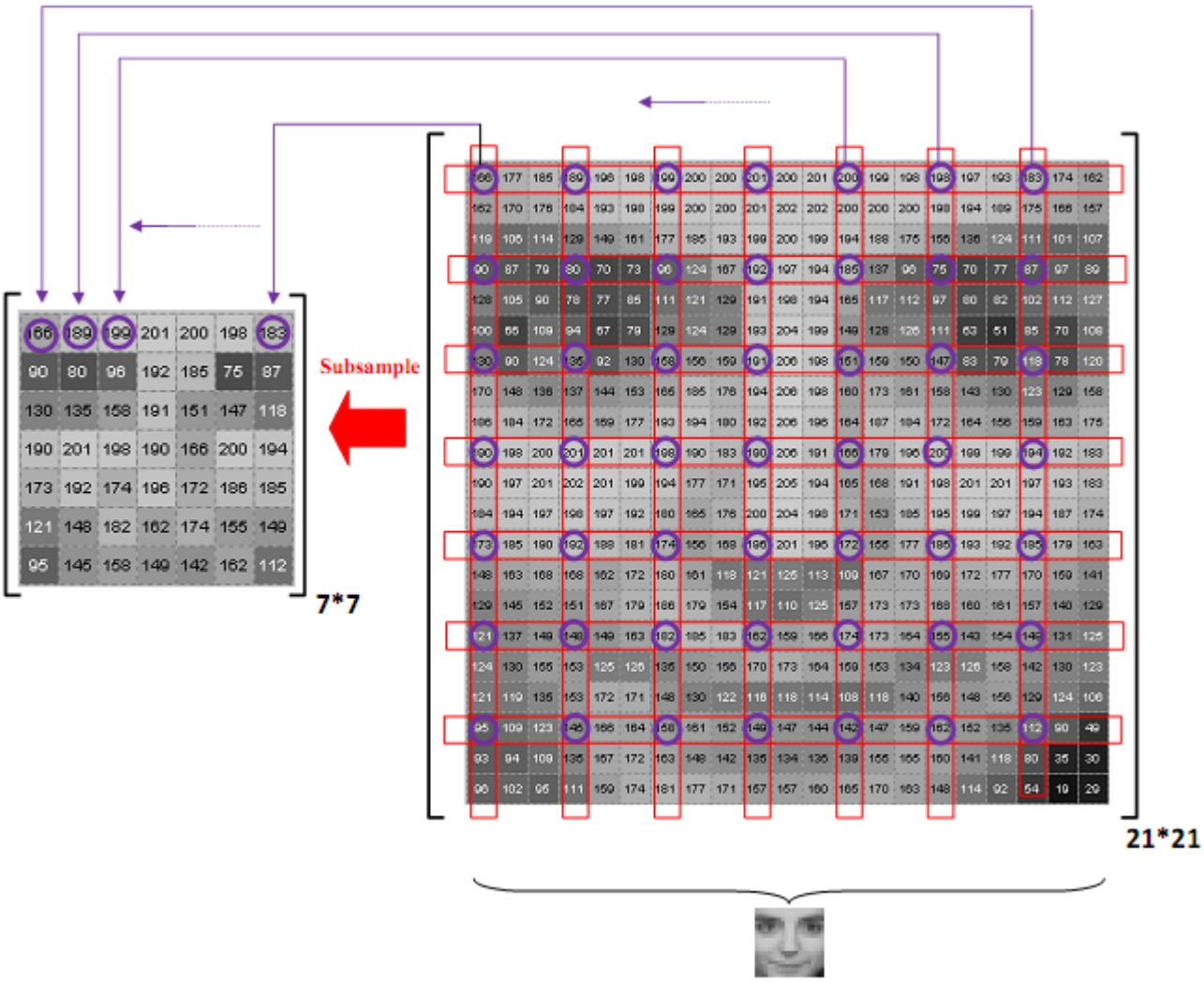


شکل(۳-۳-۲-۲). محاسبات max به صورت افقی

پس از پایان یافتن مراحل فوق و محاسبه max به‌صورت فوق به تابع C1.m بازگشته مقدار max به‌روش فوق، برای چهار تصویری که از مقایسه دو مقیاس ۱۱ و ۱۳ حاصل آمده بود محاسبه می‌گردد (به شکل(۳-۲-۲) مراجعه شود) و در c1 ذخیره می‌شود.

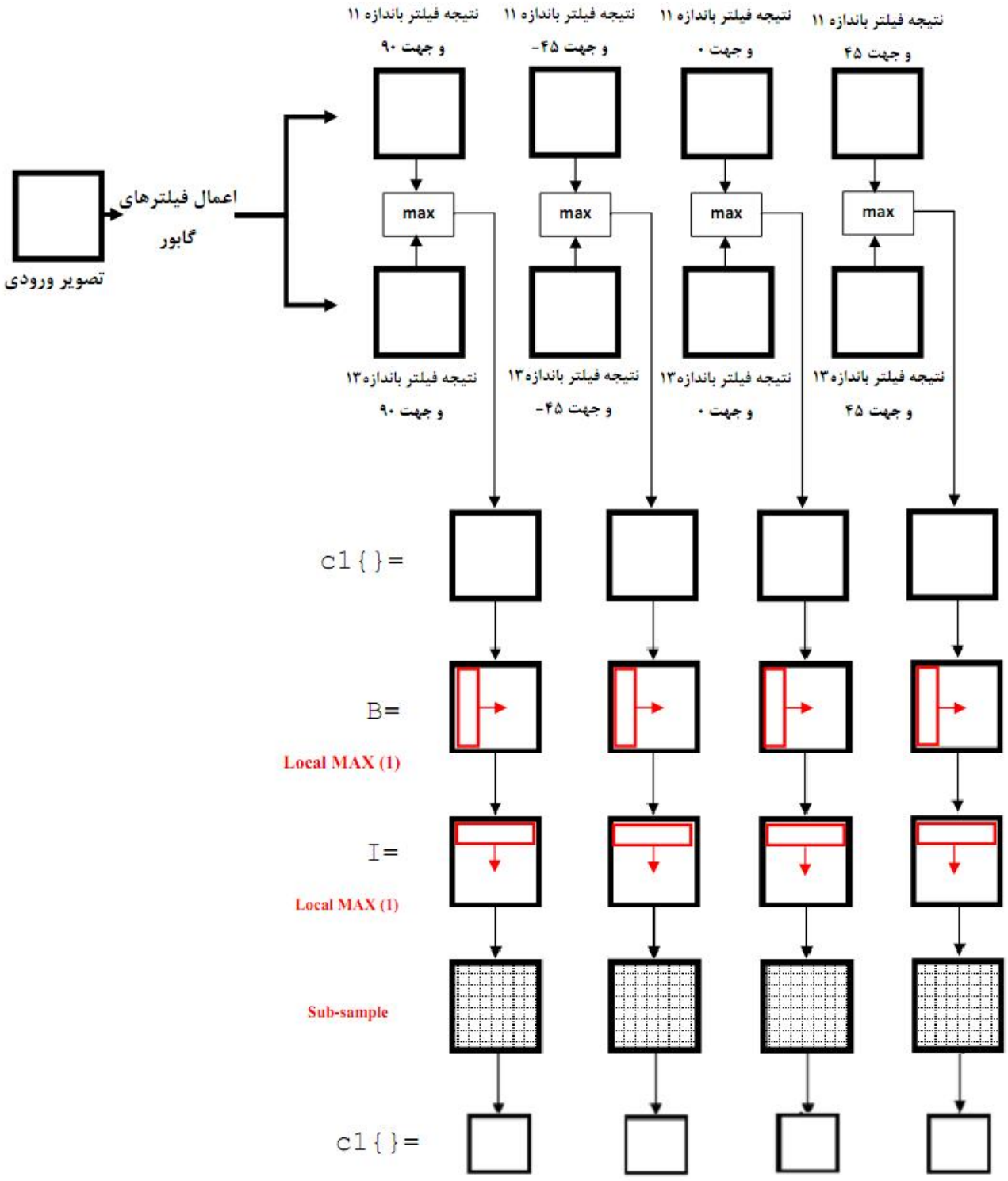
در پایان تابع C1.m به مرحله subsampling می‌رسیم در این مرحله که خطوط آن در زیر آمده است با توجه به مقدار متغیر c1SpaceSS (که در این مرحله ۱۰ است) تصویر خروجی از مرحله قبل را با توجه به شکل(۴-۲-۲) subsample می‌کنیم با انجام این کار اندازه تصاویر کوچک شده و در نهایت در c1{} ذخیره می‌شود.

```
%   (3) subsample
for iBand = 1:numScaleBands
    sSS=ceil(c1SpaceSS(iBand)/c1OL);
    clear T;
    for iFilt = 1:numSimpleFilters
        T(:,:,iFilt) = c1{iBand}(1:sSS:end,1:sSS:end,iFilt);
    end
    c1{iBand} = T;
end
```



شکل(۴-۲-۲). نحوه subsampling به‌عنوان مثال(بدون توجه به مقدار c1SpaceSS).

در این‌جا کار تابع C1.m به‌پایان رسید و خروجی این تابع S1 و C1 می‌باشد. برای جمع‌بندی مراحل، شکل (۵-۲-۲) را در نظر بگیرید توجه کنید که این مراحل تنها برای یک تصویر ورودی است.



شکل(۵-۲-۲).جمع بندی مراحل در C1.m.

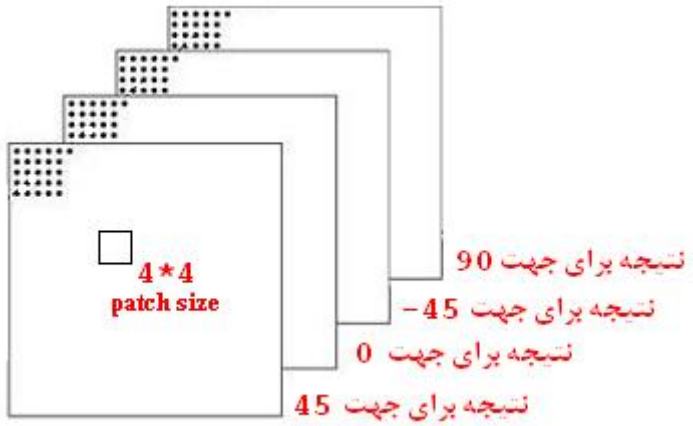
در این‌جا کار تابع C1.m پایان یافته است و به تابع extractRandC1Patches.m باز می‌گردیم. مراحل بعدی استخراج Patch ها از نتایج حاصل از تابع C1.m است که در نهایت درون cPatches{} ذخیره می‌شود. توجه شود که خطوط مربوط به این

مراحل در تابع extractRandC1Patches.m در زیر مشخص شده است.

.

```
for i = 1:numPatchesPerSize,
    ii = floor(rand*nImages) + 1;
    fprintf(1,'.');
    stim = cItrainingOnly{ii};
    img_siz = size(stim);
    [c1source,s1source] = C1(stim, filters, fSiz, c1SpaceSS,c1ScaleSS, c1OL);
    b = c1source{1}; %new C1 interface;
    bsize(1) = size(b,1);
    bsize(2) = size(b,2);
    for j = 1:numPatchSizes,
        xy = floor(rand(1,2).*(bsize-patchSizes(j)))+1;
        tmp = b(xy(1):xy(1)+patchSizes(j)-1,xy(2):xy(2)+patchSizes(j)-1,:);
        pind(j) = pind(j) + 1;
        cPatches{j}(:,pind(j)) = tmp(:);
    end
end
```

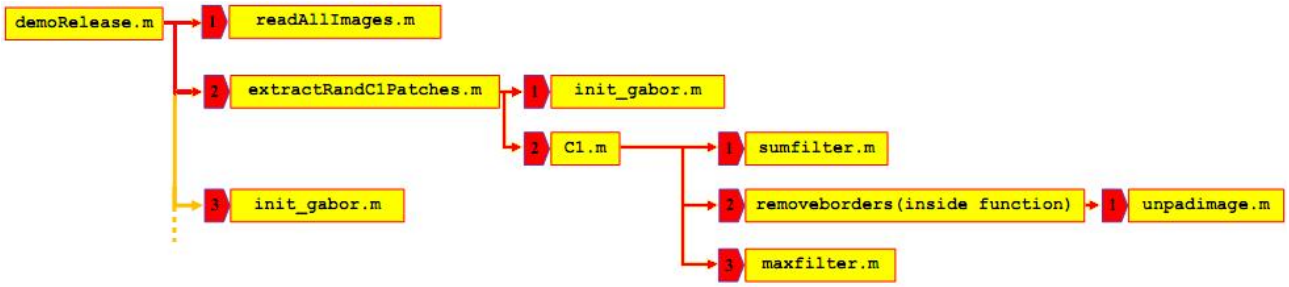
این مراحل (انتخاب Patch ها یا تکه‌های تصویر) به‌صورت کاملا تصادفی بوده و تکه‌های تصویر به‌صورت تصادفی برای چهار انـدازه Patch یعنی [4,8,12,16] انتخاب می‌شوند و درون cPatches{} قرار می‌گیرند. پس برای یک تصـویر چهـار Patch در انـدازه‌هـای [4,8,12,16] به‌صورت تصادفی انتخاب می‌گردد. حلقه for اولی ۲۵۰ بار تکرار شده که یعنی ۲۵۰ تصویر به‌صورت تصادفی از داده‌هـای آموزش از نوع train_set.pos انتخاب می‌گردد و برای هر تصویر به‌صورت جداگانه چهار Patch با اندازه‌های[4,8,12,16] بـرای ۴ جهت انتخاب می‌شود، یعنی برای یک تصویر ۱۶Patch خواهیم داشت. شکل(۱-۲) مرحله انتخاب تصادفی Patchها رانشان می‌دهد.



شکل(۱-۲). استخراج تصادفی Patchها، مربع در هر مکانی در تصویر ممکن است قرار گیرد(مثال برای Patch با اندازه ۴*۴).

باید توجه کرد که بعد از subsample در تابع C1.m چهار تصویر خروجی حاصل آمد(برای۴جهت) که برای انتخاب Patch ها باید از چهار تصویر به صورت شکل (۱-۲) استفاده کرد. به‌عنوان مثال۴ Patch با اندازه ۴ استخراج کرده کـه پـس از reshape کـردن درون cPatches{} که ساختار آن در شکل (۲-۱-۲) نشان داده شده قرار می‌گیرد.

در این مرحله کار تابع extractRandC1Patches.m نیز به‌پایان می‌رسد و تمامی Patchها استخراج می‌شوند. خلاصه توابع استفاده شده در این مرحله با رنگ قرمز(فلش‌ها و خطوط قرمز) در شکل(۲-۲) مشخص شده.

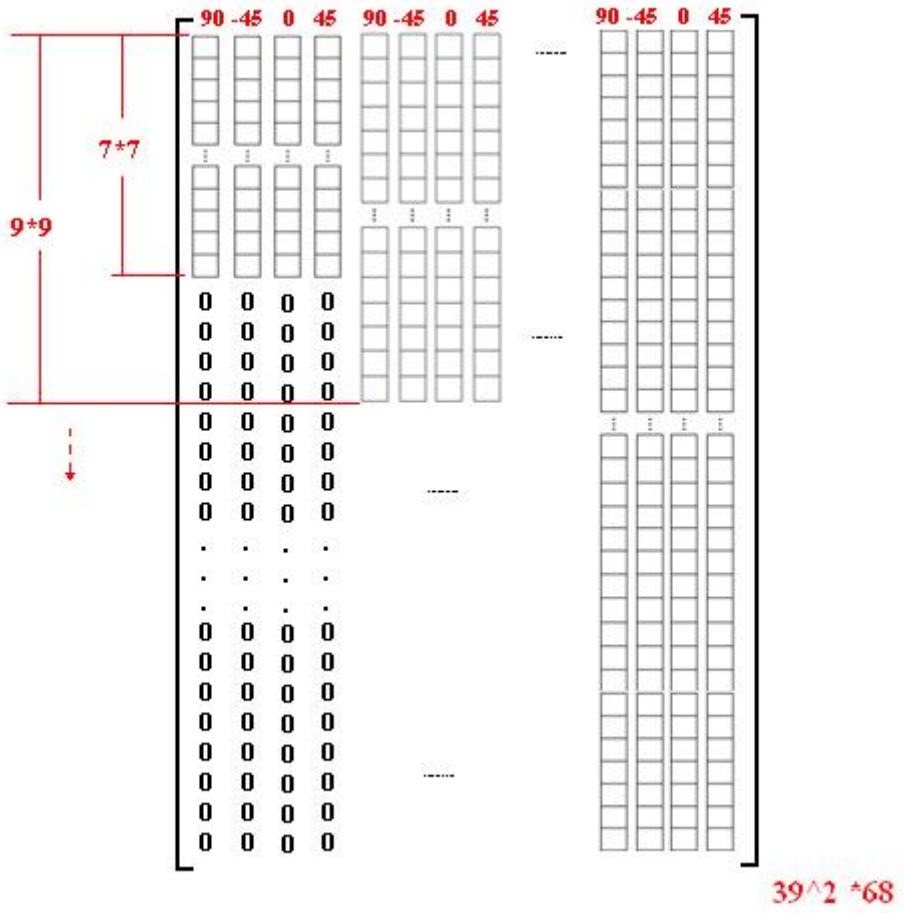


شکل(۲-۲). خلاصه توابع استفاده شده برای استخراج Patchها.

پس از اتمام مرحله استخراج Patchها به فایل اصلی `demoRelease.m` باز خواهیم گشت مرحله بعدی که در زیر آمده ساخت فیلترها برای تمامی اندازه‌ها و جهت‌ها می‌باشد. به نوعی فیلترهای جدول (۲-۲) به‌طور کامل ساخته می‌شوند. در این صورت ما ۶۴ فیلتر خواهیم داشت. نحوه تنظیم و انتخاب پارامترها در خطوط زیر نشان داده شده است.

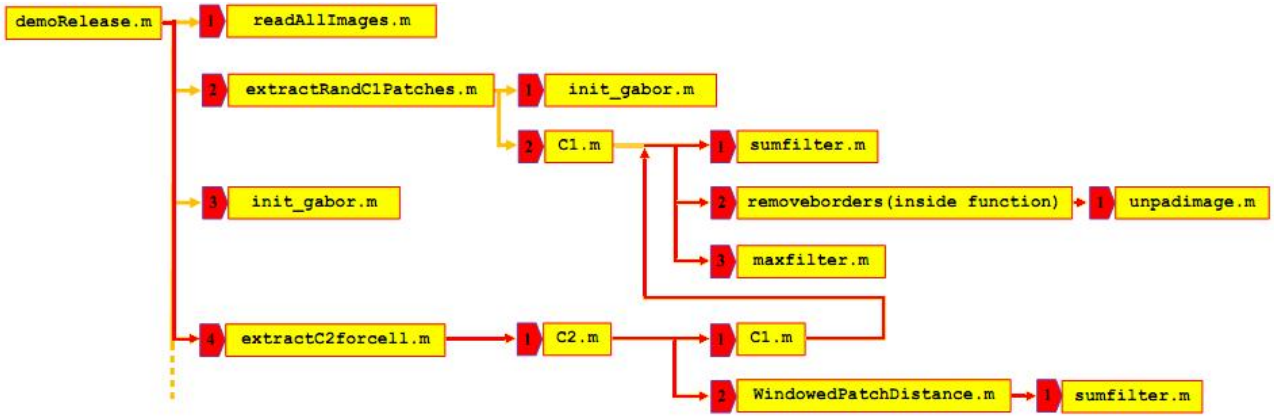
```
%----Settings for Testing --------%
rot = [90 -45 0 45];
c1ScaleSS = [1:2:18];
RF_siz    = [7:2:39];
c1SpaceSS = [8:2:22];
minFS     = 7;
maxFS     = 39;
div = [4:-.05:3.2];
Div       = div;
%--- END Settings for Testing --------%
```

بعد از انتخاب پارامترها دوباره تابع `init_gabor.m` فراخوانی می‌شود، توضیحات مربوط به این تابع در بخش ۱-۲ آمده، تنها تفاوت این مرحله تعداد فیلترهایی است که خروجی تابع `init_gabor.m` در این مرحله می‌سازد. با توجه به پارامترهای فوق خروجی تابع `init_gabor.m` در این مرحله یک ماتریس با تعداد سطر $۳۹^۲$ و تعداد ستون ۶۸ می‌باشد که در شکل(۲) نشان داده شده است. اندازه فیلترها از ۷ شروع شده و تا ۳۷ ادامه می‌یابد که در چهار جهت [45 0 45-90] می‌باشند در این صورت ما برای هر اندازه فیلتر چهار فیلتر خواهیم داشت. اندازه فیلترها در بردار `fSiz[]` ذخیره می‌شوند.



شکل(۲). ماتریس `filters`.

ساختن فیلترها برای تمامی مقیاس‌ها و جهت‌ها با فراخوانی تابع `init_gabor.m` به‌پایان می‌رسد. مرحله بعدی محاسبه و استخراج ویژگی‌های C2 برای تمامی تصاویر آموزش و تست از نوع `train_set.pos` و `train_set.neg` و `test_set.pos` و `test_set.neg` است. این مرحله بافراخوانی تابع `extractC2forcell.m` آغاز می‌شود. برای فهم بهتر ادامه مطالب ترتیب فراخوانی توابع در تابع `extractC2forcell.m` به‌صورت یک نمودار در شکل(۳) آمده که توضیحات مربوط به هر بخش در ادامه داده خواهد شد. (خطوط قرمز رنگ).



شکل (۳). ترتیب فراخوانی توابع در تابع `extractC2forcell.m`.

درون تابع extractC2forcell.m ابتدا تابع C2.m فراخوانی می‌شود درون تابع C2.m دوباره تابع C1.m فراخوانی شـده، ولی با آرگومان‌های ورودی متفاوت که در ادامه توضیح داده می‌شوند. فراخوانی تابع C1.m فراخوانی سه تـابع unpadimage.m, sumfilter.m, maxfilter.m را به‌دنبال دارد که عملکرد سه تابع قبلا توضیح داده شده. بعد از اتمام کـار تـابع C1.m تـابع بعدی که در C2.m فراخوانی می‌شود WindowedPatchDistance.m است و درون تابع WindowedPatchDistance.m مجددا تابع sumfilter.m با آرگومان‌های ورودی متفاوت فراخوانی می‌شود.

۳. تابع extractC2forcell.m

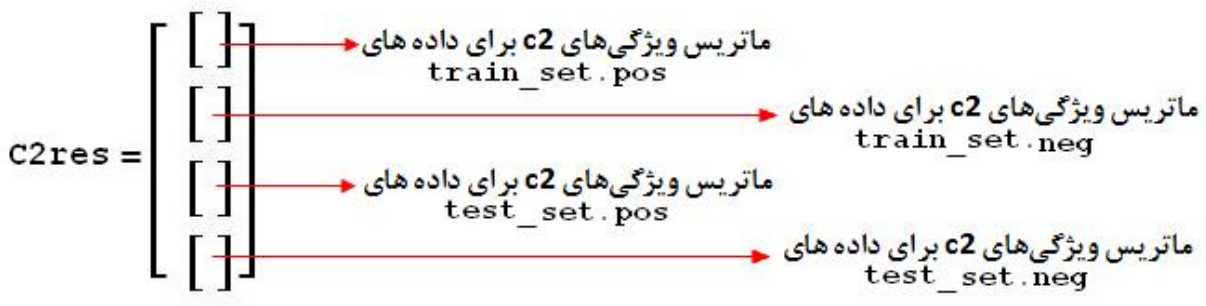
mC2=extractC2forcell(filters,fSiz,c1SpaceSS,c1ScaleSS,c1OL,cPatches,cImages, numPatchSizes)		
Input arguments	filters:	یک ماتریس که تمامی فیلتر ها را بعد از reshape کردن در خود جای می‌دهد(اندازه آن در این مرحله ۶۸*۳۹۲ است).
	fSiz:	برداری که تعداد دارایه‌های آن برابر تعداد فیلترهاست(۶۸*۱).
	c1SpaceSS	برداری که محدوده pooling را مشخص می‌نماید [8:2:22].
	c1ScaleSS	بردار scaleها [1:2:18].
	c1OL:	متغیر با مقدار ۲.
	cPatches:	سلول patchهای استخراج شده از دادهای train در مراحل قبل
	cImages:	تصاویر سلول i ام cI{i} (تصاویر ورودی).
	numPatchSizes:	تعداد جهت‌های استفاده شده برای ساخت فیلترها که ۴ جهت می‌باشد.
output arguments	mC2:	ماتریس نتایج استخراج C2 برای یک سلول از cI{}.

جدول(۱-۳)

حلقه forی که در زیر آمده (در تابع demoRelease.m) ۴ بار تکرار می‌شود. هر بار تکرار برای یک دسته از داده‌هایی است که در cI{i} ذخیره شده، به‌این معنی که بار اول داده‌های train_set.pos خوانده شده و مراحل شکل (۳) انجام می‌شود. بار دوم داده‌های train_set.neg خوانده می‌شود و در ۲ تکرار باقی مانـده داده‌هـای test_set.pos و test_set.neg خوانـده می‌شود.

```
for i = 1:4,
  C2res{i} =
extractC2forcell(filters,fSiz,c1SpaceSS,c1ScaleSS,c1OL,cPatches,cI{i},numPatchSizes);
end
```

پس از استخراج ویژگی C2 آنها را درون C2res{i} ذخیره می‌کنیم که ساختار آن در شکل(۱-۳) نشان داده شده است.



شکل(۱-۳).

نقش تابع extractC2forcell.m استخراج C2 برای هر تصویر در cI{} است یعنی در این تابع برای تمامی تصاویر موجود در یک سلول cI{} ویژگی‌های C2 استخراج می‌گردد. فراخوانی مجدد این تابع، C2 را برای تصاویر موجود در سلول دیگر cI{} استخراج می‌کند.

جهت بررسی نحوه عملکرد این تابع به خود تابع مراجعه کرده. خطوطی که در ابتدای تابع آمده‌اند و در زیر نشان داده شده‌اند تنها نقش flip کردن داده‌های موجود در cPatches{} را دارند این کار برای استفاده از دستور conv2 در MATLAB انجام شده زیرا دستور conv2 بسیار سریع‌تر از دستور filter2 عمل می‌کند. برای فهم بهتر flip شدن شکل(۲-۳) در نظر بگیرید.

```
numPatchSizes = min(numPatchSizes,length(cPatches));
%all the patches are being flipped. This is becuase in matlab conv2 is much
faster than filter2
for i = 1:numPatchSizes,
    [siz,numpatch] = size(cPatches{i});
    siz = sqrt(siz/4);
    for j = 1:numpatch,
        tmp = reshape(cPatches{i}(:,j),[siz,siz,4]);
        tmp = tmp(end:-1:1,end:-1:1,:);
        cPatches{i}(:,j) = tmp(:);
    end
end
```

$$a = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 6 & 7 & 8 & 9 & 10 \\ 11 & 12 & 13 & 14 & 15 \\ 16 & 17 & 18 & 19 & 20 \\ 21 & 22 & 23 & 24 & 25 \end{bmatrix} \Rightarrow f = \begin{bmatrix} 25 & 24 & 23 & 22 & 21 \\ 20 & 19 & 18 & 17 & 16 \\ 15 & 14 & 13 & 12 & 11 \\ 10 & 9 & 8 & 7 & 6 \\ 5 & 4 & 3 & 2 & 1 \end{bmatrix}$$

شکل(۲-۳). نحوه flip شدن

که پس از انجام مرحله فوق دوباره آنها را در cPatches{} ذخیره کرده. گام بعدی خواندن یکایک تصاویر در سلول i ام cI{i} و استخراج C2 است. خلاصه مراحل به صورت زیر است.

۱. یک تصویر در سلول i ام cI{i} خوانده می‌شود.
۲. تابع C2.m فراخوانی می‌شود. این تابع ۴ بار برای اندازه Patch های [4,8,12,16] در یک حلقه for فراخوانی شده و نتایج در iC2[] ذخیره می‌شود(برای ۴ اندازه Patch، [4,8,12,16]).
۳. نتایج به‌دست آمده در ۲ دورن mC2[] ذخیره می‌شود.
۴. برو به گام ۱.

حلقه فوق به تعداد تصاویر موجود در سلول سلول i ام cI{i} تکرار می‌شود. و خروجی تابع extractC2forcell.m که mc2[] می‌باشد را تولید می‌کند. نحوه استخراج C2 (بررسی چگونگی عملکرد تابع c2.mو توابع دیگر در ادامه توضیح داده می‌شود).

۳.۱. تابع c2.m

[tmpC2,tmp,c1] = C2(stim,filters,fSiz,c1SpaceSS,c1ScaleSS,c1OL,cPatches{j})		
Input arguments	stim:	یکی از تصاویر سلول i ام cI{i} (تصاویر ورودی).
	filters:	یک ماتریس که تمامی فیلترها را بعد از reshape کردن در خود دارد(اندازه آن در این مرحله ۶۸*۳۹^2 است).
	fSiz:	برداری که تعداد دارایه‌های آن برابر تعداد فیلترهاست(۶۸*۱).
	c1SpaceSS	برداری که محدوده pooling را مشخص می‌نماید [8:2:22].
	c1ScaleSS	بردار scaleها [1:2:18].
	c1OL:	متغیر با مقدار ۲.
	cPatches{i}:	واحد i ام سلول patchهای استخراج شده از دادهای train در مراحل قبل.
output arguments	tmpC2:	ماتریس نتایج استخراج C2 برای یکی از تصاویر سلول i ام cI{i} (تصاویر ورودی).
	tmp:	سلول نتایج S2 برای یکی از تصاویر سلول i ام cI{i} (تصاویر ورودی).
	c1:	نتایج C1 برای یکی از تصاویر سلول i ام cI{i} (تصاویر ورودی).

جدول(۱-۱-۳)

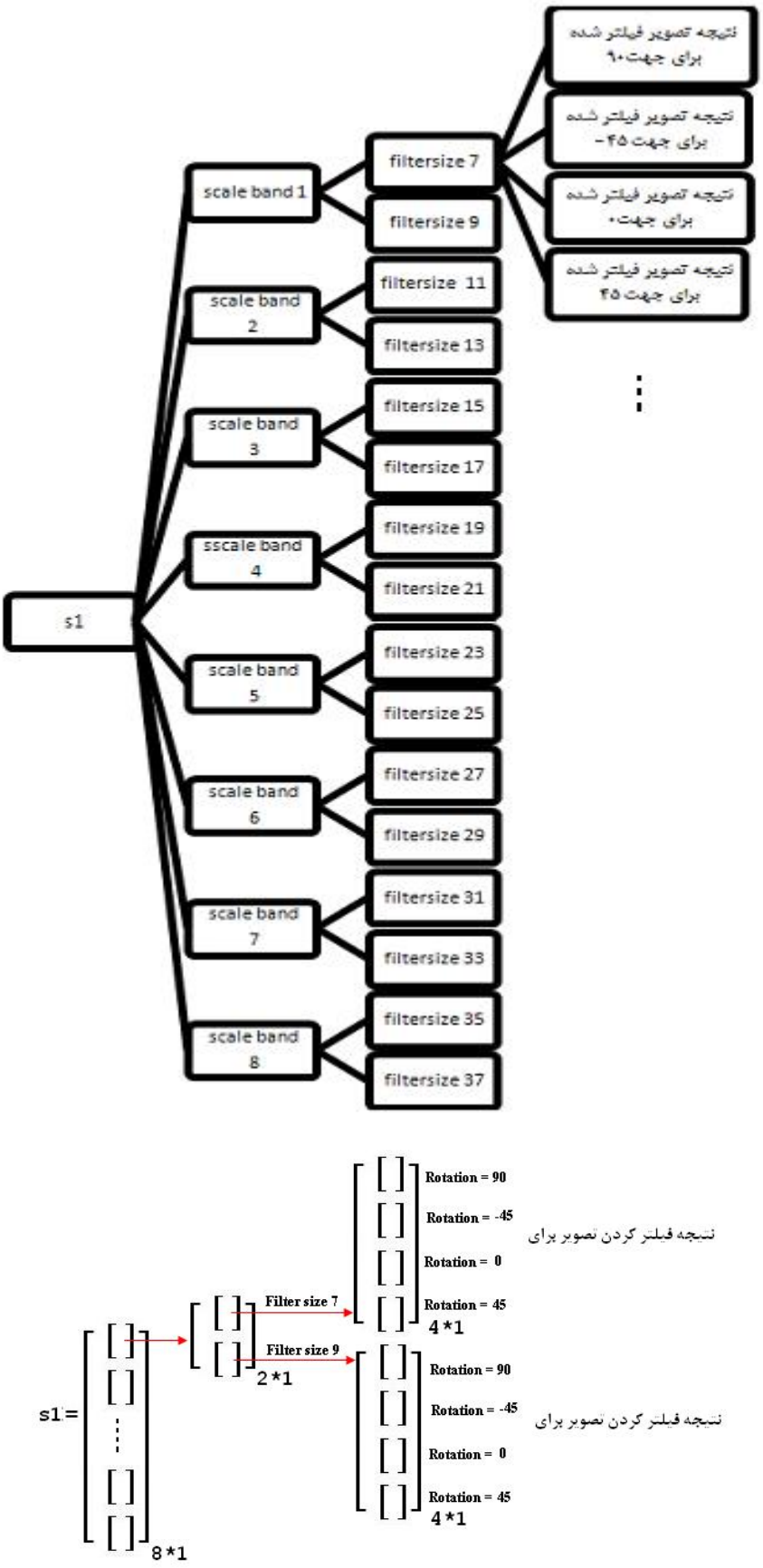
همان‌طور که در توضیحات مربوط به تابع extractC2forcell.m آمده، تابع C2.m برای هر تصویر موجود در سلول i ام cI{i} چهار بار (برای اندازه Patchها [4,8,12,16] فراخوانی می‌گردد. پس ورودی تابع یک تصویر است و خروجی ویژگی‌های C2. در ابتدای تابع C2.m دوباره تابع C1.m فراخوانی می‌شود، با این تفاوت که آرگومان‌های ورودی متفاوتی نسبت به قبل دارد. توضیحات مربوط C1.m در بخش ۲-۲ آمده است. در این‌جا تنها به آرگومان‌های ورودی و خروجی آن اشاره می‌شود.

آرگومان‌های ورودی C1.m در این مرحله :

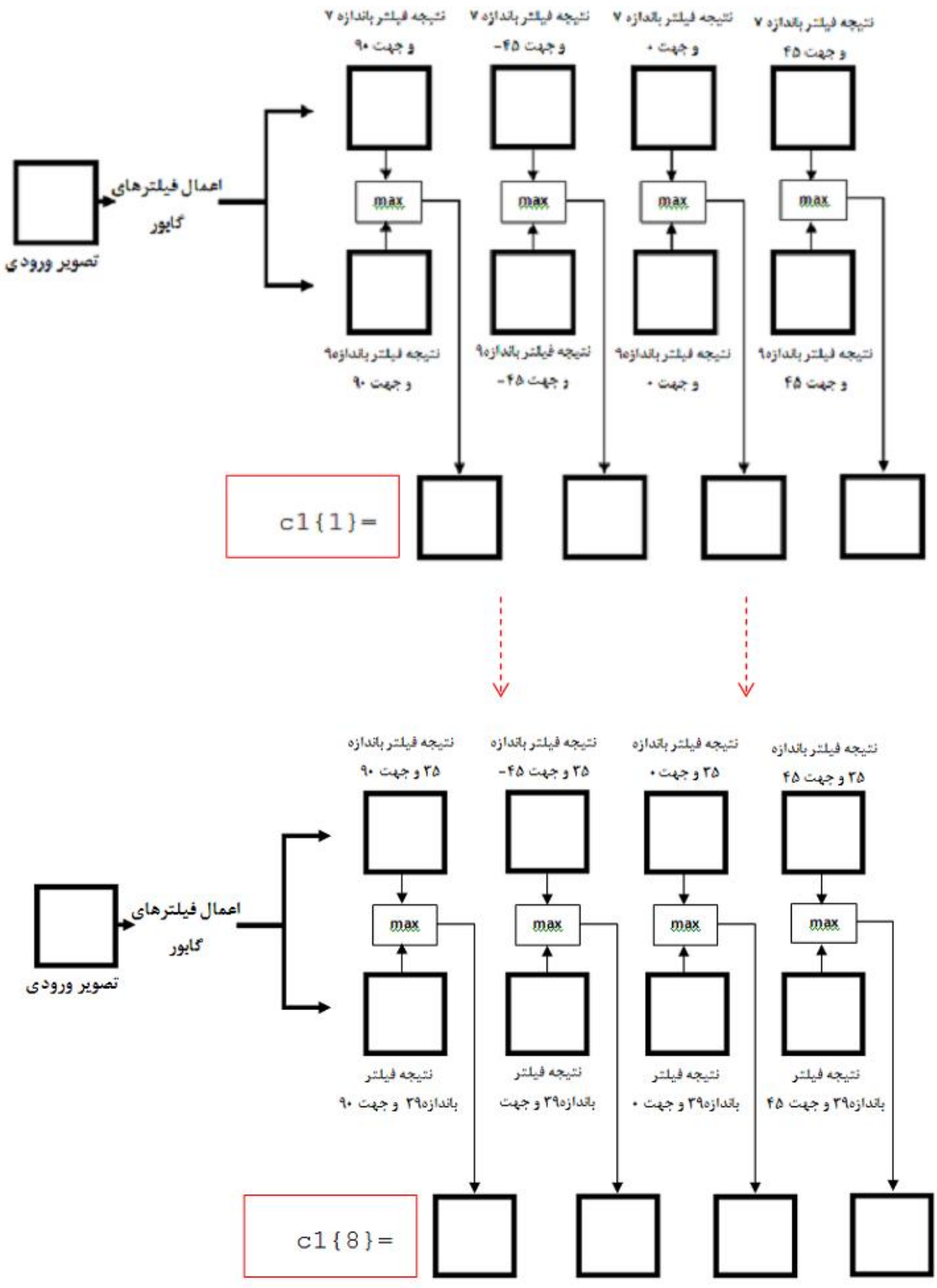
همان‌طور که مشخص است تعداد فیلترها و اندازه آنها و همچنین متغیرهای c1ScaleSS وc1SpaceSS تغییر کرده است.

آرگومان‌های خروجی C1.m در این مرحله :

تابع C1.m دو آرگومان خروجی دارد، S1 و C1 که نتایج تصویر فیلتر شده و max گیری شده‌اند. در این‌جا تنها تعداد تصاویر خروجی متفاوت است زیرا تعداد فیلترها و مقیاس آنها افزایش یافته، شکل(۱-۱-۳) و (۲-۱-۳) خروجی تابع C1.m را در این مرحله نشان می‌دهد.

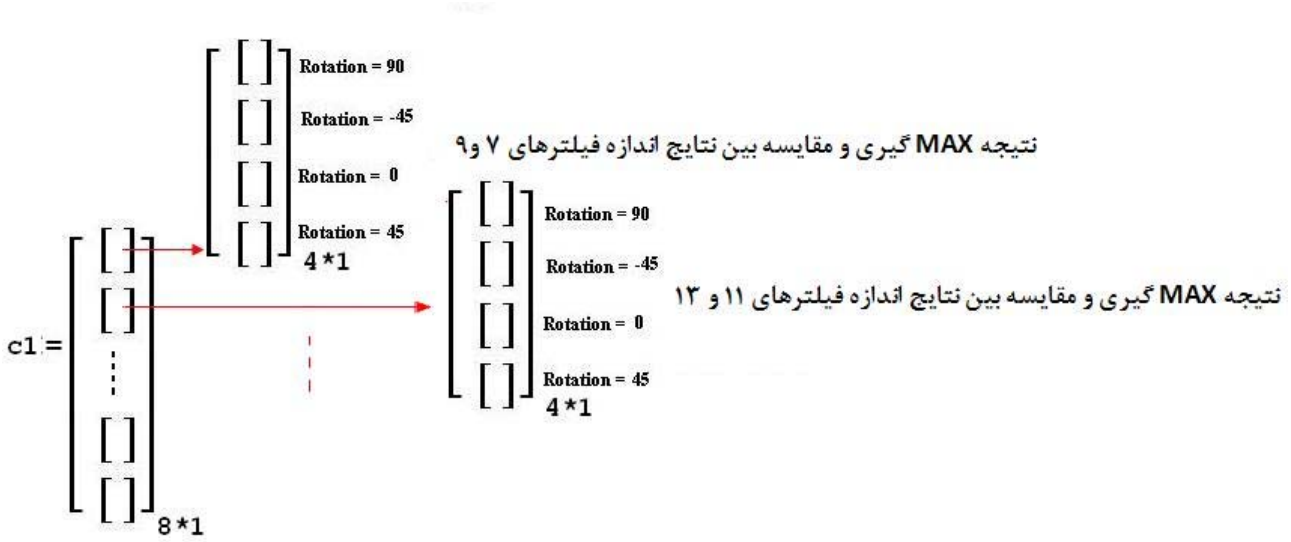


شکل(۱-۱-۳).خروجی تابع C1.m در این مرحله برای آرگومان S1.



شکل(۲-۱-۳).خروجی تابع `c1.m` در این مرحله برای آرگومان `c1`.

نتایج c1 بعد از subample شدن با Spaceهای متفاوت [8,10,12,14,16,18,20,22] (نحوهsubample شدن در بخش ۲-۲ آمده)دوباره در c1 ذخیره می‌شوند، پس ساختار خروجی تابع C1.m در این مرحله به‌صورت شکل(۳-۱-۳) می‌باشد.

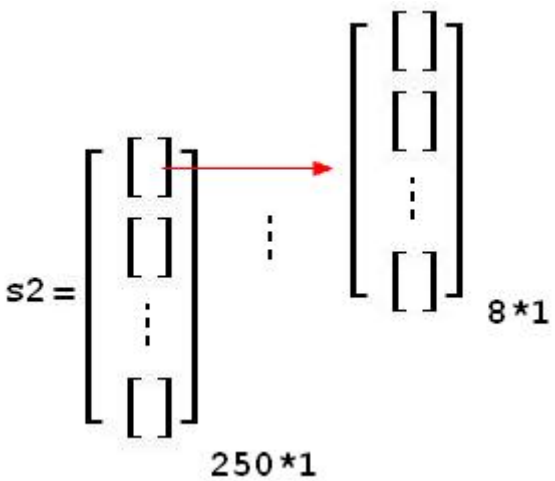


شکل(۳-۱-۳). ساختار خروجی تابع C1.m در این مرحله برای آرگومان خروجی c1.

بعد از اتمام مراحل فوق به تابع C2.m بازگشته مرحله بعدی ساخت S2 می‌باشد. همان‌طور که در مقالات آمده S2 همانند یک شبکه RBF است(مراکز یا centerهای شبکه)، که در این‌جا باید centerهای این شبکه استخراج گردد. این کار با کمک Patchهایی که از داده‌های train_set.pos به‌دست آمده و در cPatches{} ذخیره شده انجام می‌شود. در ابتدا برای Patchها با اندازه ۴ و در فراخوانی بعدی C2.m بترتیب برای ۸و۱۲و۱۶ این کار صورت می‌گیرد.

در ابتدا یک سلول ۱*۲۵۰ با استفاده از دستورات زیر ساخته می‌شود. این سلول برای ذخیره‌سازی s2 ها می‌باشد. پس از آن وارد فاز محاسبه s2 می‌شویم برای این‌کار ابتدا در هر سلول s2 یک سلول ۱*۸ می‌سازیم. شکل(۴-۱-۳).

```
s2 = cell(n_rbf_centers,1);
```



شکل(۴-۱-۳). ساختار سلول s2.

پس از آن تابع WindowedPatchDistance.m فراخوانی می‌شود. این تابع ۸ بار برای پر کردن ۸ سلول نشان داده شده در شکل (۴-۱-۳) فراخوانی می‌گردد.

۳.۱.۱. تابع `WindowedPatchDistance.m`

`D = WindowedPatchDistance(Im,Patch)`		
Input arguments	`Im:`	یکی از سلول‌های ۸ گانه `c1` که در مرحله قبل بدست آمد.
	`Patch:`	یکی از patchهای استخراج شده از دادهای train در مراحل قبل(مثلا یک ماتریس (4*4*4)).
output arguments	`D:`	نتیجه محاسبه فاصله اقیلدسی .

جدول(۱-۱-۱-۳)

همان‌طور که قبلا ذکر شد `c1` یک سلول با طول ۸ است که در هر واحد آن ۴ تصویر upsample شده با مقدارهای متفاوت برای چهار جهت [90,-45,0,45] وجود دارد، در هر بار فراخوانی تابع `WindowedPatchDistance.m` یکی از این واحدهای `c1` خوانده می‌شود (یعنی یک ۴ تصویر) و یک Patch با اندازه مثلا A و وجهت‌های [90,-45,0,45] که در تابع `C1.m` ساخته شده بود نیز خوانده می‌شود.

به‌طور کلی نقش این تابع محاسبه فاصله اقیلدسی بین Patchخوانده شده و تصویرهای واحد `i`ام `c1{}` است. که به‌طور کلی رابطه زیر را دارد.

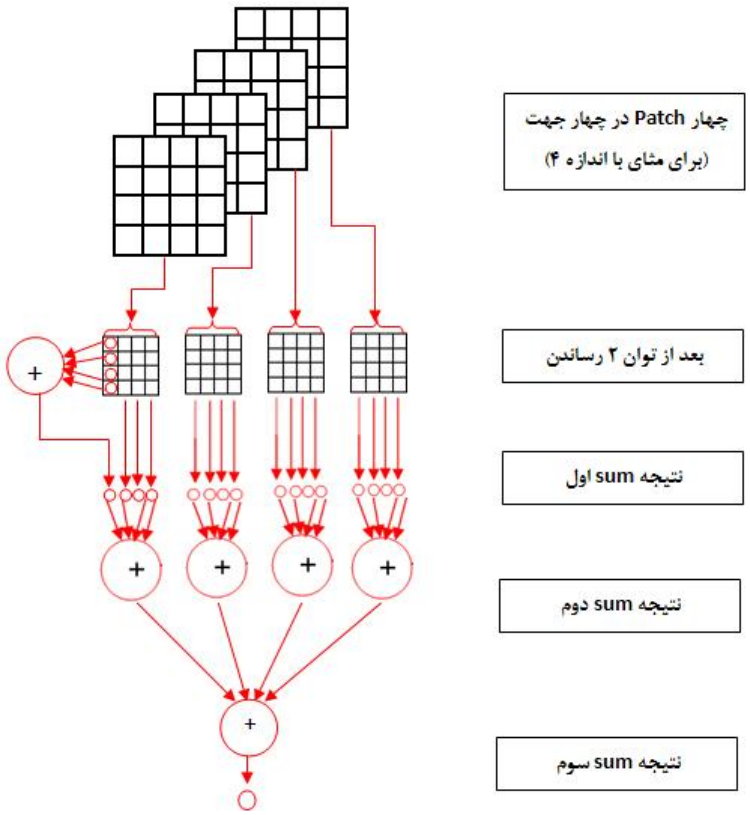
```
% sum_over_p(W(p)-I(p))^2 is factored as
% sum_over_p(W(p)^2) + sum_over_p(I(p)^2) - 2*(W(p)*I(p));
```

علامت ⁕ عمل کانولوشن را نشان می‌دهد.

مراحل کار به‌صورت زیر است :

۱. در ابتدا مقدار `sum_over_p(W(p)^2)` با دستور زیر محاسبه می‌گردد. این دستور به صورت گرافیکی در شکل (۱-۱-۱-۳) نشان داده شده است.

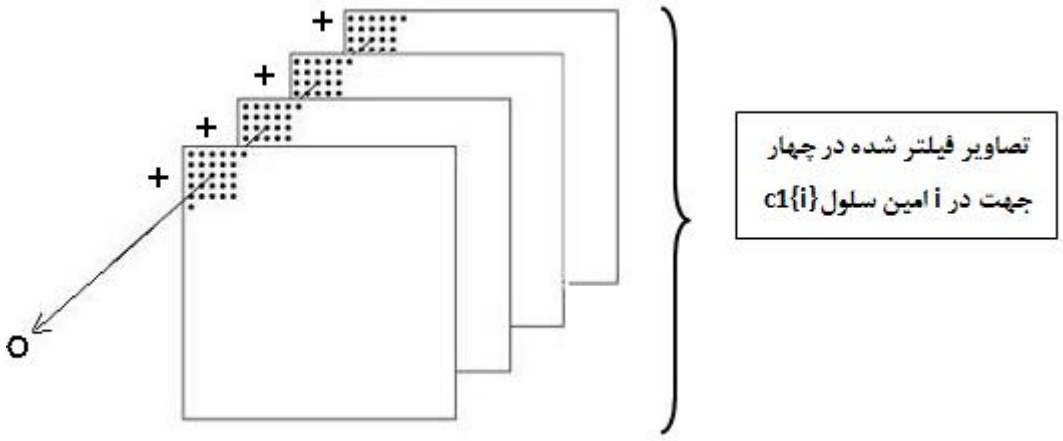
```
Psqr = sum(sum(sum(Patch.^2)));
```



شکل (۱-۱-۱-۳). نحوه محاسبه `sum_over_p(W(p)^2)`.

۲. سپس مقدار sum_over_p(I(p)^2) با دستور زیر محاسبه می‌گردد. (به توضیحات sum(A,dim) در MATLABمراجعه گردد). شکل(۲-۱-۱-۳).

```
Imsq = Im.^2;
Imsq = sum(Imsq,3);
```



شکل(۲-۱-۱-۳).

۳. بعد از آن تابع sumfilter.m فراخوانی می‌شود (توضیحات آن در بخش ۱-۲-۲داده شده).

```
sum_support = [ceil(s(2)/2)-1,ceil(s(1)/2)-1,floor(s(2)/2),floor(s(1)/2)];
Imsq = sumfilter(Imsq,sum_support);
```

۴. مرحله بعد محاسبه کانولوشن (W(p)*I(p)) است یعنی هر تصویر در Im با یک Patch با جهت یکسان با آن کانوال می-شود و نتیجه در PI ذخیره می‌گردد.

```
PI = zeros(size(Imsq));
for i = 1:dIm
    PI = PI + conv2(Im(:,:,i),Patch(:,:,i), 'same');
end
```

۵. نتیجه نهایی با استفاده از خط آخر تابع بدست می‌آید.

```
D = Imsq - 2 * PI + Psqr + 10^-10;
```

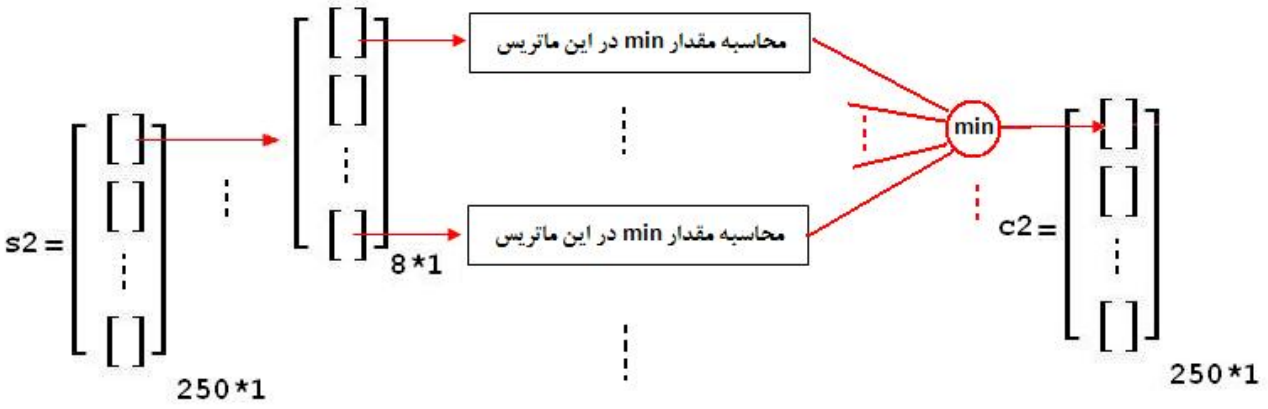
همان‌طور که قبلا بیان شد تابع WindowedPatchDistance.m ۸ مرتبه در هر سلول s2 فراخوانی می‌شود تمام سلول‌های s2 به این ترتیب پر می‌شوند. پس از پایان این مرحله به تابع C2.m باز می‌گردیم.

مرحله بعدی محاسبه ویژگی‌های C2 است. این مرحله با محاسبه حداقل فاصله بین مراکز و ورودی تعیین می‌گردد. خطوط مربوط به این مرحله در زیر آمده.

```
%Build c2:
% calculate minimum distance (maximum stimulation) across position and scales
c2 = inf(n_rbf_centers,1);
for iCenter = 1:n_rbf_centers
    for iBand = 1:nbands
        c2(iCenter) = min(c2(iCenter),min(min(s2{iCenter}{iBand})));
```

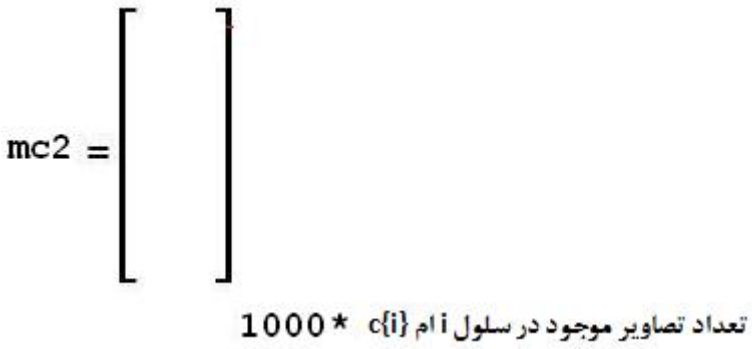
```
    end
end
```

در ابتدا یک ماتریس ۱*۲۵۰ با مقدار درایه‌های `inf` ساخته می‌شود به‌نام `c2` سپس مقدار `min` در بین هر واحد `s2` که شامل ۸ ماتریس حاصله از مراحل قبل است محاسبه شده این مقدار `min` درون `c2` قرار می‌گیرد. شکل(۳-۱-۱-۳).



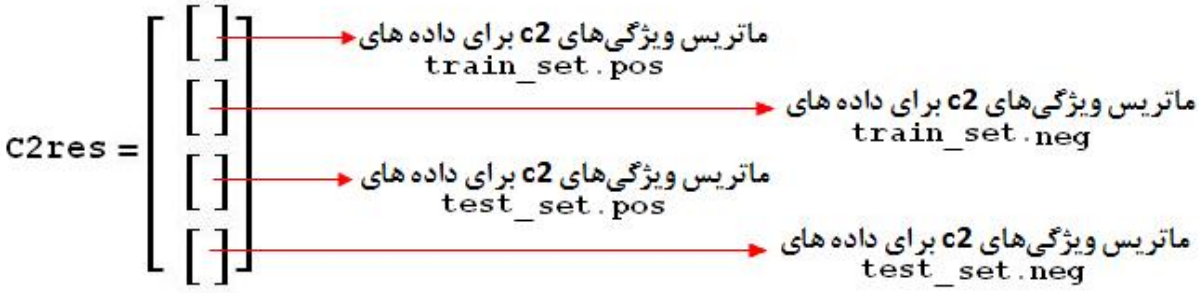
شکل(۳-۱-۱-۳). محاسبه مقدار `min` در بین هر واحد `s2` و ذخیره آن در `c2`.

پس از کامل شدن `c2` به تابع `extractC2forcell.m` بازگشته. خروجی `extractC2forcell.m` یک ماتریس است که ساختار آن در شکل(۴-۱-۱-۳) آمده.



شکل(۴-۱-۱-۳). ساختار ماتریس حاصل از خروجی `extractC2forcell.m`

بعد از پایان کار تابع `extractC2forcell.m` به تابع `demoRelease.m` بازگشته در این صورت `C2res{}` به‌صورت شکل(۵-۱-۱-۳) است.



شکل(۵-۱-۱-۳). ساختار `C2rees{}`

مرحله بعدی طبقه‌بندی C2 می‌باشد، با استفاده از توابع زیر.

```
LSnn.m    - Nearest Neighbor classifier train
CLSnnC.m  - Nearest Neighbor classifier test
```

```
CLSosusvm.m   - SVM train (a wrapper function for osusvm)
CLSosusvmC.m  - SVM test  (a wrapper function for osusvm)
```

تمامی توابع با جزئیات مطالع و بررسی شده(بجز توابع مربوط به طبقه‌بندی کننده‌ها).