Code Vectorization

By : Xiao zhang
Don’t waste life on LOOP!

Perform pixel-level operation on image.

The loop in matlab or python code is less efficient than C code.

Use the built-in function to replace loop.
MATLAB

img = zeros(4000,4000);

for i = 1:size(img,1)
    for j = 1:size(img,2)
        img(i,j) = img(i,j) + 1;
    end
end

Run Time: 0.57s

img = img + 1;

Run Time: 0.06s

PYTHON

Img = np.zeros((4000,4000))

for i in range(img.shape[0]):
    for j in range(img.shape[1]):
        Img[i,j] += 1

Run Time: 6.47s

Img = Img + 1

Time: 0.029s

PYTHON for loop with range is incredibly slow !!! But numpy built in function is efficient
Img = np.zeros((4000,4000))

Img = Img + 1

Scalar value 1 broadcasting to matrix(4000,4000)
Broadcasting
In some application, e.g. deep learning, it preferred data with mean value 0. So we need to preprocess the image by extracting the mean before putting into neural network. The mean value just 1x3 vector for RGB channels.

Suppose you have 1000 image and each image has equal size 500*300*3
And you want to extract mean value from all the image

```matlab
size(img)
1000, 500 ,300 ,3
N      H     W    C

size(mean_val)
3
C

For n = 1:N
    For h = 1:H
        For w = 1:W
            For c = 1:C
                img(n,h,w,c) = img(n,h,w,c) - mean_val(c)
            End
        End
    End
end
```
In some application, e.g. deep learning, it preferred data with mean value 0. So we need to preprocess the image by extracting the mean before putting into neural network. The mean value just 1x3 vector for RGB channels.

Suppose you have 1000 image and each image has equal size 500*300*3
And you want to extract mean value from all the image

\[
\text{size(img)} \\
1000, 500, 300, 3 \\
\text{Reshaped\_mean\_val = reshape(c, 1,1,1,3);} \\
\text{Img = img - Reshaped\_mean\_val} \\
\text{size(Reshaped\_mean\_val)} \\
1, 1, 1, 3
\]
size(img) = (1000, 500, 300, 3)
N H W C

size(mean_val) = (3, 3)
C

img = img - reshape(c, 1, 1, 1, 3);

Broadcasting will check each dimension:
For each dimension:
  If have same size in this dimension:
    Do nothing
  else:
    If one of them has size 1:
      Repeat element to match the other size
    Else:
      Raise error
MATLAB
Suppose we have 4-D variable img for all image data
%size(img) = 1000,500,500,3
%First we compute mean value for each channel.
R_img = img(:,:,1);
G_img = img(:,:,2);
B_img = img(:,:,3);
Img_mean = [mean(R_img(:)),mean(G_img(:)),mean(B_img(:))];
%size(Img_mean) = 1,3
%reshape img for broadcasting
Img_mean = reshape(Img_mean,1,1,1,3);
%size(Img_mean) = 1,1,1,3
Img_extracted_mean = img - Img_mean;

PYTHON
#img.shape  1000,500,300,3
#compute mean for all channel
R_mean = np.mean(img[:,:,:,0])
G_mean = np.mean(img[:,:,:,1])
B_mean = np.mean(img[:,:,:,2])
Mean_val = np.array([R_mean,G_mean,B_mean])

reshaped_mean_val = np.reshape(Mean_val,[1,1,1,3])

Img_extracted_mean = img - new_mean_val
Implicit broadcasting by repeating elements

\[
\begin{array}{cc}
1 & 2 \\
3 & 4 \\
\end{array}
\] \quad + \quad \begin{array}{c}
1 \\
2 \\
\end{array} \quad = \quad ?
Implicit broadcasting by repeating elements

\[
\begin{array}{c}
\begin{array}{cc}
1 & 2 \\
3 & 4 \\
\end{array} \\
\begin{array}{cc}
1 & 2 \\
3 & 4 \\
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{c}
1 \\
2 \\
\end{array} \\
\begin{array}{c}
1 \\
2 \\
\end{array}
\end{array}
\]

Repeat column

\[
\begin{array}{c}
\begin{array}{cc}
1 & 1 \\
2 & 2 \\
\end{array}
\end{array}
\]

\[
\begin{array}{c}
\begin{array}{cc}
1 & 2 \\
3 & 4 \\
\end{array}
\end{array}
\]

Implicit broadcasting by repeating elements

\[
\begin{bmatrix}
1 \\
2
\end{bmatrix} + \begin{bmatrix}
2 & 3 & 4
\end{bmatrix} = ?
\]
Implicit broadcasting by repeating elements

\[
\begin{align*}
\begin{bmatrix}
1 \\
2
\end{bmatrix}
& \quad + \\
\begin{bmatrix}
2\; & 3\; & 4
\end{bmatrix}
& \quad = \\
\begin{bmatrix}
2\; & 3\; & 4
\end{bmatrix}
& \quad + \\
\begin{bmatrix}
2\; & 2\; & 2 \\
2\; & 2\; & 2
\end{bmatrix}
& \quad = \\
\begin{bmatrix}
3\; & 4\; & 5 \\
4\; & 5\; & 6
\end{bmatrix}
\end{align*}
\]
Matlab and python will try expand dim if necessary.

A has size (4,2,3)
B has size(4,1)  compute A-B
C has size(1,3)

In **Matlab**, the dimension checking will be conducted from **left to right**.

```
<table>
<thead>
<tr>
<th>A</th>
<th>4</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>4</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
```

- 4 vs 4: Do nothing (Check pass)
- 2 vs 1: Broadcasting (Check pass)
- 3 vs None: Expand dim broadcasting (Check pass)

Implicit Expand dims

- B 4 1
- B 4 1
- B 4 2
- B 4 1

Broadcasting

- B 4 1
- B 4 1
- B 4 3
Matlab and python will try expand dim if necessary.

A has size (4,2,3)
B has size(4,1)                      compute A-C
C has size(1,3)

In **Matlab**, the dimension checking will be conducted from **left to right**.

So that A - C is not valid

Checking direction

<table>
<thead>
<tr>
<th></th>
<th>4</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

4 vs 1
Broadcasting
Check pass

2 vs 3
Raise error
Check failed
Matlab and python will try expand dim if necessary.

A has size (4,2,3)
B has size(4,1)  compute A-C
C has size(1,3)

In **PYTHON**, the dimension checking will be conducted from Right to left. So that A - C is valid

```
A      4                               2                                         3
C                                        1                                        3
C     1                                 1                                        3
C      4                                 2                                       3
```

4 vs None  Expand dims  broadcasting  Check pass
2 vs 1  broadcasting  Check pass
3 vs 3  Do nothing  Check pass

Implicit  Expand dims

```
C  1
C  4
```

broadcasting
Matlab and python will try expand dim if necessary.

A has size (4,2,3)
B has size(4,1)
C has size(1,3)

In **PYTHON**, the dimension checking will be conducted from **Right to left**.
And that A - B is not valid

A      4                                  2                                    3
      B                                           4                                      1

Checking direction

2 vs 4  Raise error  3 vs 1  broadcasting
Check failed  Check pass
Broadcasting could be dangerous

Suppose you have a 3x3 matrix $A$ and a 3 element vector $B$. You want to subtract each row of $A$ by the corresponding element in $B$.

Broadcasting $B$ to 3x3 by repeating each element in row direction
MATLAB code:

A = [1,2,3;4,5,6;7,8,9];
B = [1,2,3];

A - B

\[
\begin{array}{ccc}
0 & 0 & 0 \\
3 & 3 & 3 \\
6 & 6 & 6 \\
\end{array}
\]

\[
\begin{array}{ccc}
0 & 1 & 2 \\
2 & 3 & 4 \\
4 & 5 & 6 \\
\end{array}
\]
Checking direction

A 3 3

B 1 3

3 vs 1 broadcasting
Check pass

B = [1, 2, 3];
size(B)
1, 3

Broadcasted B
3x3

A 3x3

WRONG!

In matlab.

3 vs 3
Do nothing
Check pass

 checked direction
Python code

```python
A = (np.arange(9)+1).reshape(3,3)
A
array([[1, 2, 3],
       [4, 5, 6],
       [7, 8, 9]])

B = np.array([1,2,3])
C = A-B
C
array([[0, 0, 0],
       [3, 3, 3],
       [6, 6, 6]])

B.shape
(3,)
```

```
In Python, single dimension vectors have only 1 dimension by default. B = np.array([1,2,3])
B.shape (3,)
But the unexpected broadcasting will not pop out an error. To avoid confusion and control the broadcasting direction. We recommend:

**Always check the shape before using broadcasting**

Explicitly reshaping the array or matrix to the desired shape and then using implicit broadcasting.

<table>
<thead>
<tr>
<th>MATLAB</th>
<th>PYTHON</th>
</tr>
</thead>
<tbody>
<tr>
<td>A - reshape(B,3,1)</td>
<td>A - np.reshape(B,[3,1])</td>
</tr>
</tbody>
</table>

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2</td>
<td>0 1 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2 3 4</td>
<td>2 3 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4 5 6</td>
<td>4 5 6</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Matlab support scalar broadcasting in the very early version but it just introduced the implicit broadcasting feature since R2016b.

```
Img = zeros(500,300,3);
V = zeros(1,1,3);
```

So in matlab R2016a or early version, you can’t do:
```
Img + V
Error using +
Matrix dimensions must agree.
```

But it’s now valid for R2016b and its later version.
For early version or explicit broadcasting,
Use built-in function `bsxfun` or `repmat` instead
Example:
Compute gaussian distribution
Compute gaussian distribution
For the range that $x = [0,1]$ and $y = [0,1]$

$$f(x, y) = \frac{1}{2\pi\sigma} \exp\left(-\frac{(x - \mu_x)^2 + (y - \mu_y)^2}{2\sigma^2}\right)$$

$$\mu_x = 0.5$$
$$\mu_y = 0.5$$
$$\sigma = 0.2$$
Meshgrid

X_line = [1,2,3];
Y_line = [4,5,6];
[mesh_x, mesh_y] = meshgrid(x_line,y_line)

(x_line is the line in width and y is the line in height)

\[ f(x, y) = \frac{1}{2\sigma \pi} \exp\left(-\frac{(x - \mu_x)^2 + (y - \mu_y)^2}{2\sigma^2}\right) \]
\[
f(x, y) = \frac{1}{2\sigma \pi} \exp\left(-\frac{(x - \mu_x)^2 + (y - \mu_y)^2}{2\sigma^2}\right)
\]

**MATLAB**

```matlab
X_line = 0:0.001:1;
Y_line = 0:0.001:1;
[mesh_x, mesh_y] = meshgrid(X_line,Y_line);
Constant_term = 1/(0.2*pi);
val = exp(-((mesh_x-0.5).^2+(mesh_y-0.5).^2)/(2*0.2^2));
val = Constant_term * val;
imagesc('XData',[0,1],'YData',[0,1],'CData',val);
axis image
colorbar;
```

**PYTHON**

```python
x = np.arange(0,1,0.001)
y = np.arange(0,1,0.001)
mesh_x, mesh_y = np.meshgrid(x,y)
constant_term = 1/(0.2*np.pi)
val = np.exp(-1*((mesh_x - 0.5)**2 + (mesh_y - 0.5)**2)/(2*(0.2)**2))
val = val * constant_term
```
Example:

Compute multi-channel convolution
Multi channel convolution computation

Image patch $M_1$

$M_1(:,:,1)$

$2\ 1\ 2$

$1\ 3\ 1$

$M_1(:,:,2)$

$2\ 1\ 2$

$2\ 1\ 2$

$1\ 0\ 1$

$M_1(:,:,3)$

$2\ 2\ 1$

$1\ 2\ -1$

$2\ 1\ 2$

Convolution kernel $F_1$

$F_1(:,:,1)$

$0\ 1\ 0$

$0\ 0\ 0$

$1\ 0\ 0$

$F_1(:,:,2)$

$0\ 1\ 0$

$0\ 0\ 0$

$F_1(:,:,3)$

$0\ 1\ 0$

$0\ 0\ 0$

$0\ 0\ 1$
Multi channel convolution computation

Image patch

Convolution kernel

Compute center value for each channel
Sum up the result from all channel
Just flip the kernel on H and W space

\[
\begin{align*}
&M1(:,:,1) \times F1(:,:,1) + M1(:,:,2) \times F1(:,:,2) + M1(:,:,3) \times F1(:,:,3) \\
= & (1 \times 0 + 2 \times 1 + 1 \times 0) + (2 \times 1 + 1 \times 2 + 2 \times 0) + (1 \times 0 + 2 \times 1 + 1 \times 0) \\
= & 3 + 2 + 2 = 7
\end{align*}
\]
MATLAB

img_patch = zeros(3,3,3);
img_patch(:,:,1) = [1,2,1;2,1,2;1,2,1];
img_patch(:,:,2) = [1,2,-1;2,1,2;1,3,1];
img_patch(:,:,3) = [2,2,1;2,1,2;1,0,1];

kernel = zeros(3,3,3);
kernel(:,:,1) = [0,1,0;0,1,0;0,0,0];
kernel(:,:,2) = [0,1,0;0,0,0;1,0,0];
kernel(:,:,3) = [0,1,0;0,0,0;0,0,1];

Flipped_kernel = flipud(fliplr(kernel));
val_matrix = img_patch.*Flipped_kernel;
Result = sum(val_matrix(:));

Result
7.0

PYTHON

Img_patch = np.zeros((3,3,3))
img[:,:,0] = np.array([[1,2,1],[2,1,2],[1,2,1]])
img[:,:,1] = np.array([[1,2,-1],[2,1,2],[1,3,1]])
img[:,:,2] = np.array([[2,2,1],[2,1,2],[1,0,1]])

kernel = np.zeros((3,3,3))
kernel[:,:,0] = np.array([[0,1,0],[0,1,0],[0,0,0]])
kernel[:,:,1] = np.array([[0,1,0],[0,0,0],[1,0,0]])
kernel[:,:,2] = np.array([[0,1,0],[0,0,0],[0,0,1]])

Flipped_kernel = kernel[::-1,:,::-1]
Result = np.sum(Flipped_kernel*img)

Result
7.0
Example:

Compute multi-kernel multi-channel convolution
(image to column)
Input: feature map or image
\( H \times W \times C_1 \)

Convolutional kernel group
\( C_2 \times K \times K \times C_1 \)

Output: feature map
\( H \times W \times C_2 \)

C2 kernels
Each kernel has size \( K \times K \times C_1 \)

RGB image \( C_1 = 3 \)
Gray image \( C_1 = 1 \)
Zero padding input feature map M1
For input kernel size K
P = K-1

Convolution kernel
k*k, k = 5, P = 4
If $k \times k$ sampling window move 1 pixel each time, it will move $W$ times on each row and $H$ times on column.

Convolution kernel $k \times k$, $k = 5$
Naive implementation

For each sampling window in feature map
  Img_patch = get_patch(img, window)
For kernel in all c2 convolution kernels
  compute_center_val(kernel, img_path)
End

Input feature map H*W*C1
We padded it to (H+P)*(W+P)*C1
Suppose sampling window move 1 pixel each time
Outer loop executed H*W times
Inner loop executes C2 times
For each sampling window in feature map
  \[\text{Img\_patch} = \text{get\_patch}(\text{img, window})\]
For kernel in all c2 convolution kernels:
  \[\text{compute\_center\_val}(\text{kernel, img\_path})\]
End
End

Output dim
\[H \times W \times C2\]
The actual size of feature map
\((H+P) \times (W+P) \times C_1\)
Where \(P\) denotes the padding size to make output has same size of input
Padded Feature Map M1
\((H+P)*(W+P)*C_1\)
Dim = 3

Feature map M2
\((H*W)*(C_1*K*K)\)
Dim = 2

\[\begin{array}{cccc}
K*K & K*K & \cdots & K*K \\
\end{array}\]
Padded Feature Map M1
\((H+P)*(W+P)*C1\)
Dim = 3

Feature map M2
\((H*W)*(C1*K*K)\)
Dim = 2
Padded Feature Map \( M_1 \)
\((H+P)\times(W+P)\times C_1\)
Dim = 3

Feature map \( M_2 \)
\((H\times W)\times(C_1\times K\times K)\)
Dim = 2

**im2col**
Padded Feature Map M1
\((H+P)*(W+P)*C1\)
Dim = 3

Feature map M2
\((H*W)*(C1*K*K)\)
Dim = 2
Each Kernel $K \times K \times C_1$

Total $C_2$ kernels

Conv kernel $F_1$
$C_2 \times K \times K \times C_1$
Dim = 4

Flip kernel
flatten

Conv kernel $F_2$
$C_2 \times (K \times K \times C_1)$
Dim = 2

Reshape

(k \times k \times C_1)
Feature map $M_2$

$((K \times K \times C_1) \times (H \times W))$

Kernel matrix $F_{2, \text{transpose}}$

$C_2$
For each sampling window in feature map:
   \( \text{Img\_patch} = \text{get\_patch(img, window)} \)
   For kernel in all \( c2 \) convolution kernels:
      \( \text{conv(kernel, img\_patch)} \)
   End
End
Formulate the convolution operation as matrix multiplication.

Conv kernel F1
C2*K*K*C1
Dim = 4

Feature Map M1
H*W*C1
Dim = 3

Flip kernel
Conv kerne F2
C2*(K*K*C1)
Dim = 2

Zero padding
Feature map M2
(H*W)*(K*K*C1)
Dim = 2

im2col
Conv kerne F2.transpose
(K*K*C1)*C1
Dim = 2

Matrix multiplication
Output
(H*W)*C2
reshape
Output
H*W*C2
Feature Map M1
H*W*C1
Dim = 3

Feature map M2
(H*W)*(C1*K*K)
Dim = 2

M1 has H*W*C1 element
M2 has H*W*(C1*K*K) element

The cost to killing loop:
more storage.

Trade off between space and time