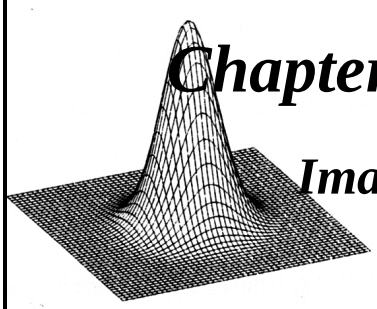


Chapter 6

Image Compression

فشرده‌سازی تصویر

پیش نمایش(*preview*)

در فشرده‌سازی تصویر مقدار داده‌های مورد نیاز برای نمایش تصویر دیجیتالی کاهش می‌یابد. برای انجام این کار یک یا چند عامل از داده‌های اضافی حذف می‌شوند:

(۱) افزونگی کدها(Coding redundancy) زمانی استفاده می‌شود که از حداقل کدهای مورد نیاز برای طول تصویر بهره‌برداری شود.

(۲) افزونگی بین عناصر تصویری(Interpixel redundancy): که نتیجه رابطه متقابل عناصر تصویری است

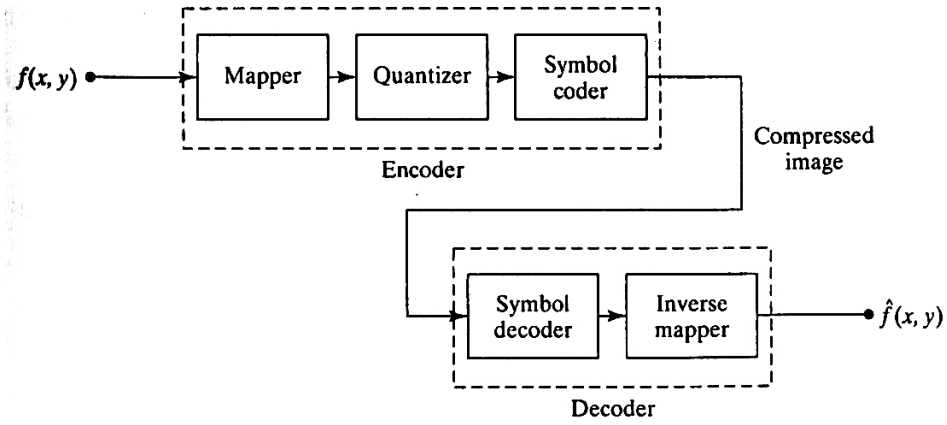
(۳) افزونگی روانی – تجسمی(Psychovisual redundancy): که با توجه به داده‌هایی که در چشم انسان به حساب نمی‌آیند انجام می‌شود (یعنی اطلاعاتی که از نظر دیداری ضروری نیست). در این فصل هر یک از این افزونگی‌ها را بررسی کرده و شگردهای بهره‌برداری از آنها را شرح می‌دهیم و ۲ استاندارد فشرده‌سازی با پسوند jpeg, jpeg2000 را شرح می‌دهیم. مفاهیم ارائه شده در اینجا در این استانداردها یکی می‌شوند و این شگردها با داده‌های ۳ نوع افزونگی فوق کار می‌کنند.

از آنجائی که جعبه ابزار پردازش تصویر تابعهایی برای فشرده‌سازی تصویر ندارد، نکته اصلی در این فصل روشهای کاربردی شگردهای فشرده‌سازی در نرم افزار MATLABاست. مثلاً یک تابعC فراخوان‌پذیر ایجاد می‌کنیم که نحوه مدیریت داده‌هایی با طول متغیر را در سطح بیت نشان می‌دهد. اهمیت این نکته از آن جهت است که رمزگذاری طول متغیر نکته اصلی فشرده‌سازی تصویر است ولی نرم افزارMATLAB برای پردازش ماتریسهایی حاوی داده‌هایی با طول ثابت مطلوب است. حین ابداع این تابع فرض شده است که خوانندگان از زبان C اطلاع دارند و نکته اصلی این مبحث کنش و واکنش نرم افزار MATLAB با برنامه‌های C و Fortran است که خارج از این نرم افزار اجرا می‌شوند. اهمیت این نکته در زمانی است که بخواهیم از تابع‌های M در برنامه‌های Fortran و C استفاده کنیم و نیاز به افزایش شتاب تابعهای M بردار‌بندی شده باشد. (مثلاً وقتی نتوان یک حلقه for را به قدر کافی بردار‌بندی کرد). تابعهای ابداع شده در این فصل و توانایی نرم افزار MATLAB برای کار با برنامه‌های C و Fortran طوری که تابعهای فوق به صورت متعارف و تعبیه شده به کار برده شوند حاکی از آن است که نرم افزار MATLAB ابزاری کارآمد برای ایجاد نمونه‌های اصلی الگوریتمها و سیستم‌های فشرده‌سازی است.

۶.۱. تاریخچه(Background)

همان طور که در تصویر ۶.۱ دیده می‌شود، سیستم‌های فشرده‌سازی تصویر از ۲ بلوک متمایز رمزگذار(Encoder) و رمزگشا(Decoder) تشکیل شده‌اند. تصویر f(x, y) به رمزگذار خورانده می‌شود و مجموعه‌ای از نمادها از داده‌های ورودی ایجاد می‌شود که از آنها برای نمایش تصویر استفاده می‌شود. اگر n1, n2 به ترتیب تعداد واحدهای حامل اطلاعات (بر حسب بیت) در تصویرهای اصلی و رمزگذاری شده باشند، فشرده‌سازی را می‌توان از طریق ارقام با نسبت فشرده‌سازی سنجید.

$$CR = \frac{n_1}{n_2}$$



تصویر ۶.۱: نموداری از سیستم فشرده‌سازی تصویر

وقتی نسبت فشرده‌سازی (۱۰:۱) باشد، یعنی تصویر اصلی در هر واحد در داده‌های فشرده شده ۱۰ واحد حامل اطلاعات دارد. در نرم افزار MATLAB نسبت تعداد بیتهای مورد استفاده برای نمایش ۲ پرونده تصویری و متغیرهای مربوطه با تابعهای M زیر محاسبه می‌شود.

```
function cr = imratio(f1, f2)
%IMRATIO Computes the ratio of the bytes in two images/variables.
%     CR = IMRATIO(F1, F2) returns the ratio of the number of bytes in
%     variables/ files F1 and F2. If F1 and F2 are an original and
%     compressed image, respectively, CR is the compression ratio.

error(nargchk(2, 2, nargin));       % Check input arguments
cr = bytes(f1) / bytes(f2);              % Compute the ratio

%--------------------------------------------------------------------------%
function b = bytes(f)
% Return the number of bytes in input f. If f is a string, assume
% that it is an image filename; if not, it is an image variable.
if ischar(f)
```

```
      info = dir(f);                b = info.bytes;
elseif isstruct(f)
      % MATLAB's whos function reports an extra 124 bytes of memory
      % per structure field because of the way MATLAB stores
      % structures in memory. Don't count this extra memory; instead,
      % add up the memory associated with each field.
      b = 0;
      fields = fieldnames(f);
      for k = 1:length(fields)
            b = b + bytes(f.(fields{k}));
      end
else
      info = whos('f');       b = info.bytes;
end
```

مثلاً فشرده‌سازی تصویری که با محتوای jpeg رمزگذاری شده است در عکس ۲.۴.سی فصل ۲ از طریق فرمولهای زیر قابل محاسبه است:

```
>> r = imratio(imread('bubbles25.jpg'), 'bubbles25.jpg')
r =
     35.1612
```

توجه داشته باشید که imratio تابع داخلی b=bytes(f) است که برای تولید تعداد بایتها در سه مورد زیر به کار می‌رود:

(۱) پرونده a file (۲) متغیرهای ساختاری a structure variable (۳) متغیر غیرساختاری a nonstructure variable.

اگر f متغیر غیرساختاری باشد، تابع whos که برای سنجش اندازه آن بر حسب بایت به کار برده می‌شود. اگر f اسم یک پرونده باشد، تابع dir کار مشابهی انجام می‌دهد. در ترکیب استفاده شده dir یک ساختار تولید می‌کند که field's name, date, bytes, isdir در آن مشخص شده است. آنها به ترتیب نام پرونده، تاریخ اجرای تغییرات، اندازه آن بر حسب بایت، و مسیر بودن آن را مشخص می‌کنند (اگر isdir ،۱ باشد مسیر است و اگر ۰ باشد مسیر نیست) اگر f یک ساختار باشد، bytes مکرراً خود را فرامی‌خواند تا تعداد بایتهای تخصیص داده شده به هر میدان ساختار جمع بسته شود. بنا بر این بالاسری متغیر ساختاری (۱۲۴ بیت در هر میدان) حذف می‌شود و فقط تعداد بایتهای مورد نیاز برای داده‌های میدان تولید می‌شود. تابع fieldname برای به دست آوردن فهرست میدانها در f است و عبارتهای زیر

```
for k = 1:length(fields)
            b = b + bytes(f.(fields{k}));
```

برای اجرای تکرارها هستند. توجه کنید که در فراخوانی مکرر بایتها از اسامی میدانها با ساختار پویا استفاده شده است. اگر s ساختار و f یک متغیر رشته‌ای حاوی نام میدان باشد در عبارتهای زیر

```
S.(F) = foo;
Field = S.(F);
```

برای تعیین یا اخذ محتویات ساختار f از ترکیب نام میدان و ساختار پویا استفاده می‌شود.

برای نمایش یا استفاده از یک تصویر رمزگذاری شده فشرده ابتدا باید آن را به رمزگشا (تصویر ۶.۱) داد تا تصویر خروجی f(x, y) بازسازی شود. ممکن است f'(x, y) نماد دقیقی از f(x, y) باشد یا نباشد. اگر باشد سیستم را عاری ازخطا(error free)،محافظ اطلاعات

(information preserving) یا بدون خسارت(lossless) می‌نامیم. اگر نباشد یعنی تصویر بازسازی شده تحریف شده است. در حالت دوم که آن را فشرده‌سازی اتلاف شده (lossy compression) می‌نامیم، می‌توان خطای e(x, y) را بین f(x, y) و f'(x, y) برای هر مقدار از x وy تعریف کرد.

$$e(x,y) = \hat{f}(x,y) - f(x,y)$$

طوری که مجموع خطاهای بین دو تصویر به شرح زیر است:

?

و جذر میانگین مربعات بین f(x, y) و f'(x, y) ریشه دوم خطا به توان ۲ است که میانگین آن در ارایه m*n گرفته شده است

?

.

تابع M زیر e_{rms} را محاسبه کرده و اگر e_{rms} غیر از صفر باشد e(x, y) و پیشینه نمای آن را نشان می‌دهد. از آنجائی که e(x, y) می‌تواند مقادیر مثبت یا منفی داشته باشد، برای تولید پیشینه نما به جای imhist (که فقط با داده‌های تصویری کار می‌کند) از hist استفاده می‌شود.

```
function rmse = compare(f1, f2, scale)
%COMPARE Computes and displays the error between two matrices.
%     RMSE = COMPARE(F1, F2, SCALE) returns the root-mean-square error
%     between inputs F1 and F2, displays a histogram of the difference,
%     and displays a scaled difference image. When SCALE is omitted, a
%     scale factor of 1 is used.

% Check input arguments and set defaults.
error(nargchk(2, 3, nargin));
if nargin < 3
      scale = 1;
end
% Compute the root-mean-square error
e = double(f1) – double(f2);
[m, n] = size(e);
rmse = sqrt(sum(e(:) .^ 2) / (m * n));

% Output error image & histogram if an error(i.e., rmse ~= 0).
If rmse
      % Form error histogram.
      Emax = max(abs(e:)));
      [h, x] = hist(e(:), emax;
if length(h) >= 1
      figure; bar(x, h, 'k');
      % Scale the error image symmetrically and display
      emax = emax / scale;
      e = mat2gray(e, [-emax, emax]);
      figure; imshow(e);
end
end
```

رمزگذار تصویر ۶.۱ مسئول کاهش افزونگی رمزگذاری، بین پیکسلی و روانی-دیداری تصویر ورودی است. نقشه‌بردار در اولین مرحله فرایند رمزگذاری تصویر ورودی را به قالبی غیرتصویری تبدیل می‌کند که برای کاهش افزونگی عناصر بین تصویری طراحی شده است. در مرحله دوم یک بلوک مبدل دقت داده‌های خروجی ابزار نقشه‌بردار را مطابق با معیارهای وفاداری از پیش تعیین شده کاهش می‌دهد تا داده‌هایی که از نظر روانی – دیداری اضافه هستند حذف شوند. این عملیات برگشت‌ناپذیر است و اگر فشرده‌سازی بدون خطا مورد نظر باشد باید حذف شود. در مرحله سوم و نهایی این فرایند یک رمزگذار نمادین رمزی ایجاد می‌کند که افزونگی رمزگذاری را در مبدل داده‌های خروجی کاهش می‌دهد و داده‌های خروجی را مطابق با رمز نقشه برداری می‌کند.

رمزگشای تصویر ۶.۱ ،دو مولفه دارد: رمزگشای نمادین (symbol decoder) و نقشه بردار معکوس (inverse mapper). این قطعه‌ها معکوس اجرا می‌شوند، عملیات معکوس رمزگذار نمادین و نقشه بردار جلوی یکدیگر را می‌گیرند زیرا تبدیل برگشت‌ناپذیر است و بلوک تبدیل معکوس درج نمی‌شود.

۶.۲. افزونگی رمزگذاری (Coding Redundancy)

فرض کنید متغیر تصادفی متمایز $k=1, 2,\ldots l$ با احتمالات $p_r(r_k)$ نمایانگر سطوح خاکستری یک تصویر با سطح ال خاکستری (l_gray_level) باشد. R_1 همچون فصل ۳ مطابق با سطح خاکستری ۰ است(زیرا ارایه‌های MATLAB نمی‌توانند صفر باشند)

$$p_r(rk) = \frac{nk}{n} \quad k = 1,2,...,L$$

در اینجا n_k تعداد دفعاتی است که k امین سطح خاکستری در تصویر ظاهر می‌شود و n تعداد کل عناصر تصویری موجود در عکس است. اگر تعداد بیتهای مورد نیاز برای نمایش مقدار r_k معادل $l(r_k)$ باشد میانگین بیتهای مورد نیاز برای نمایش هر عنصر تصویری به شرح زیر است:

$$L_{avg} = \sum_{k=1}^{L} l(r_k)\, p_r(r_k)$$

یعنی میانگین طول کلمات رمزی تخصیص داده شده مقادیر گوناگون سطح خاکستری با جمع مجموع تعداد بیتهای مورد استفاده برای نمایش هر سطح خاکستری و احتمال وقوع سطح خاکستری مشخص می‌شود. بنا بر این کل تعداد بیتهای مورد نیاز برای رمزگذاری تصویرM*Nبرابر است با MNL_{avg}.

وقتی سطوح خاکستری یک تصویر با استفاده از رمز دودویی m بیت طبیعی نمایش داده می‌شوند، سمن راست معادله قبل به m بیت کاهش می‌یابد.

r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$	Code 2	$l_2(r_k)$
r_1	0.1875	00	2	011	3
r_2	0.5000	01	2	1	1
r_3	0.1250	10	2	010	3
r_4	0.1875	11	2	00	2

جدول ۶.۱: یک مثال از افزونگی رمزگذاری

یعنی وقتی m جایگزین $l(r_k)$ شود $l_{avg}=m$ می‌شود. سپس m ثابت خارج از مجموع برده می‌شود و فقط مجموع $p_r(r_k)$ برای $1<k<l$ باقی می‌ماند. که مساوی ۱ است. همان طور که در جدول ۶.۱ دیده می‌شود، وقتی سطوح خاکستری یک تصویر با استفاده از کد دودویی طبیعی رمزگذاری می‌شوند افزونگی رمزگذاری همیشه وجود دارد. در این جدول رمزگذاری طول ثابت و متغیر تصویر ۴ سطحی دیده می‌شود که توزیع سطح خاکستری آن در ستون ۲ نشان داده شده است. میانگین طول رمزگذاری دودویی ۲ بیتی در ستون ۳ دو بیت است. میانگین تعداد بیتهای مورد نیاز در رمزگذاری ۲ در ستون ۵ با رابطه زیر حاصل می‌شود و

?

نسبت فشرده‌سازی حاصل از آن $cr=2/1825=1103$ است. مبنای فشرده‌سازی رمز ۲ آن است که کلمات آن طول متغیر دارند. بنا بر این کوتاهترین کلمات به سطوح خاکستری تخصیص داده می‌شوند که به تعداد دفعات بیشتری در تصویر وجود داشته باشند.

سوال آن است که برای نمایش سطوح خاکستری تصویر چند بیت واقعاً مورد نیاز است؟ حداقل تعداد داده مورد نیاز برای نمایش تصویر بدون از دست دادن اطلاعات چقدر است؟ برای پاسخ دادن به این موضوع و پرسشهای مرتبط به فرضیه اطلاعات مراجعه می‌کنیم. اصل حاکم بر آن این است که الگوی تولید اطلاعات بر اساس فرایند احتمالات است و به شکلی مطابق با حس ششم سنجیده می‌شود. طبق این فرضیه تعداد واحدهای اطلاعاتی رویداد تصادفیE با احتمال $P(E)$ به شرح زیر است:

?

اگر $P(E)=1$ باشد یعنی این رویداد همواره اتفاق افتد، $I(E)=0$ می‌شود و هیچ اطلاعاتی به آن مرتبط نیست. از آنجائی که هیچ نوع ابهامی با این رویداد ارتباط ندارد، وقوع این رویداد باعث انتقال هیچ اطلاعاتی نمی‌شود. با توجه به این رویدادهای تصادفی و مجموعه وقایع

احتمالی $[a_1,a_2,\ldots,a_j]$ با احتمالات $\{p(a_1),p(a_2),..p(a_j)\}$ میانگین اطلاعات منبع خروجی که درجه بی‌نظمی منبع نامیده می‌شود به شرح زیر است:

?

اگر تصویری به عنوان نمونه‌ای از منبع خاکستری ساطع کننده آن تعبیر شود، می‌توان برای الگوبرداری احتمالات نماد آن منبع از پیشینه نمای(histogram) خاکستری تصویر مشاهده شده استفاده کرد و برآورد درجه یک درجه بی‌نظمی منبع را تخمین زد.

?

```
function h = entropy(x, n)
%ENTROPY Computes a first-order estimate of the entropy of a matrix.
%     H = ENTROPY(X, N) returns the first-order estimate of matrix X
%     with N symbols (N = 256 if omitted) in bits/symbol. The estimate
%     assumes a statistically independent source characterized by the
%     relative frequency of occurrence of the elements in X.

error(nargchk(1, 2, nargin));       % Check input arguments
if nargin < 2
      n = 256;                      % Default for n.
end

x = double(x);                      % Make input double
xh = hist(x(:), n);                       % Compute N-bin histogram
xh = xh / sum(xh(:));               % Compute probabilities
% Make mask to eliminate 0's since log2(0) = -inf.
i = find(xh);

h = -sum(xh(i). *log2(xh(i)));            % Compute entropy
```

به استفاده از تابع find در نرم افزار MATLAB توجه کنید، که برای نمایش شاخصهای عناصر غیرصفر پیشینه نمای x_h به کار برده شده است. عبارت find (x) معادل عبارت find (x~=0) است. تابع درجه بی‌نظمی برای ایجاد بردار شاخصهای I در پیشینه نمای x_h از find استفاده می‌کند که برای حذف کلیه عناصر با مقدار صفر از محاسبه درجه بی‌نظمی در عبارت نهایی به کار برده می‌شوند. اگر این کار انجام نشود، تابع log2 داده‌های خروجی h را به زور به Nan می‌دهد (0*-inf یک عدد نیست)در حالتی که نماد احتمالات صفر باشد.

مثال ۶.۱: محاسبه برآورد شاخص درجه بی‌نظمی:

یک تصویر ساده ۴*۴ را در نظر بگیرید که پیشینه نمای آن (p در کد بعدی) احتمالات نماد را در جدول ۶.۱ الگوبرداری کند. ترتیب عبارتها در خط فرمان زیر چنین تصویری را ایجاد می‌کند و شاخص درجه بی‌نظمی آن را محاسبه و ارزیابی می‌کند.

```
>> f = [119 123 168 119; 123 119 168 168];
>> f = [f; 119 119 107 119; 107 107 119 119;]
f =
      119   123   168   119
      123   119   168   168
      119   119   107   119
      107   107   119   119
p = hist(f(:), 8);
p = p / sum(p)
p =
     0.1875 0.5   0.125           0         0  0     0     0.1875
h = entropy(f)
h =
    1.7806
```

رمز ۲ جدول ۶.۱ با l_{avg}=1.81 نزدیک به برآورد شاخص درجه بی‌نظمی است و کد دودویی حداقل طول تصویر f است. توجه داشته باشید که سطح خاکستری ۱۰۷ مطابق با r_1 و کلمات رمزی دودویی ۲(۰۱۱)مطابق با جدول ۶.۱ و ۱۱۹ مطابق با r_2 و کدهای $(1)_2$، ۱۲۳ و ۱۶۸ به ترتیب مطابق با ۲(۰۱۰) و ۲(۰۰) هستند.

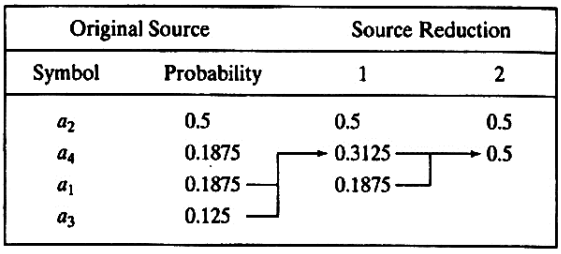
۶.۲.۱. رمزهای هافمن(Huffman codes)

هنگام رمزگذاری سطوح خاکستری(gray levels) یک تصویر و یا داده‌های خروجی عملیات نقشه برداری سطح خاکستری (اختلاف پیکسلها، طول و غیره) کدهای هافمن کمترین تعداد نمادهای رمزی (مثل بیت) را در هر نماد منبع (مثلاً مقیاس خاکستری) دارند و محدودیت آنها آن است که نمادهای منبع یک به یک رمزگذاری می‌شوند.

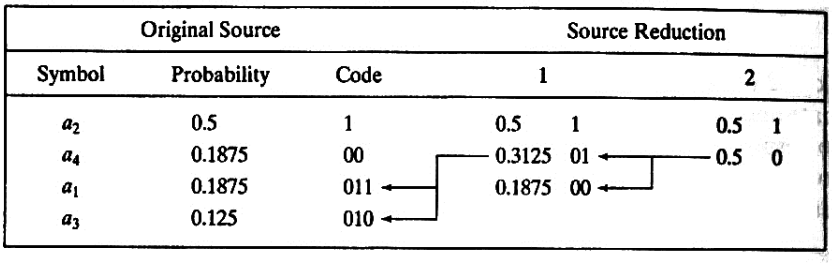
اولین گام در روش هافمن ایجاد واحدهای کاهش منبع با مرتب کردن احتمالات نمادهای مورد نظر و ادغام نمادهای کم احتمال با نمادهای تکی است که در مرحله کاهش بعدی جایگزین آنها می‌شوند. درتصویر۶.۲(a)این فرایند برای توزیع سطح خاکستری در جدول ۶.۱ نشان داده شده است. در سمت چپ مجموعه مقدماتی نمادهای منبع و احتمالات آنها از بالا به پایین بر حسب مقادیر احتمالات مرتب شده‌اند. برای کاهش اولین منبع دو احتمال آخر یعنی ۰.۱۲۵ و ۰.۱۸۷۵ با هم ادغام می‌شوند تا یک نماد ترکیبی با احتمال ۰.۳۱۲۵ ایجاد شود. این نماد ترکیبی و احتمال مرتبط با آن در اولین ستون کاهش منبع قرار داده می‌شوند طوری که احتمالات منبع کاهش یافته از حداکثر تا حداقل احتمالات مرتب می‌شوند. این فرایند آنقدر تکرار می‌شود تا منبع کاهش یافته با ۲ نماد در منتها الیه سمت راست تشکیل شود.

گام دوم در روش هافمن رمزگذاری هر منبع کاهش یافته با کوچکترین منبع و کار با منبع اصلی است. رمزهای دودویی با حداقل طول در منبع‌های دو نمادی از نمادهای ۰ و ۱ تشکیل می‌شوند. همان طور که در تصویر ۶.۲ب می‌بینید، این نمادها به دو نماد سمت راست تخصیص داده شده‌اند (این تخصیص اختیاری است با معکوس‌سازی ترتیب ۰ و ۱ نیز کار به همین شکل اجرا می‌شود). وقتی نماد کاهش

یافته منبع با احتمال ۰.۵ با ادغام دو نماد در منبع کاهش یافته سمت چپ تولید می‌شود صفری که برای رمزگذاری آن به کار برده شده است به هر دو نماد تخصیص داده می‌شود و ۰ و ۱ هر کدام اختیاری به هر کدام از آنها متصل می‌شوند تا بتوان آنها را از یکدیگر تشخیص داد.



Original Source		Source Reduction	
Symbol	Probability	1	2
a_2	0.5	0.5	0.5
a_4	0.1875	0.3125	0.5
a_1	0.1875	0.1875	
a_3	0.125		



Original Source			Source Reduction			
Symbol	Probability	Code	1		2	
a_2	0.5	1	0.5	1	0.5	1
a_4	0.1875	00	0.3125	01	0.5	0
a_1	0.1875	011	0.1875	00		
a_3	0.125	010				

تصویر ۶.۲: هافمن (a) کاهش منبع (b) روش تخصیص رمزها

این عملیات در هر یک از منابع کاهش یافته آنقدر تکرار می‌شود که به منبع اصلی دست یابند. رمز نهایی در منتها الیه سمت چپ ستون ۳ در تصویر ۶.۲(b) دیده می‌شود.

رمز هافمن در تصویر ۶.۲ب و جدول ۶.۱ یک رمز فوری منحصر به فرد قابل رمزگشایی است. این یک رمز بلوکی است زیرا هر نماد منبع به توالیهای ثابتی از نمادهای رمز ترسیم می‌شود. از آن جهت فوری است که هر کلمه رمزی در رشته نمادهای رمز می‌تواند بدون مرجع‌گذاری نمادهای بعدی رمزگشایی شود. یعنی کلمات بدون رمز در هر یک از رمزهای هافمن به صورت یک در میان پیشوند کلمات هستند. آنها به شکلی منحصر به فرد قابل رمزگشایی هستند زیرا رشته نمادها را فقط در یک جهت می‌توان رمزگشایی کرد. بنا بر این برای رمزگشایی هر یک از نمادهای رمزگذاری شده هافمن عناصر تشکیل شده در رشته از چپ به راست بررسی می‌شوند. در تصویر ۴*۴ در مثال ۶.۱ یک رمزگذاری بالا به پایین و چپ به راست بر اساس رمز هافمن در تصویر ۶.۲ب رشته ۲۹ بیتی زیر را تولید می‌کند: ۱۰۱۰۱۰۱۱۱۱۱۱. از

آنجائی که از یک بلوک رمز منحصر به فرد فوری قابل رمزگشایی استفاده می‌کنیم نیازی به درج جداساز بین عناصر تصویری رمزگذاری شده نیست. در بررسی رشته حاصل از آن از چپ به راست معلوم شد که اولین رمز معتبر ۱ است که رمز نماد a2 یا سطح خاکستری ۱۱۹ است. کلمه رمزی معتبر بعدی ۰۱۰ است که مطابق با سطح خاکستری ۱۲۳ است. اگر کار را به همین منوال ادامه دهیم بالاخره به یک تصویر رمزگشایی شده می‌رسیم که معادل f در مثال فوق است.

روشهای کاهش منبع و تخصیص رمزها که در بالا با تابع M اجرا شدند هافمن نامیده می‌شوند.

```
function CODE = huffman(p)
%HUFFMAN Builds a variable-length Huffman code for a symbol source.
%     CODE = HUFFMAN(P) returns a Huffman code as binary strings in
%     cell array CODE for input symbol probability vector P. Each word
%     in CODE corresponds to a symbol whose probability is at the
%     corresponding index of P.
%
%     Based on huffman5 by Sean Danaher, University of North Umbria,
%     Newcastle UK. Available at the MATLAB Central File Exchange:
%     Category General DSP in Signal Processing and Communications.

% Check the input arguments for reasonableness.
error(nargchk(1, 1, nargin));
if (ndims(p) ~= 2) | (min(size(p)) > 1) | ~isreal(p) | ~isnumeric(p)
      error('p must be a real numeric vector.');
end

% Global variable surviving all recursions of function 'makecode'
global CODE
CODE = cell(length(p), 1);          % Init the global cell array

if length(p)>1                      % When more than one symbol…
      p = p / sum(p);               % Normalize the input probabilities
      s = reduce(p);                % Do Huffman source symbol reductions
      makecode(s, []);              % Recursively generate the code
else
      CODE = {'1'};                 % Else, trivial one symbol case!
end;
%--------------------------------------------------------------------------------
----%
function s = reduce(p);
% Create a Huffman source reduction tree in a MATLAB cell structure
% by performing source symbol reductions until there are only two
% reduced symbols remaining
s = cell(length(p), 1);

% Generate a  starting tree with symbol nodes 1, 2, 3, … to
% reference the symbol probabilities.
for I = 1:length(p)
      s{i} = i;
end

while numel(s) > 2
      [p, i] = sort(p);       % Sort the symbol probabilities
      p(2) = p(1) + p(2);           % Merge the 2 lowest probabilities
      p(1) = [];              % and prune the lowest one

      s = s(i);               % Reorder tree for new probabilities
```

```
      s{2} = {s{1}, s{2}};    % and merge & prune its nodes
      s(1) = [];              % to match the probabilities
end
%-------------------------------------------------------------------------------
----%
function makecode(sc, codeword)
% Scan the nodes of a Huffman source reduction tree recursively to
% generate the indicated variable length code words.
% Global variable surviving all recursive calls
global CODE
if isa(sc, 'cell')                              % For cell array nodes,
      makecode(sc{1}, [codeword 0]);      % add a 0 if the 1st element
      makecode(sc{2}, [codeword 1]);      % or a 1 if the 2nd
else                                      % For leaf (numeric) nodes,
      CODE{sc} = char('0' + codeword);    % create a char code string
end
```

در خط فرمان زیر برای تولید رمز در تصویر ۶.۲ از هافمن استفاده می‌شود.

```
>> p = [0.1875    0.5   0.125      0.1875];
>> c = Huffman(p)

c o=
      '011'
      '1'
      '010'
      '00'
```

توجه داشته باشید که داده خروجی آرایه‌ای نویسه‌دار با طول تغییرپذیر است که هر ردیف آن یک رشته از صفرها و یکها است که رمز دودویی نماد شاخص گذاری شده مطابق با آن در پی است. مثلاً ۰۱۰ در ارایه شاخص ۳ رمز سطح خاکستری با احتمال ۰.۱۲۵ است.

شناسه ورودی p در قسمتهای اول هافمن (بردار احتمالات نماد ورودی نمادهای رمزگذاری شده) از نظر معقول بودن بررسی می‌شود و رمز تغییرپذیر کلی به صورت ارایه سلولهای MATLAB با length به اندازه p ردیف و یک ستون تعریف شده است شروع می‌شود. کلیه متغیرهای کلی نرم افزار MATLAB باید بر حسب تابعهایی بیان شوند که با عبارتهایی شبیه به عبارت زیر مرجع گذاری می‌شوند.

```
global X Y Z
```

در این عبارت متغیرهای x، y و z که در آنها بیان می‌شوند، وقتی چند تابع یک متغیر کلی را بیان کنند، یک نسخه واحدی از آن متغیر را دارند. تابع داخلی و متداول هافمن با نام makecode خصوصیات مشترکی با متغیر کلی code دارد. توجه داشته باشید که معمولاً اسامی متغیرهای کلی با حروف بزرگ نوشته می‌شود. متغیرهای غیرکلی متغیرهای موضعی هستند و فقط در اختیار تابعهایی هستند که در آنها تعریف می‌شوند (نه در اختیار سایر تابعها یا فضای کاری مبنا). آنها معمولاً با حروف کوچک نوشته می‌شوند. در هافمن برای اجرای code از تابع cell استفاده می‌شود که ترکیب آن به شرح زیر است:

```
X = cell(m, n)
```

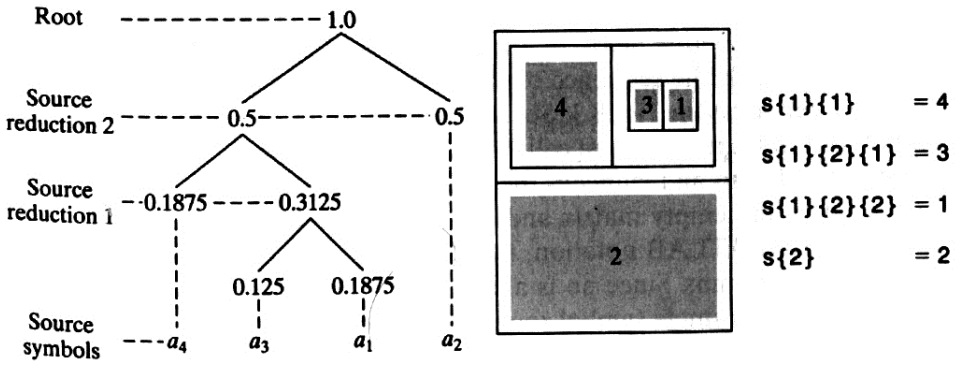
این دستور،تعداد m*n آرایه از ماتریسهای خالی ایجاد می‌کند که بر اساس سلول یا محتوا مرجع گذاری می‌شوند. پرانتز برای شاخص‌گذاری سلولها به کار برده می‌شود. از "()" برای شاخص‌گذاری محتوا استفاده می‌شود بنا بر این x(1)=[] شاخص است و عنصر ۱ را از ارایه سلولی برمی‌دارد در حالی که x{1} = [] مجموعه اولین عنصر آرایه سلول را در ماتریس خالی قرار می‌دهد.

یعنی x{1} اشاره به محتویات اولین عنصر دارد. آرایه x: x(1) بیشتر اشاره به عنصر دارد تا محتویات آن. از آنجائی که می‌توان آرایه‌های سلولی را در سایر ارایه‌های سلولی تعبیه کرد ترکیب x{1}{2} اشاره به محتوای عنصر دوم آرایه سلولی دارد که در اولین عنصر آرایه سلولی ایکس است.پس از راه اندازی code و هنجارمند کردن بردار احتمالات داده‌های ورودی در عبارت p = p/sum (p) کد هافمن برای بردار احتمالات هنجارمند شده طی ۲ مرحله ایجاد می‌شود. مرحله اول که با عبارت اصلی روال s=reduce (p) راه اندازی می‌شود، تابع درونی reduce را فراخوانی می‌کند که وظیفه آن کاهش منبع است که در تصویر ۶.۲(a)دیده می‌شود. در این روند عناصر ارایه سلولی که در ابتدا خالی بوده است و با code منطبق شده است با شاخصهای مربوطه راه اندازی می‌شوند. یعنی s{1}=1, s{2}=2 و غیره. سپس سلول معادل نمودار دودویی کاهشهای منبع در حلقه while numel (s) > 2 ایجاد می‌شود. در هر دفعه تکرار این حلقه بردار p بر حسب افزایش احتمالات مرتب می‌شود. این کار با تابع sort (یعنی مرتب سازی) انجام می‌شود که ترکیب کلی آن به شرح زیر است:

```
[y, i] = sort(x)
```

داده‌های خروجی y عناصر مرتب شده ایکس هستند و i طوری است که y=x(i) است. پس از مرتب شدن p این دو احتمال در p(2) با هم ادغام می‌شوند و p(1) بریده می‌شود. سپس سلول کاهش منبع دوباره مرتب می‌شود تا منطبق بر p بر اساس بردار شاخص i با استفاده از s=s(i) باشد. بالاخره s{2} با آرایه سلولی دو عنصری حاوی شاخصهای احتمال ادغام شده از طریق s{2}={s{1},s{2}} جایگزین می‌شود. (این یک نمونه از شاخص‌گذاری محتوا است). شاخص‌گذاری سلول برای بریدن اولین عنصر از دو عنصر ادغام شده s(1) از طریق s(1)=[] به کار برده می‌شود. این فرایند آنقدر تکرار می‌شود تا فقط دو عنصر در s باقی بمانند.

```
celldisp(s);
cellplot(s);
```



داده‌های خروجی فرایند احتمالات نمادها در جدول ۶.۱ و تصویر ۶.۲.(a)در تصویر ۶.۳ نشان داده شده است. تصویرهای ۶.۳.ب و سی با درج بین ۲ عبارت آخر روال اصلی هافمن ایجاد شدند.

تابع celldisp نرم افزار MATLAB محتویات آرایه سلول را مکرراً نمایش می‌دهد. تابع cellplot تصویر گرافیکی یک آرایه سلولی به صورت جعبه‌های تودرتو است. به تطابق یک به یک عناصر آرایه سلولی در تصویر ۶.۳.ب و گره‌های درخت کاهش منبع در تصویر ۶.۳.(a) توجه کنید.

(۱) هر یک از مسیرهای دو شاخه در این نمودار (که نمایانگر کاهش منبع است) مطابق با یک ارایه سلولی دو عنصری در s است.

(۲) هر یک از آرایه‌های سلولی دو عنصری حاوی شاخصهای نمادهایی است که در کاهش منبع مطابق با آنها ادغام شدند.

مثلاً ادغام نمادهای a1 , a3 در انتهای این نمودار آرایه سلولی دو عنصری s{1}{2} را ایجاد می‌کند. در اینجا s{1}{2}{1}=3 و s{1}{2}{2}=1 به ترتیب شاخصهای نماد a1 ,a3 هستند. ریشه این درخت آرایه سلولی دو عنصری فوقانی است.

گام نهایی در فرایند تولید رمز (یعنی تخصیص رمزها بر اساس آرایه سلولی کاهش منبع s) با فراخوانی عبارت نهایی هافمن یعنی makecode(s,[]) اجرا می‌شود. این فراخوانی باعث راه اندازی فرایند تخصیص رمزهای تکراری بر اساس روش تصویر ۶.۲(b) می‌شود. گرچه تکرار باعث صرفه جویی در ذخیره سازی (زیرا مقادیر پردازش شده باید جای دیگر نگهداری شوند) و یا افزایش سرعت نمی‌شود ولی مزیتش آن است که رمز فشرده است و درک آن آسان است مخصوصاً هنگام کار با داده‌هایی که به شکل تکراری تعریف شده‌اند مانند نمودارهای درختی. هر یک از تابعهای نرم افزار MATLAB را می‌توان به صورت تکراری استفاده کرد یعنی می‌توان آنها را مستقیماً یا غیرمستقیم فراخوانی کرد. هنگام بازگشت در فراخوانی تابعها مجموعه تازه‌ای از متغیرهای موضعی ایجاد می‌شوند که مستقل از همه مجموعه‌های قبلی هستند.

تابع درونی makecode دو نوع داده ورودی را قبول می‌کند: codeword که آرایه‌ای از صفرها و یکها است و sc که عنصر آرایه سلولی کاهش منبع است. وقتی sc خودش یک ارایه سلولی باشد، حاوی دو نماد (ترکیبی) منبع است که در فرایند کاهش منبع با یکدیگر ادغام شده‌اند. از آنجائی که آنها باید انفرادی رمزگذاری شوند، mkecode مکرراً در عناصر فراخوانی می‌شود که همراه با دو کلمه رمزی است که روزینه می‌شوند (صفرها و یک به کلمه رمز ورودی ضمیمه می‌شوند. وقتی sc حاوی ارایه سلول نباشد، شاخص نماد منبع اصلی است و رشته دودویی ایجاد شده از کلمه رمز ورودی با استفاده از code (sc) = char (0 +codeword) به آن تخصیص داده می‌شود. همان طور که گفتیم، تابع char در نرم افزار MATLAB آرایه‌های حاوی اعداد صحیح مثبت را که نمایانگر رمزهای نویسه هستند به آرایه

نویسه MATLAB تبدیل می‌کند (۱۲۷ رمز اول اسکی هستند). مثلاً char('0'+[010]) رشته '010' را تولید می‌کند زیرا افزودن صفر به کد اصلی به ازای هر صفر یک صفر اسکی تولید می‌کند در حالی که افزودن ۱ به اسکی صفر فقط کد اسکی ۱ را ایجاد می‌کند.

در جدول ۶.۲ جزئیات روال فراخوانی makecode که نتایج آن در آرایه سلولی کاهش منبع تصویر ۶.۳ دیده می‌شود. برای رمزگذاری چهار نماد منبع هفت فراخوانی مورد نیاز است. اولین فراخوانی (ردیف ۱ ازجدول ۶.۲) از روال اصلی هافمن انجام می‌شود و فرایند رمزگذاری را با داده‌های ورودی کلمه رمز راه اندازی می‌کند و sc به ترتیب به ماتریس خالی و آرایه سلولی s تخصیص داده می‌شود. {1*2cell} بر اساس نشانه‌گذاریهای نرم افزار MATLAB به معنی آرایه سلولی تک ردیفه دو ستونی است. از آنجائی که sc همیشه آرایه سلولی در اولین فراخوانی است (منبع نماد تکی یک استثنا است) دو فراخوانی مکرر ردیفهای ۲ و ۷ جدول انجام می‌شود. اولین فراخوانی دو فراخوانی دیگر را نیز راه اندازی می‌کند که ردیفهای ۳ و ۴ هستند. و دومین مورد ردیفهای ۵ و ۶ را فراخوانی می‌کند.

Call	Origin	sc	codeword
1	main routine	{1x2 cell} [2]	[]
2	makecode	[4] {1x2 cell}	0
3	makecode	4	0 0
4	makecode	[3] [1]	0 1
5	makecode	3	0 1 0
6	makecode	1	0 1 1
7	makecode	2	1

جدول ۶.۲: فرایند تخصیص رمز در ارایه سلولی کاهش منبع در تصویر ۶.۳

هر دفعه که sc یک ارایه سلولی نباشد، همچون ردیفهای ۳، ۵، ۶ و ۷ جدول تکرارهای اضافی باید انجام شوند. رشته رمز از codeword ایجاد می‌شود و به نماد منبعی تخصیص داده می‌شود که شاخص آن sc بوده است.

۶.۲.۲. رمزگذاری هافمن (Huffman Encoder)

رمزگذاری هافمن به خودی خود فشرده‌سازی محسوب نمی‌شود. برای درک روش فشرده‌سازی تعبیه شده در رمزهای هافمن نمادهایی که رمزها در آن ایجاد شده است چه از نوع سطوح خاکستری باشند و یا داده‌های خروجی سایر عملیات ترسیم سطوح خاکستری باشند باید مطابق با رمزهای تولید شده تبدیل یا نقشه برداری (یعنی رمزگذاری) شوند.

مثال ۶.۲:

یک تصویر ساده ۴*۴ شانزده بایتی را در نظر بگیرید:

```
>> f2 = uint8([2   3  4  2;  3  2  4  4;  2  2  1  2;  1  1  2  2])
f2 =
      2     3     4     2
      3     2     4     4
      2     2     1     2
      1     1     2     2
>> whos('f2')
      Name        Size        Bytes       Class
      F2          4×4         16          uint8 array
Grand total is 16 elements using 16 bytes
```

هر عنصر تصویری در f_2 یک بایت حاوی ۸ بیت است. برای نمایش کل تصویر از ۱۶ بایت استفاده می‌شود. از آنجائی که سطوح خاکستری f_2 احتمالات یکسان ندارند، یک رمز با طول متغیر (همان طور که در بخش آخر گفتیم) مقدار حافظه مورد نیاز برای نمایش تصویر را کاهش می‌دهد. تابع هافمن چنین رمزی را محاسبه می‌کند.

```
>> c = Huffman(hist(double(f2(:)), 4))
c =
      '011'
      '1'
      '010'
      '00'
```

از آنجائی که رمزهای هافمن بر اساس تعداد دفعات نسبی وقوع نمادهای منبع رمزگذاری شده هستند نه خود نمادها، c مانند رمزی است که برای تصویر مثال ۶.۱ ایجاد شد. در واقع می‌توان با ترسیم سطوح خاکستری ۱۰۷و ۱۱۹، ۱۲۳، و ۱۶۸ به ترتیب تا ۱، ۲، ۳ و ۴ تصویر f_2 را از f در مثال ۶.۱ به دست آورد.

یک روش ساده برای رمزگذاری اف۲ بر اساس رمز c اجرای عملیات جستجوی ساده است.

```
>> h1f2 = c(f2(:))'
h1f2 =
      Columns 1 through 9
      '1'   '010' '1'   '011' '010' '1'   '1'   '011' '00'
      Columns 10 through 16
      '00'  '011' '1'   '1'   '00'  '1'   '1'
>> whos('h1f2')
      Name        Size        Bytes       Class
      h1f2        1×16        1530        cell array
Grand total is 45 elements using 1530 bytes
```

در اینجا f_2 یک ارایه دو بعدی از نوع uint8 است که به ارایه سلول ۱*۱۶ تبدیل می‌شود h1f2 جابجایی است که لایه را فشرده می‌کند. عناصر h1f2 رشته‌هایی با طول متغیر هستند که مطابق با عناصر تصویری f_2 از بالا به پایین و چپ به راست بررسی می‌شوند. در این تصویر رمزگذاری شده از ۱۵۳۰ بایت استفاده می‌شود که ۱۰۰ برابر حافظه مورد نیاز توسط f_2 هستند.

استفاده از ارایه سلولی h1f2 منطقی است چون که یکی از سازه‌های استاندارد داده‌های MATLAB است که برای کار با آرایه‌های داده‌های نامشابه به کار برده می‌شود. این اختلاف در مورد h1f2 مربوط به طول رشته‌های نویسه است و بهای پرداخت شده برای کار با آن

از طریق آرایه سلولی جزو هزینه‌های عمومی حافظه محسوب می‌شود (که در آرایه سلولی درج شده است) که برای ردیابی موقعیت عناصری با طول متغیر به کار برده می‌شود. با تبدیل h1f2 به یک آرایه نویسه دو بعدی متعارف می‌توان از هزینه‌های عمومی اجتناب کرد.

```
>> h2f2 = char(h1f2)'
h2f2 =
      1010011000011011
        1  11    1001    0
        0  10    1   1
>> whos('h2f2')
      Name        Size        Bytes       Class
      h2f2        3×16        96          char array
Grand total is 48 elements using 96 bytes
```

در اینجا آرایه سلولی h1f2 به آرایه نویسه ۳*۱۶ h2f2 تبدیل می‌شود. هر ستون از h2f2 مطابق با یک عنصر تصویری از f_2 است که از بالا به پایین و چپ به راست بررسی می‌شوند. بلوکها برای اندازه گیری صحیح آرایه درج می‌شوند از آنجائی که برای هر ۰ و ۱ کلمه رمز به دو بایت نیاز است کل حافظه مورد استفاده توسط h2f2 ۹۶ بایت است که ۶ مرتبه بیشتر از ۱۶ بایت اصلی مورد نیاز برای f_2 است. می‌توان فضاهای خالی درج شده را با استفاده از روابط زیر حذف کرد:

```
>> h2f2 = h2f2(:);
>> h2f2(h2f2 == '  ') = [ ];
>> whos('h2f2')
      Name        Size        Bytes       Class
      h2f2        29×1        58          char array
Grand total is 29 elements using 58 bytes
```

ولی حافظه مورد نیاز بیشتر از ۱۶ بایت اصلی f_2 است.

برای فشرده‌سازی f_2 رمز c باید در سطح بیت اعمال شود و چندین عنصر تصویری رمزگذاری شده در یک بایت جمع شوند.

```
>> h3f2 = mat2huff(f2)
h3f2=
      size: [4 4]
      min: 32769
      hist: [3 8 2 3]
         code: [43867 1944]

>> whos('h3f2')
      Name        Size        Bytes       Class
      h3f2        1×1         518         struct array
Grand total is 13 elements using 518 bytes
```

گرچه تابع mat2huff ساختار h3f2 را تولید می‌کند که نیاز به ۵۱۸ بایت حافظه دارد اکثر آن با یکی از دو مورد زیر مرتبط است: (۱) هزینه‌های عمومی متغیر سازه (استفاده از imratio در بخش ۶.۱ برای یادآوری این MATLAB که نرم افزار MATLAB در هر میدان از سازه از ۱۲۴ بایت برای هزینه‌های عمومی استفاده می‌کند) و (۲) mat2huff اطلاعاتی برای تسهیل رمزگشاییهای آتی تولید

می‌کند. صرف نظر از این هزینه‌های عمومی که هنگام بررسی تصویرهای کاربردی با اندازه عادی قابل چشم پوشی است mat2huff، f_2 را با عامل ۴:۱ فشرده‌سازی می‌کند ۱۶ پیکسل ۸ بیتی f_2 به ۲ کلمه ۱۶ بیتی فشرده می‌شوندعناصر رمز h3f2 به شرح زیر است:

```
>> hcode = h3f2.code;
>> whos('hcode')
      Name        Size        Bytes       Class
      hcode       1×2         4           uint16 array
Grand total is 2 elements using 4 bytes

>> dec2bin(double(hcode))
ans =
      1010101101011011
      0000011110011000
```

توجه داشته باشید که dec2bin برای نمایش هر یک از بیتهای h3f2 به کار برده می‌شوند و از بیتهای ۱۶ مدولی آخر (یعنی ۳ صفر آخر) صرف نظر می‌شود. رمزگذاری ۳۲ بیت معادل بلوکهای ۲۹ بیت فوری منحصر به فرد است که قبلاً در بخش ۶.۲.۱ تولید کردیم.

همان طور که در مثال قبل گفتیم تابع mat2huff حاوی اطلاعات لازم برای رمزگشایی آرایه رمزگذاری شده (مثلاً احتمالات نمادها و ابعاد اصلی) است که در یکی از متغیرهای تکی نرم افزار MATLAB قرار دارند. اطلاعات این ساختار در مطالب راهنمای بخش mat2huff درج شده است.

```
function y=mat2huff(x)
%MAT2HUFF Huffman encodes a matrix.
%     Y = MAT2HUFF(X) Huffman encodes matrix X using symbol
%     probabilities in unit-with histogram bins between X's minimum
%     and maximum values. The encoded data is returned as a structure
%     Y:
%           Y.code The Huffman-encoded values of X, stored in
%                  a uint16 vector. The other fields of Y contain
%                  additional decoding information, including:
%     Y.min        The minimum values of X plus 32768
%     Y.size       The size of X
%     Y.hist             The histogram of X
%
%     If X is logical, uint8, uint16, unit32, int8, int16, or double,
%     with integer values, it can be input directly to MAT2HUFF. The
%     minimum value of X must be represent able as an int16.
%
%     If X is double with non-integer values---for example, and image
%     with values between 0 and 1---first scale X to an appropriate
%     integer range before the call. For example, use Y =
%     MAT2HUFF(255*X) for 256 gray level encoding.
%
%     NOTE: The number of Huffman code words is round(max(X(:))) –
%     round(min(X(:))) + 1. You may need to scale input X to generate
%     codes of reasonable length. The maximum row or column dimension
%     of X is 65535.
%
%     See also HUFF2MAT.

If ndims(x) ~= 2 | ~isreal(x) | (~isnumeric(x) & ~islogical(x))
    error('X must be a 2-D real numeric or logical matrix.');
end
```

```
% Store the size of input x.
y.size = unit32(size(x));

% Find the range of x values and store its minimum value biased
% by  +32768 as a UINT16.
x = round(double(x));
xmin = min(x(:));
xmax = max(x(:));
pmin = double(int16(xmin));
pmin = uint16(pmin + 32768);        y.min = pmin;

% Compute the input histogram between xmin and xmax with unit
% width bins, scale to UINT16, and store.
x = x(:)';
h = histc(x, xmin:xmax);
if max(h) > 65535
      h = 65535 * h / max(h);
end

% Code the input matrix and store the result.
Map = huffman(double(h));                % Make Huffman code map
hx = map(x(:) – xmin + 1);               % Map image
hx = char(hx)';                          % Convert to char array
hx = hx(:)';
hx(hx == '  ') = [ ];                          % Remove blanks
ysize = ceil(length(hx) / 16);                 % Compute encoded size
hx16 = repmat('0', 1, ysize * 16);       % Pre-allocate module-16 vector
hx16(1:length(hx)) = hx;                       % Make hx module-16 in length
hx16 = reshape(hx16, 16, ysize);         % Reshape to 16-character words
hx16 = hx16' – '0';                            % Convert binary string to decimal
twos = pow2(15:-1:0);
y.code = uint16(sum(hx16 .* twos(ones(ysize, 1), :), 2))';
```

توجه داشته باشید که عبارت y=mat2huff (x) هافمن ماتریس x را با استفاده از محفظه‌های پیشینه نما به پهنای واحد بین مقادیر حداقل و حداکثر x رمزگذاری می‌کند. وقتی داده‌های رمزگذاری شده درy بعداً رمزگشایی می‌شوند کد هافمن مورد نیاز برای رمزگشایی آن باید دوباره از y.min که حداقل مقدارx است و y.hist که پیشینه نمای x است ساخته شود. Mat2huff به جای حفظ رمز هافمن اطلاعات مورد نیاز برای تولید مجدد آن را نگه می‌دارد. با توجه به این موضوع و ابعاد اصلی ماتریس x که در y.size ذخیره می‌شود تابع huff2mat بخش ۶.۲.۳ (بخش بعد) برای رمزگشایی y.code و بازسازی x به کار برده می‌شود.

مراحل مورد نیاز برای تولید y.code به طور خلاصه در اینجا بیان شده است:

۱. ابتدا پیشینه نمای h را با داده‌های ورودی x بین حداقل و حداکثر مقادیر x با استفاده از محفظه‌هایی به پهنای واحد محاسبه کنید و مقیاس آن را طوری تعیین کنید که در بردار uint16 جا شود.

۲. برای ایجاد رمز هافمن از Huffman استفاده کنید که در اینجا نقشه‌ای مبتنی بر پیشینه نمای h مقیاس بندی شده است.

۳. داده‌های ورودی x را با استفاده از نقشه ترسیم کنید (تا یک آرایه سلولی ایجاد شود) و سپس آن را به آرایه نویسه‌ای h_x تبدیل کنید. فضاهای خالی درج شده همچون h2f2 در مثال ۶.۲ را برطرف کنید.

۴. بردار h_x را طوری درست کنید که نویسه‌های آن به صورت قطعات ۱۶ واحدی در آید. این کار با ایجاد یک بردار نویسه ۱۶ مدولی انجام می‌شود که h_{x16} را در رمز نگه می‌دارد و عناصر h_x را به آن کپی می‌کند، و آن را در ردیفهای ۱۶ واحدی در آرایه ysize شکل‌دهی می‌کند. از بخش ۴.۲ به خاطر داریم که تابع ceil اعداد را به سمت مثبت بینهایت گرد می‌کند. تابع کلی نرم افزار MATLAB به شرح زیر است:

```
y = reshape(x, m, n)
```

در اینجا m از ماتریس n تولید می‌شود که عناصر آن به صورت ستونی از x گرفته می‌شوند. اگر ایکس تعداد m*n عدد عنصر نداشته باشد خطایی تولید می‌شود.

۵. عناصر ۱۶ نویسه‌ای h_{x16} را به اعداد دودویی ۱۶ بیتی تبدیل کنید (یعنی uint16s). این ۳ عبارت در رابطه فشرده زیر جایگزین می‌شوند: y=uint16. آنها هسته اصلی bin2dec هستند. که معادل اعشاری رشته دودویی را تولید می‌کند ولی سریعتر هستند زیرا جنبه‌های عمومی آنها کاهش یافته است. تابع pow2y مطلب برای تولید ارایه‌ای به کار برده می‌شود که عناصر آن ۲ به توان y هستند. یعنی twos= pow2 آرایه ۳۲۷۶۸ را تولید می‌کند.

مثال ۶.۳:برای تشریح بیشتر عملکرد فشرده‌سازی روش رمزگذاری هافمن یک تصویر تک رنگ ۸ بیتی ۵۱۲*۵۱۲ را در عکس ۶.۴(a)نظر بگیرید. فشرده‌سازی این تصویر با استفاده از mat2huff با اجرای دستورات زیر روی خط فرمان تحقق می‌یابد.

```
>> f = imread('Tracy.tif');
>> c = mat2huff(f);
>> cr1 = imratio(f, c)
cr1=
    1.219
```

تصویر ۶.۴: یک تصویر ۸ بیتی ۵۱۲*۵۱۲ تک رنگ از یک زن و بزرگنمایی چشم راست او

با حذف کردن داده‌های اضافی موجود در رمزگذاری دودویی ۸ بیتی متعارف تصویر به ۸۰٪ اندازه اصلی خودش (حتی با احتساب اطلاعات عمومی رمزگشایی) فشرده می‌شود.

از آنجائی که داده‌های خروجی mat2huff یک سازه هستند آنها را با استفاده از تابع save روی دیسک ذخیره می‌کنیم:

```
>> save SqueezeTracy c;
>> cr2 = imratio('Tracy.tif', 'SqueezeTracy.mat')
cr2 =
      1.2365
```

تابع save برای ذخیره سازی پرونده با سایر پسوندها است و در این مورد پرونده ایجاد شده را با پسوند mat. ذخیره می‌کند. در اینجا پرونده حاصل از آن به نام SqueezeTracy.mat یک پرونده از نرم افزار MATLAB نامیده می‌شود. این پرونده حاوی داده‌های دودویی و اسامی متغیرها و مقادیر محیط کار است. همچنین متغیر انفرادی c را نیز دارد. علت اختلاف اندک نسبت فشرده‌سازی cr1, cr2 که در بخش قبل محاسبه کردیم داده‌های متفرقه پرونده نرم افزار MATLAB است.

۱.۲.۳. رمزگشایی هافمن (Huffman Decoding)

تصویرهای رمزگذاری شده هافمن کاربرد اندکی دارند مگر آن که بتوان آنها را دوباره به صورت تصاویر اصلی رمزگشایی کرد. اگر y=mat2huff x در داده‌های خروجی بخش قبل باشد، ابتدا رمزگشا باید رمزی را که در هافمن برای رمزگشایی x به کار می‌رود باز کند (که این کار بر اساس پیشینه نما (histogram) و اطلاعات ایکس انجام می‌شود) و سپس داده‌های رمزگذاری شده را به صورت معکوس ترسیم کند (که آنها برای بازسازی x از y استخراج می‌شوند). همان طور که در فهرست تابعهای x=huff2mat(y) دیده می‌شود، این فرایند را می‌توان به ۵ مرحله اساسی تجزیه کرد:

۱. استخراج ابعاد m, n و حداقل مقدار xmin (داده‌های خروجی x) از سازه داده‌های ورودی y

۲. برای بازسازی رمز هافمنی که برای رمزگذاری x به کار رفته است پیشینه نمای آن را به تابع هافمن بدهید. رمز تولید شده در فهرست به نام map ظاهر می‌شود.

۳. ساختار اطلاعاتی جدولهای داده‌ها و انتقال را ایجاد کنید تا با یک سری جستجوهای دودویی کارآمد رایانه‌ای رمزگشایی داده‌های رمزگذاری شده در y.code تسریع شود.

۴. داده‌های رمزگذاری شده و ساختار مربوطه را به تابع c به نام unravel منتقل کنید. این تابع زمان مورد نیاز برای اجرای جستجوهای دودویی را کمینه می‌کند و بردار رمزگشایی شده داده‌های x را که از نوع double یا مضاعف است ایجاد می‌کند.

۵. xmin را به هر عنصر ایکس اضافه کنید و شکل آن را تغییر دهید تا مطابق با ابعاد x اصلی شود (یعنی m عدد ردیف و n عدد ستون)

یکی از خصوصیات منحصر به فرد huff2mat ادغام یکی از تابعهای c نرم افزار MATLAB به نام unravel است (به مرحله ۴ مراجعه کنید) که باعث می‌شود رمزگشایی تصویرهایی با تفکیک پذیری عادی فوری انجام شود.

```
function x= huff2mat(y)
%HUFF2MAT decodes a Huffman encoded matrix.
%     X = H UFF2MAT(Y) decodes a Huffman encoded structure Y with uint16
%     fields:
%     Y.min          Minimum values of X plus 32768
%     Y.size         Size of X
%     Y.hist               Histogram of X
%     Y.code         Huffman code
%
%     The output X is of class double.
%
%     See also MAT2HUFF.

If –isstruct(y) | ~isfield(y, 'min') |  ~isfield(y, 'size') | …
      ~isfield(y, 'hist') | ~isfield(y, 'code')
   error('The input must be a structure as returned by MAT2HUFF.');
end

sz = double(y.size);    m = sz(1);        n = sz(2);
xmin = double(y.min) – 32768;             % Get X minimum
map = huffman(double(y.hist));                  % Get Huffman code (cell)

% Create a binary search table for the Huffman decoding process.
% 'code' contains source symbol strings corresponding to 'link'
% nodes, while 'link' contains the address (+) to node pairs for
% node symbol strings plus '0' and '1' or addresses (-) to decoded
% Huffman codewords in 'map'. Array 'left' is a list of nodes yet to
% be processed for 'link' entries.

code = cellstr(char('', '0', '1'));             % Set starting conditions as
link = [2; 0; 0];       left = [2 3]                  % 3 nodes w/2 unprocessed
found = 0;  tofind = length(map);         % Tracking variables

while length(left) & (found < tofind)
      look = find(strcmp(map, code{left(1)}));        % Is string in map?
      if look                                 % Yes
            link(left(1)) = -look;        % Point to Huffman map
            left = left(2:end);                 % Delete codes found
            found = found + 1;                  % Increment codes found
      else                                    % No, add 2 nodes & pointers
            len = length(code);                 % Put pointers in node
            link(left(1)) = len + 1;

            link = [link; 0; 0];                % Add unprocessed nodes
            code{end + 1} = stract(code{ left(1)}, '0');
            code{end + 1} = stract(code{ left(1)}, '1');

            left = left(2:end);                 % Remove processed nodes
            left = [left len + 1 len + 2];      % Add 2 unprocessed nodes
      end
end
x = unravel(y.code', link, m * n);        % Decode using C 'unravel'
x = x + xmin – 1;                         % X minimum offset adjust
x = reshape(x, m, n);                     % Make vector an array
```

همان طور که قبلاً گفتیم، رمزگشایی بر اساس huff2mat مبتنی بر جستجوهای دودویی و یا تصمیم به رمزگشایی بر اساس نتایج دوگانه است.

هر عنصر از رشته رمزگذاری شده هافمن که به ترتیب بررسی می‌شود که البته باید ۰ یا ۱ باشد باعث راه اندازی روند رمزگشایی دودویی بر اساس جدول داده‌های خروجی و انتقال می‌شود. هر یک از عناصر ارایه link سه حالته مطابق با رشته‌های دودویی رمزگذاری شده هافمن در آرایه سلولی مطابق با آنها هستند. در ابتدا code=cellstr(char(‘’,’0’, ‘1’)) است رشته تهی کد (۱) نقطه مبداء (و یا حالت رمزگشایی مقدماتی) است که در همه رمزگشاییهای رشته هافمن به کار برده می‌شود. عدد ۲ در پیوند (۱)،دو حالت رمزگشای محتمل را شناسایی می‌کند که ۰ یا ۱ را به رشته تهی متصل می‌کنند. اگر بیت رمزگذاری شده بعدی هافمن صفر باشد، حالت رمزگشای بعد link(2) است (از آنجائی که code(2)=0 است رشته تهی با صفر ارتباط دارد). اگر ۱ باشد حالت جدید link(3) می‌شود. توجه داشته باشید که ورودی‌های آرایه پیوند صفر هستند یعنی طوری پردازش نشده‌اند که نمایانگر تصمیم گیری صحیح در ترسیم رمز هافمن باشند. در حین ساخت این پیوند اگر هر یک از رشته‌های ۰ یا ۱ در نقشه یافت شود، (یعنی یک رمز معتبر هافمن باشد) صفر منطبق با آن در پیوند جایگزین مقدار منفی شاخص نقشه مطابق با آن می‌شود (که همان مقدار رمزگشایی شده است). در غیر این صورت یک پیوند جدید با مقدار مثبت درج می‌شود تا اشاره به ۲ حالت جدید داشته باشد. (که اینها رمزهای محتمل هافمن هستند) که منطقاً یا بر اساس ۰۰ و ۰۱ هستند و یا ۱۰ و ۱۱.

این عناصر پیوندی پردازش نشده جدید link را افزایش می‌دهند (رمز آرایه سلول نیز باید update شود) فرایند ساخت آنقدر ادامه می‌یاد که دیگر هیچ عنصر پردازش نشده‌ای در link باقی نماند. به جای بررسی مستمر عناصر پردازش نشده در این پیوند huff2mat یک ارایه ردیابی به نام left دارد که با ۲و ۳ راه اندازی می‌شود و به هنگام سازی(update) می‌شود تا شاخصهای عناصر پیوندی بررسی نشده را نیز در بر بگیرد.

جدول ۶.۳ جدول این پیوند است که برای رمز هافمن در مثال ۶.۲ ایجاد شده است. اگر هر شاخص پیوند یک حالت رمزگشای I محسوب شود، هر یک از رمزگذاری‌های دودویی (در بررسی رشته رمزگذاری شده از چپ به راست) و یا داده‌های خروجی رمزگشایی شده هافمن با link(i) مشخص می‌شود.

۱. اگر link (i)<0 باشد (یعنی منفی باشد) یکی از کلمات رمز هافمن رمزگشایی شده است. در اینجا link(i) داده خروجی رمزگشایی شده است و قدر مطلق در | | قید شده است.

۲. اگر link(i) >0 باشد، (یعنی مثبت باشد) و بیت رمزگذاری شده بعدی ۰ باشد حالت رمزگشایی بعدی شاخص link i است یعنی فرض می‌کنیم i = link(i) است.

۳. اگر link(i) >0 باشد و بیت رمزی شده بعدی ۱ باشد حالت رمزگشایی بعدی شاخص link(i) +1 است یعنی i = link(i) + 1

Index i	Value in link(i)
1	2
2	4
3	−2
4	−4
5	6
6	−3
7	−1

جدول ۶.۳: جدول رمزگشایی ارایه سلولی کاهش منبع در تصویر ۶.۳

همان طور که قبلاً گفتیم ورودی‌های پیوندهای مثبت مطابق با رمزگشایی‌های انتقالی دودویی هستند در حالی که ورودی‌های منفی مقادیر رمزگشایی شده داده‌های خروجی را تعیین می‌کنند. بعد از رمزگشایی هر یک از کدهای هافمن یک جستجوی دودویی در شاخص link آغاز می‌شود که توالی انتقال حالت حاصل از آن به شرح زیر است:i=1,3,1,2,5,6,1,… توالی داده‌های خروجی مطابق با آن ..,_,_,|3_|,_,_,|2_|,_ است. خط فاصله نمایانگر نبود داده‌های خروجی است. داده‌های خروجی رمزگشایی شده ۲ و ۳ پیکسل اول ستون اول تصویر آزمایشی f_2 در مثال ۶.۲ است.

تابع unravel زبان C سازه فوق را قبول می‌کند و از آن برای جستجوی دودویی برای رمزگشایی داده‌های ورودی h_x استفاده می‌کند. عملکرد اساسی آن در نمودار ۶.۵ ترسیم شده است که مطابق با فرایند تصمیم گیری تشریح شده در جدول ۶.۳ است. تغییراتی انجام شده است تا شاخص بندی آرایه‌های C از صفر به جای یک جبران شود.

می‌توان تابعهای زبان C و Fortran را در نرم افزار Matlab ادغام کرد تا به دو هدف دست یافت:

(۱) می‌توان برنامه‌های Fortran و C را بدون نیاز به بازنویسی آنها به صورت پرونده‌های M از نرم افزار MATLAB فراخوانی کرد.

(۲) آنها باعث تسریع محاسبه برخی پرونده‌های M نرم افزار MATLAB می‌شوند که معمولاً با سرعت کافی انجام نمی‌شوند ولی می‌توان برای افزایش کارآیی آنها را درC یا Fortran رمزگذاری کرد. صرف نظر از استفاده از C یا Fortran تابع‌های حاصل از آن را پرونده‌های MEX می‌نامیم. رفتار آنها طوری است که گویی پرونده‌های M یا تابعهای عادی نرم افزار MATLAB هستند. ولی بر خلاف پرونده‌های M آنها را باید قبل از فراخوانی با استفاده از پرونده آغازگر MEX در نرم افزار MATLAB تدوین و مرتبط کرد. مثلاً برای تبدیل و پیوند تابع unravel در محیط ویندوز عبارت زیر در خط فرمان نرم افزارMatlab باید نوشته شود:

```
>> mex unravel.c
```

سپس یک پرونده MEX به نام unravel.dll با پسوند dll ایجاد می‌شود. هر نوع متن راهنما به صورت یک پرونده M جداگانه با همان نام و پسوند m. ایجاد می‌شود.

کد منبع پرونده C MEXبه نام unravel پسوند c. دارد و به شرح زیر است:

Unravel.c

که یک رشته به طول متغیر رمزگذاری شده را رمزگشایی می‌کند (و برداری با اعداد صحیح ۱۶ بیتی است) و روند مرتب سازی آن دودویی از msb به lsb است که بر اساس جدول انتقال انجام می‌شود.

```
/*============================================
* unravel.c
* Decodes a variable length coded bit sequence (a vector of
* 16-bit integers) using a binary sort from the MSB to the LSB
* (across word boundaries) based on a transition table.
*============================================
#include "mex.h"
void unravel(unsigned short *hx, double *link, double *x,
      double xsz, int hxsz)
{
      int I = 15, j = 0, k = 0, n = 0;          /* Start ar root node, 1st  */
                                          /* hx bit and x element */
while (xsz – k)   {                             /* Do until x is filled */
      if (*(link + n) > 0)       {                  /* Is there a link? */
            if ((*hx + j) >> j) & 0x0001)       /* Is bit a 1? */
                  n = *(link + n);              /* Yes, get new node */
            else n = *(link + n) – 1;                 /* It's 0 so get new node */
            if (i) i--; else {j++; i = 15;}           /* Set i, j to next bit */
            if (j > hxsz)                             /* Bits left to decode? */
                  mexErrMsgTxt("Out of code bits ???");
            }
      else        {                             /* It must be a leaf node */
            * (x + k++) = -*(link + n);         /* Output value */
            n = 0;            }                       /* Start over at root */
}
if (k == xsz – 1)                               /* Is one left over? */
      * (x + k++) = -*(link + n);
}
void mexFunction(int nlhs, mxArray *plhs[],
                              int nrhs, const mxArray *prhs[])
{
      double *link, *x, xsz;
      usigned short *hx;
      int hxsz;

/* Check inputs for reasonableness */
if (nrhs != 3)
      mexErrMsgTxt("Three inputs required.");
else if (nlhs > 1)
      mexErrMsgTxt("Too output argument.");
```

```
/* Is last input argument a scalar? */
if (!mxIsDouble(prhs[2])            || mxIsComplex(prhs[2]) ||
      mxGetN(prhs[2]) * mxGetM(prhs[2]) != 1)
      mexErrMsgTxt("Input XSIZE must be a scalar.");
/* Create input matrix pointers and get scalar */
hx = mxGetPr(prhs[0]);              /* UINT16 */
link = mxGetPr(prhs[1]);                  /* DOUBLE */
xsz = mxGetScalat(prhs[2]);         /* DOUBLE */

/* Get the number of elements in hx */
hxsz = mxGetM(prhs[0]);

/* Create 'xsz' x 1 output matrix */
plhs[0] = mxCreateDoubleMatrix(xsz, 1, mxREAL);

/* Get C pointer to a copy of  the output matrix */
x = mxGetpr(plhs[0]);

/* Call the C subroutine */
unravel(hx, link, x, xsz, hxsz);
}
```

متن راهنما در پرونده M به نام unravel.m ایجاد می‌شود.

```
%UNRAVEL Decodes a variable-length bit stream.
%     X =   UNRAVEL(Y, LINK, XLEN) decodes UINT16 input vector Y based on
%     transition and output table LINK. The elements of Y are
%     considered to be a contiguous stream of encoded bits--i.e., the
%     MSB of one element follows the LSB of the previous element. Input
%     XLEN is the number of code words in Y, and thus the size of output
%     vector X (class DOUBLE). Input LINK is a transition and output
%     table (that drives a series of binary searches):
%
%     1. LINK(0) is the entry point for decoding, i.e., state n = 0.
%     2. If  LINK(n) < 0, the decoded output is | LINK(n)|; set n = 0.
%     3. If  LINK(n) > 0, get the next encoded bit and transition to
%           state [LINK(n) – 1] if the bit is 0, else LINK(n).
```

پرونده C MEX، unravel.c همچون سایر پرونده‌های C MEX از دو بخش متمایز تشکیل شده است که آنها را روال محاسباتی و دروازه می‌نامیم.

روال محاسباتی که آن را روال گره‌گشا نیز می‌نامیم حاوی رمز C است که فرایند رمزگشایی تصویر ۶.۵ را اجرا می‌کند.

نام همیشگی روال دروازه MexFunction است که بین روال محاسباتی unravel در c و پرونده M فراخوان تابع huff2mat ارتباط ایجاد می‌کند

از میانجی استاندارد پرونده Mex نرم افزار MATLAB استفاده می‌کند که بر اساس موارد زیر است:

۱. چهار پارامتر استاندارد از داده‌های ورودی و خروجی:

این پارامترها شماره شناسه‌های سمت چپ داده‌های خروجی (عدد صحیح)، و آرایه‌ای از اشاره‌گرهای سمت چپ شناسه‌های داده‌های خروجی (که همگی آرایه‌های نرم افزار MATLAB هستند) می‌باشند. تعداد شناسه‌های داده‌های ورودی سمت راست (یک عدد صحیح دیگر)، و آرایه اشاره‌گرهای شناسه‌های داده‌های ورودی سمت راست (همگی آرایه‌های نرم افزار MATLAB هستند)

۲. مجموعه‌ای از تابعهای میانجی برنامه‌های کاربردی که در نرم افزار Matlab تهیه شده است. این تابعها که پسوند mx دارند برای ایجاد، دسترسی به، مدیریت، و یا تخریب ساختار آرایه‌های نوع mx به کار برده می‌شوند. مثلاً

mxCalloc داده‌های حافظه را همچون یک تابع استاندارد C Calloc به شکل پویا تخصیص می‌دهد. تابعهای مرتبط mxMalloc, mxRealloc هستند که به جای تابعهای Cmalloc, realloc به کار برده می‌شوند.

mxGetScalar کمیت عددی را از داده‌های ورودی آرایه prhs استخراج می‌کند. سایر تابعهای mxGet مانند mxGetM, mxGetn, mxGetString سایر انواع داده را استخراج می‌کنند.

mxCreateDoubleMatrix یک آرایه MATLAB برای plhs ایجاد می‌کند. سایر تابعهای MxCreate مانند mxCreateString, mxCreateNumericArray ایجاد سایر انواع داده‌های را تسهیل می‌کنند.

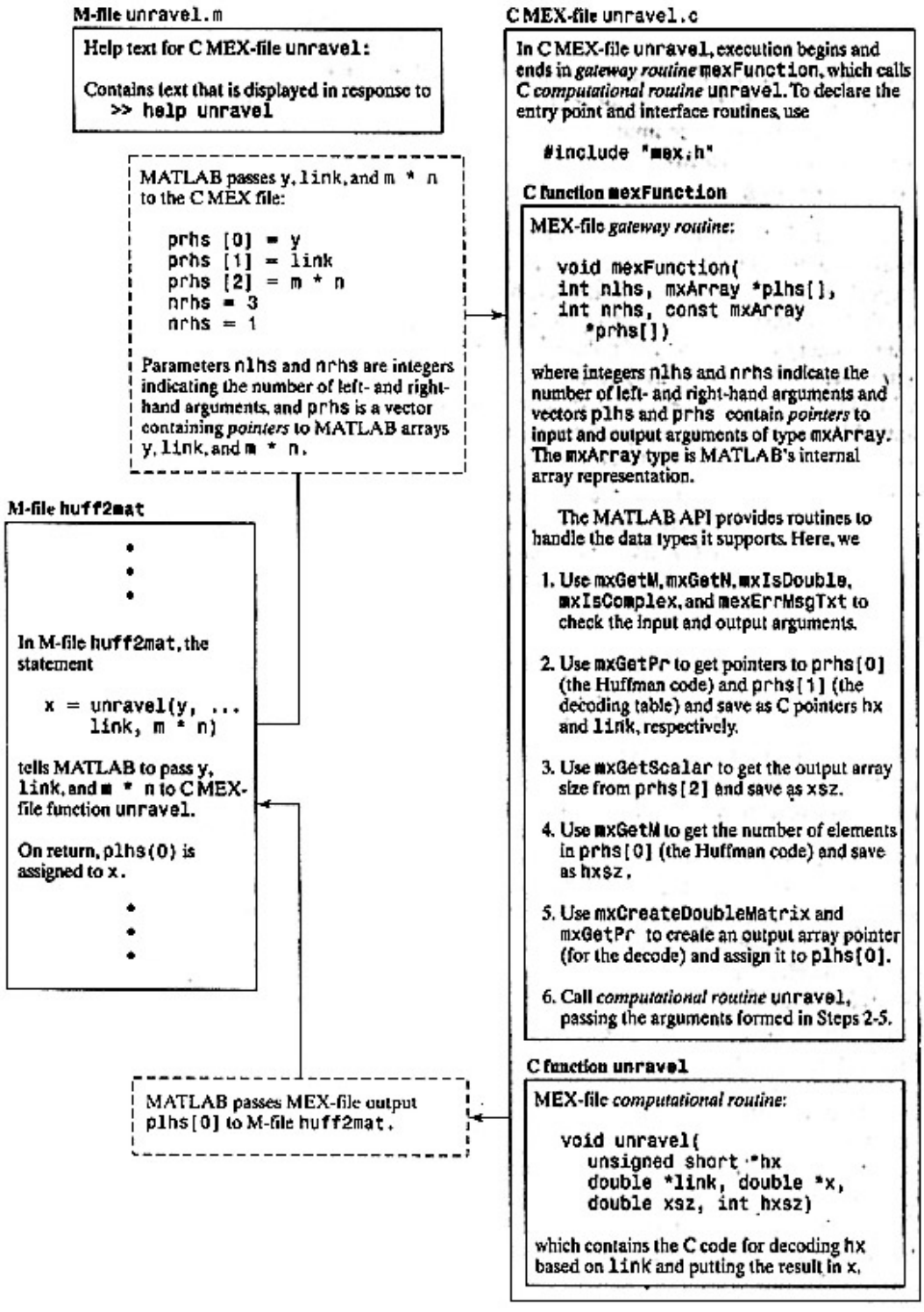
تابعهای میانجی با پسوند mex عملیاتی را در محیط نرم افزار MATLAB انجام می‌دهند. مثلاً mexErrMsgTxt یک پیام را در محیط کار نرم افزار MATLAB قرار می‌دهد.

تابعهای اصلی میانجی mex، mx که در بخش ۲ فهرست قبل به آنها اشاره شد در پرونده‌های سرنویس نرم افزار MATLAB با اسامی mex.h, matrix.h هستند. هر دو در مسیر بیرونی نصب نرم افزار MATLAB هستند. <matlab> ریشه درایوی است که نرم افزار MATLAB روی آن نصب می‌شود.

سرنویس mex.h باید در آغاز همه پرونده‌های mex درج شود. (به عبارت درج پرونده c "mex.h" #include توجه کنید که دارای پرونده سرنویس metrix.h است.

نمونه‌های اصلی روالهای میانجی mex, mx که در این پرونده‌ها درج شده است پارامترهای مورد استفاده را تعریف می‌کنند و نشانه‌های ارزشمند ی درباره عملیات کلی آنها دارند. سایر اطلاعات دردفترچه راهنمای میانجی‌های بیرونی نرم افزار MATLAB(MATLAB External Interfaces Refrences) قید شده است.

بحث قبل در تصویر ۶.۶ خلاصه شده است و دارای جزئیات ساختار پرونده‌های c mex است و جریان اطلاعات را بین آنها و پرونده M huff2mat نشان می‌دهد. گرچه بر اساس رمزگشایی هافمن ایجاد شده است، ولی مطالب آن را می‌توان به راحتی به سایر تابعهای MATLAB وC و Fortran نیز تعمیم داد.



تصویر ۶.۶: کنش و واکنش پرونده mhuff2mat و تابع unravel فراخوان‌پذیر در نرم افزار MATLAB توجه داشته باشید که c mex file unravel ۲ کاربرد دارد: به عنوان روال دروازه mex function و روال محاسباتی unravel متن راهنما در پرونده دیگری به نام unravel است.

تصویر رمزگذاری شده هافمن در مثال ۶.۳ را می‌توان با توالی فرمانهای زیر رمزگشایی کرد:

مثال ۶.۴: رمزگشایی با huff2mat

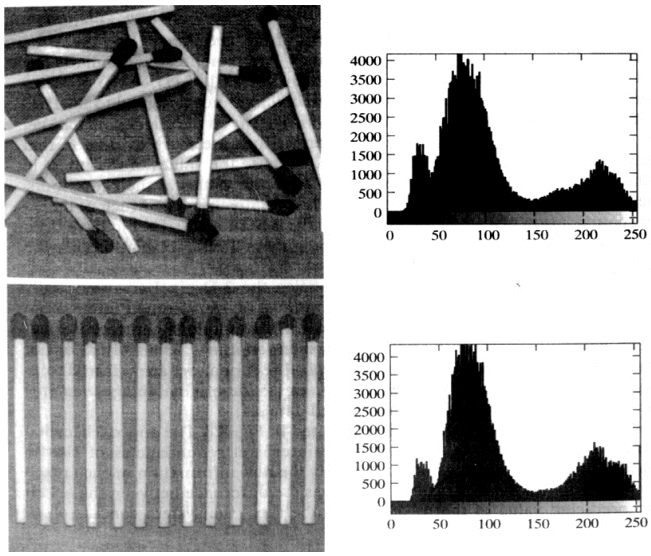
تابع load file متغیرهای MATLAB را از پرونده‌ها می‌خواند آنها را داخل محیط کار بارگذاری می‌کند. اسامی متغیرها به ترتیب ذخیره سازی و بارگذاری حفظ می‌شوند.

```
>> load SqueezeTracy;
>> g = huff2mat(c);
>> f = imread('Tracy.tif');
>> rmse = compare(f, g)
rmse =
      0
```

توجه داشته باشید که در این فرایند رمزگذاری و رمزگشایی اطلاعات حفظ می‌شوند. خطای جذر میانگین مربعات بین تصویرهای اصلی و نافشرده شده صفر است. از آنجائی که بیشتر رمزگشایی در پرونده‌های c mex انجام می‌شود huff2mat اندکی سریعتر از همتای رمزگذار خود است. به استفاده از تابع بارگیری(load function) برای به دست آوردن داده‌های خروجی رمزگذاری شده در مثال ۶.۳ توجه کنید.

۶.۳. افزونگی بین عناصر تصویری(Interpixel Redundancy)

تصاویر ۶.۷(a) و(c) و ۶.۷(b) و(d) پیشینه نماهای یکسان دارند. این سابقه نماه(histogram)ا سه حالته هستند و مقادیر سطح خاکستری عمدتاً به ۳ شکل در آنها دیده می‌شود.



تصویر ۶.۷: ۲ تصویر و پیشینه نماهای سطح خاکستری آنها

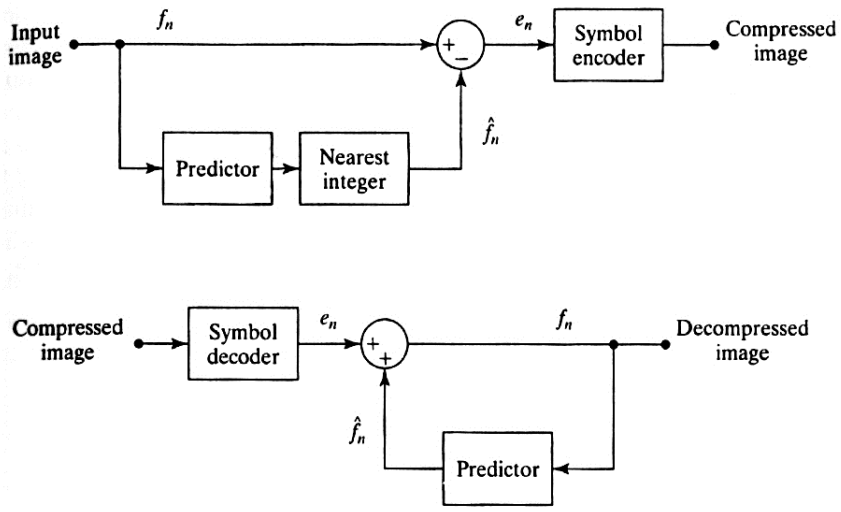
از آنجائی که سطوح خاکستری تصاویر برابر نیستند، رمزگذاری با طول متغیر برای کاهش افزونگی رمزگذاری به کار می‌رود که ریشه آن رمزگذاری طبیعی دودویی پیکسلها است.

```
>> f1 = imread('Random Matches.tif');
>> c1 = mat2huff(f1);
>> entropy(f1)
ans =
      7.4253
>> imratio(f1, c1)
ans =
      1.0704
>> f2 = imread('Aligned Matches.tif');
>> c2 = mat2huff(f2);
>> entropy(f2)
ans =
      7.3505
>> imratio(f2, c2)
ans =
      1.0821
```

توجه داشته باشید که در اینجا شاخص درجه بی‌نظمی ۲ تصویر یکسان است و با mat2huff به طور یکسان فشرده‌سازی شده‌اند (نسبت فشرده‌سازی ۱.۰۷۰۴ به ۱.۰۸۲۱ است). بنا بر این رمزگذاری با طول متغیر برای استفاده از روابط ساختاری آشکار بین موارد همراستا در تصویر ۶.۷(c) نیست. گرچه ارتباطهای عناصر تصویری در این عکس مشخص است ولی در تصویر ۶.۷(a) نیز دیده می‌شوند. از آنجائی که مقادیر پیکسلهای هر تصویر را می‌توان از مقادیر مجاور آنها پیش بینی کرد این عناصر تصویری حامل اطلاعات اندکی هستند. این پیکسلها برای نمایش تصاویر اطلاعات اضافی نیز دارند که می‌توان آن را بر اساس مقادیر مجاور حدس زد. این روابط مبنای خصوصیات اضافی بی‌خاصیت پیکسلها هستند.

برای کاهش افزونگی بین پیکسلها ارایه پیکسل ۲ بعدی که معمولاً برای مشاهده با چشم استفاده می‌شود تبدیل به قالبی کارآمد می‌شود که جنبه تصویری ندارد. مثلاً اختلاف بین عنصرهای مجاور برای نمایش تصویر به کار برده می‌شوند. در تبدیلهایی از این نوع (که افزونگی بین عناصر تصویری برطرف می‌شود) نقشه برداری نامیده می‌شوند. اگر بتوان عناصر تصویر اصلی را با داده‌های تبدیل شده بازسازی کرد نقشه برداری برگشت‌پذیر نامیده می‌شوند.

یک روش نقشه برداری ساده در تصویر ۶.۸ نشان داده شده است. نام این روش پیش بینی رمزگذاری بدون خسارت(lossless predictive coding) است. برای حذف افزونگی بین عناصر تصویری مجاور، اطلاعات جدید هر عنصر استخراج و رمزگذاری می‌شود. اطلاعات جدید هر عنصر تصویری تفاوت بین مقدار واقعی و پیش بینی شده آن عنصر است. همان طور که دیده می‌شود این سیستم از یک رمزگذار و یک رمزگشا تشکیل شده است.



که هر کدام از آنها یک پیش‌بینی کننده یکسان دارد. وقتی عناصر تصویری به صورت پی‌درپی به نام f_n به رمزگذار داده می‌شوند ابزار پیش بینی کننده مقدار قابل پیش بینی آن عنصر تصویری را بر اساس مقادیر داده‌های ورودی قبلی تعیین می‌کند. سپس داده‌های خروجی پیش‌بینی کننده به نزدیکترین عدد صحیح گرد می‌شوند تا f_n را تشکیل دهند و اختلاف یا خطای پیش‌بینی را ایجاد می‌کنند

?

که با استفاده از رمزی با طول متغیر (با نماد رمزگذار) رمزگذاری می‌شود تا عنصر بعدی جریان فشرده داده‌ها را تشکیل دهد. رمزگذار تصویر ۶.۸(b) , e_n را از کلمات رمزی دریافت شده با طول متغیر بازسازی می‌کند و عملیات معکوس را انجام می‌دهد.

?

می‌توان برای تولید $\hat{f}_n$ از روشهای سازگار گوناگون موضعی و کلی استفاده کرد. این پیش‌بینی‌ها در اکثر موارد با ترکیب خطی m عدد عنصر تصویری انجام می‌شود یعنی این که

?

در اینجا m ترتیب ابزار پیش‌بینی کننده خطی است. round تابعی است که برای گرد کردن و رسیدن به نزدیکترین عدد صحیح به کار برده می‌شود (مانند تابع مطابق با آن در نرم افزار Matlab) و ai در$m, \ldots, 2, i=1$ ضرایب پیش‌بینی‌ها هستند. این معادله در رمزگذاری خطی تک بعدی به شکل زیر نوشته می‌شود:

?

هر یک از متغیرهای زیرنویس به صورت تابعی از مختصات فضایی x و y بیان می‌شود. توجه داشته باشید که تابع f(x, y) تابع عناصر تصویری قبلی در خط پیمایش جاری است.تابعهای M mat2Lpc, Lpc2mat برای اجرای رمزگذاری همراه با پیش‌بینی و رمزگشای (به غیر از مراحل نمادین رمزگذاری و رمزگشایی) هستند. تابع رمزگذار mat2Lpc برای پیش‌بینی هر یک از عناصر تصویری و قرار دادن آن در داده‌های ورودی x از حلقه استفاده می‌کند. بعد از هر دفعه تکرار xs که به صورت کپی از ط است یک ستون به سمت راست منتقل می‌شود و لایه‌گذاری سمت چپ صفر است و ضربدر یک ضریب پیش‌بینی مناسب می‌شود و با مجموع پیش‌بینی‌های p جمع می‌شود. از آنجائی که تعداد ضرایب پیش‌بینی خطی معمولاً اندک است، کل این فرایند سریع انجام می‌شود. در فهرست زیر توجه داشته باشید که اگر فیلتر پیش‌بینی f مشخص نشود از یک فیلتر تک عنصری با ضریب ۱ استفاده می‌شود.

```
function y = mat2lpc(x, f)
%MAT2LPC Compresses a matrix using 1-D lossless predictive coding.
%     Y = MAT2LPC(X, F) encodes matrix X using 1-D lossless predictive
%     coding. A linear prediction of X is made based on the
%     coefficients in F. If F is omitted, F = 1 (for previous pixel
%     coding) is assumed. The prediction error is then computed and
%     output as encoded matrix Y.
%
%     See also LPC2MAT.

error(nargchk(1, 2, nargin));              % Check input arguments
if nargin < 2                                    % Set default filter if omitted
      f = 1;
end

x = double(x);                             % Ensure double for computations
[m, n] = size(x);                          % Get dimensions of input matrix
p = zeros(m, n);                           % Init linear prediction to 0
xs = x;     zc = zeros(m, 1);              % Prepare for input shift and pad

for j = 1:length(f)                              % For each filter coefficient…
      xs = [zc xs(:, 1:end – 1)];                % Shift and zero x
      p = p + f(j) * xs;                         % Form partial prediction sums
end
y = x – round(p);                          % Compute the prediction error
```

تابع رمزگشای Lpc2mat عملیات معکوس همتای رمزگذار خود با نام mat2lpc را انجام می‌دهد. همان طور که در فهرست زیر دیده می‌شود، از حلقه‌ای با n عدد تعداد دفعات تکرار استفاده می‌کند که در اینجا n تعداد ستونهای ماتریس رمزگذاری شده داده‌های ورودی y است. در هر دفعه تکرار یک ستون از داده‌های خروجی رمزگشایی شده x محاسبه می‌شود زیرا هر ستون رمزگشایی شده برای محاسبه همه ستونهای بعدی به کار می‌رود. برای کاهش زمان مصرف شده در حلقه فوق قبل از شروع حلقه مقدار ایکس پیشاپیش به حداکثر لایه‌گذاری تخصیص داده می‌شود. توجه داشته باشید که محاسبات مورد نیاز برای تولید پیش‌بینی‌ها با همان ترتیبی که در Lpc2mat برای جلوگیری از خطای گرد کردن نقطه شناور انجام می‌شد انجام می‌شوند.

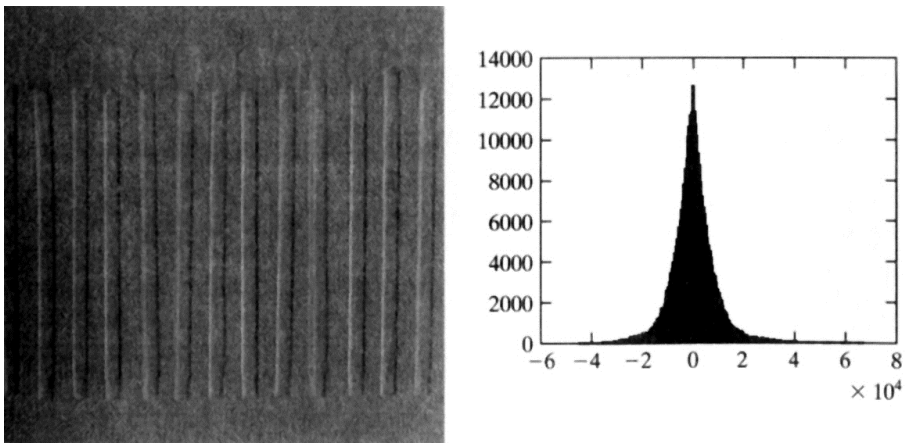
```
function x = lpc2mat(y, f)
%LPC2MAT Decompresses a 1-D lossless predictive encoded matrix.
%     X = LPC2MAT(Y, F) decodes input matrix Y based on linear
%     prediction coefficients in F and the assumption of 1-D lossless
%     predictive coding. If F is omitted, filter F = 1 (for previous
%
%     See also MAT2LPC.

error(nargchk(1, 2, nargin));              % Check input arguments
if nargin < 2                                    % Set default filter if omitted
      f = 1;
end

f = f(end:-1:1);                           % Reverse the filter coefficients
[m, n] = size(y);                          % Get dimensions of output matrix
order = length(f);                               % Get order of linear predictor
f = repmat(f, m, 1);                       % Duplicate filter for vector zing
x = zeros(m, n + order);                         % pad for 1st 'order' column
decodes
% Decode the output one column at a time. Compute a prediction based
% on the 'order' previous elements and add it to the prediction
% error. The result is appended to the output matrix being built.
for j = 1:n                              % For each filter coefficient…
      jj = j + order;
      x(:, jj) = y(:, j) + round(f(:, order:-1:1) .* …
                                 x(:, (jj – 1): -1:(jj – order)), 2));
end
x = x(:, order + 1:end);                         % Remove left padding
```

مثال ۶.۵: رمزگذاری بدون اتلاف پیش‌بینی

رمزگذاری تصویر ۶.۷(c) با استفاده از ابزار پیش‌بینی خطی درجه ۱ در نظر بگیرید:

$$\hat{f}(x,y) = round\left[af(x,y-1)\right]$$

پیش‌بینی کننده‌ای از این نوع را عملیات پیش‌بینی کننده عناصر قبلی می‌نامیم و این نوع روش رمزگذاری و پیش‌بینی را رمزگذاری تفاضلی می‌نامیم. تصویر دارای خطای پیش‌بینی که در حالت a=1 ایجاد شده است در عکس ۶.۹(a) دیده می‌شود. در اینجا سطح خاکستری ۱۲۸ مطابق با خطای پیش‌بینی صفر است.



تصویر ۶.۹: (a) تصویر خطای پیش‌بینی تصویر ۶.۷ (b) پیشینه نمای خطای پیش‌بینی

خطاهای مثبت و منفی غیرصفر به ترتیب توسط mat2gray سایه‌های روشن و تیره خاکستری می‌شوند.

```
>> f = imread('Aligned Matches.tif');
>> e = mat2huff(f);
>> imshow(mat2gray(e));
>> entropy(e)
ans =
      5.9727
```

توجه داشته باشید که شاخص درجه بی‌نظمی خطای پیش‌بینی e بسیار کمتر از همین شاخص در تصویر اصلی f است. مقدار این شاخص از ۷.۳۵۰۵ (که در اوایل این بخش محاسبه شد) به ۵.۹۷۲۷ بیت در هر پیکسل کاهش یافته است در صورتی که در تصویرهای m بیتی برای نمایش دقیق توالی خطای حاصل از آنها باید m+1 بیت مورد نیاز است. کاهش شاخص درجه بی‌نظمی نشان می‌دهد که تصویر خطای پیش‌بینی را می‌توان کارآمدتر رمزگذاری کرد تا تصویر اصلی هدف نقشه برداری شود. بنا بر این

```
>> c = mat2huff(e);
>> cr = imratio(f, c)
cr =
      1.3311
```

همان طور که انتظار می‌رود می‌بینیم نسبت فشرده‌سازی از ۱.۰۸۲۱ (وقتی سطوح خاکستری مستقیماً با هافمن رمزگذاری می‌شوند) به ۱.۳۳۱۱ افزایش یافته است.

سابقه نمای خطای پیش‌بینی e در تصویر ۶.۹(b) نشان داده شده است و محاسبه آن به شرح زیر است:

```
>> [h, x] = hist(e(:) * 512, 512);
>> figure; bar(x, h, 'k');
```

توجه داشته باشید که نزدیک به صفر به نقطه اوج خود می‌رسد و در قیاس با میزان سطوح خاکستری تصویرهای ورودی اختلاف اندکی دارد (تصویر ۶.۷ d). در این حالت همان طور که در شاخصهای محاسبه شده درجه بی‌نظمی دیدیم در فرایندهای ایجاد اختلاف و پیش‌بینی اکثر افزونگی‌های بین پیکسلها از بین می‌رود. با نمایش روش رمزگذاری بدون اتلاف اطلاعات یعنی با رمزگشایی C و مقایسه آن با تصویر اولیه f این مثال را به پایان می‌رسانیم.

```
>> g = lpc2mat(huff2mat(c));
>> compare(f, g)
ans =
      0
```

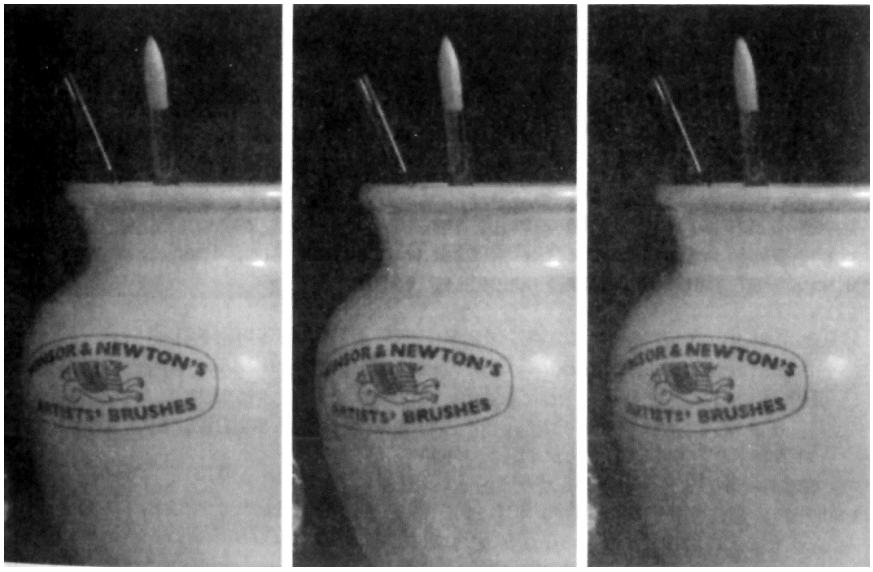
۶.۴. افزونگی روانی- دیداری (Psychovisual Redundancy)

این نوع افزونگی بر خلاف رمزگشایی و افزونگی بین پیکسلی با اطلاعات تصویری سنجش‌پذیر واقعی سر و کار دارد. حذف آن از آن جهت مطلوب است که این اطلاعات برای پردازش تصویری عادی ضروری نیست. از آنجائی که حذف داده‌های اضافی مربوط به جنبه روانی – دیداری باعث از دست دادن اطلاعات کمی می‌شود آن را کمی سازی می‌نامیم. این کلمه فنی با کاربرد این واژه سازگار است یعنی نقشه برداری از حیطه گسترده‌ای از داده‌های ورودی برای ایجاد تعداد محدودی داده‌های خروجی. از آنجائی که این فرایند برگشت‌ناپذیر است (یعنی اطلاعات دیداری از بین می‌رود) کمی سازی معمولاً منجر به فشرده‌سازی همراه با از دست دادن اطلاعات می‌شود.

مثال ۶.۶: فشرده‌سازی با کمی سازی

عکسهای تصویر ۶.۱۰ را در نظر بگیرید. در تصویر ۶.۱۰(a) یک عکس تک رنگ با سطوح خاکستری ۲۵۶ نشان داده شده است. تصویر ۶.۱۰(b) همان تصویر بعد از کمی سازی ۴ بیتی در ۱۶ سطح محتمل است. نسبت فشرده‌سازی ۲:۱ است. توجه داشته باشید که محدوده‌سازی کاذب در نواحی هموار تصویر اصلی دیده می‌شود. این تاثیر تصویری طبیعی نمایش سطوح خاکستری تصویر است.

در تصویر ۶.۱۰(c) بهبود وضعیت با کمی سازی دیده می‌شود که از خصوصیات چشم انسان هنگام تماشای تصویر بهره‌برداری شده است. گرچه فشرده‌سازی حاصل از این نوع کمی سازی ۲:۱ است برای کاهش محدوده سازی کاذب دادنه‌های بیشتر ولی نه چندان ناراحت کننده کنده دیده می‌شوند. توجه داشته باشید که در هر ۲ مورد نافشرده‌سازی غیرضروری و غیرممکن است. (یعنی کمی سازی عملیات برگشت‌ناپذیر است).



تصویر ۶.۱۰: (a) تصویر اصلی ۰(b)کمی سازی یکسان تا ۱۶ سطح ((c)می سازی ای جی اس تا ۱۶ سطح

روش مورد استفاده برای تولید تصویر ۶.۱۰(c) را کمی سازی بهینه شده مقیاس خاکستری (improved gray_scale (IGS) quantization) می‌نامیم. در این حالت حساسیت چشم نسبت به لبه‌ها در نظر گرفته می‌شود و با تخصیص یک عدد تصادفی به هر عنصر که از بیتهای تحتانی پیکسلهای مجاور تولید می‌شود قبل از کمی سازی نتایج آن را تجزیه می‌کنند. از آنجائی که بیتهای پایین رتبه تصادفی هستند ارقام تصادفی تخصیص داده می‌شوند که بستگی به خصوصیات موضعی تصویر و لبه‌های مصنوعی مربوط به محدوده سازی کاذب دارند. تابع quantize هم کمی سازی بهینه مقیاس خاکستری و هم برش سنتی پایین رتبه (low_order) بیتها را انجام می‌دهد. توجه داشته باشید که اجرای کمی سازی بهینه مقیاس خاکستری برداری انجام می‌شود تا داده‌های ورودی x هر دفعه به اندازه یک ستون پردازش شوند. برای ایجاد ستونی از نتایج ۴ بیتی در تصویر ۶.۱۰(c) مجموع ستون s که در ابتدا صفر هستند به صورت مجموع ستون ایکس و ۴ بیت مجموع کنونی ایجاد می‌شود. اگر ۴ بیت مهم هر یک از ایکسها ۲(۱۱۱۱) باشد در عوض ۲(۰۰۰۰) افزوده می‌شود. بنا بر این ۴ بیت مهم مجموع حاصل از آن به عنوان عناصر تصویری رمزگذاری شده ستون پردازش شده به کار برده می‌شوند.

```
function y = quantize(x, b, type)
%QUANTIZE Quantizes the elements of a UINT8 matrix.
%     Y = QUANTIZE(X, 8, TYPE) quantizes X to B bits. Truncation is
%     used unless TYPE is 'igs' for Improved Gray Scale quantization.
error(nargchk(2, 3, nargin));             % Check input arguments
if ndims(x) ~= 2 | ~isreal(x) | …
      ~isnumeric(x) | ~isa(x, 'uint8')
      error('The input must be a UINT8 numeric matrix.');
end

% Create bit masks for the quantization
lo = uint8(2 ^ (8 – b)  - 1);
hi = uint8(2 ^ 8 – double(1o) – 1);

% Perform standard quantization unless IGS is specified
if nargin < 3 | ~strcmpi (type, 'igs')
      y = bitand(x, hi);
% Else IGS quantization. Process column-wise. If the MSB's of the
% pixel are all 1's, the sum is set to the pixel value. Else, add
% the pixel value to the LSB's the previous sum. Then take the
% MSB's of the sum as the quantized value.
else
      [m, n] = size(x);                          s = zeros(m, 1);
      hitest = double(bitad(x, hi) ~= hi);             x = double(x);
      for j = 1:n
            s = x(:, j) + hitest(:, j) .* double(bitand(uint8(s), 1o));
            y(:, j) = bitand(uint8(s), hi);
      end
end
```

کمی سازی بهینه مقیاس خاکستری در اکثر روشهای کمی سازی که تاثیر مستقیم بر سطوح خاکستری تصویرهای هدف فشرده‌سازی دارند دیده می‌شود.

معمولاً میزان تفکیک پذیری فضایی و مقیاس خاکستری کاهش می‌یابد. اگر تصویر برای کاهش افزونگی بین پیکسلها نقشه برداری شده باشد، کمی سازی می‌تواند منجر به کاهش کیفی سایر جنبه‌ها شود مثلاً لبه‌های تار (از دست رفتن اطلاعات فرکانس بالا) وقتی از فرکانس تبدیل دو بعدی برای ارتباط‌زدایی داده‌ها استفاده می‌شود.

مثال ۶.۷: ادغام کمی سازی بهینه مقیاس خاکستری روشهای رمزگذاری هافمن و پیش‌بینی بدون اتلاف اطلاعات

گرچه کمی سازی که برای ایجاد تصویر ۶.۱۰(c) به کار برده شده است اکثر افزونگی‌های روانی – تصویری با تاثیر اندک بر کیفیت تصویر برطرف می‌کند، ولی برای فشرده‌سازی بیشتر از شگردهای ۲ بخش قبل برای کاهش افزونگی بین عناصر تصویری و رمزگذاری استفاده می‌شود. در واقع می‌توان میزان فشرده‌سازی کمی سازی بهینه مقیاس خاکستری را تا ۲ برابر افزایش داد. در فرمانهای زیر کمی سازی بهینه مقیاس خاکستری، رمزگذاری داده‌ها برای پیش‌بینی بدون اتلاف اطلاعات، و رمزگذاری هافمن برای فشرده‌سازی تصویر ۶.۱۰(a)تا یک چهارم اندازه اصلی آن به کار برده شده است.

```
>> f = imread('Brushes.tif');
>> q = quantize(f, 4, 'igs');
>> qs = double(q) / 16;
>> e = mat2lpc(qs);
>> c = mat2huff(e);
>> imratio(f, c)
ans =
      4.1420
```

نتیجه رمزگذاری C را می‌توان با معکوس کردن روند عملیات (بدون معکوس کردن کمی سازی) انجام داد.

```
>> ne = huff2mat(c);
>> nqs = lpc2mat(ne);
>> nq = 16 * nqs;
>> compare(q, nq)
ans =
      0
>> rmse = compare(f, nq)
rmse =
      6.8382
```

توجه داشته باشید که خطای جذر میانگین مربعات تصویر نافشرده شده حدود ۷ سطح خاکستری است و ریشه این خطا مرحله کمی سازی است.

۶.۵. فشرده‌سازی تصویرها با پسوند jpeg

شگردهای فصلهای قبل بر اساس کار روی عناصر تصویری است. و بنا بر این روشهای دامنه فضایی(spatial domain methods) هستند. در این بخش یک سری روشهای فشرده‌سازی را در نظر می‌گیریم که بر اساس تغییر تبدیل تصویر هستند. هدف ما معرفی تبدیلهای ۲ بعدی در فشرده‌سازی تصویر است.

برای تشریح نحوه کاهش افزونگی تصویر در بخشهای ۶.۲ تا ۶.۴ با مثالهای بیشتر و برای مطلع کردن خوانندگان از روشهای هنری فشرده‌سازی تصویر باید بگوییم که استانداردهای عرضه شده (که موارد نزدیک به آنها در نظر گرفته می‌شوند) برای کار با انواع تصویرها و شرایط فشرده‌سازی است.

در رمزگذاری تبدیلی مانند یک تبدیل خطی برگشت‌پذیر همچون DFT فصل ۴ یا تبدیل کسینوسی متمایزDCT

$$T(u,v) = \sum_{x=0}^{M-1}\sum_{y=0}^{N-1} f(x,y)a(u)a(v)\cos\left[\frac{(2x+1)u\pi}{2M}\right]\cos\left[\frac{(2x+1)v\pi}{2N}\right]$$

$$a(u) = \begin{cases} \sqrt{\frac{1}{M}} & u = 0 \\ \sqrt{\frac{2}{M}} & u = 1,2,...,M-1 \end{cases}$$

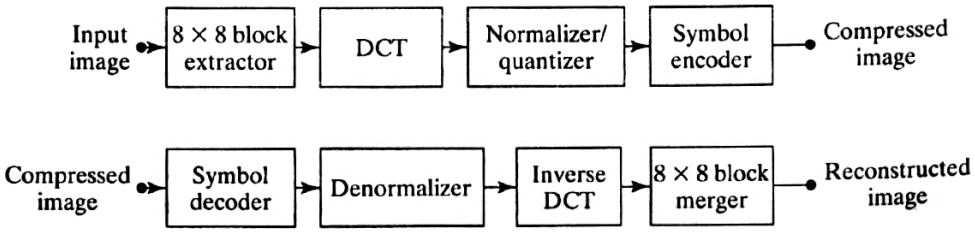
$$\hat{T}(u,v) = round\left[\frac{T(u,v)}{Z(u,v)}\right]$$

αv برای نقشه برداری تصویر روی ضرایب تبدیل به کار برده می‌شود که پس از آن کمی سازی و رمزگذاری می‌شوند. در اکثر تصویرهای طبیعی بیشتر ضرایب بزرگی اندکی دارند و می‌توان آنها را با تحریف اندک تصویر کمی سازی کرد.

Jpeg ۶.۵.۱

یکی از محبوبترین و کاملترین استانداردهای فشرده‌سازی تصویرهای ثابت پسوند jpeg (مخفف گروه متخصصان تصویربرداری مشترک)(Joint Photographic Expert Group) است. در سیستم رمزگذاری مبنای این پسوند که بر اساس تبدیل کسینوس متمایز است و برای اکثر برنامه‌های کاربردی کافی است تصویرهای ورودی و خروجی محدود به ۸ بیت می‌شوند در صورتی که مقادیر ضرایب تبدیل کسینوسی متمایز محدود به ۱۱ بیت می‌شوند. همان طور که در نمودار ساده شده تصویر ۶.۱۱aدیده می‌شود، این فشرده‌سازی در ۴ مرحله پی درپی به نامهای استخراج تصویر فرعی ۸*۸، محاسبه تبدیل کسینوسی متمایز کمی سازی و تخصیص رمزی با طول متغیر انجام می‌شود.

مرحله اول در فشرده‌سازی با پسوند jpeg تقسیم تصویر ورودی به قطعات تصویری بدون همپوشی به اندازه ۸*۸ است. سپس آنها از چپ به راست و بالا به پایین پردازش می‌شوند. حین پردازش هر یک از قطعات ۸*۸ پیکسلهای ۶۴ آن با تفریق 2m-1 جابجا می‌شوند که در اینجا ۲ تعداد سطوح خاکستری تصویر است و تبدیل کسینوسی متمایز ۲ بعدی آن محاسبه می‌شود.



16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

تصویر ۶.۱۲: (a) آرایه هنجارمند کننده تصویرهای jpeg در حالت پیش فرض (b) توالی ترتیب ضریب زیگزاگ jpeg

سپس ضرایب حاصل از آن همزمان مطابق با رابطه زیر هنجارمند و کمی سازی می‌شوند.

?

در اینجا T(u, v) (u,v=0,1,…,7) ضرایب هنجارمند سازی و کمی سازی هستند و T (u, v) تبدیل کسینوسی متمایزDCT تصویر ۸*۸ f(x, y) است و z(u, v) آرایه هنجارمند تبدیل(Transform normalization) است که مانند تصویر ۶.۱۲(a) است. با مقیاس بندی z(u, v) می‌توان انواع نسبتهای فشرده‌سازی را اجرا و خصوصیات تصویر را بازسازی کرد.

پس از کمی سازی ضرایب تبدیل کسینوسی متمایزDCT هر بلوک عناصر T(u, v) مطابق با الگوی زیگزاگ تصویر ۶.۱۲(b) ثبت می‌شوند. از آنجائی که آرایه مرتب شده تک بعدی (ضرایب کمی سازی شده) بر حسب افزایش فرکانس فضایی به شکل کیفی مرتب می‌شود رمزگذار نماد تصویر ۶.۱۱(a) برای بهره‌برداری از اجراهای طولانی صفر طراحی شده است که بعد از ثبت به دست می‌آیند. ضرایب

AC غیرصفر T(u, v) به غیر از u=v=0 با استفاده از کدی با طول متغیر رمزگذاری می‌شوند که مقدار ضریب و تعداد صفرهای قبل را تعریف می‌کند. ضریب کسینوس متمایز به شکلی متفاوت رمزگذاری می‌شود که بستگی به ضریب DC تصویر فرعی قبل دارد. جدول رمزگذاری DC و AC هافمن طبق استانداردها ارائه شده است. ولی کاربر می‌تواند جدولهای سفارشی و آرایه‌های هنجارسازی خودش را ایجاد کند که می‌توانند با خصوصیات تصویر فشرده شده وفق داده شوند.

تشریح اجرای کامل استانداردهای jpeg در این فصل امکان‌پذیر نیست ولی پرونده‌هایM file, m زیر فرایند رمزگذاری مبنای مشابه دارند:

```
function y = im2jpeg(x, quality)
%IM2JPEG Compresses an image using a JPEG approximation.
%     Y = IM2JPEG(X, QUALITY) Compresses image X based on 8 x 8 DCT
%     transforms, coefficient quantization, and Huffman symbol
%     coding. Input QUALITY determines the amount of information that
%     is lost and compression achieved. Y is an encoding structure
%     containing fields:
%
%     Y.size            size of X
%     Y.numblocks Number of 8-by-8 encoded blocks
%     Y.quality         Quality factor (as percent)
%     Y.huffman         Huffman encoding structure, as returned by
%                       MAT2HUFF
%
%     See also JPEG2IM.
error(nargchk(1, 2, nargin));       % Check input arguments
if nargin(x) ~= 2 | ~isreal(x) | ~isnumeric(x) | ~isa(x, 'uint8')
      error('The input must be a UINT8 image.');
end
if nargin < 2
      quality = 1;                        % Default value for quality.
End
m = [ 16    11    10    16    24    40    51    61    % JPEG normalizing array
      12    12    14    19    26    58    60    55    % and zig-zag reordering
      14    13    16    24    40    57    69    56    % Pattern.
      14    17    22    29    51    87    80    62
      18    22    37    56    68    109   103   77
      24    35    55    64    81    104   113   92
      49    64    78    87    103   121   120   101
      72    92    95    98    112   100   103   99] * quality;

order =  [1   9    2    3    10   17   25   18   11   4    5    12   19   26   33   …
          41   34   27   20   13   6    7    14   21   28   35   42   49   57   50   …
          43   36   29   22   15   8    16   23   30   37   44   51   58   59   52   …
          45   38   31   24   32   39   46   53   60   61   54   47   40   48   55   …
          62   63   56   64   ];
  [xm, xn] = size(x);                     % Get input size.
  x = double(x) – 128;                % Level shift input
  t = dctmtx(8);                      % Compute 8 x 8 DCT matrix

  % Compute DCTs of 8x8 blocks and quantize the coefficients.
  y = blkproc(x, [8 8], 'p1 * x * p2', t, t');
  y = blkproc(x, [8 8], 'round(x ./ p1)', m);

  y = im2col(y, [8 8], 'distinct');         % Break 8x8 blocks into columns
```

```
xb = size(y, 2);                    % Get number of blocks
y = y(order, :);                    % Reorder column elements
eob = max(x(:)) + 1;                % Create end-of-block symbol
r = zeros(numel(y) + size(y, 2), 1);
count = 0;
for j = 1:xb                                % Process 1 block (col) at a time
      i = max(find(y(:, j)));               % Find last non-zero element
      if isempty(i)                         % Ni nonzero block values
            i = 0;
      end
      p = count + 1;
      q = p + I;
      r(p:q) = [y(1:i, j); eob];            % Truncate trailing 0's, add EOB,
      count = count + i + 1;        % and add to output vector
end
r((count + 1):end) = [];                    % Delete unused portion of r
%     Y.size            = uint16([xm xn]);
%     Y.numblocks = uint16(xb);
%     Y.quality         = uint16(quality * 100);
%     Y.huffman         = mat2huff(r);
```

تابع im2jpeg مطابق با نمودار تصویر ۱۱.۶.(a)قسمتهای ۸*۸ متمایز و یا قطعه‌های تصویر ورودیx را تک به تک (به جای پردازش یکجای تصویر) پردازش می‌کند. blkproc, im2col ۲ تابع مخصوص پردازش قطعه‌ها هستند که برای ساده سازی محاسبات به کار برده می‌شوند. تابع blkproc که ترکیب استاندارد آن به شرح زیر است:

```
B = blkproc(A, [M N], FUNCTION, P1, P2, …),
```

فرایند کار با تصویرها را سریع و خودکار می‌کند. تصویر ورودی (a)را همراه با اندازه قطعات آماده پردازش دریافت می‌کند سپس از تابعی برای پردازش آنها استفاده می‌شود که پارامترهای ورودی اختیاری p1, p2 و غیره را برای تابع پردازش دارند. سپس تابع a blkproc را به قطعات m*n (همراه با هر نوع لایه‌گذاری صفر) تجزیه می‌کند. با هر بلوک . پارامترهای p1, p2 تابع FUN را فراخوانی می‌کند و نتایج را در تصویر خروجی ب قرار می‌دهد.

دومین تابع مخصوص پردازش بلوک که توسط im2jpeg استفاده می‌شود im2col است. وقتی blkproc برای اجرای عملیات روی قطعات مناسب نباشد، im2col برای مرتب سازی داده‌های ورودی به کار برده می‌شود طوری که بتوان عملیات را به کلی ساده‌تر و کارآمدتر انجام داد. (مثلاً با بردار‌بندی عملیات). داده‌های خروجی im2col ماتریسی هستند که هر یک از ستونهای آن حاوی عناصر یک قطعه متمایز از تصویر ورودی است. قالب استاندارد آن به شرح زیر است:

```
B = im2col(A, [M N], 'distinct')
```

در اینجا پارامترهای A,B,[M,N] قبلاً در تابع blkproc تعریف شدند. رشته distinct به im2col می‌گوید که قطعات پردازش شده همپوشی ندارند. رشته sliding ایجاد یک ستون را در b به ازای هر عنصر تصویری a نشان می‌دهد (گویی یک قطعه در تصویر درج شده است).

تابع blkproc در im2jpeg برای تسهیل محاسبات تبدیل کسینوسی متمایز و کمی سازی و هنجارزدایی ضریب به کار برده می‌شوند در حالی که im2col برای ساده سازی تنظیم مجدد ضرایب کمی شده و کشف اجراهای صفر به کار برده می‌شود. Im2jpeg بر خلاف استاندارد jpeg فقط اجراهای آخر صفر را در هر قطعه ضریب مرتب شده نشان می‌دهد و کل اجرای آن را با نماد eob جایگزین می‌کند. گرچه نرم افزار MATLAB یک تابع کارآمد مبتنی بر FFT برای تبدیل کسینوسی متمایزDCT تصویرهای بزرگ دارد (که در مطالب راهنمای تابع dct2 تشریح شده است) ولی im2jpeg از یک فرمول ماتریسی دیگر استفاده می‌کند.

$$T = HFH^T$$

در اینجا f بلوک ۸*۸ تصویر f(x, y) است و H تبدیل ۸*۸ تبدیل کسینوسی متمایز و ماتریس ایجاد شده توسط dctmtx8. است و T نتیجه تبدیل کسینوسی متمایز f است.

T برای نشان دادن عملیات جابجایی به کار می‌رود. در صورت انجام نشدن کمی سازی معکوس تبدیل کسینوسی متمایز تی به شرح زیر است:

$$F = H^T TH$$

این فرمول مخصوصاً وقتی موثر است که تصویرهای مربع شکل کوچک (مانند تبدیل کسینوسی متمایز ۸*۸) تبدیل می‌شوند. بنا بر این عبارت زیر

```
y = blkproc(x, [8 8], 'P1 * x * P2', h, h')
```

تبدیل کسینوسی متمایز تصویرx را در قطعات ۸*۸ با استفاده از ماتریس تبدیل کسینوسی متمایزh انجام می‌دهد و جابجایی h به صورت پارامترهایp_1 و p_2 ضرب ماتریس تبدیل کسینوسی متمایزDCT هستند.

برای نافشرده کردن تصویر im2jpeg قطعات باید پردازش و بر اساس ماتریس تبدیل شوند. تابع jpeg2im عملیات معکوس‌سازی را (به استثنای کمی سازی) یک به یک انجام می‌دهد.

```
A = col2im(B, [M N], [MM NN],  'distinct')
```

```
function x= jpeg2im(y)
%JPEG2IM Decode an IM2JPEG compressed image.
%     X = JPEG2IM(Y) decodes compresses image Y, generationg
%     reconstructed approximation X. Y is a structure generated by
%     IM2JPEG
%
```

```
   %     See also IM2JPEG.
   error(nargchk(1, 1, nargin));       % Check input arguments

   m = [ 16    11    10    16    24    40    51    61    % JPEG normalizing array
         12    12    14    19    26    58    60    55    % and zig-zag reordering
         14    13    16    24    40    57    69    56    % Pattern.
         14    17    22    29    51    87    80    62
         18    22    37    56    68    109   103   77
         24    35    55    64    81    104   113   92
         49    64    78    87    103   121   120   101
         72    92    95    98    112   100   103   99];

order =  [1   9    2    3    10   17   25   18   11   4    5    12   19   26   33   …
          41   34   27   20   13   6    7    14   21   28   35   42   49   57   50   …
          43   36   29   22   15   8    16   23   30   37   44   51   58   59   52   …
          45   38   31   24   32   39   46   53   60   61   54   47   40   48   55   …
          62   63   56   64   ];
   rev = order;                                % Compute inverse ordering
   for k = 1:length(r=order)
         rev(k) = find(order == k);
   end
   m = double(y.quality) / 100 * m;    % Get encoding quality.
   Xb = double(y.numblocks)            % Get x blocks.
   sz = double(y.size);
   xn = sz(2);                         % Get x columns.
   xm = sz(1);                         % Get x rows.
   x = huff2mat(y.huffman);            % Huffman decode.
   eob = max(x(:));                    % Get end-of-block symbol
   z  = zeros(64, xb);   k = 1;        % Form block columns by copying
   for j = 1:xb                              % successive values from x into
         for i = 1:64                        % columns of z, while changing
               if  x(k) == eob         % to the next column whenever
                     k = k + 1: break;       % an EOB symbol is found.
               else
                     z(i, j) = x(k);
                     k = k + 1;
               end
         end
   end

   z = z(rev, :);                                        % Restore order
   x = col2im(z, [8 8], [xm xn], 'distinct');            % Form matrix blocks
   x = blkproc(x , [8 8], 'x .* P1', m);                 % Denormalize DCT
   t = dctmtx(8);                                  % Get 8 x 8 DCT matrix
   x = blkproc(x , [8 8], 'P1 * x * p2', t', t);         % Compute block DCT-1
   x = uint8(x + 128);                                   % Level shift
```

برای باز یک تصویر ۲ بعدی از ستونهای ماتریس z که هر یک از ستونهای ۶۴ عنصری یک قطعه ۸*۸ تصویر بازسازی شده است از تابعهای عمومی استفاده می‌کند. پارامترهای A,B,[M,N], distinct قبلاً برای تابع im2col تعریف شده‌اند در حالی که آرایه mm, mn, ابعاد تصویر خروجی a را نشان می‌دهد.

مثال ۶.۸:

در تصاویر ۶.۱۳(a) و (b), ۲ تصویر رمزگذاری شده و رمزگشایی شده با jpeg و مقادیر تقریبی تصویر تک رنگ ۶.۴(a) دیده می‌شوند. نسبت فشرده‌سازی نتیجه اول ۱۸ به ۱ است با اجرای مستقیم آرایه هنجارسازی در تصویر ۶.۱۲(a)به دست آمد.

دومین مورد که تصویر اصلی را به نسبت ۴۲ به ۱ فشرده‌سازی کرده است با ضرب کردن آرایه هنجارسازی در ۴ ایجاد شد.

اختلاف بین تصویر اصلی عکس ۶.۴(a) و تصویر بازسازی شده عکس ۶.۱۳(a) و(b) در تصویرهای ۶.۱۳(c) و(d) به ترتیب نشان داده شده است. هر دو تصویر برای نمایش بهتر خطاها مقیاس بندی شده‌اند.

خطاهای rms مطابق با آنها به ترتیب ۲.۵ و ۴.۴ در سطوح خاکستری هستند.

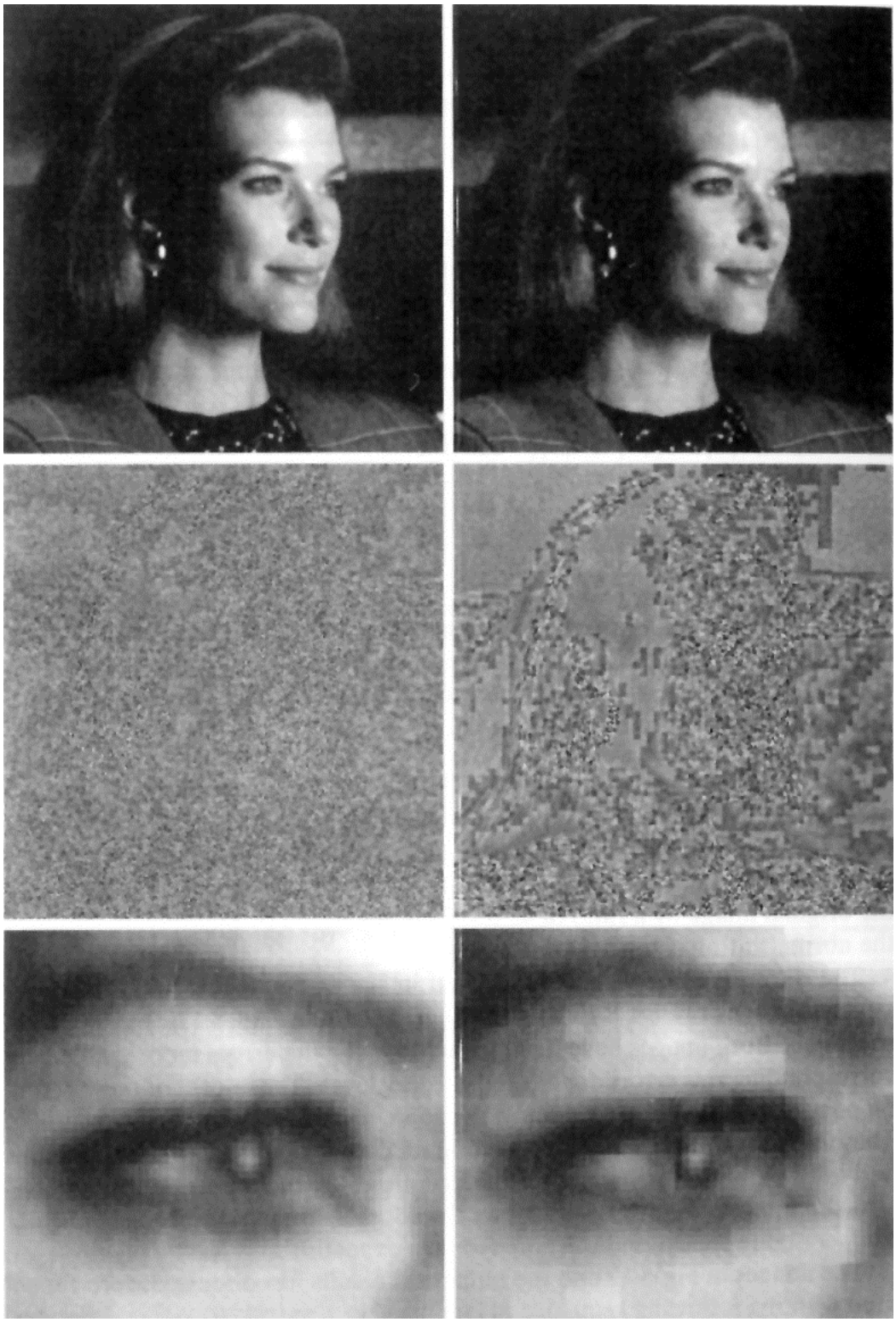
تاثیر این خطاها بر کیفیت تصویر در تصویرهای بزرگنمایی شده ۶.۱۳(e) و (f) به خوبی مشهود است.

این تصویرها به ترتیب قسمتهای بزرگ شده عکسهای ۶.۱۳(a) و (b) هستند و بهتر می‌توان اختلافهای ظریف بین تصویرهای بازسازی شده را ارزیابی کرد. تصویر اصلی بزرگ شده در تصویر ۶.۴(b) دیده می‌شود.

به ابزار قطعه‌بندی که در تصویرهای بزرگ شده دیده می‌شود توجه کنید.

تصویرهای عکس ۶.۱۳ و نتایج فوق با فرمانهای زیر به دست آمده است:

```
>> f = imread(Tracy.tif');
>> c1 = im2jpeg(f);
>> f1 = jpeg2im(c1);
>> imratio(f, c1)
ans =
      18.2450
>> compare(f, f1, 3)
ans =
      2.4675
>> c4 = im2jpeg(f, 4);
>> f4 = jpeg2im(c4);
>> imratio(f, c4)
ans =
      41.7826
>> compare(f, f4, 3)
ans =
      4.4184
```



تصویر ۶.۱۳: ستون چپ ساخت تصویر تقریبی عکس ۶.۴ با استفاده از تبدیل کسینوسی متمایز و آرایه هنجارسازی تصویر سمت راست:

کسب همان نتایج با استفاده از آرایه هنجارسازی که به مقیاس ۴ درآمده است.

این نتایج با نتایج به دست آمده در محیط رمزگذاری مبنای jpeg تفاوت دارد زیرا im2jpeg محصولی ایجاد می‌کند که شبیه به استاندارد jpeg با رمزگذاری هافمن است. ۲ تفاوت عمده وجود دارد:

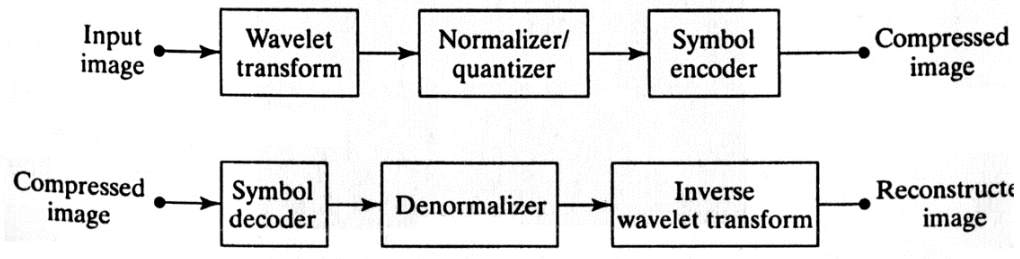
(۱) کلیه دفعات اجرای صفرهای ضریب در حالت استاندارد دارای رمزگذاری هافمن هستند در حالی که im2jpeg فقط اجرای آخر هر قطعه را رمزگذاری می‌کند.

(۲) رمزگذاری و رمزگشایی حالت استاندارد بر اساس رمز پیش فرض هافمن است در حالی که im2jpeg اطلاعات مورد نیاز برای بازسازی تصویر با کد هافمن را در مبنای تصویر قرار می‌دهد. نسبت فشرده‌سازی با استفاده از این استاندارد ۲ برابر می‌شود.

۶.۵.۲. بررسی پسوند jpeg2000

پسوند فوق همچون پسوند jpeg معمولی بر اساس این استدلال است که ضرایب تبدیل که عناصر تصویری را ارتباط‌زدایی می‌کنند موثرتر از عناصر اصلی قابل رمزگذاری هستند. اگر تابعهای مبنای تبدیل که امواج کوچکی در jpeg2000 هستند اکثر اطلاعات تصویری مهم را به تعداد محدودی از ضرایب فشرده کنند مابقی ضرایب با دانه‌های درشت کمی سازی می‌شوند و با اندکی تحریف تصویر به صفر بریده می‌شوند.

یک سیستم رمزگذاری ساده شده jpeg2000 در تصویر ۶.۱۴ نشان داده شده است. اولین مرحله رمزگذاری انتقال عناصر تصویری با کم کردن ۲^{m-1}است که در اینجا 2^m تعداد سطوح خاکستری(gray_level) تصویر است. تبدیل متمایز تک بعدی ردیفها و ستونهای تصویر قابل محاسبه است.



تبدیل استفاده شده در فشرده‌سازی بدون خطا دومتعامدی است. که مقیاس بندی ضرایب ۵-۳ و برداری از امواج کوچک دارد. در برنامه‌هایی که اتلاف اطلاعات داریم از برداری با مقیاس بندی ضریب ۹-۷ استفاده می‌شود (به تابع wavefilter فصل ۷ مراجعه کنید). به

هر حال تجزیه اولیه منجر به ایجاد ۴ مقوله فرعی می‌شود: شبیه سازی تصویر با تفکیک پذیری پایین، و خصوصیات فرکانس افقی، عمودی و اریب تصویر.

اگر فرایند تجزیه N_L دفعه تکرار شود، و تکرارهای بعدی محدود به ضریبهای تجزیه‌گر قبل باشد یک تبدیل با مقیاس N_L تولید می‌شود. مقیاسهای مجاور با توان ۲ مرتبط می‌شوند و پایینترین مقیاس نوع تعریف شده تصویر اصلی را دارد. همان طور که در تصویر ۶.۱۵ حدس زده می‌شود، نشانه‌گذاری استاندارد در مورد $N_L = 2$ خلاصه شده است و یک مقیاس تبدیل عمومی N_L و$N_L + 1$ 3 واحد فرعی دارد که ضرایب آن با a_b نشان داده شده است. در این استاندارد تعداد مقیاس‌های محاسبه شده مشخص نمی‌شود.

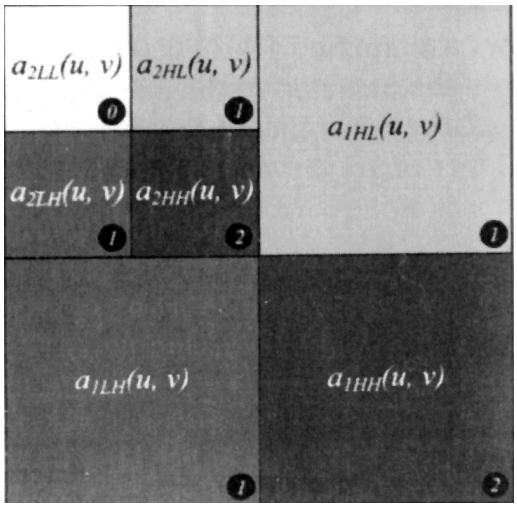
پس از محاسبه تبدیل موجی مقیاس N_L کل تعداد ضرایب تبدیل معادل تعداد نمونه‌های تصویر اصلی است ولی اطلاعات تصویری اصلی در چند ضریب متمرکز شده است. برای کاهش تعداد بیتهای مورد نیاز برای نمایش آنها ضریب $a_b(u, v)$ باند فرعیb به مقدار $q_b(u, v)$ با استفاده از رابطه زیر کمی سازی می‌شود:

؟

در اینجا عملگرهای "sign" و "floor" همچون تابعهای هم اسم نرم افزار MATLAB هستند. در این مرحله کمی سازی ،اندازه Δ_b برابر است با:

؟

R_b حیطه اسمی پویای باند فرعی b است وμ_b , ε_b تعداد بیتهای تخصیص داده شده به نما و جزء اعشاری لگاریتم ضرایب باند فرعی هستند. حیطه پویای باند فرعیb مجموع بیتهای مورد نیاز برای نمایش تصویر اصلی و تحلیل باند فرعی b است.



تحلیل بیتهای باند فرعی مطابق با الگوی ساده تصویر ۶.۱۵ است. مثلاً در باند فرعی b=1HH دو تحلیل بیتی وجود دارد.

برای فشرده‌سازی بدون خطا $\mu_b = 0$ و$R_b = \varepsilon_b$ که در نتیجه $\Delta_b = 1$ است. در فشرده‌سازی‌های برگشت‌ناپذیر هیچ مرحله کمی سازی خاصی مشخص نمی‌شود. در عوض تعداد بیتهای جزء اعشاری لگاریتم و نما بر اساس باند فرعی به نام کمی سازی علنی در اختیار رمزگشا قرار داده می‌شود و یا فقط بر اساس باند فرعی $N_L LL$ با عنوان کمی سازی نهفته (implicit quantization)انجام می‌شود. در مورد دوم، مابقی باندهای فرعی با استفاده از پارامترهای باند فرعی $N_L LL$ کمی سازی می‌شوند. فرض کنید ε . μ , تعداد بیتهای تخصیص داده شده به باند فرعی $N_L LL$ هستند. پارامترهای درون یابی شده باند فرعی ب به شرح زیر است:

؟

در اینجا nsd_b تعداد سطوح تجزیه باند فرعی از تصویر اصلی نسبت به باند فرعی b است.

در مرحله نهایی رمزگذاری ضرایب کمی سازی شده بر اساس اصول ریاضی رمزگذاری می‌شوند. گرچه رمزگذاری ریاضی (aritmetic coding) در این فصل بررسی نشده است ولی یک روش رمزگذاری با طول متغیر است که همچون رمزگذاری هافمن برای کاهش افزونگی رمزگذاری طراحی شده است.

تابع سفارشی im2jpeg2k فرایندی شبیه به رمزگذاری jpeg2000 در تصویر ۶.۱۴(a) دارد. تنها استثنا رمزگذاری نمادین ریاضی است. همان طور که در فهرست زیر دیده می‌شود رمزگذاری هافمن که با رمزگذاری صفر تقویت شده جایگزین حالت ساده شده است.

```
function y = im2jpeg2k(x, n, q)
%IM2JPEG Compresses an image using a 2000 approximation.
%     Y = IM2JPEG2K(X, N, Q) Compresses image X using an N-scale JPEG
%     2k wavelet transform, implicit or explicit coefficient
%     quantization, and Huffman symbol coding augmented by zero
%     run-length coding. If quantization vector Q contains two
%     elements, they are assumed to be implicit quantization
%     parameters; else, it is assumed to contain explicit subband step
%     size. Y is an encoding structure containing Huffman-encoded
%     data and additional parameters needed by JPEG2K2IM for decoding.
%
%     See also JPEG2K2IM.
global RUNS
error(nargchk(3, 3, nargin));       % Check input arguments
if nargin(x) ~= 2 | ~isreal(x) | ~isnumeric(x) | ~isa(x, 'uint8')
      error('The input must be a UINT8 image.');
end
if length(q) ~= 2 & length(q) ~= 3 * n + 1
      error('The quantization step size vector is bad.');
end
% Level shift the input and compute its wavelet transform.
x = double(x) – 128;
[c, s] = wavefast(x, n, 'jpeg.7');
% Quantize the wavelet coefficients.
q = stepsize(n, q);
sgn = sign(c);    sgn(find(sgn == 0) = 1;       c = abs(c);
```

```
for k = 1:n
      qi = 3* k – 2;
      c = wavepaste('h', c, s, k, wavecopy('h', c, s, k) / q(qi));
      c = wavepaste('v', c, s, k, wavecopy('v', c, s, k) / q(qi+1));
      c = wavepaste('d', c, s, k, wavecopy('d', c, s, k) / q(qi+2));
end
c = wavepaste('a', c, s, k, wavecopy('a', c, s, k) / q(qi+3));
c = floor(c);

% Run-length code zero runs of more 10. Begin by creating
% a special code for 0 runs ('zrc') and end-of-code ('eoc') and
% making a run-length table.
Zrc = min(c(:)) – 1;    eoc = zrc – 1;    RUNS = [65535]

% Find the run transition points: 'plus' contains the index of the
% start of a zero run; the corresponding 'minus' is its end + 1.
z = c == 0;             z = z – [0 z(1:end – 1)];
plus = find(z == 1);    minus = find(z == -1);

% Remove any terminating zero run form 'c'.
if length(plus) ~= length(minus)
      c(plus(end):end) = [];  c = [c eoc];
end

% Remove all other zero runs (based on 'plus' and 'minus') from 'c'.
for i = length(minus):-1:1
      run = minus(i) – plus(i);
      if run > 0
            ovrflo = floor(run / 65535);  run = run – ovrflo * 65535;
            c = [c(1:plus(i) – 1) repmat([zrc 1], 1, ovrflo) zrc …
                  runcode(run) c(minus(i):end)];
      end
end

% Huffman encode and add misc. information for decoding.
y.runs                  = uint16(RUNS);
y.s               = uint16(s(:));
y.zrc             = uint16(-zrc);
y.q               = uint16(100 * q');
y.n               = uint16(n);
y.huffman         = mat2huff(c);

%-------------------------------------------------------------------------------
----%
function y = runcode(x)
% Find a zero run in the run-length table. If not found, create a
% new entry in the table. Return the index of the run.
global RUNS
y = find(RUNS == x);
if length(y) ~= 1
      RUNS = [RUNS; x];
      y = length(RUNS);
end

%-------------------------------------------------------------------------------
----%
function q = stepsize(n, p)
% Create a subband quantization array of step ordered by
% decomposition (first to last) and subband (horizontal, vertical),
```

```
% diagonal, and for final decomposition the approximation subband).
if length(p) == 2                    % Implicit Quantization
      q = [ ];
      qn = 2 ^ (8 – p(2) + n) * (1 + p(1) / 2 ^ 11);
      for k = 1:n
            qk = 2 ^ - k * qn;
            q = [q (2 * qk) (2 * qk) (4 * qk)];
      end
      q = [q qk];
else                              % Explicit Quantization
      q = p;
end

q = round(q * 100) / 100;     % Round to 1/100th place
if any(100 * q > 65535)
      error('The quantizing steps are not UINT16 represent able.');
end
if any(q == 0)
      error(A quantizing steps of 0 is not allowed.');
end
```

رمزگشاهای jpeg2000 صرفاً روند عملیات فوق را معکوس می‌کنند. پس از رمزگشایی این ضرایب ریاضی باندهای فرعی تصویر اصلی منتخب کاربر بازسازی می‌شوند. گرچه این رمزگذار بیتهای M_b را برای باندهای فرعی خاصی رمزگذاری می‌کند ولی کاربر ممکن است با توجه به جریانهای تعبیه شده فقط تصمیم به رمزگشایی بیتهای N_b خاصی بگیرد. برای این کار ضرایب با استفاده از مرحله $2^{M_b}-\Delta_b$. N_bکمی سازی می‌شوند. بیتهای رمزگشایی نشده صفر می‌شوند و ضرایب حاصل از آن با استفاده از رابطه زیر هنجارزدایی می‌شوند.

$$R_{qb}(u,v)=\begin{cases}(\bar{q}_b(u,v)+2^{M_b-N_b(u,v)}).\Delta_b & \bar{q}_b(u,v)>0\\ (\bar{q}_b(u,v)-2^{M_b-N_b(u,v)}).\Delta_b & \bar{q}_b(u,v)<0\\ 0 & \bar{q}_b(u,v)=0\end{cases}$$

در اینجا $R_{qb}(u, v)$ یک ضریب تبدیل هنجارزدایی شده است و $N_b(u, v)$ تعداد بیتهای رمزگشایی شده $q_b(u, v)$ است. سپس ضرایب هنجارزدایی شده به صورت معکوس تبدیل می‌شوند و جابجا میشوند تا محصولی شبیه به تصویر اصلی حاصل شود. تابع سفارشی jpeg2k2im شبیه این روند را ایجاد می‌کند و فشرده‌سازی im2jpeg2k را معکوس می‌کند.

```
function x = jpeg2k2im(y)
%JPEG2K2IM Decoded an IM2JPEG2K compressed image.
%     X = JPEG2K2IM(Y) decoded compresses image Y, reconstructing an
%     structure returned by IM2JPEG2K.
%
%     See also IM2JPEG2K.
error(nargchk(1, 1, nargin));       % Check input arguments
% Get decoding parameters: scale, quantization vector, run-length
% table size, zero run code, end-of-data code, wavelet bookkeeping
% array, and run-length table.
n = double(y.n);
q = double(y.q) / 100;
runs = double(y.runs);
rlen = length(runs);
```

```
zrc = -double(y.zrc);
eoc = zrc – 1;
s = reshape(s, n + 2, 2);

% Compute the size of the wavelet transform.
Cl = prod(1, :)):
for i = 2:n + 1
      cl = cl + 3 * prod(s(i, :));
end

% Perform Huffman decoding followed by zero run decoding.
r = huff2mat(y.huffamn);
c = [];           zi = find(r == zrc);          i = 1;
for j = 1:length(zi)
      c = [c r(i:zi(j) – 1) zeros(1, runs(r(zi(j) + 1)))];
end
zi = find(r == eoc);                        % Undo terminating zero run
if length(zi) == 1                          % or last non-zero run.
      c = [c r(i:zi – 1)];
      c = [c zeros(1, c1 – length))];
else
      c = [c r(i:end)];
end

%Denormalize the coefficients.
c = c + (c > 0) – (c < 0);
for k = 1:n
      qi = 3 * k – 2;
      c = wavepaste('h', c, s, k, wavecopy('h', c, s, k) / q(qi));
      c = wavepaste('v', c, s, k, wavecopy('v', c, s, k) / q(qi+1));
      c = wavepaste('d', c, s, k, wavecopy('d', c, s, k) / q(qi+2));
end
c = wavepaste('a', c, s, k, wavecopy('a', c, s, k) / q(qi+3));

% Compute the inverse wavelet transform and level shift.
x = waveback(c, s, 'jpeg9.7', n);
x = uint8(x + 128);
```

اختلاف اصلی بین سیستم مبتنی بر امواج کوچک در تصویر ۶.۱۴ و سیستم jpeg مبتنی بر تبدیل کسینوسی متمایز در تصویر ۶.۱۱ حذف مراحل پردازش تصویر فرعی است. از آنجائی که تبدیلهای امواج کوچک از نظر محاسباتی موثر و ذاتاً موضعی هستند (یعنی عملکرد اساسی آنها مدت محدود دارد) نیازی به تقسیم تصویر به قطعات کوچکتر نیست. همان طور که در مثال زیر می‌بینیم، حذف مرحله تقسیم باعث حذف ابزار قطعه بندی می‌شود که در نسبتهای فشرده‌سازی بالا خصوصیاتی شبیه به تبدیل کسینوسی متمایز ایجادDCT می‌کند.

دو نوع شبیه سازی خصوصیات تصویر تک رنگ ۶.۴(a) در تصویر ۶.۱۶ دیده می‌شود. تصویر ۶.۱۶(a) از رمزگذاری بازسازی شد که تصویر اصلی را به نسبت ۴۲ به ۱ فشرده کرده بود. تصویر ۶.۱۶(b) از رمزگذاری ۸۸:۱ ایجاد شد. این دو نتیجه با استفاده از تبدیل ۵ مقیاسی و کمی سازی نهفته به ترتیب 8=. μ

8.۵=. μ به دست آمد. از آنجائی که im2jpeg2k فقط رمزگذاری شبیه به jpeg2000ایجاد می‌کند، نسبتهای فشرده‌سازی قید شده فقط با نسبتهای رمزگذاری واقعی jpeg2000 تفاوت دارند. در واقع نسبتهای واقعی ۲ برابر افزایش می‌یابد.

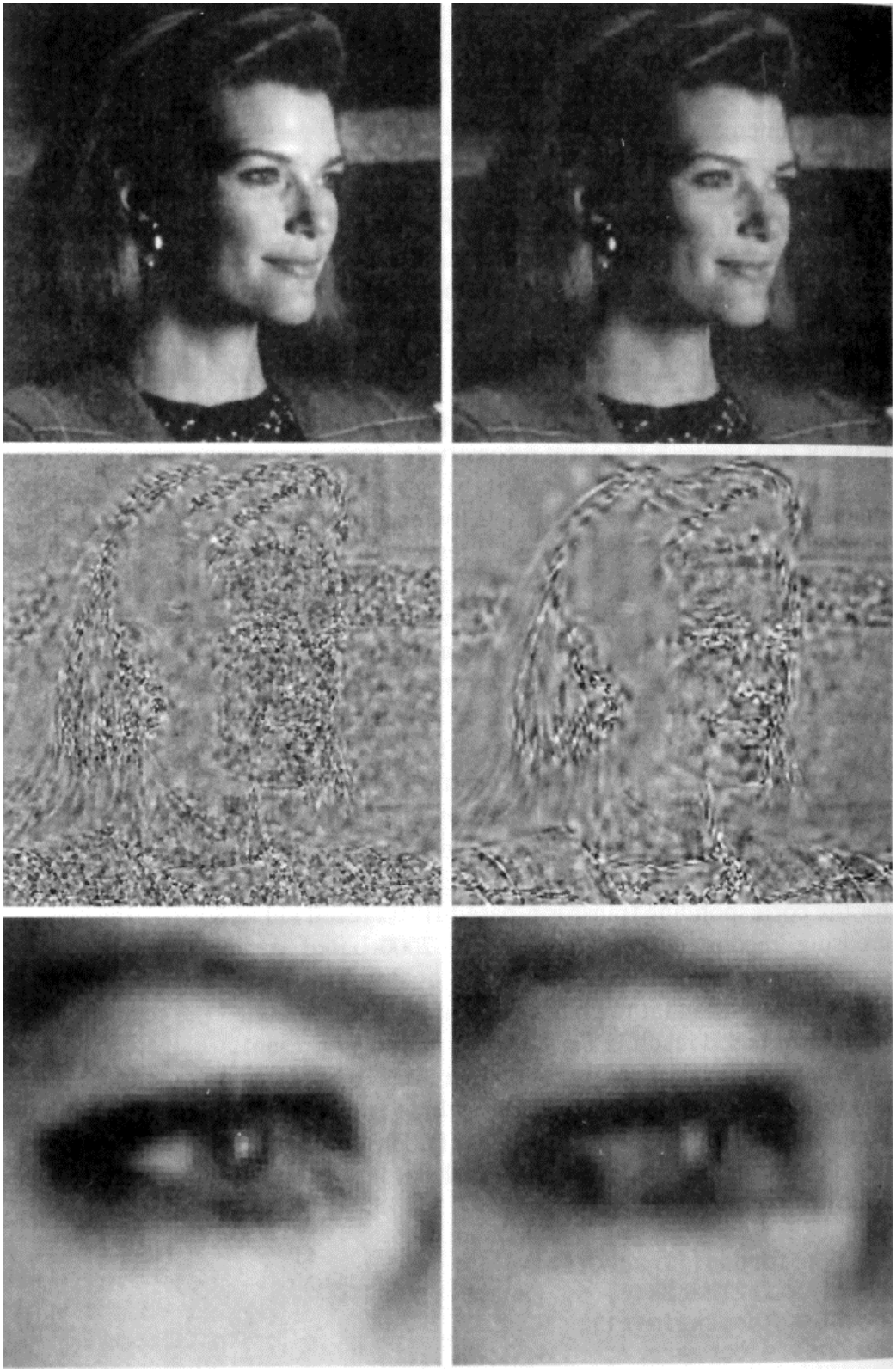
از آنجائی که فشرده‌سازی ۴۲:۱ نتایج در ستون سمت چپ تصویر ۶.۱۶ همانند فشرده‌سازی تحقق یافته در ستون سمت راست تصویر ۶.۱۳ است (مثال ۶.۸) تصویرهای ۶.۱۶.e، c از نظر کیفی و کمی با نتایج تبدیل تصویرهای ۶.۱۳ b، d و f قابل قیاس هستند. بعد از مقایسه این تصاویر کاهش چشمگیر خطا در تصویرهای jpeg2000 دیده می‌شود. در واقع خطای rms نتایج مبتنی بر jpeg2000 در تصویر ۶.۱۶.(a) در سطوح خاکستری ۳.۷ است که در تقابل با مقدار ۴.۴ در نتایج تصویر ۶.۱۳(b) است.

به غیر از کاهش خطای بازسازی رمزگذاری jpeg2000 کیفیت تصویر را به شکلی خاص بهبود بخشیده است. این موضوع در تصویر ۶.۱۶.(e) مشهود است. توجه داشته باشید که ابزار قطعه بندی که در نتایج تبدیل تصویر۶.۱۳.(f) دیده می‌شد دیگر وجود ندارد.

وقتی سطح فشرده‌سازی همچون تصویر ۶.۱۶.(b) به ۸۸:۱ می‌رسد، بافت لباس زن و تاری چشمهایش کاهش می‌یابد.

هر ۲ تاثیر در تصویرهای ۶.۱۶.(b) و(f) دیده می‌شود. خطای rms این بازسازیها در سطوح خاکستری ۵.۹ است. نتایج تصویر ۶.۱۶ با فرمانهای زیر بازسازی شدند.

```
>> f = imread('Tracy.tif');
>> c1 = im2jpegek(f, 5, [8 8.5]);
>> f1 = jpeg2k2im(c1);
>> rms1 =  compare(f, f1)
rms1 =
      3.6931
>> cr1 = imratio(f, c1


cr1 =
      42.1589
>> c2 = jm2jpeg2k(f, f, [8 7]);
>> f2 = jpeg2k2im(c2);
>> rms2 = compare(f, f2)
rms2 =
      5.9172
>> cr2 = imratio(f, c2)
cr2 =
      87.7323
```



تصویر ۶.۱۶. ستون چپ: شبیه سازی تصویر ۶.۴ با استفاده از ۵ مقیاس و کمی سازی نهفته با

ستون راست: کسب نتایج مشابه با e0=7

وقتی یک بردار دو عنصری(two_element) به صورت شناسه ۳ (agrument3)از im2jpeg2k ارائه می‌شود کمی سازی نهفته انجام می‌شود. اگر طول این بردار ۲ نباشد، کمی سازی نهفته در تابع فرض می‌شود و اندازه مراحل $3\ N_L + 1$ باید مشخص شود (در اینجا N_L تعداد مقیاسهای محاسبه شده است). این مقدار برای هر یک از باندهای فرعی تجزیه ۱ است. آنها باید بر اساس سطح تجزیه و نوع باند فرعی (به صورت افقی، عمودی یا اری(b)مرتب شوند. مثلاً رابطه زیر

```
>> c3 = im2jpeg2k(f, 1, [1 1 1 1]);
```

یک تبدیل تک مقیاسی را محاسبه کرده و از کمی سازی علنی بهره می‌گیرد. کلیه چهار باند فرعی با استفاده از مرحله $\Delta_1=1$ کمی سازی می‌شوند. یعنی ضرایب تبدیل به نزدیکترین عدد صحیح گرد می‌شوند. این حداقل خطا را برای اجرای im2jpeg2k ایجاد می‌کند و خطای rms و نسبت فشرده‌سازی حاصل از آن به شرح زیر است:

```
>> f3 = jpeg2k2im(c3);
>> rms3 = compare(f, f3)
rms3 =
      1.1234
>> cr3 = imratio(f, c3)
cr3 =
      1.635
```

خلاصه مطالب (Summary)

مطالب این بخش اصول فشرده‌سازی تصویرهای دیجیتالی از طریق رمزگشایی و حذف افزونگی روانی – تصویری و بین پیکسلی است. روال‌هایی برای حذف افزونگی در نرم افزار MATLAB و توسعه ابزار پردازش تصویر ابداع شده است. معیارهای استاندارد فشرده‌سازی تصویرهای Jpeg و Jpeg2000 نیز ارائه شده است. برای کسب اطلاعات بیشتر در خصوص حذف افزونگی تصویرها و شگردهای تشریح نشده در اینجا و استانداردهای مربوط به مجموعه تصویرهای مخصوص (مانند تصویرهای دودویی) کنید.