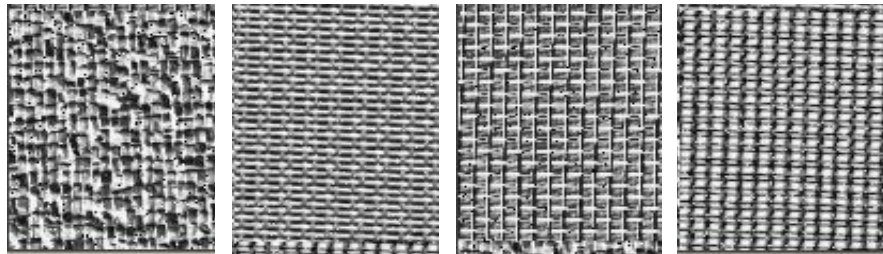# Lab 06

Basic Image Processing Algorithms
Fall 2024

Texture segmentation with Laws filters

Aim is to recognize trained textures on an image. **The algorithm has two phases:**
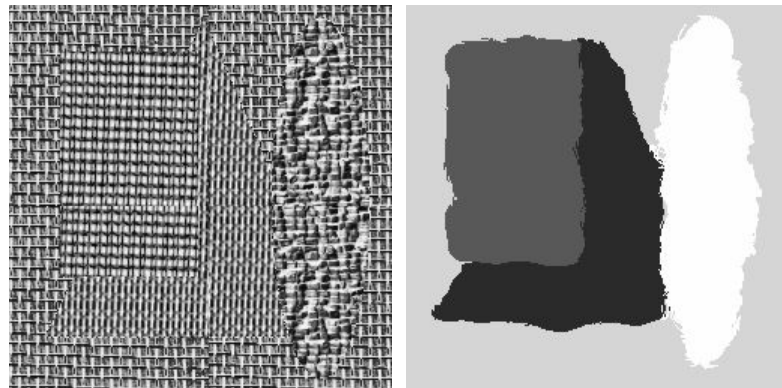
Phase 1: Training

Calculate a set of scalars representing each texture, using different training images.

Training images

Phase 2: Recognition

Find the known textures in the picture and classify its pixels into the appropriate bins.

Input test image   Output of the classifier

2

Laws filter – mathematical background

Laws filter uses different types of **kernels** for convolution and then a **summation** on the convolved images is carried out resulting in a scalar **number** which represents a texture-kernel pair.

The convolution kernels are created from vector-vector products of the following 1D filters:

$$L = \frac{1}{6} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \qquad E = \frac{1}{2} \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \qquad S = \frac{1}{2} \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}$$

Level detection filter   Edge detection filter   Spot detection filter

Laws filter – kernels

$$L = \frac{1}{6} \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} \qquad E = \frac{1}{2} \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix} \qquad S = \frac{1}{2} \begin{bmatrix} 1 \\ -2 \\ 1 \end{bmatrix}$$

$$H_1 = LL^T = \frac{1}{36} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \qquad H_2 = LE^T = \frac{1}{12} \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \qquad H_3 = LS^T = \frac{1}{12} \begin{bmatrix} 1 & -2 & 1 \\ 2 & -4 & 2 \\ 1 & -2 & 1 \end{bmatrix}$$
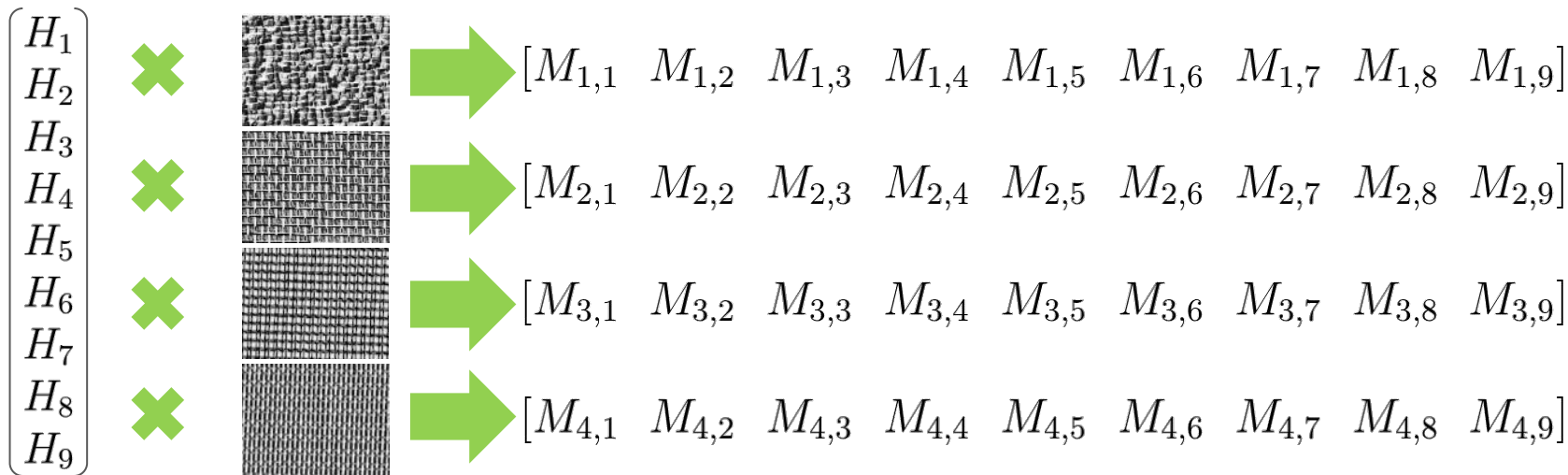
$$H_4 = EL^T = \frac{1}{12} \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \qquad H_5 = EE^T = \frac{1}{4} \begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix} \qquad H_6 = ES^T = \frac{1}{4} \begin{bmatrix} 1 & -2 & 1 \\ 0 & 0 & 0 \\ -1 & 2 & -1 \end{bmatrix}$$

$$H_7 = SL^T = \frac{1}{12} \begin{bmatrix} 1 & 2 & 1 \\ -2 & -4 & -2 \\ 1 & 2 & 1 \end{bmatrix} \qquad H_8 = SE^T = \frac{1}{4} \begin{bmatrix} 1 & 0 & -1 \\ -2 & 0 & 2 \\ 1 & 0 & -1 \end{bmatrix} \qquad H_9 = SS^T = \frac{1}{4} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

Phase 1: Training

In the training phase, every texture sample gets convolved (and then summed) with the 9 different Laws filter kernels, resulting in 9 scalar numbers for a texture sample. The model of the texture is the 9-element long vector built from the $M_{n,k}$ numbers, where n is the ID of the texture sample, and k is the kernel index (H_k).

$$\begin{pmatrix} H_1 \\ H_2 \\ H_3 \\ H_4 \\ H_5 \\ H_6 \\ H_7 \\ H_8 \\ H_9 \end{pmatrix}$$

$$[M_{1,1} \quad M_{1,2} \quad M_{1,3} \quad M_{1,4} \quad M_{1,5} \quad M_{1,6} \quad M_{1,7} \quad M_{1,8} \quad M_{1,9}]$$

$$[M_{2,1} \quad M_{2,2} \quad M_{2,3} \quad M_{2,4} \quad M_{2,5} \quad M_{2,6} \quad M_{2,7} \quad M_{2,8} \quad M_{2,9}]$$

$$[M_{3,1} \quad M_{3,2} \quad M_{3,3} \quad M_{3,4} \quad M_{3,5} \quad M_{3,6} \quad M_{3,7} \quad M_{3,8} \quad M_{3,9}]$$

$$[M_{4,1} \quad M_{4,2} \quad M_{4,3} \quad M_{4,4} \quad M_{4,5} \quad M_{4,6} \quad M_{4,7} \quad M_{4,8} \quad M_{4,9}]$$

The training procedure is the following:

for each texture $(n = 1, 2, ..., N)$:

    for each filter $(k = 1, 2, ..., 9)$:

Convolve the training texture T_n with the appropriate kernel: $A = T_n \otimes H_k$

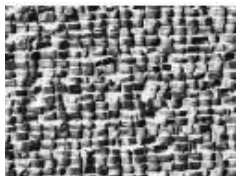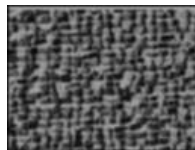Compute the k-th element of the vector describing the n-th texture:

$$M_{n,k} = \frac{1}{hw} \sum_{x=1}^{w} \sum_{y=1}^{h} A^2(x, y)$$

Where h and w are the height and width of the training texture image T_n

The different
Laws filters

Filtered training
texture image

Squared averaging

$$H_1 = \frac{1}{36} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$$M_{i,1} = \frac{1}{hw} \sum_{x=1}^{w} \sum_{y=1}^{h} A_1^2(x,y)$$

The i-th training
texture image

$$A_1 = T_i \otimes H_1$$

$$H_2 = \frac{1}{12} \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

$$M_{i,2} = \frac{1}{hw} \sum_{x=1}^{w} \sum_{y=1}^{h} A_2^2(x,y)$$

$$T_i \in [0,1]^{(h \times w)}$$

$$A_2 = T_i \otimes H_2$$

$$H_9 = \frac{1}{4} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

$$M_{i,9} = \frac{1}{hw} \sum_{x=1}^{w} \sum_{y=1}^{h} A_9^2(x,y)$$
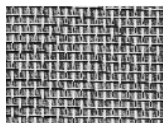
$$A_9 = T_i \otimes H_9$$

7

Phase 1: Training
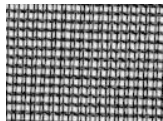
Finally, the result should be a vector of 9 scalar values for every texture.
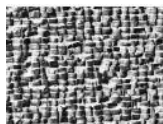
$$[M_{1,1} \quad M_{1,2} \quad M_{1,3} \quad M_{1,4} \quad M_{1,5} \quad M_{1,6} \quad M_{1,7} \quad M_{1,8} \quad M_{1,9}]$$

$$[M_{2,1} \quad M_{2,2} \quad M_{2,3} \quad M_{2,4} \quad M_{2,5} \quad M_{2,6} \quad M_{2,7} \quad M_{2,8} \quad M_{2,9}]$$

$$[M_{3,1} \quad M_{3,2} \quad M_{3,3} \quad M_{3,4} \quad M_{3,5} \quad M_{3,6} \quad M_{3,7} \quad M_{3,8} \quad M_{3,9}]$$

$$[M_{4,1} \quad M_{4,2} \quad M_{4,3} \quad M_{4,4} \quad M_{4,5} \quad M_{4,6} \quad M_{4,7} \quad M_{4,8} \quad M_{4,9}]$$

In the program code we will store the "descriptors" of the textures in a so-called "model" matrix. We call the rows of this matrix "classes" and by "class index" we mean the row index (starting from 1).

```
MODEL = [M11 M12 M13 M14 M15 M16 M17 M18 M19
         M21 M22 M23 M24 M25 M26 M27 M28 M29
         M31 M32 M33 M34 M35 M36 M37 M38 M39
         M41 M42 M43 M44 M45 M46 M47 M48 M49];
```

With this matrix $M_{i,j}$ = `MODEL(i,j)`, the parameters of *Class1* are in `MODEL(1,:)`.

Recognition takes the texture model (the $M_{i,j}$ scalars from the training phase) and an input image which consists of arbitrary regions of the pre-trained textures.

Pre-trained textures

input image

classification result
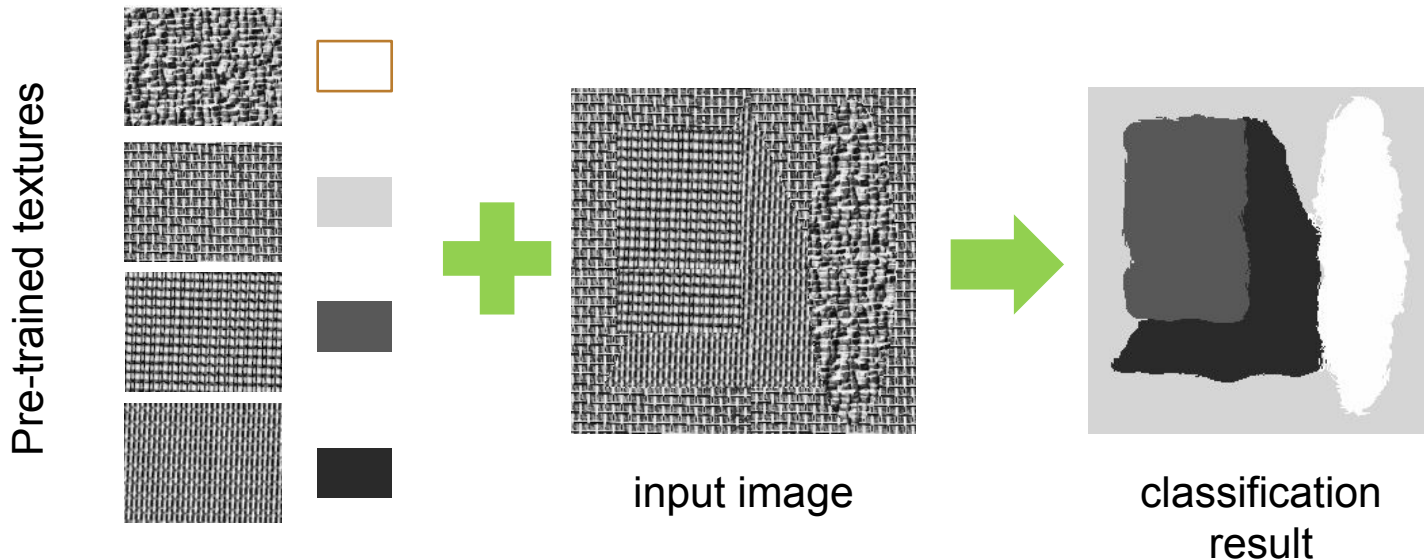
Phase 2: Recognition

The recognition procedure is the following:

for each filter ($k = 1, 2, ..., 9$):

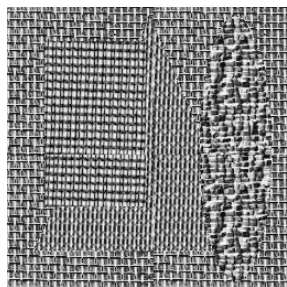Convolve the input image with the appropriate kernel: $B = I \otimes H_k$

Square the result and apply a 15×15 averaging: $B^* = B^2 \otimes N$

Where `N = 1/(15*15) * ones(15)`

The different
Laws filters

Filtered input image

Squared, averaged result

The input image
to be recognized

$$H_1 = \frac{1}{36} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

$B_1 = I \otimes H_1$

$$B_1^* = B_1^2 \otimes N$$

$$H_2 = \frac{1}{12} \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

$B_2 = I \otimes H_2$

$$B_2^* = B_2^2 \otimes N$$

$$H_9 = \frac{1}{4} \begin{bmatrix} 1 & -2 & 1 \\ -2 & 4 & -2 \\ 1 & -2 & 1 \end{bmatrix}$$

$B_9 = I \otimes H_9$

$$B_9^* = B_9^2 \otimes N$$

12

Every B_i^* output of the previous step is a matrix (which has the same size as the image). If the image is $h \times w$ and we concatenate our B_i^* outputs along the 3rd dimension then we get an $(h \times w \times 9)$ size matrix (Let's call it `BB`.)

The task is to classify every pixel. (Classify `:=` tell whether a pixel at location (x, y) belongs to *Class1* or *Class2* or … or *ClassN* based on the texture in its 15×15 neighborhood.)

For classification; at every pixel location we use the 9-element vector of the `BB` matrix at (x, y) and the pre-trained classes stored in `MODEL`.

Phase 2: Recognition

After creating the ($h \times w \times 9$) size `BB` matrix we create an empty `ClassMap` and continue the recognition procedure with a pixel-by-pixel comparison; implementing the following formula:

$$\texttt{ClassMap}(x, y) = \arg \min_{n} \sum_{k} \left| B_k^*(x, y) - M_{n,k} \right|$$

$$n = 1, 2, ..., N \qquad k = 1, 2, ..., 9$$
$$x = 1, 2, ..., w \qquad y = 1, 2, ..., h$$

That is in a more pseudocode way:

for each pixel (x, y):

   for each texture class $(n = 1, 2, ..., N)$:

      Compute the summed absolute difference between the 9-element vectors `BB(x,y,:)` and `MODEL(n,:)` and store the result in a temporary variable: `sum_abs_diff(n)`

   After computing the sum of absolute differences for every class, find the `n` where the value of `sum_abs_diff` is the smallest (i.e. pick the class where the difference between the image texture and the training texture is the smallest). Put this `n` into `ClassMap(x,y)`.
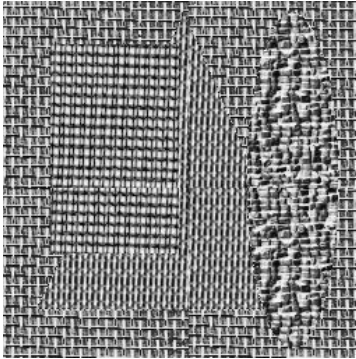
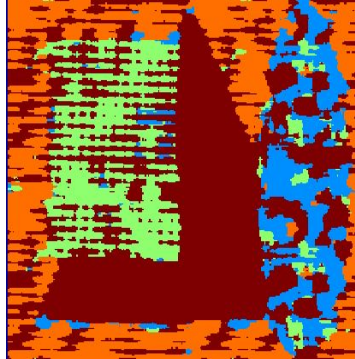   Continue with the next pixel...

Results

Output „winner class" maps for the four textures – enhanced with some morphology

Input image

Laws segmentation results

Ground truth

One more thing: Majority voting

Majority voting is a very common technique to smooth noisy images.

If we know that the image consist of large "constant" areas (i.e. the background, rectangle, triangle and ellipse in this case) then we can know "for sure" that a *Class1* label on its own surrounded by *Class2* labels is more likely to be an error than an actual good classification.
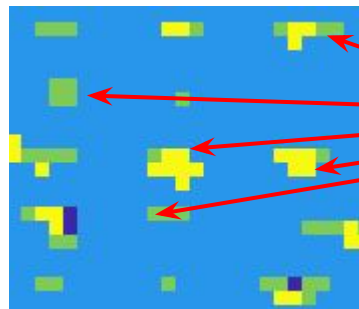
These kind of islands are very likely to be noise.
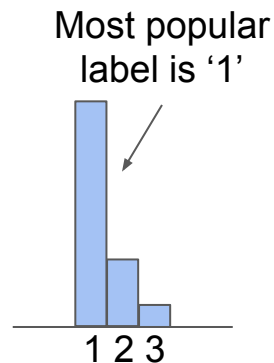
One more thing: Majority voting

The idea of majority voting is that we define a window and count the "votes" (occurrences of the different class labels) inside the region. Then we get the "most popular vote" (the class label with the most frequent occurrence). Change all class labels in this window to the most popular one; this makes the "unpopular" (unlikely) class labels disappear.

Content of ClassMap

Selected window

Most popular label is '1'

Histogram of class labels

Updated map after voting

18

One more thing: Majority voting

To implement majority voting you will need the diameter of the window (`w_dia`). You have to run this window through your input matrix (`IN`) <u>without overlapping</u>. Inside the window use `mode()` to get the most popular element and replace every element inside the window with the majority vote.

Use `x = 1:w_dia:width` and `x:x+w_dia-1` vectors to iterate through your image with step size `w_dia`. (same with `y`)

In this example the window diameter is 3.

Note that there are no overlapping windows.

Also note that if the number of remaining elements is less than a full window then we work with a smaller area.

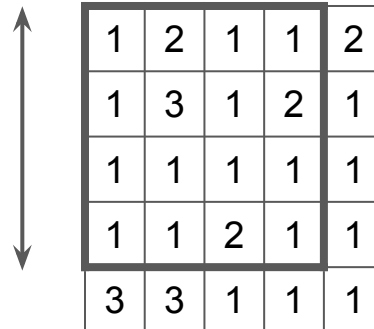| 1	2	1	1	2	1	2	1	1	2	1	2	1	1
1	3	1	2	1	1	3	1	2	1	1	3	1	2
1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	2	1	1	1	1	1	2	1	1	1	1	1
3	3	1	1	1	3	3	3	1	1	3	3	3	3

19

One more thing: Majority voting

Pseudocode for majority voting:

for x=1:w_dia:size(IN,1)
    for y=1:w_dia:size(IN,2)
        $start_x = x$; $end_x = \min\{x + \texttt{w\_dia} - 1; \texttt{size}(IN,1)\}$;
        $start_y = y$; $end_y = $ *similar*
        $SelectedWindow = IN(start_x{:}end_x, \; start_y{:}end_y)$
        $MajorityValue = \text{mode}\{SelectedWindow\}$
        $OUT(start_x{:}end_x, \; start_y{:}end_y) = MajorityValue$

In this example
w_dia = 4

1	2	1	1	2
1	3	1	2	1
1	1	1	1	1
1	1	2	1	1
3	3	1	1	1

20

Now please

download the 'Lab 06' code package

Exercise 1

Implement the function `laws_kernel` in which:
- Check if the input argument (`k`) is valid. It should be a number `1 <= k <= 9`. If this condition not holds, throw an error (use the `error()` function).
- If `k` is valid return the `k`-th Laws kernel. You can do this by computing the appropriate vector product or by hard-coding every kernel in the function; it's up to you. See Slide 4 for the list of the Laws kernels.

You should return a 2D double matrix. Don't forget the normalization factors!

Run `test1.m` which will test your implementation.

Exercise 2

Implement the function `training_phase` in which:
- The input is a cell of training textures: `T_cell ={T1, T2, ..., TN}`
- Get the number of training textures (`N`); create the empty (`N×9`) `MODEL` matrix.
- For each texture realize the training steps described in Slide 6. Fill all the values of `MODEL`. Use `conv2` with the option `'same'` for convolution to get rid of the padding. Use your previously implemented function to get the kernels.

You can assume that all training textures are grayscale images, double, [0,1] range. **The training images might have different sizes!** (I.e. it can happen that T1 is 10×50, T2 is 33×33 etc…). **Pay attention to the order of the training textures!** (I.e. the M values describing T2 must be in the second row of `MODEL`).

Run `test2.m` which will test your implementation.

Exercise 3

Implement the function `recognition_phase` in which:
- The two inputs are the input image and the pre-trained `MODEL` matrix.
- Knowing the size of the input image create the same-sized empty `ClassMap`.
- Realize the recognition steps described on Slides 11 and 15. Fill all the values of `ClassMap`. Use `conv2` with the option `'same'` for convolution to get rid of the padding. Use your previously implemented function to get the kernels.

You can assume that the input is a grayscale image, double, from [0,1] range. **Pay attention to the order of the training textures!** (I.e. if the closest match is with the second row of `MODEL` then you should write 2 in the corresponding location of `ClassMap`.)

**Run `test3.m` which will test your implementation.**

Exercise 4

Implement the function `majority_voting` in which:
- The input is a matrix containing the class labels.
- Knowing the size of the input, create the same-sized empty `OUT`.
- Realize the majority voting described on Slide 20. Fill all the values of `OUT`. Use `mode` and the smart, non-overlapping indexing.
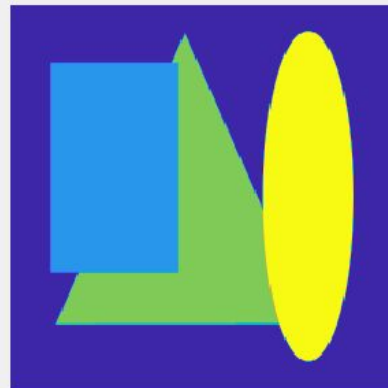
Run `test4.m` which will test your implementation.

If you are ready with all the function implementations, run `final_script.m` and check the results.
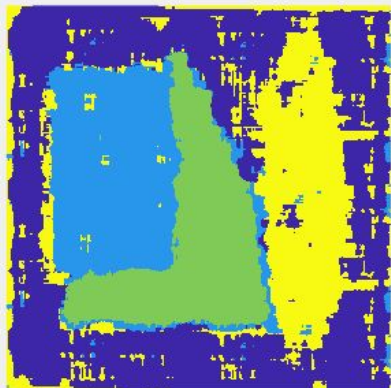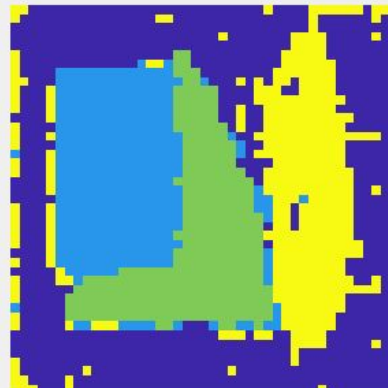
Original input
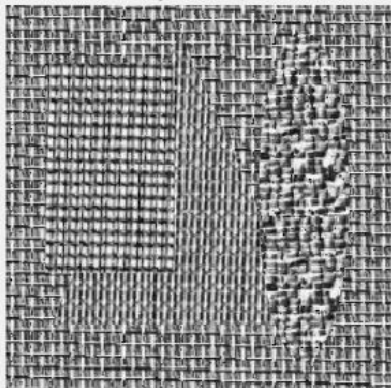
Ground truth

Raw output
84.2148% accurate

Output + majority voting
86.705% accurate

THE END