

UNIVERSITY OF GHENT  
DEPARTMENT OF TELECOMMUNICATION  
AND INFORMATION PROCESSING

SUMMER INTERNSHIP  
COURSE: COMPUTER GRAPHICS

Assignment: Optimizing and making code for context-aware patch-based image inpainting more  
user friendly

Student

Mihailo Drljača

Mentor

Ph.D. Aleksandra Pižurica

## 1. INTRODUCTION

Image inpainting is an image processing task of filling in the missing region in an image in a visual plausible way. Applications are wide such as image restauration, photo editing, image coding and transmission etc. Context-aware patch-based image inpainting using Markov random field modelling is a novel method developed by Ph.D. Aleksandra Pižurica and Ph.D. Tijana Ružić. The main idea of this method is to employ contextual (textural) descriptors to guide and improve the inpainting process. Two most important contributions are:

- Context-aware patch selection strategy
- Interface method for global MRF-based inpainting

Context-aware patch selection strategy is not limited to global inpainting. It aims at improving and accelerating the search for candidate patches in patch-based methods in general.

In this particular realization of the method normalized texton histograms computed from Gabor filter responses are used as contextual descriptors. In general any other contextual descriptors can be used.

There are two strategies for dividing an image into regions based on the context:

- Division into fixed-size square non-overlapping blocks
- Division into blocks of adaptive size

In this particular realization of the inpainting method division into blocks of adaptive size is used.

Interface method for global MRF-based inpainting uses novel optimization approach that makes it suitable in case of large number of labels.

Code is written in Matlab. Original version consists of 144 functions written in Matlab. There is many functions that are similar to each other because the code was written during the research. These similar functions are differing vary little because they were written to be optimal for a specific use.

Task was to select and gather functions that are essential to the method so that optimized version of the program can be delivered to the various users. Besides that optimized version of the code is commented so that further revisions can be easily made. Parts of the code that are prompting user to interact with program makes it more user friendly. Every function that is called in the main function is assigned with an explanation that can be read by using help function in Matlab. Script is called **content\_aware\_mrf\_inpainting** and it also contains an explanation. Version of script file that is realized as a function called **content\_aware\_mrf\_inpainting\_func**.

## 2. SCRIPT FILE

Code is realized as script that is calling all functions. Structure of program is given below:

### 1. **content\_aware\_mrf\_inpaintings**

#### 1.1. **textonGenerateContrastNorm**

##### 1.1.1. **createGabor**

#### 1.2. **decomposition\_adaptive\_blocksize**

##### 1.2.1. **splitVertically**

##### 1.2.2. **splitHorizontally**

##### 1.2.3. **chiTestTexton**

#### 1.3. **textonHistBlock**

#### 1.4. **gist\_matches\_all\_adaptive\_blocksize\_weighted**

##### 1.4.1. **find\_neighbours\_adaptive**

#### 1.5. **find\_label\_pos\_per\_adaptive\_blocksize**

#### 1.6. **comp\_data\_gist\_ver4\_adaptive\_scaled\_weighted**

##### 1.6.1. **compdata.c**

#### 1.7. **label\_extraction\_gist\_weighted**

##### 1.7.1. **find\_first**

##### 1.7.2. **label\_pruning\_art\_fast**

##### 1.7.3. **find\_neighbours**

##### 1.7.4. **calculate\_weighted\_dif\_fast1**

###### 1.7.4.1. **overlap\_region**

###### 1.7.4.2. **calculatediffhelper.c**

##### 1.7.5. **upade\_priority\_scaled**

## 1.8. **compute\_pot\_art**

### 1.8.1. **find\_neighbours**

### 1.8.2. **comp\_matrix\_pot\_art**

#### 1.8.2.1. **overlap\_region**

#### 1.8.2.2. **calculatepotentialhelper.c**

## 1.9. **compute\_cost\_art**

### 1.9.1. **getpatch**

### 1.9.2. **ssd3**

## 1.10. **NCMP**

### 1.10.1. **find\_neighbours**

### 1.10.2. **normalise\_new\_inf**

## 1.11. **output\_art\_mincut**

### 1.11.1. **binomialFilter**

### 1.11.2. **getpatch**

### 1.11.3. **dpmain**

#### 1.11.3.1. **dp1**

#### 1.11.3.2. **dp2**

#### 1.11.3.3. **randmin**

### 1.11.4. **dp**

### 1.11.5. **rconv2**

## 1.12. **show\_order**

Script file is called **content\_aware\_mrf\_inpainting**. After defining the name of the script, explanation of the script file is given. It can be called with **help** function. (0) After the explanation all used variables are described.

```
>> help content_aware_mrf_inpainting
ADDME: context-aware global MRF-based inpainting method
Image inpainting is an image processing task of filling in the missing
Region in an image in a visually plausible way. First the user is asked to
Load an image, which is going to be inpainted and the mask of the image.
Next step is to employ contextual (textural) descriptors to guide and improve
The inpainting process. Contextual descriptors are normalized texon histograms
Computed from Gabor filter responses. Image is divided into regions based on
The context into blocks of adaptive size. This strategy is called top-down
Splitting procedure, which is based on contextual descriptors. Inpainting is
MRF-based.

Based on the following paper:
Context-aware patch-based image inpainting using Markov random field
Modeling by Tijana Ruzic and Aleksandra Pizurica Member, IEEE

Code written by PhD Tijana Ruzic
Code optimized and made more user-friendly by student Mihailo Drljaca
```

*Picture 2.1. Matlab command window*

Before creating contextual descriptors image and mask have to be loaded. User is prompt to insert path to the mask and the image. Then user has to insert its name with an extension. Matlab function **input** is being used to prompt the user. Functions **imread** and **im2double** are used to load the image and mask. After that the mask and the image are plotted. (Picture 2.4)

```
%ADDIND PATH TO THE FOLDER WITH IMAGES
img_path = input('Enter the path of the image: ','s');
addpath(img_path);

%LOADING IMAGE THAT IS GOING TO BE INPAINTED
img = input('Enter the name of the picture with the extention: ','s');
fillRegion = input('Enter the name of the mask with the extention: ','s');

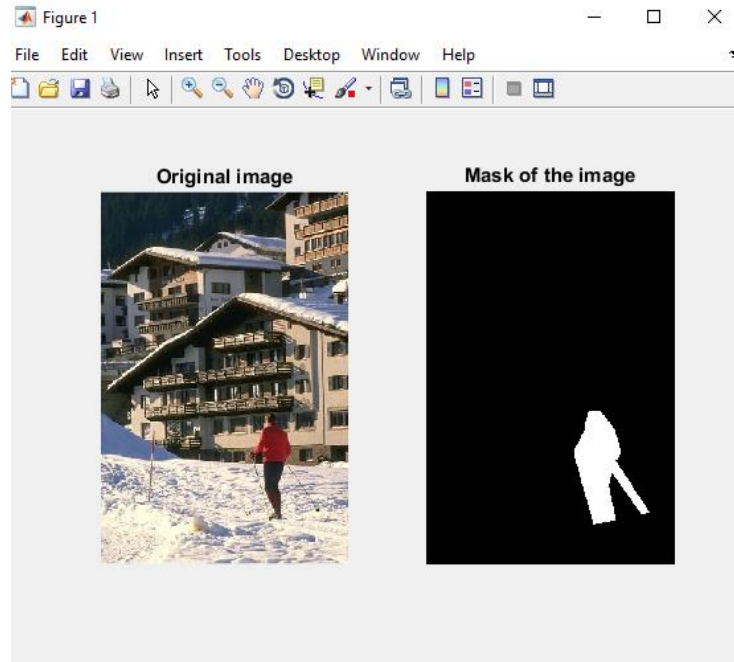
img = im2double(imread(img));
fillRegion = im2double(imread(fillRegion));

figure(1);
subplot(1,2,1);
imshow(img);
title('Original image');
subplot(1,2,2);
imshow(fillRegion);
title('Mask of the image');
```

*Picture 2.2. Part of the code that is responsible for loading image and mask*

```
>> content_aware_mrf_inpainting
Enter the path of the image: c:/Users/Mihailo/Desktop/praksa/slike_mail
Enter the name of the picture with the extension: skiresort.ppm
Enter the name of the mask with the extension: skiresortMask.pgm
```

Picture 2.3. User interface – Matlab command window



Picture 2.4. Plotted image and mask

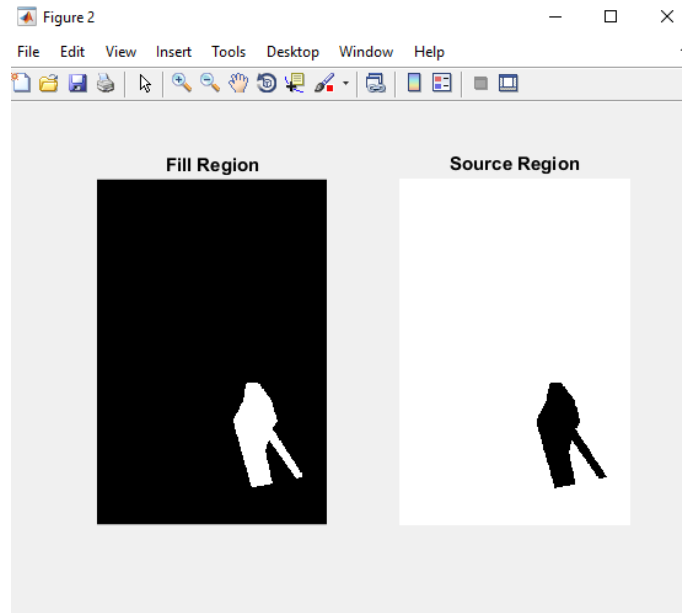
Next step is to define minimum block size of the image. Default parameter is  $f = 8$ . User is prompt to decide weather he is going to use default value or not. Length and width are divided by  $f$  and stored into  $bsmin$  parameter. Source and fill region are plotted. (Picture 2.6)

```
%DEFINING MINIMUM IMAGE BLOCK
yes_no = 0;
while(yes_no == 0)
    yes_no = input('Use default min block size i.e. f = 8: Y/N ', 's');
    if(yes_no == 'y' || yes_no == 'Y')
        f = 8;
    elseif(yes_no == 'n' || yes_no == 'N')
        f = input('Enter the max number of blocks along one side of the image: ');
    else
        yes_no = 0;
    end
end

bsmin = floor(size(fillRegion)./f);
bsmin = [bsmin(1) bsmin(2)];

img = img(1:bsmin(1)*f, 1:bsmin(2)*f, :);
fillRegion = fillRegion(1:bsmin(1)*f, 1:bsmin(2)*f);
sourceRegion = 1-fillRegion;
```

Picture 2.5. Part of the code that is responsible for defining minimum block size



*Picture 2.6. Plotted source and fill region*

For context-aware patch algorithm following parameters have to be loaded.

- nTex – number of textons
- orientationsPerScale - number of orientations over 3 scales
- Ts - block similarity threshold used while decomposing image
- Tb - block similarity threshold while context aware patch selection is being performed
- nGistMatches - max number of blocks from where the patches are considered

Once again user can choose between default and personalized values.

```
Use default min block size i.e. f = 8: Y/N y
Default parameters are:
nTex = 18
orientationsPerScale = [6 6 6]
Ts = 0.15
Tb = 0.15
nGistMatches = 10
Use default parameters: Y/N |
```

*Picture 2.7. Parameters displayed in Matlab command window*

After loading of the parameters contextual descriptors are being calculated. Function that is responsible for this part of the method is **textonGenerateContrastNorm**. Further explanation of the function is given in Matlab help selection. (Picture 2.8)

```
>> help textonGenerateContrastNorm
ADDME: creating contextual descriptors
Contextual descriptors characterize spatial content and texture within
blocks. Texture is extracted using multi-channel filtering. Contextual
descriptors are implemented as texton histograms.
```

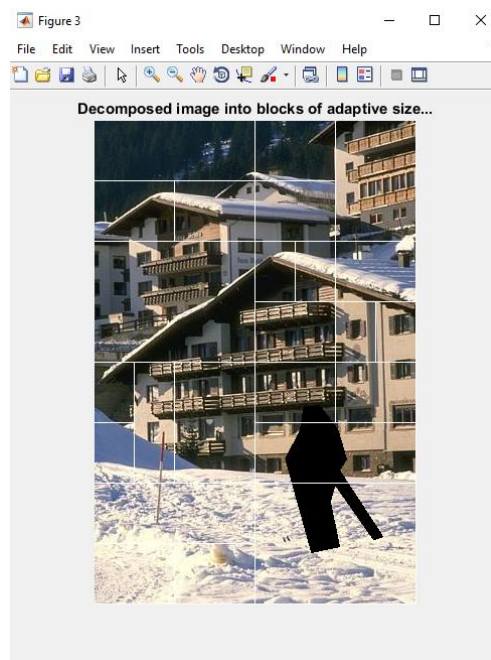
Picture 2.8. Matlab command window

Next step is to divide image into blocks of adaptive size. Function that is responsible for division is **decomposition\_adaptive\_blocksize**. (Picture 2.9)

```
>> help decomposition_adaptive_blocksize
ADDME: top-down splitting procedure
Divides an image into blocks of adaptive size depending on the 'homogeneity'
of their texture. Directional flag is assigned to each block. This flag
determines the direction along which the evaluation of the blocks
homogeneity will have the priority. Splitting along one side of the block
is constrained by block similarity threshold.
```

Picture 2.9. Matlab command window

After decomposition processed image is displayed. (Picture 2.10)



Picture 2.10. Decomposed image



Part of the code that is responsible for previously described process is shown below.

```
%CALCULATING CONTEXTUAL DESCRIPTORS
disp('Contextual descriptors are being calculated i.e. mapTextons...');
[mapTextons, C, gfull] = textonGenerateContrastNorm ...
    (img,fillRegion,nTex,orientationsPerScale);

%DIVISON INTO BLOCKS OF ADAPTIVE SIZE
disp('Image is being decomposed into blocks of adaptive size...');
[allBlocks1,backupBlockPos,blocks] = decomposition_adaptive_blocksize...
    (img,fillRegion,mapTextons,nTex,bsmin,Ts,r);

figure(3);
imshow(blocks);
title('Decomposed image into blocks of adaptive size...');
```

Picture 2.11. Part of the code responsible for creating contextual descriptors and dividing image into blocks of adaptive size

To perform inpainting it is obligatory to find positions of the contextually similar blocks. Functions that are employed in this section are **textonHistBlock** and **gist\_matches\_all\_adaptive\_blocksize\_weighted**.

```
>> help gist_matches_all_adaptive_blocksize_weighted
ADDME: context-aware patch selection
Fiding positions of the blocks that are contextually similar. Block is
considerd to be similar to it self only if the block is reliable.
Reliable block is block with more than half known pixels. If the block is
unreliable similar blocks will be found based on the neighbouring blocks
of the unreliable block.
```

Picture 2.12. Matlab command window

```
%COMPUTING HISTOGRAM FOR INE TEXTURE FEATURE FOR EACH BLOCK
disp('computing histogram for one texture feature for each block...');
gstartTexton = textonHistBlock(mapTextons, ~fillRegion, allBlocks1, nTex);

%CALCULATING GIST MATCHES ONLY FOR BLOCKS THAT INTERSECT THE TARGET REGION
disp('Calculate gist matches only for blocks that intersect the target region');
[gmatchesTexton,nbrsTexton,gerrorsTexton] = gist_matches_all_adaptive_blocksize_weighted ...
    (gstartTexton, ~fillRegion, allBlocks1, nGistMatches, Tb);
```

Picture 2.13. Part of the code responsible for finding positions of the blocks that are contextually similar

For MRF- based inpainting following parameters have to be loaded.

- L – number of labels kept after pruning
- ITER- number of iterations
- gap - determines the patch size as  $2\text{gap}+1$

Once again user can choose between default and personalized values.

```
Default parameters are:  
L = 10  
ITER = 10  
gap = 6
```

Picture 2.14. Parameters displayed in Matlab command widow

Part of the code that is responsible for patch-based inpainting process uses memory efficient optimization method. This method consists of three steps. Initialization, label pruning and inference.

Function responsible for initialization is **comp\_data\_gist\_ver4\_adaptive\_scaled\_weighted**. Before executing initialization it is necessary to find all labels positions with function called **find\_label\_pos\_per\_adaptive\_block\_overlap**.

```
%FINDING LABEL POSITION  
disp('Finding label position...');  
tic;  
labBlock = find_label_pos_per_adaptive_block_overlap(sourceRegion, gap, allBlocks1);  
T2 = toc;  
  
%EFFICIENT ENERGY OPTIMIZATION STEP 1  
%ASSIGNING PRIORITIES TO MRF NODES  
disp('Efficient energy optimization step 1(initialization)...');  
tic;  
[coordy, coordx, priority, labels, diff, diff_ssd, nrLabelsPerBlock, nodeDblock] = ...  
    comp_data_gist_ver4_adaptive_scaled_weighted(img, sourceRegion, labBlock, ...  
    gmatchesTexton, Tuncertain, Tb, gap, allBlocks1, gerrorsTexton);  
T3 = toc;
```

Picture 2.15. Part of the code responsible for initialization of energy efficient optimization

```
>> help comp_data_gist_ver4_adaptive_scaled_weighted
ADDME: Efficient energy minimization step1(INITIALIZATION)
Assigning priorities to MRF nodes which determine theid visiting order in
the next phase (label pruning)
```

Picture 2.16. Matlab commad prompt window

Second step label pruning is realized with **label\_extraction\_gist\_weighted**.

```
%EFFICIENT ENERGY OPTIMIZATION STEP 2
%LABEL PRUNING
disp('Efficient energy optimization step 2(label pruning)...');
tic;
[labelsNew, order, priorityNew] = label_extraction_gist_weighted ...
    (img, sourceRegion, coordy, coordx, priority, diff, diff_ssd, labels, ...
    nrLabelsPerBlock, nodeDblock, L, Tsim, Tuncertain, gap);
T4 = toc;
```

Picture 2.17. Part of the code responsible for label pruning

```
>> help label_extraction_gist_weighted
ADDME: Efficient energy optimization step2(LABEL PRUNING)
After computing the label-pruning distance measure for each node, nodes
are visited in the order of their priority keeping L labels with the
smallest distance measure and discarding the rest. For interior nodes,
(label cost is 0) only available information is the one coming from
the neighbors. Each nod is visited only once during label pruning. Once
chosen set of labels per node remains fixed throughout the rest of the
energy optimization algorithm.
```

Picture 2.18. Matlab command prompt

Third and last step of energy optimization is neighbor consensus message passing. Function is call **NCMP**. Before calling NCMP label cost and pairwise potential have to be calculated. Function that calculates label cost is called **compute\_cost\_art**. Inside **compute\_cost\_art** there is **ssd3** function that calculates common sum of squared differences. Pairwise potential is calculated with **compute\_pot\_art** function.

```
>> help compute_pot_art
ADDME: pairwise potential
Pairwise potential Vjk(xj,xk) is similarly as label cost defined as the
SSD between labels centered at xj and xk in their nodes' overlap region.
Check the function compute_cost_art for more detail
```

Picture 2.19. Matlab commad prompt

```
>> help compute_cost_art
ADDME: label cost
Label cost  $V_i(x_i)$  measures the agreement of a node with its labels. Common sum of squared differences(SSD) is used as a distance measure between the values of the known pixels in the  $W \times W$  neighborhood of the node  $i$  and the corresponding pixel values of the label at  $x_i$ . If the  $W \times W$  neighborhood of a node is completely inside target region the label cost is zero.
```

Picture 2.20. Matlab commad prompt

```
%COMPUTING PAIRWISE POTENTIAL MATRIX  $V_{jk}(x_j, x_k)$ 
disp('Computing the pairwise potential matrix  $V_{jk}(x_j, x_k) \dots$ ');
tic;
pot = compute_pot_art(img, coordy, coordx, labelsNew, gap);

%COMPUTING LABEL COST  $V_i(x_i)$ 
disp('Computing the label cost  $V_i(x_i) \dots$ ');
cost = compute_cost_art(img, sourceRegion, coordy, coordx, labelsNew, gap);
loc = exp(-cost);
[max_value, InitMask] = max(loc, [], 2);

%EFFICIENT ENERGY OPTIMIZATION STEP 3
%NEIGHBOURHOOD-CONSENSUS MESSAGE PASSING(NCPM)
disp('Efficient energy optimization step 3(neighbourhood-consensus message passing) \dots');
[beliefs, OutMask, iter] = NCMP(loc, pot, InitMask, coordx, coordy, gap, ITER);
T5 = toc;
```

Picture 2.21. Part of the code responsible for final step of energy efficient optimization

When the processing of missing region filled with patches of the original image is done, function **output\_art\_mincut**, name and path of the inpainted image and image representing filling order have to be given.

```
%PROCESS ORIGINAL MISSING REGION FILLED WITH PATCHES
disp('Processing original missing region filled with patches \dots');
target1 = output_art_mincut(OutMask, labelsNew, order, coordy, coordx, img, sourceRegion, gap);

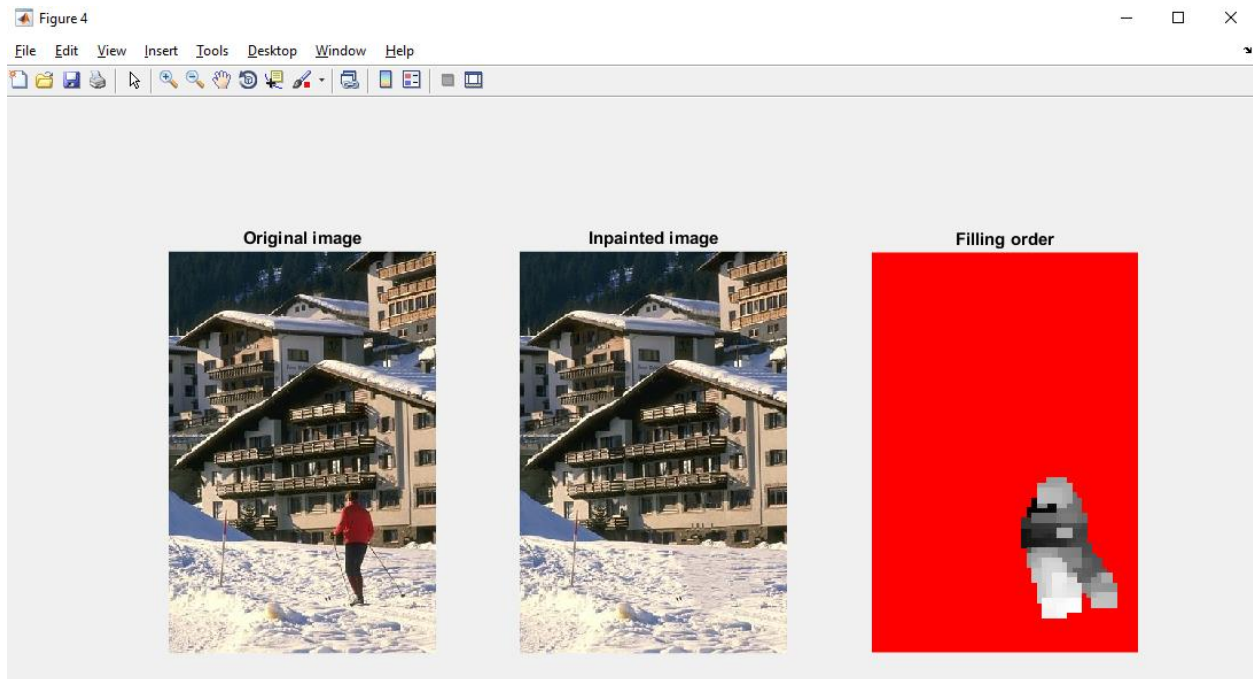
%GIVING THE NAME TO THE INPAINTED IMAGE
img_out_path = input('Enter the path of the out image: ', 's');
addpath(img_out_path);
img_out_path = strcat(img_out_path, '/');

nameout = input('Enter the name of the inpainted image with the extention: ', 's');
nameout = strcat(img_out_path, nameout);
imwrite(target1, nameout, 'bmp');

orderImg = show_order(img, sourceRegion, order, coordx, coordy, gap);
nameout1 = input('Enter the name of the filling order image with the extention: ', 's');
nameout1 = strcat(img_out_path, nameout1);
imwrite(uint8(orderImg), nameout1, 'bmp');
```

Picture 2.22. Part of the code responsible for storing inpainted image

After finishing the inpainting process original image, inpainted image and filling order image are displayed.



*Picture 2.23. Original image, inpainted image and filling order image*

### 3. FUNCTION

In addition to script file function called **content\_aware\_mrf\_inpainting\_func** is realized. All of the parametars that were prompted from the user in scrip file are now requested as parametars of the function. Function returns path of the inpainted image and filling order image.

```
function [path_out_img, path_out_fill] = content_aware_mrf_inpainting_func1  
(inpaintImg, inpaintMask, param_f, number_of_textons, orientations_per_scale,  
threshold_sim, constrained_source_region, number_of_labels, number_of_iterations,  
patch_size)
```

Picture 2.24. Function *content\_aware\_mrf\_inpainting\_func*

inpaintImg	- name of original image enter as string inside '' with extention
inpaintMask	- name of the mask enter as strig inside '' with extention
param_f	- defining min image block
number_of_textons	- nuber of textons
orientations_per_scale	- number of orientations over 3 scales enter inside []
threshold_sim	- if orientations per scale is: [6 6 6] recommended value is 0.15 [8 8 8] recommended value is 0.2
constrained_source_region	- max number of blocks from where the patches are considered recommended value is 10
number_of_labels	- number of labels kept after pruning recommended value is 10
number_of_iterations	- recommended value is 10
patch_size	- defined as gap = 2*gap+1 recommended value is 6

Picture 2.25. Parametars of the function

Based on the following paper: T. Ružic and A. Pižurica, “Context-aware patch-based image inpainting using Markov random field modeling,” IEEE Transactions on Image Processing, vol. 24, no. 1, pp. 444-456, Jan 2015