International Journal of Bifurcation and Chaos, Vol. 18, No. 12 (2008) 3551–3609

IMITATION OF VISUAL ILLUSIONS VIA OPENCV AND CNN

MAKOTO ITOH
Department of Information and Communication Engineering,
Fukuoka Institute of Technology,
Fukuoka 811-0295, Japan

LEON O. CHUA
Department of Electrical Engineering and Computer Sciences,
University of California, Berkeley,
Berkeley, CA 94720, USA

Received February 26, 2008; Revised April 20, 2008

Visual illusion is the fallacious perception of reality or some actually existing object. In this paper, we imitate the mechanism of Ehrenstein illusion, neon color spreading illusion, watercolor illusion, Kanizsa illusion, shifted edges illusion, and hybrid image illusion using the Open Source Computer Vision Library (OpenCV). We also imitate these illusions using Cellular Neural Networks (CNNs). These imitations suggest that some illusions are processed by high-level brain functions. We next apply the morphological gradient operation to anomalous motion illusions. The processed images are classified into two kinds of images, which correspond to the central drift illusion and the peripheral drift illusion, respectively. It demonstrates that the contrast of the colors plays an important role in the anomalous motion illusion. We also imitate the anomalous motion illusions using both OpenCV and CNN. These imitations suggest that some visual illusions may be processed by the illusory movement of animations.

Keywords: Visual illusion; OpenCV; CNN; programming function; template; equilibrium state; nonlinear operator; Hough transform; morphological gradient; thresholding; edge detection; watershed segmentation; optical flow; animation; Ehrenstein illusion; neon color spreading illusion; watercolor illusion; Kanizsa illusion; Fraser illusion; shifted edges illusion; hybrid image illusion; central drift illusion; peripheral drift illusion.

1. Introduction

Human visual systems see something that is not present, or incorrectly see what is present. The information gathered by the eye is processed by the brain to give a percept that does not tally with a physical measurement of the stimulus source. *Visual illusion*[1] is defined as the fallacious perception of reality or some actually existing object. *Illusory colors* [Werner *et al.*, 2007] are defined as the colors that the brain is tricked into seeing. The study of illusory colors demonstrates that color processing in the brain occurs hand in hand with processing of other properties; such as, shape and boundary. Recently, great attention has been drawn to *anomalous motion illusion* [Kitaoka, 2007], which is characterized by illusory motion in a stationary image. In some anomalous motion illusions, color enhances the illusion, namely, it gives a much stronger illusion. Research on visual illusion can provide fundamental insights

[1]For more details of visual illusions, see "*optical illusion*," Wikipedia: The Free Encyclopedia.

into the general brain mechanisms of perception and cognition.

Cellular Neural Network (CNN) [Chua, 1998; Chua & Roska, 2002] is a dynamic nonlinear system defined by coupling only identical simple dynamical systems, called *cells*, located within a prescribed sphere of influence, such as nearest neighbors. Because of its simplicity, and ease for chip (hardware) implementation, CNN has found numerous applications in *Image and Video Signal Processing, Robotic and Biological Visions*, *and Higher Brain Functions*. It is a well-known fact that for many brainlike computations, the CNN universal chip [Chua, 1998; Chua & Roska, 2002] is far superior to any equivalent DSP implementation by at least three orders of magnitude in either speed, power or area. The CNN has the ability to mimic high level brain functions. Many well-known visual illusions have been simulated by CNN image processing [Chua, 1998; Chua & Roska, 2002; Itoh & Chua, 2007].

Open Source Computer Vision Library (OpenCV)[2] is a library of programming functions originally *developed by Intel and optimized for their processors*. This library is mainly aimed at real-time image processing (computer vision), and includes a collection of algorithms and sample code for various computer vision problems. OpenCV's application areas include *Human-Computer Interaction*; *Object Identification*, *Segmentation and Recognition*; *Face Recognition*; *Gesture Recognition*; *Motion Tracking*, *Ego Motion*, *Motion Understanding*; *Structure from Motion*; *and Mobile Robotics*. The OpenCV is not designed as a neural network, but it has the Machine Learning Library, which includes feedforward artificial neural networks, more particularly, multilayer perceptrons.

The difference between OpenCV and CNNs is described as follows: The image processing of CNNs is dynamic, however, that of the OpenCV is static, since the CNN is defined by a system of differential equations, and the OpenCV is usually defined by nonlinear or linear functions. If we integrate OpenCV into CNN image processing, we can use both dynamic and static properties.

In this paper, we imitate the mechanism of *Ehrenstein illusion*, *neon color spreading illusion*, *watercolor illusion*, *Kanizsa illusion*, *shifted edges illusion* [Kitaoka, 2007], and *hybrid image illusion* [Oliva *et al.*, 2006] using OpenCV programming functions. We also imitate them by using CNN templates with the help of OpenCV, thereby allowing us to simulate many visual illusions by equilibrium states of CNNs. Our imitations via OpenCV and CNN suggest that some color illusions are processed by high-level brain functions. We next apply *morphological gradient* operation to *anomalous motion illusions*. The processed output images are classified into two kinds of images. One is a pale or a single colored image and the other is a bright colored image. They seem to emulate the type of illusion which occurs in the *central* vision (*central drift illusion* [Kitaoka, 2007]), and in the *peripheral* vision (*peripheral drift illusion* [Kitaoka, 2007]), respectively. This suggests that color contrast plays an important role in anomalous motion illusion. Finally, we imitate anomalous motion illusions, and glare effect illusion, by using the OpenCV's optical flow and the shift motion CNN templates. These imitations suggest that some visual illusions may be processed by the illusory movement of animations.

2. OpenCV Functions

OpenCV is a collection of programming functions that implement some popular image processing and computer vision algorithms developed by Intel and optimized for their processors. We introduce the basic definition of *Hough Transform*, *morphological gradient*, *thresholding*, *Canny edge detector*, and *optical flow*, in the *OpenCV reference manual*. These functions are basically *nonlinear* operators, and they are used to imitate visual illusions.

2.1. *Hough transform*

Hough Transform[3] is a popular method for extracting geometric primitives from raster images.[4] The simplest version of the algorithm just detects lines, but it is easily generalized to find more complex features. To illustrate the idea, let us start with a straight line (Fig. 1). In the image space, the straight line can be described as $y = mx + n$ and is plotted for each pair of values (x, y). However, the characteristics of that straight line is not x or y,

[2]Open Source Computer Vision Library: http://www.intel.com/technology/computing/opencv/

[3]For more information, see "*Hough transform*," in the OpenCV Reference Manual or Wikipedia.

[4]A raster image, also called a bitmap, is a way to represent digital images. The raster image takes a wide variety of formats, including the familiar gif, jpg, and bmp.

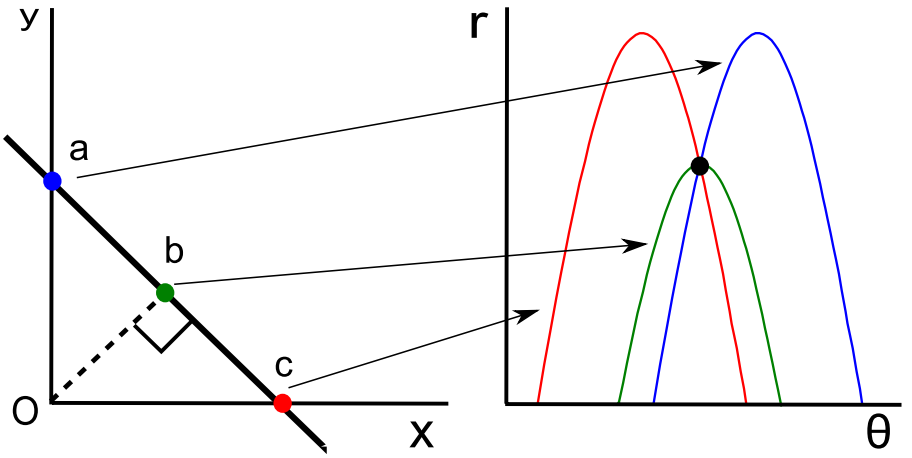

Fig. 1. Hough transform. The points a, b and c are transformed into the blue, green and red lines on the (r,θ)-plane, respectively. The black point where the lines intersect gives a distance and angle.

but its slope m and intercept n. Based on that fact, the straight line $y = mx + n$ can be represented as a point (n, m) in the parameter space (n versus m graph.)

Using slope-intercept parameters could make application complicated since both parameters are unbounded: As lines get more and more vertical, the magnitudes of m and n grow toward infinity. For computational purposes, it is better to parameterize the lines in the Hough transform with two other parameters, commonly called r and θ. The parameter r represents the distance between the line and the origin, while θ is the angle of the vector from the origin to this closest point. Using this parameterization, the equation of the line can be written as:

$$y = \left(-\frac{\cos\theta}{\sin\theta}\right)x + \left(\frac{r}{\sin\theta}\right), \tag{1}$$

which can be recast into

$$r = x \cdot \cos\theta + y \cdot \sin\theta. \tag{2}$$

It is therefore possible to associate to each line of an image, a couple of real numbers (r, θ) which are unique if $\theta \in [0, \pi]$ and $r \in \mathbf{R}$, or if $\theta \in [0, 2\pi]$ and $r > 0$. The (r, θ)-plane is sometimes referred to as *Hough space*.

It is well known that an infinite number of lines can go through a single point of the plane. If that point has coordinates (x_0, y_0) in the image plane, all lines that go through it obey the following equation:

$$r(\theta) = x_0 \cdot \cos\theta + y_0 \cdot \sin\theta. \tag{3}$$

This corresponds to a sinusoidal curve in the (r, θ) plane, which is unique to that point. If the curves corresponding to two points are superimposed, the location (in the Hough space) where they cross corresponds to lines (in the original image space) that pass through both points. More generally, a set of points that form a straight line will produce sinusoids which cross at the parameters for that line. Thus, the problem of detecting colinear points can be converted to the problem of finding concurrent curves (Fig. 1).

Consider next the integration kernel

$$h(x, y, r, \theta) = I(x, y)\delta[x \cdot \cos\theta + y \cdot \sin\theta - r], \tag{4}$$

where I is the image magnitude in pattern space and δ is the Dirac delta function that is normally integrated into a sinusoidal string of accumulator bins. Typically, I is binary valued taking on unit value at points on image lines. An accumulator bin thus receives a unit count where singularities appear in h. The Hough transformation is *nonlinear*, since the integration kernel (4) can be written as

$$h(x, y, r, \theta) = I(x, y)\delta\Big[(x^2 + y^2)^{\frac{1}{2}} \times \cos\Big(\theta - \tan^{-1}\Big(\frac{y}{x}\Big)\Big) - r\Big], \tag{5}$$

which shows the nonlinear affect of (x, y) on (r, θ).

Practically, the Hough transform algorithm uses an array called accumulator to detect the existence of a line. Every pixel in an image may belong to many lines described by a set of parameters r and θ. In other words, the accumulator is defined which is an integer array $A(r, \theta)$ containing only zeroes initially. For each nonzero pixel in the image all accumulator elements corresponding to lines that contain the pixel are incremented by 1. Then a threshold is applied to distinguish lines and noise features, that is, select all pairs (r, θ) for which

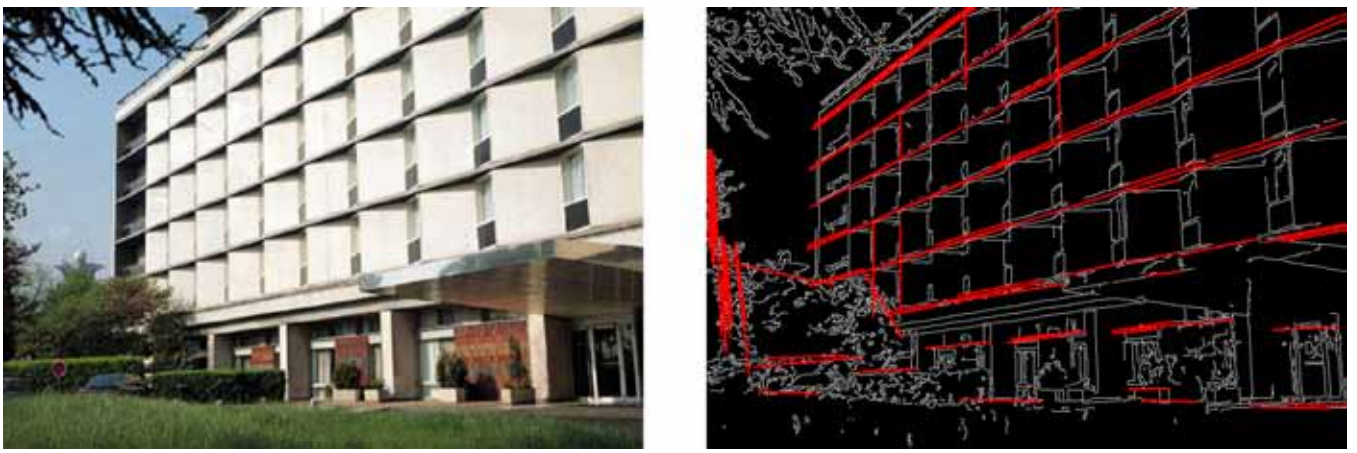

Fig. 2. Hough transform. Using the probabilistic Hough transform, many straight lines are detected from the input image, and are marked in red (right). This input image (left) is included in the OpenCV reference manual.

$A(r, \theta)$ is greater than the threshold value. All such pairs characterize detected lines. In the probabilistic (random) Hough transforms, not all input pixels are mapped to the accumulator (selecting a point randomly from the image space), and thus the runtime is decreased (Fig. 2).

Although the version of the transform described above applies only to finding straight lines, a similar transform can be used for finding any shape which can be represented by a set of parameters. A circle, for instance, can be transformed into a set of three parameters, representing its center and radius, so that the Hough space becomes three dimensional. Arbitrary ellipses and curves can also be found this way, as can any shape easily expressed as a set of parameters. For more complicated shapes, the generalized Hough transform is used, which allows a feature to vote for a particular position, orientation and/or scaling of the shape using a predefined look-up table.

2.2. *Morphological gradient*

Mathematical morphology[5] is a set-theory method of image analysis. Mathematical morphology examines the geometrical structure of an image by probing it with small patterns, called "structuring elements", of varying size and shape, just the way a blind man explores the world with his fingers or a stick. This procedure results in *nonlinear* image operators which are well-suited to exploring geometrical and topological structures. Morphological operators can be used for noise filtering, merging or splitting image regions, as well as for region boundary detection.

Two basic morphological operations are erosion, or thinning, and dilation, or thickening (Figs. 3 and 4). All operations involve an image A, called the object of interest, and a kernel element B, called the structuring element. The element B is most often a square or a circle, but could be any shape. Just like in convolution, B is a kernel or template with an anchor point.

If B_z is the spatial translation of B along the boundary of an image, then the *dilation* of object A by the structuring element B is defined by

$$A \oplus B = \{z : B_z \cap A \neq 0\}. \tag{6}$$

Equation (6) implies that every pixel is in the set $A \oplus B$, if the intersection is not null. That is, a pixel under the anchor point of B is marked "on", if at least one pixel of B is inside of A. The basic effect of dilation on binary images is to enlarge the areas of foreground pixels at their borders. The areas of foreground pixels thus grow in size, while the background holes within them shrink. Grayscale dilation brightens small dark areas, and very small dark holes might be totally removed.

Erosion of object A by structuring element B is defined by

$$A \ominus B = \{z : B_z \subseteq A\}. \tag{7}$$

That is, a pixel under the anchor of B is marked "on", if B is entirely within A. The basic effect of erosion on binary images is to remove any

[5]For more information, see "*morphology*," in the OpenCV Reference Manual or "*mathematical morphology*," in Wikipedia.

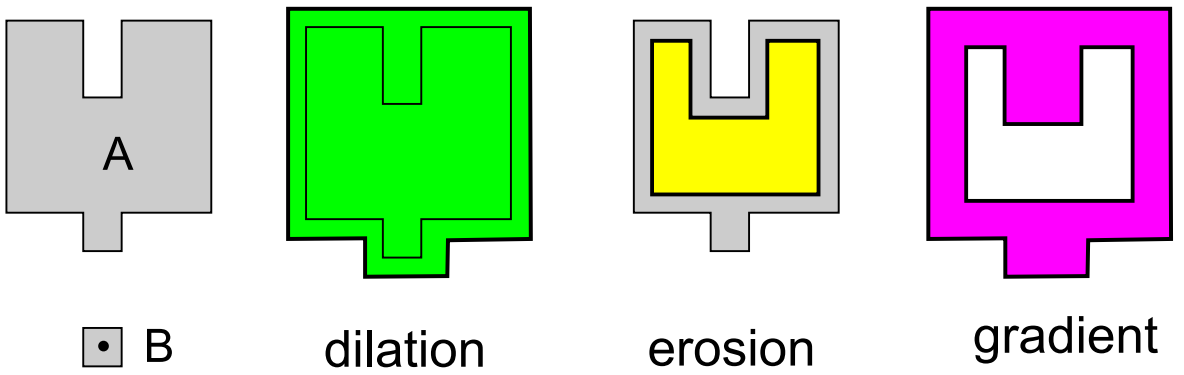

Fig. 3. Morphology operations. The symbols A and B are the object image and the kernel element, respectively. The dilation, erosion, and gradient images are colored in green, yellow and pink, respectively.

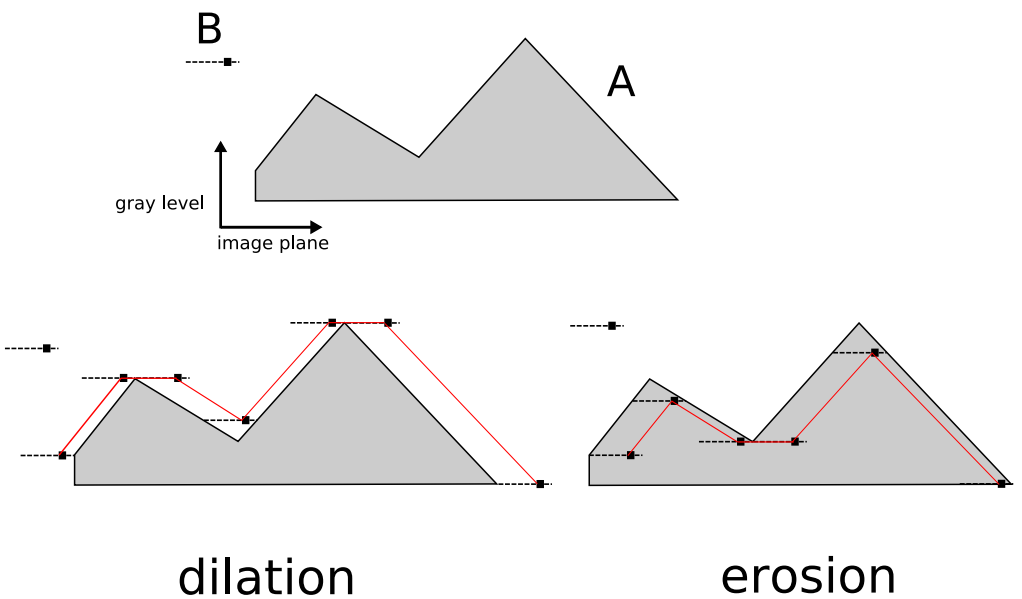

Fig. 4. Dilation and erosion of grayscale image. The symbols A and B are the object image and the flat structuring element, respectively. The dilation and erosion of a gray level image A by a flat structuring element B are shown in red lines.

foreground pixel that is not completely surrounded by other foreground pixels.

Erosion and dilation in gray levels can be done using a flat structuring element B as shown in Fig. 4. The flat structuring element B has an anchor slightly to the right of the center as shown by the dark mark on B. The dilation and erosion of a gray level image A by a flat structuring element B are shown in red lines. Grayscale erosion darkens small bright areas, and very small bright areas like noise spikes or small spurs might be totally removed.

A *morphological gradient* of object A by structuring element B is defined by

$$\text{grad}(A) = \frac{(A \oplus B) - (A \ominus B)}{2}. \qquad (8)$$

The areas with the steepest bright-to-dark or dark-to-bright transitions are highlighted using this operation (Fig. 5).

2.3. *Thresholding*

Thresholding[6] is a *nonlinear* operation which converts a gray scale image into a binary image. It is also the simplest method of image segmentation (Fig. 6). Individual pixels in a grayscale image are marked as "object" pixels if their value is greater than some threshold value (assuming an object to be brighter than the background) and as "background" pixels otherwise. Typically, an object pixel is given a value of "1" while a background

[6]For more information, see "*thresholding*," in the OpenCV Reference Manual or Wikipedia.

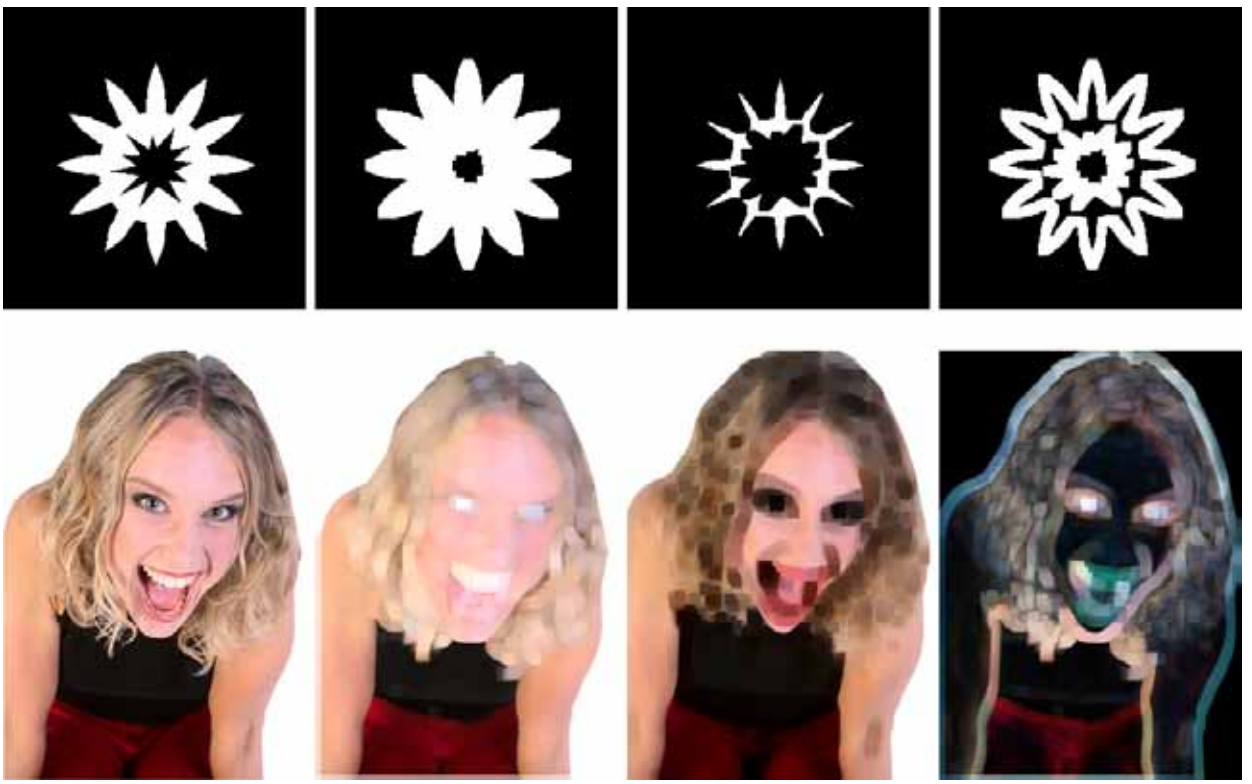

Fig. 5. Morphology operations. Input, dilation, erosion, and gradient images are illustrated from left to right. The kernel element B is a 3×3 square.

Fig. 6. Thresholding. The input image (left) is converted to the black-and-white thresholding image (center) and adaptive thresholding image (right).

pixel is given a value of "0". The key parameter in thresholding is obviously the choice of the threshold. Several different methods for choosing a threshold exist. The simplest method would be to choose the mean or median value, the rationale being that if the object pixels are brighter than the background, they should also be brighter than the average. Thresholding functions are used mainly for two purposes: masking out some pixels that do not belong to a certain range, for example, to extract blobs of certain brightness or color from the image; converting grayscale image to bilevel or black-and-white image. Usually, the resultant image is used as a mask or as a source for extracting higher-level topological information, e.g. contours, skeletons, lines, etc.

Adaptive thresholding changes the threshold dynamically over the image, whereas the conventional thresholding operator uses a global threshold for all pixels. The algorithm will consider each

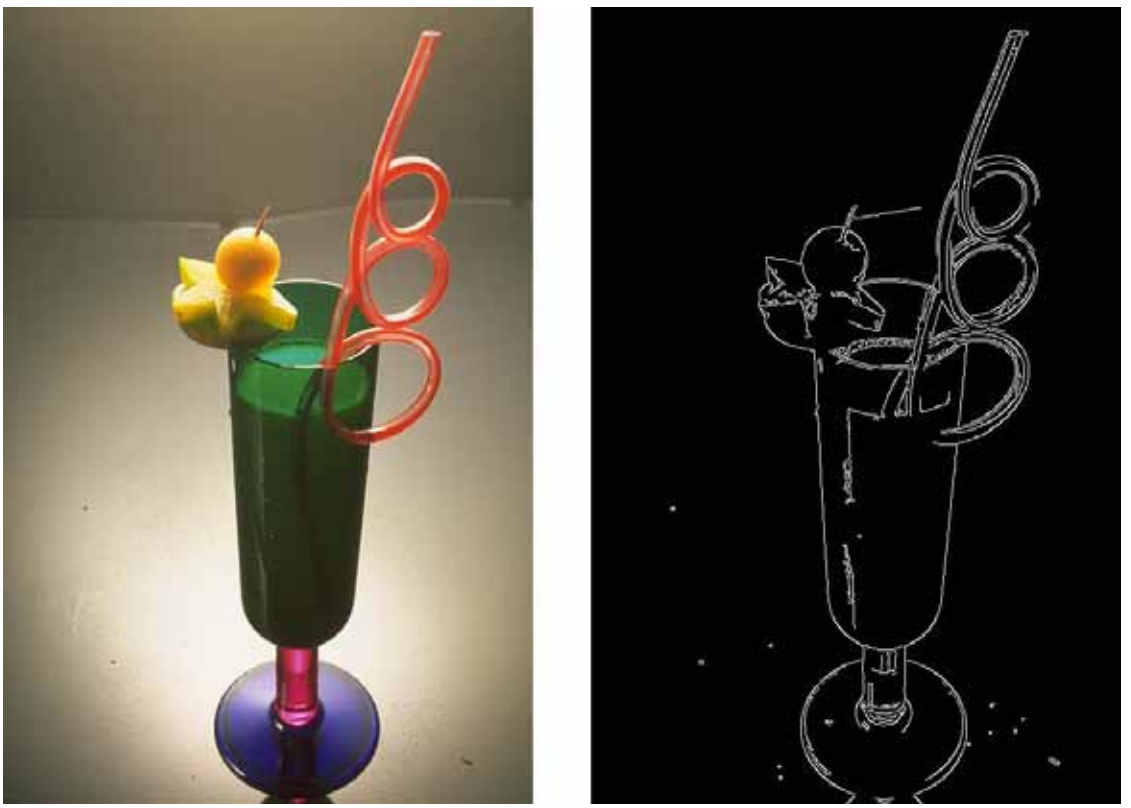

Fig. 7. Canny edge detector. The input image (left) is converted to a black and white image with edges (right).

pixel one at a time, calculate the mean of the local neighborhood and thresholds the current pixel to white if the difference between the calculated mean and the current pixel value is lower than the mean offset.

2.4. *Canny edge detector*

Edges are the boundaries separating regions with different brightness or color. Canny edge detector is a *nonlinear* operator which converts a grayscale image into a binary image where non-zero pixels mark detected edges. The *Canny edge detection algorithm* was developed by John F. Canny in 1986 [Canny, 1986] and uses a multistage algorithm to detect a wide range of edges in images. *Canny edge detector*[7] in the OpenCV takes grayscale image on input and returns bilevel image where white pixels mark detected edges (Fig. 7). We next show the outline of a simple four-stage Canny edge detector algorithm.

- ***Step 1.*** Image smoothing
  The image data is smoothed by a Gaussian filter.
- ***Step 2.*** Differentiation
  The smoothed image is differentiated with respect to the directions x and y. From the computed gradient values x and y, the magnitude and the angle of the gradient can be calculated using the hypotenuse and arctangent functions.
- ***Step 3.*** Nonmaximum suppression
  The edges can be located at the points of local maximum gradient magnitude. After the gradient has been calculated at each point of the image, a search is then carried out to determine if the gradient magnitude assumes a local maximum in the gradient direction. This will give a thin line in the output image.
- ***Step 4.*** Edge thresholding
  Canny edge detector uses the so-called "hysteresis" thresholding, which requires two thresholds: upper and lower edge values. Considering a line segment, if a value lies above the upper threshold limit it is immediately accepted. If the value lies below the low threshold it is immediately rejected. Points which lie between the two limits are accepted if they are connected to pixels which exhibit strong response. The likelihood

[7]For more information, see "*Canny edge detector*," in the OpenCV Reference Manual or Wikipedia.

of streaking is reduced drastically since line segment points must fluctuate above the upper limit and below the lower limit for streaking to occur [Canny, 1986].

2.5. *Watershed segmentation*

Watershed segmentation[8] is a way of automatically separating or cutting apart particles that touch (Fig. 8). This method can also be explained by a metaphor based on the behavior of water in a landscape. When it rains, drops of water falling in different regions will follow the landscape downhill. The water will end up at the bottom of valleys. For each valley there will be a region from which all water drains into it. In other words, each valley is associated with a catchment basin, and each point in the landscape belongs to exactly one unique basin.

The Watershed function in the OpenCV implements one of the variants of watershed, nonparametric marker-based segmentation algorithm, described in [Meyer, 1992]. Before passing the image to the function, we have to outline roughly the desired regions in the image markers with positive indices, i.e. every region is represented as one or more connected components with pixel values $1, 2, 3$, etc. Those components will be "seeds" of the future image regions. All other pixels in markers, whose relation to the outlined regions is not known and is defined by an algorithm should be set to 0's. On the output of the function, each pixel in markers is set to one of the values of the "seed" components, or to -1 at boundaries between the regions. Hence, the watershed segmentation is a *nonlinear* operator which converts a grayscale or color image into an image whose pixel values are -1 or $1, 2, 3$, etc.

2.6. *Optical flow*

Optical flow[9] is a concept which approximates the motion of objects within a visual representation (Fig. 9). Estimating the optical flow is useful in

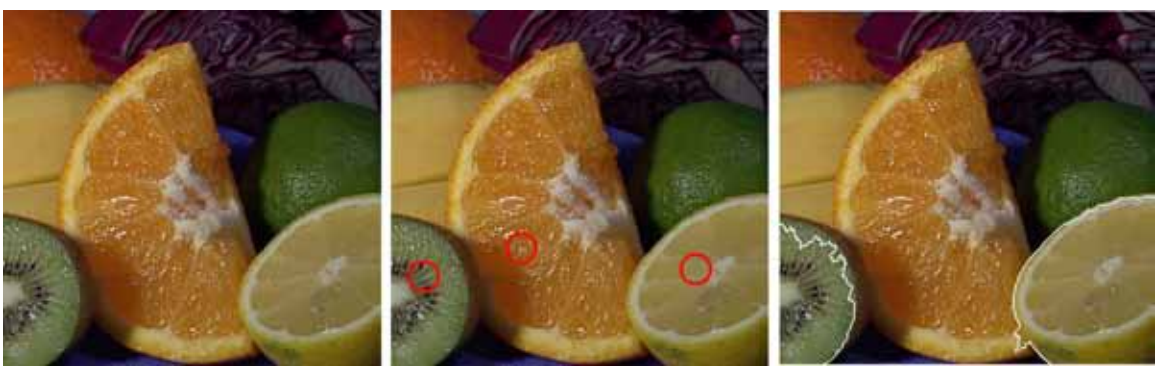

Fig. 8. Watershed (marker-based) segmentation. Observe that three regions are separated by two white boundaries. Input image, marked image (by red circles), and watershed segmentation image are illustrated from left to right. The input image is included in the OpenCV reference manual.

Fig. 9. Optical flow. The motion of image is marked by red lines (right). It is calculated for two input images (left and center), which are included in the OpenCV's web reference manual.

[8]For more information, see "*watershed*," in the OpenCV sample program or Wikipedia.
[9]For more information, see "*optical flow*," in the OpenCV Reference Manual or Wikipedia.

pattern recognition, computer vision, and other image processing applications. It is closely related to motion estimation and motion compensation. Optical flow is defined as an apparent motion of image brightness. Let $I(x, y, t)$ be the image brightness that changes in time to provide an image sequence. Two main assumptions can be made:

1. Brightness $I(x, y, t)$ smoothly depends on coordinates x, y in greater part of the image.
2. Brightness of every point of a moving or static object does not change in time.

Let some object in an image, or some point of an object, move and after time dt the object displacement is (dx, dy). Using Taylor series for brightness $I(x, y, t)$ gives the following:

$$I(x + dx, y + dy, t + dt) = I(x, y, t) + \frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt + \mathcal{H}.\mathcal{O}.\mathcal{T}., \tag{9}$$

where the symbol $\mathcal{H}.\mathcal{O}.\mathcal{T}.$ indicates higher order terms. Then, according to Assumption 2, we have

$$I(x + dx, y + dy, t + dt) = I(x, y, t), \tag{10}$$

and

$$\frac{\partial I}{\partial x}dx + \frac{\partial I}{\partial y}dy + \frac{\partial I}{\partial t}dt + \mathcal{H}.\mathcal{O}.\mathcal{T}. = 0. \tag{11}$$

Define the components of optical flow field in x and y coordinates by

$$\frac{dx}{dt} = u, \quad \frac{dy}{dt} = v, \tag{12}$$

respectively, which means the speeds of the object moving in the x and y directions. In the limit that dt tends to zero, we obtain

$$-\frac{\partial I}{\partial t} = \frac{\partial I}{\partial x}u + \frac{\partial I}{\partial y}v, \tag{13}$$

which is usually called *optical flow constraint equation*. In this equation, $I_t \triangleq \partial I/\partial t$ measures how fast the intensity is changing with time, while $I_x \triangleq \partial I/\partial x$ and $I_y \triangleq \partial I/\partial y$ are the spatial rates of change of intensity, i.e. how rapidly intensity changes on going across the picture, so all three of these quantities can be estimated for each pixel by considering the images. Assuming that the flow (u, v) is constant in a small window of size $m \times m$ with $m > 1$, which is centered at (x, y) and numbering the pixels as $1, \ldots, n$, the following set of equations can be found:

$$\begin{aligned} I_{x1}u + I_{y1}v &= -I_{t1} \\ I_{x2}u + I_{y2}v &= -I_{t2} \\ &\vdots \\ I_{xn}u + I_{yn}v &= -I_{tn} \end{aligned} \tag{14}$$

or equivalently

$$\begin{bmatrix} I_{x1} & I_{y1} \\ I_{x2} & I_{y2} \\ \vdots & \vdots \\ I_{xn} & I_{yn} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -I_{t1} \\ -I_{t2} \\ \vdots \\ -I_{tn} \end{bmatrix} \tag{15}$$

To solve the over-determined system of equations, the least squares method is used in the Lucas–Kanade optical flow estimation. Multiplying Eq. (15) with the matrix

$$\begin{bmatrix} I_{x1} & I_{x2} & \cdots & I_{xn} \\ I_{y1} & I_{y2} & \cdots & I_{yn} \end{bmatrix} \tag{16}$$

we obtain

$$\begin{bmatrix} I_{x1} & I_{x2} & \cdots & I_{xn} \\ I_{y1} & I_{y2} & \cdots & I_{yn} \end{bmatrix} \begin{bmatrix} I_{x1} & I_{y1} \\ I_{x2} & I_{y2} \\ \vdots & \vdots \\ I_{xn} & I_{yn} \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} I_{x1} & I_{x2} & \cdots & I_{xn} \\ I_{y1} & I_{y2} & \cdots & I_{yn} \end{bmatrix} \begin{bmatrix} -I_{t1} \\ -I_{t2} \\ \vdots \\ -I_{tn} \end{bmatrix} \tag{17}$$

Solving this equation for (u, v), we get

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{n} I_{xi}^2 & \sum_{i=1}^{n} I_{xi}I_{yi} \\ \sum_{i=1}^{n} I_{xi}I_{yi} & \sum_{i=1}^{n} I_{yi}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_{i=1}^{n} I_{xi}I_{ti} \\ -\sum_{i=1}^{n} I_{yi}I_{ti} \end{bmatrix}. \tag{18}$$

This means that the optical flow can be found by calculating the derivatives of the image in all three dimensions, namely I_{xi}, I_{yi}, I_{ti}. Two other kinds of optical flow functions are given in the OpenCV library, which calculate the optical flow for two images (for more details, see the OpenCV Reference Manual).

3. CNN Templates

Cellular Neural Network [Chua, 1998; Chua & Roska, 2002] is a dynamic nonlinear system defined by coupling only identical simple dynamical systems, called cells, located within a prescribed sphere of influence, such as nearest neighbors. The dynamics of a standard cellular neural network with a neighborhood of radius r are governed by a system of $n = MN$ differential equations

$$\frac{dx_{ij}}{dt} = -x_{ij} + \sum_{k,l \in N_{ij}} (a_{k,l} y_{kl} + b_{k,l} u_{kl}) + z_{ij}, \quad (i,j) \in \{1,\ldots,M\} \times \{1,\ldots,N\} \tag{19}$$

where N_{ij} denotes the r-neighborhood of cell C_{ij}, and a_{kl}, b_{kl}, and z_{ij} denote the feedback, control, and threshold template parameters, respectively. The matrices $A = [a_{kl}]$ and $B = [b_{kl}]$ are referred to as the feedback template A and the feedforward (input) template B, respectively. The output y_{ij} and the state x_{ij} of each cell are usually related via the piecewise-linear saturation function

$$y_{ij} = f(x_{ij}) = \frac{1}{2}(|x_{ij} + 1| - |x_{ij} - 1|). \tag{20}$$

If we restrict the neighborhood radius of every cell to 1 and assume that z_{ij} is the same for the whole network, the template $\{A, B, z\}$ is fully specified by 19 parameters, which are the elements of two 3×3 matrices A and B, namely

$$A = \begin{bmatrix} a_{-1,-1} & a_{-1,0} & a_{-1,1} \\ a_{0,-1} & a_{0,0} & a_{0,1} \\ a_{1,-1} & a_{1,0} & a_{1,1} \end{bmatrix}, \quad B = \begin{bmatrix} b_{-1,-1} & b_{-1,0} & b_{-1,1} \\ b_{0,-1} & b_{0,0} & b_{0,1} \\ b_{1,-1} & b_{1,0} & b_{1,1} \end{bmatrix}, \tag{21}$$

and a real number z. The following palette is usually applied to the output y_{ij}:

output y_{ij}	color
$y_{ij} = 1 \Rightarrow$	*black*
$-1 < y_{ij} < 1 \Rightarrow$	*gray scale*
$y_{ij} = -1 \Rightarrow$	*white*

(22)

which is the *reverse* of the OpenCV's palette. In color image processing, the same template and palette are usually applied to each color component.

We next show some typical CNNs which are similar to OpenCV programming functions, and they can be applied to the imitation of visual illusions.[10]

3.1. *Morphological CNNs*

Dilation CNN [Chua, 1998; Chua & Roska, 2002; Itoh & Chua, 2003] grows a layer of pixels around objects in a binary input image in a way determined by a structuring element coded by the B template (Fig. 10). A dilation CNN template with 3×3 structuring element is given by

$$A = \begin{array}{|c|c|c|}\hline 0&0&0\\\hline 0&2&0\\\hline 0&0&0\\\hline\end{array}, \quad B = \begin{array}{|c|c|c|}\hline 1&1&1\\\hline 1&1&1\\\hline 1&1&1\\\hline\end{array}, \quad z = \boxed{9}, \tag{23}$$

where the initial condition is given by $x_{ij}(0) = u_{ij}$. The dilation CNN simply adds onto the input image one layer of black pixels on the perimeter of all black objects.

Erosion CNN [Chua, 1998; Chua & Roska, 2002; Itoh & Chua, 2003] peels off all boundary pixels of a binary input image (Fig. 10). Pixels are considered to belong to the object boundary if the structuring element, coded by a B template, does not fit completely within the object. An erosion CNN template with a 3×3 structuring element is given by

$$A = \begin{array}{|c|c|c|}\hline 0&0&0\\\hline 0&2&0\\\hline 0&0&0\\\hline\end{array}, \quad B = \begin{array}{|c|c|c|}\hline 1&1&1\\\hline 1&1&1\\\hline 1&1&1\\\hline\end{array}, \quad z = \boxed{-9}, \tag{24}$$

where the initial condition is given by $x_{ij}(0) = u_{ij}$. The erosion CNN simply peels off from the input image one layer of black pixels on the perimeter of all black objects.

Gradient CNN highlights the areas with the steepest white-to-black or black-to-white transitions, namely, extracts edges (Fig. 10). A gradient CNN template is given by

$$A = \begin{array}{|c|c|c|}\hline 0&0&0\\\hline 0&1&0\\\hline 0&0&0\\\hline\end{array}, \quad B = \begin{array}{|c|c|c|}\hline 0&0&0\\\hline 0&-1&0\\\hline 0&0&0\\\hline\end{array}, \quad z = \boxed{z_{ij}}, \tag{25}$$

which has a nonuniform threshold z_{ij}. The input and threshold images of the gradient CNN are the output images of the erosion and dilation CNNs, respectively.

[10] All numerical simulations of the CNNs are performed by using the simulator "CELL" (Circuit Lab. Department of Electric Engineering, University of Rome "Tor Vergata").

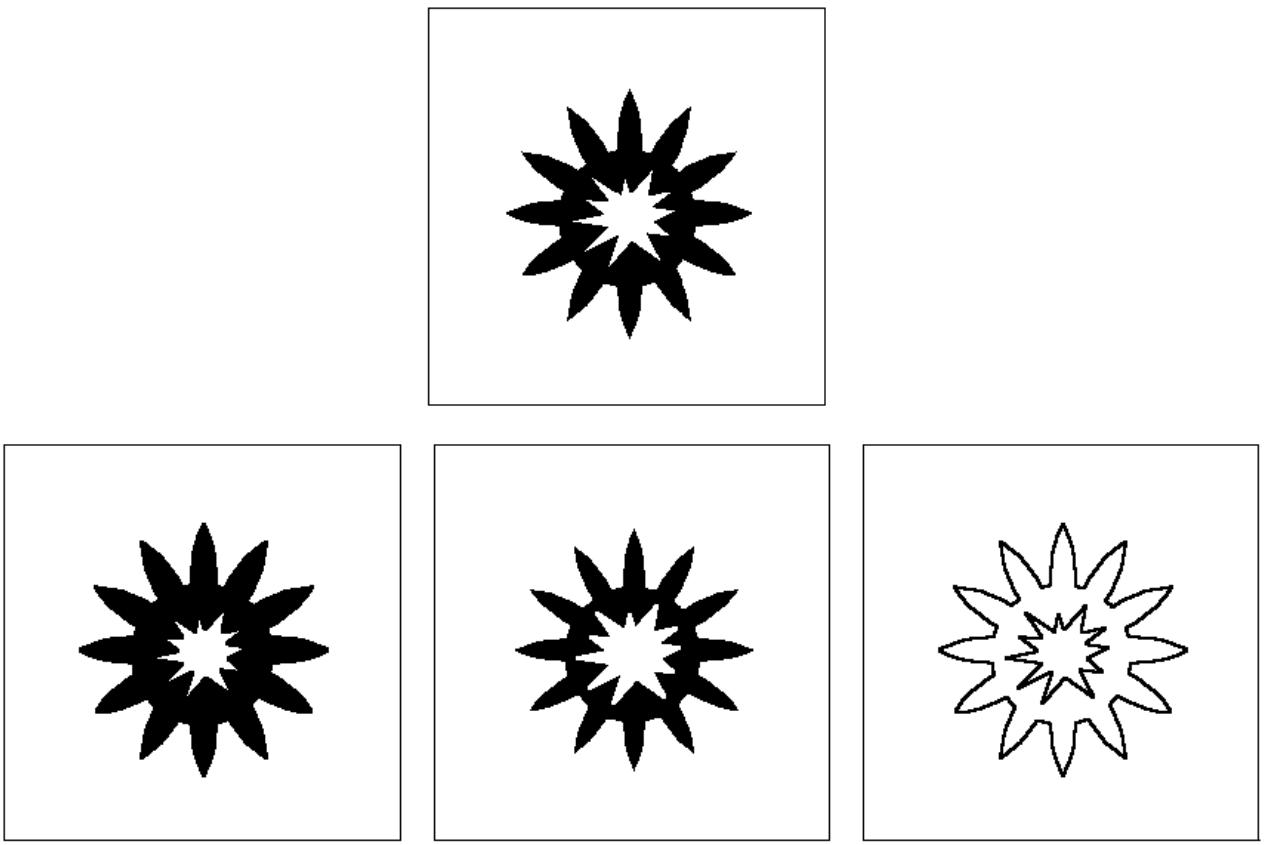

Fig. 10. Morphology CNNs. Input (top), dilation (bottom left), erosion (bottom center), and gradient images (bottom right) are illustrated. The structuring element B is given by a 3×3 pixel square.

3.2. *Thresholding CNN*

Thresholding CNN converts a grayscale input image into a binary image (Fig. 11). A thresholding CNN template is given by

$$A = \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 2 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array}, \quad B = \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array}, \quad z = \boxed{-z^*}, \qquad (26)$$

where the initial condition is given by $x_{ij} = 0$. Each pixel in an input image is converted into black (resp. white) if it has a gray scale intensity greater (resp. less) than a prescribed threshold z^*. This template is similar to that of the threshold CNN [Chua, 1998; Chua & Roska, 2002].

3.3. *Edge detection CNN*

Edge detection CNN [Chua, 1998; Chua & Roska, 2002; Itoh & Chua, 2003] extracts edges of objects in a binary input image where each black pixel with at least one white nearest neighbor is defined to be an edge cell (Fig. 12). An edge detection CNN template is given by

$$A = \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & 2 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array}, \quad B = \begin{array}{|c|c|c|}\hline -1 & -1 & -1 \\ \hline -1 & 8 & -1 \\ \hline -1 & -1 & -1 \\ \hline \end{array}, \quad z = \boxed{-2}, \qquad (27)$$

where the initial condition is given by $x_{ij}(0) = u_{ij}$. The edge detection CNN can also extract edges from a grayscale image by adjusting the threshold parameter z (Fig. 13).

3.4. *Watershed segmentation CNN*

Watershed segmentation CNN fills the marked area of a binary input image (Fig. 14). A watershed segmentation template is given by

$$A = \begin{array}{|c|c|c|}\hline 0 & 1 & 0 \\ \hline 1 & 4 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array}, \quad B = \begin{array}{|c|c|c|}\hline 0 & 0 & 0 \\ \hline 0 & -6 & 0 \\ \hline 0 & 0 & 0 \\ \hline \end{array}, \quad z = \boxed{0}. \qquad (28)$$

A small marked patch of black pixels on the initial state of x_{ij} spreads and ends up at the boundary. This template is equivalent to that of the face-vase illusion CNN [Chua, 1998; Chua & Roska, 2002; Itoh & Chua, 2003].

Fig. 11. Thresholding CNN. The woman's grayscale image (left) is converted to the black-and-white threshold image (right). The threshold z^* is set to 0.1.

Fig. 12. Edge detection CNN. The threshold image (left) is converted to the edge image (right).

3.5. *Shift translation CNN*

Shift translation CNN moves a grayscale or color image by one-pixel to one of eight possible directions (north, northeast, east, southeast, south, southwest, west, northwest) (Fig. 15). For example, an east shift translation CNN template is given by

$$A = \begin{array}{|c|c|c|}\hline 0&0&0\\\hline 0&0&0\\\hline 0&0&0\\\hline\end{array}, \quad B = \begin{array}{|c|c|c|}\hline 0&0&0\\\hline 1&0&0\\\hline 0&0&0\\\hline\end{array}, \quad z = \boxed{0}, \quad (29)$$

where the initial state of x_{ij} are set to zero or an input image, and an input image is given by a grayscale or color image. By multiple operations of shift motion CNN, an input image can be moved to any direction and by any pixels. This template is similar to that of the translation CNN [Chua, 1998; Chua & Roska, 2002].

3.6. *Shift motion CNN*

Shift motion CNN moves a color image to one of eight possible directions (north, northeast, east, southeast, south, southwest, west, northwest) *continuously* if each color component consists of a binary image (Fig. 16). For example, an east shift motion CNN template is given by

$$A = \begin{array}{|c|c|c|}\hline 0&0&0\\\hline 1&1&0\\\hline 0&0&0\\\hline\end{array}, \quad B = \begin{array}{|c|c|c|}\hline 0&0&0\\\hline 0&0&0\\\hline 0&0&0\\\hline\end{array}, \quad z = \boxed{0}, \quad (30)$$

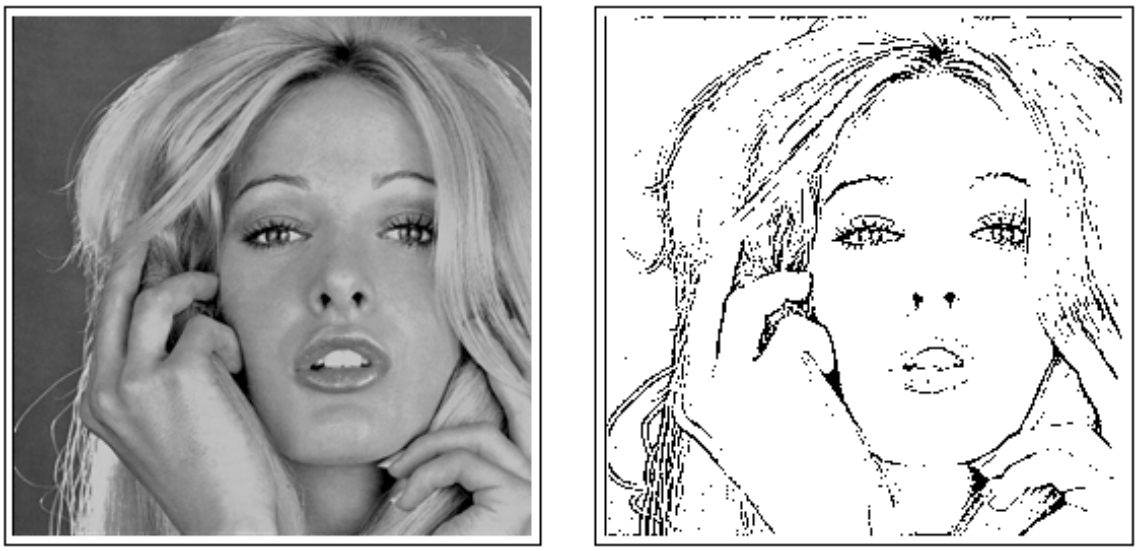

Fig. 13. Edge detection CNN. The woman's grayscale image (left) is converted to the edge image (right). The threshold z^* is set to -0.8.

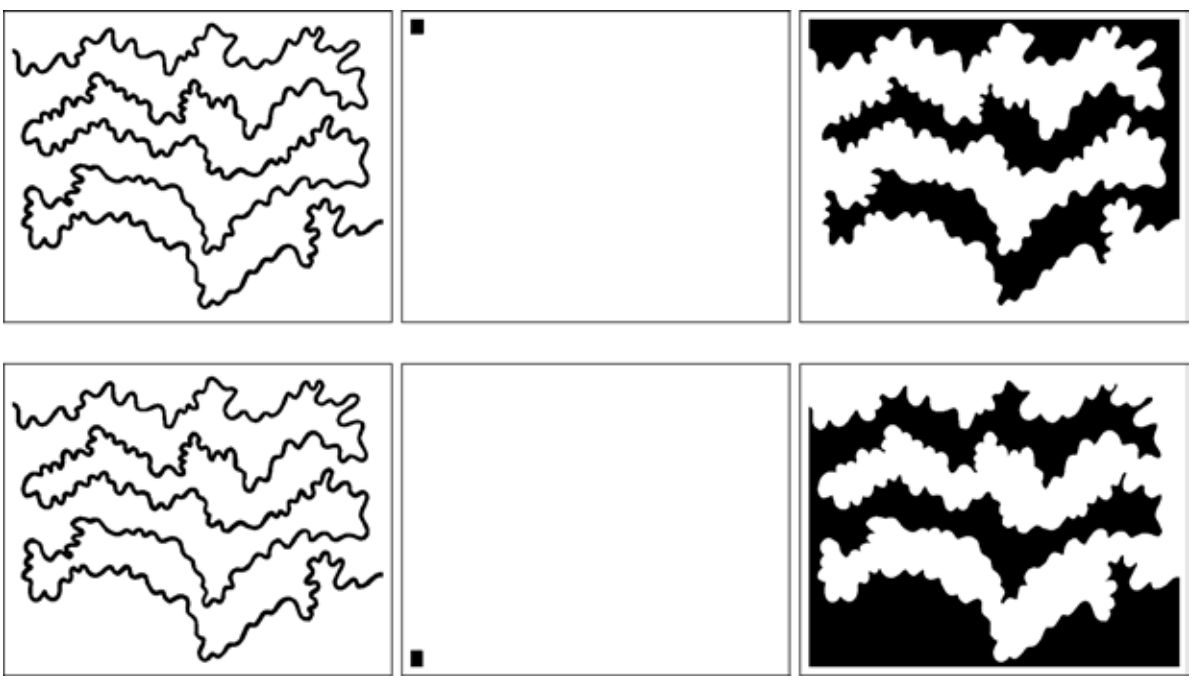

Fig. 14. Watershed segmentation CNN. The marked contiguous area of a binary input image is filled. Input image, marked image, and watershed segmentation image are illustrated from left to right.

where the initial state of x_{ij} is given by an input color image. By this template, an input image can be moved to right (east) continuously.

3.7. *Stop motion animation via CNN*

Stop motion (*or frame-by-frame*) *animation* is an animation technique which makes a physically manipulated object appear to move. If an image is displayed on the computer screen then quickly replaced by a new image that is similar to the previous image, but shifted slightly, then an illusory movement is created. Thus, the shift translation CNN can generate stop motion animations, since the given image converged to the shifted image. Furthermore, the shift translation and shift motion CNNs suggest the possibility of high speed CNN animation, which creates moving images via the

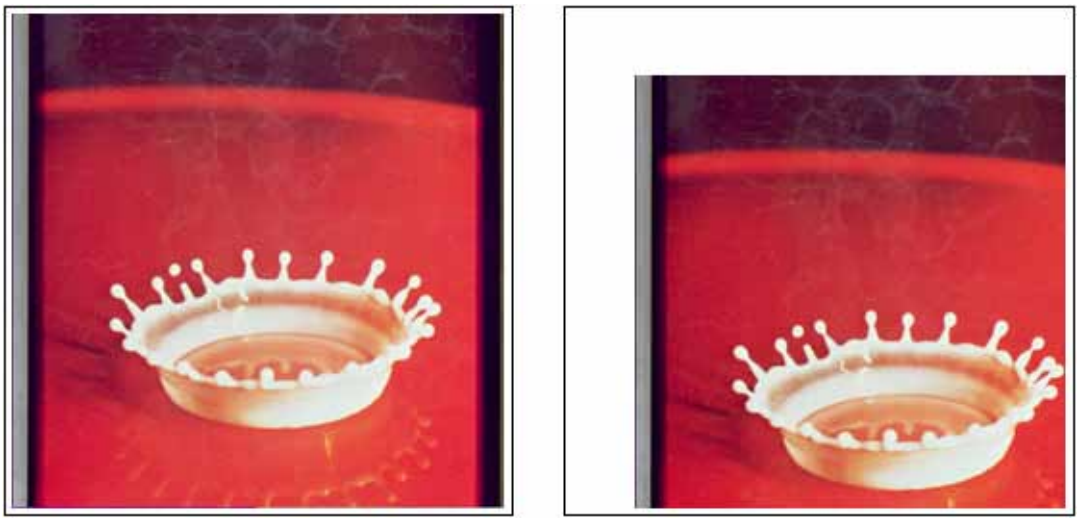

Fig. 15. Shift translation CNN. The milkdrop image (left) is moved to the southwest direction (right).

use of CNN templates and CNN universal chips. They also suggest the possibility of video data compression by sending only a few parameters of CNN templates in place of huge video images.

3.8. *Marker CNN*

Marker CNN adds white (or black) markers to a color input image (Fig. 17). For example, white marker templates are given by

$$A = \begin{array}{|c|c|c|}\hline 0&0&0\\\hline 0&1&0\\\hline 0&0&0\\\hline\end{array},\quad B = \begin{array}{|c|c|c|}\hline 0&0&0\\\hline 0&-1&0\\\hline 0&0&0\\\hline\end{array},\quad z = \boxed{-1}, \tag{31}$$

or

$$A = \begin{array}{|c|c|c|}\hline 0&0&0\\\hline 0&1&0\\\hline 0&0&0\\\hline\end{array},\quad B = \begin{array}{|c|c|c|}\hline 0&0&0\\\hline 0&1&0\\\hline 0&0&0\\\hline\end{array},\quad z = \boxed{-1}, \tag{32}$$

where the marker image is given by a binary input image u_{ij} and the initial state of x_{ij} is given by a color image. Equations (31) (resp. (32)) add white markers using the black objects (resp. white objects) of a binary input image. Red, green and blue marker CNNs are obtained by applying the marker CNN into red, green and blue color components, respectively.

3.9. *Negative image CNN*

A positive image is a normal image. A negative image is a tonal inversion of a positive image, in which light areas appear dark and vice versa. A negative color image is additionally color reversed, with red areas appearing cyan, green areas appearing magenta, and blue areas appearing yellow.

Negative image CNN transforms a positive image into a negative image (Fig. 18), whose template is given by

$$A = \begin{array}{|c|c|c|}\hline 0&0&0\\\hline 0&0&0\\\hline 0&0&0\\\hline\end{array},\quad B = \begin{array}{|c|c|c|}\hline 0&0&0\\\hline 0&-1&0\\\hline 0&0&0\\\hline\end{array},\quad z = \boxed{0}. \tag{33}$$

Since the CNN uses the reverse palette, we need the negative image CNN to imitate visual illusions.

3.10. *Painting CNN*

Painting CNN colors black objects (resp. white objects) in a binary input image with a specified color (resp. black color) (Fig. 19). The CNN templates of red, green and blue color components are given by

$$A_r = \begin{array}{|c|c|c|}\hline 0&0&0\\\hline 0&0&0\\\hline 0&0&0\\\hline\end{array},\quad B_r = \begin{array}{|c|c|c|}\hline 0&0&0\\\hline 0&-1&0\\\hline 0&0&0\\\hline\end{array},\quad z_r = \boxed{1+r}, \tag{34}$$

$$A_g = \begin{array}{|c|c|c|}\hline 0&0&0\\\hline 0&0&0\\\hline 0&0&0\\\hline\end{array},\quad B_g = \begin{array}{|c|c|c|}\hline 0&0&0\\\hline 0&-1&0\\\hline 0&0&0\\\hline\end{array},\quad z_g = \boxed{1+g}, \tag{35}$$

and

$$A_b = \begin{array}{|c|c|c|}\hline 0&0&0\\\hline 0&0&0\\\hline 0&0&0\\\hline\end{array},\quad B_b = \begin{array}{|c|c|c|}\hline 0&0&0\\\hline 0&-1&0\\\hline 0&0&0\\\hline\end{array},\quad z_b = \boxed{1+b}, \tag{36}$$

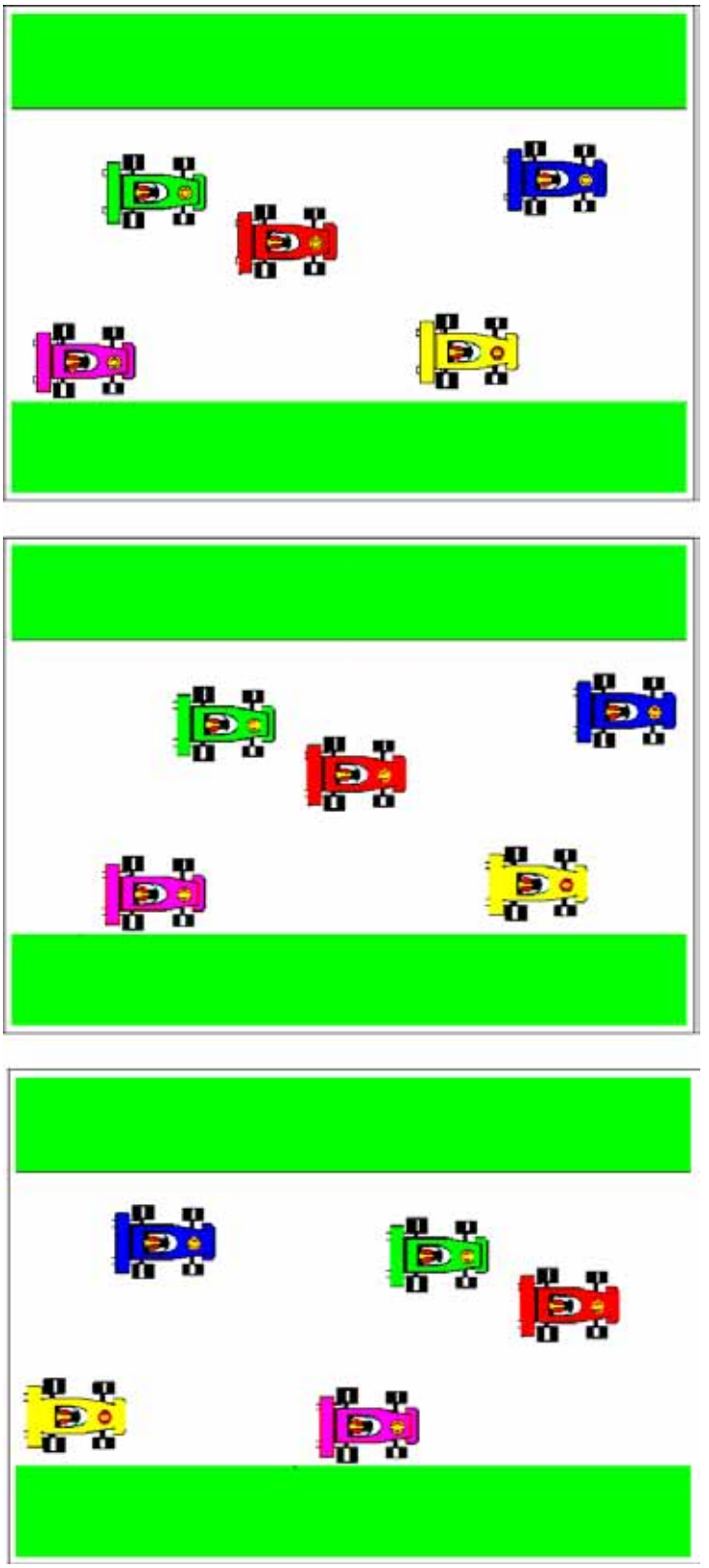

Fig. 16. Shift motion CNN with a periodic boundary. Five cars move to right ($t = 0$, 130, 520).

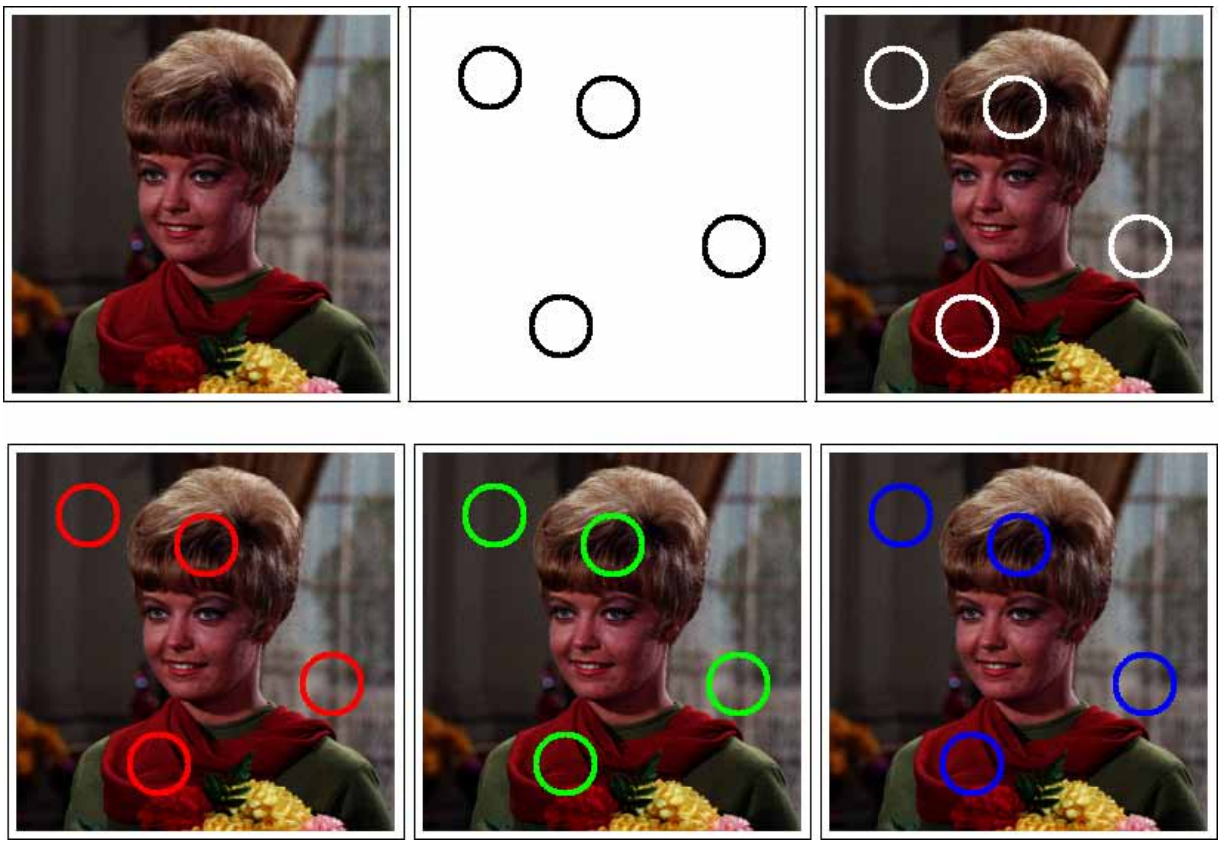

Fig. 17. Marker CNN. White circles are added to the woman's input image. Input images, initial states, and output images of these CNNs are illustrated from left to right (top). Red, green and blue circles are added to the woman's input image (bottom).

Fig. 18. Negative image CNN. The positive image (left) of parrots is transformed into the negative image (right).

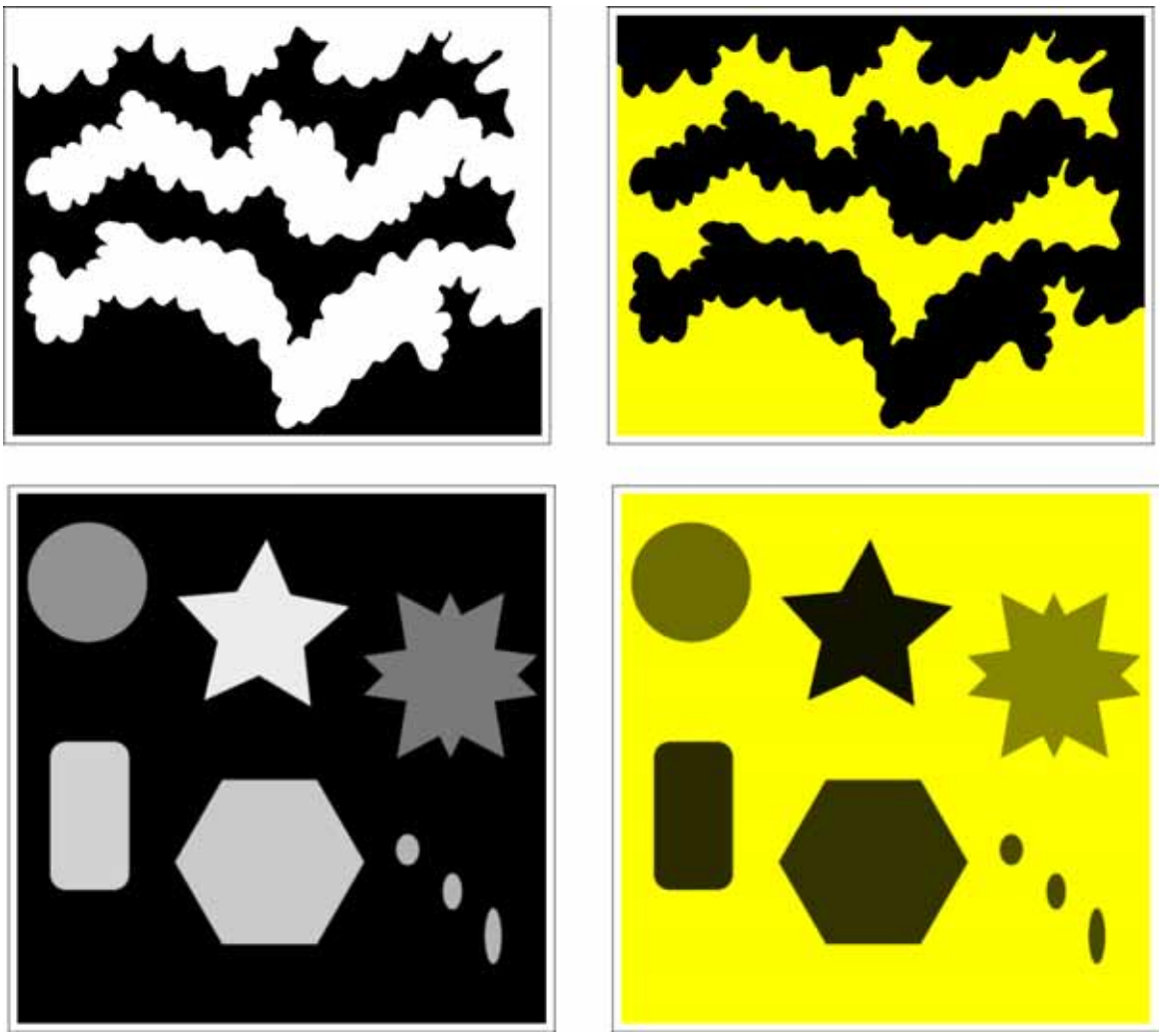

Fig. 19. Painting CNN. Black objects in a binary image (top) and a grayscale image (bottom) are painted with yellow.

respectively, where $-1 \leq r, g, b \leq 1$ and Red, Green, and Blue color values (RGB) are given by

$$(\mathrm{R}, \mathrm{G}, \mathrm{B}) = \left(-\frac{255}{2}(r-1),\ -\frac{255}{2}(g-1),\ -\frac{255}{2}(b-1)\right), \tag{37}$$

or equivalently

$$(r, g, b) = \left(\frac{255-2\mathrm{R}}{255},\ \frac{255-2\mathrm{G}}{255},\ \frac{255-2\mathrm{B}}{255}\right). \tag{38}$$

4. Visual Illusion

Visual (optical) illusion is characterized by visually perceived images that are deceptive or misleading. The information gathered by the eye is processed by the brain to give a percept that does not tally with a physical measurement of the stimulus source. We introduce some visual illusions using the references: "*optical illusion*" in Wikipedia, [Kitaoka, 2007], and [Oliva *et al.*, 2006].

4.1. *Ehrenstein illusion*

Ehrenstein illusion is a visual illusion studied by the German psychologist Walter Ehrenstein. A series of radial lines whose inward-pointing end create an illusory circle that appears to be brighter than the background (Fig. 20). A similar effect is obtained in the Kanizsa triangle illusion.

4.2. *Neon color spreading illusion*

Neon Color Spreading illusion is characterized by neonlike color spreading into a homogeneous background. If each inner endpoint of the black lines is extended with a neon colored line, a bright illusory disk is perceived. Any gap destroys the illusion (Fig. 21).

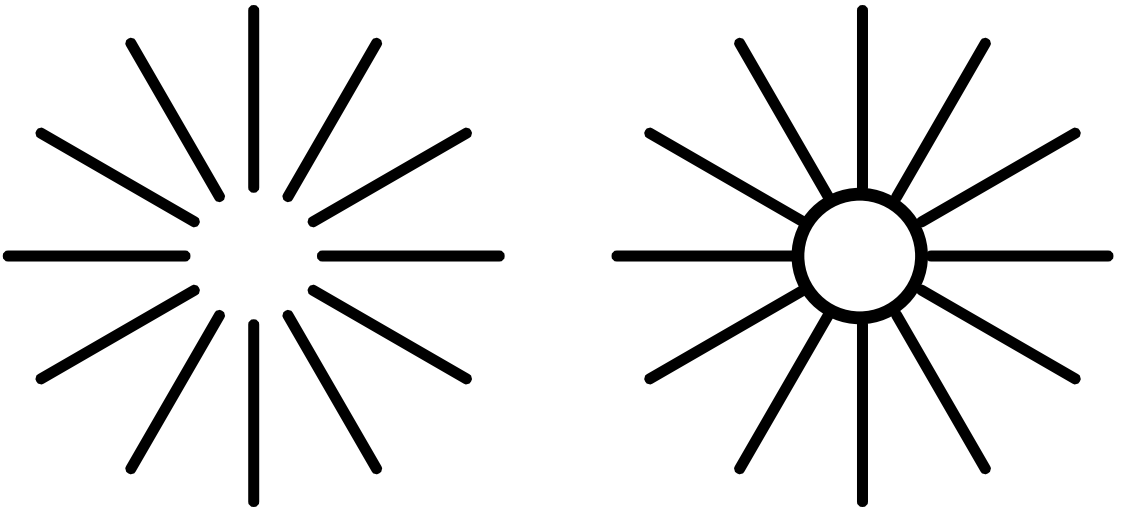

Fig. 20. Ehrenstein illusion. The ends of the dark segments produce the illusion of a circle. The apparent figure has the same color as the background, but appears brighter. Adding a circle (right) destroys the illusion of a bright central disk.

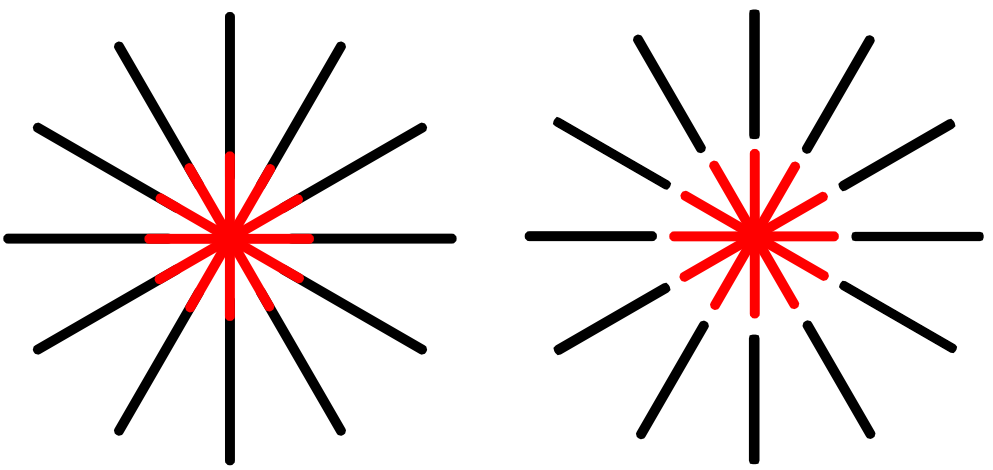

Fig. 21. Neon color spreading. If the inner endpoints of the black lines are continued with neon colored lines, a bright illusory disk is perceived. Any gap destroys the illusion.

4.3. *Kanizsa illusion*

Kanizsa illusion is a visual illusion first described by the Italian psychologist Gaetano Kanizsa in 1955. An equilateral triangle is perceived, but in fact none is drawn. This effect is known as a subjective or illusory contour. Also, the nonexistent triangle appears to be brighter than the surrounding area, but in fact it has the same brightness as the background (Fig. 22).

4.4. *Watercolor illusion*

Watercolor illusion is a phenomenon that can be observed when a figure is defined by a contour consisting of a pair of parallel lines on white background — a dark line, and a lighter colored line on the inside. The interior of the figure then appears slightly tinted with the hue of the lighter colored line (Fig. 23).

4.5. *Fraser illusion and shifted edges illusion*

Fraser illusion is a visual illusion named after the British psychologist James Fraser, who first studied the illusion in 1908. When slightly tilted line segments are aligned horizontally, the whole array appears to tilt toward the tilt of the line elements (Fig. 24).

Shifted edges illusion [Kitaoka, 2007] is a phenomenon where each row appears to tilt even

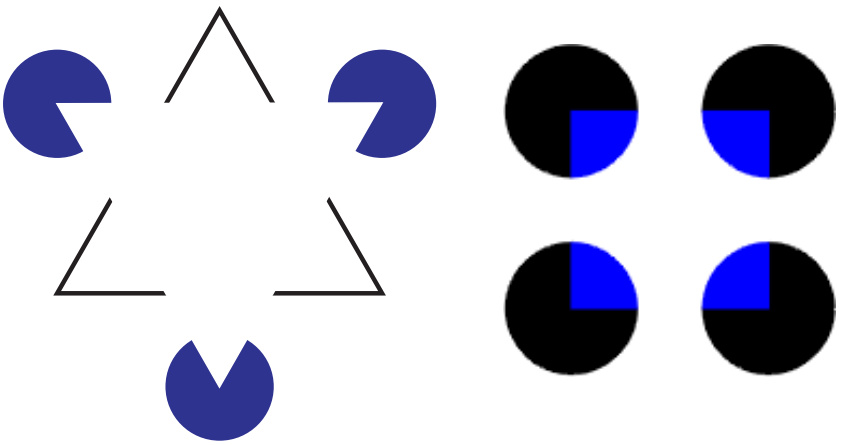

Fig. 22. Kanizsa illusion. In the left figure a white equilateral triangle is perceived, but in fact none is drawn (left). This effect is known as a subjective or illusory contour. Also, the nonexistent white triangle appears to be brighter than the surrounding area, but in fact it has the same brightness as the background. In the case of Varin's figure (right), a blue square appears in front of four black circles.

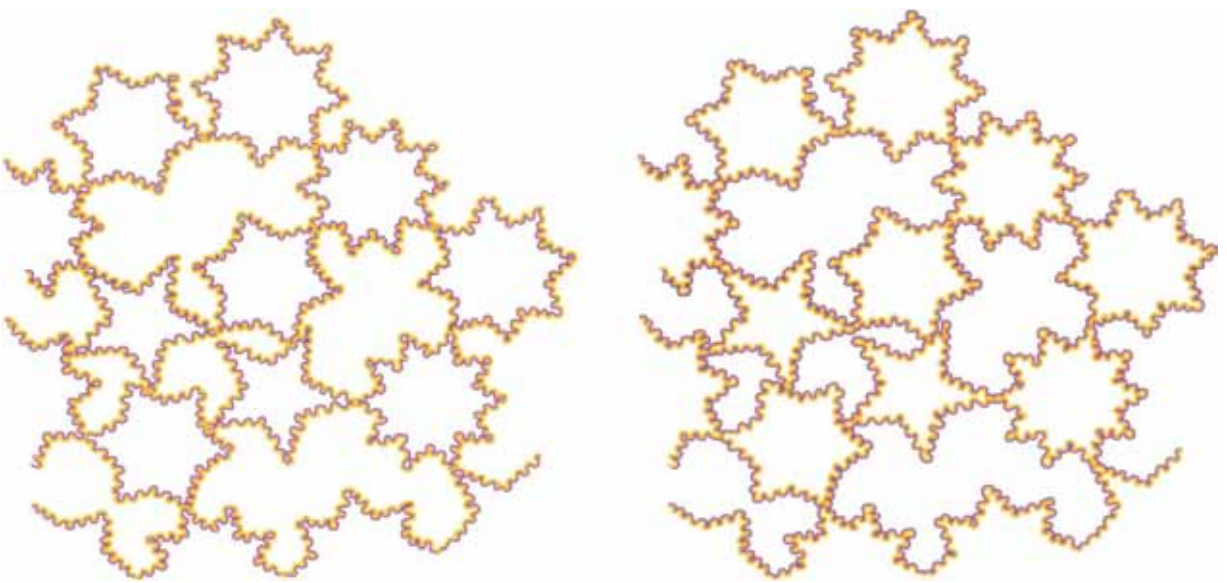

Fig. 23. Watercolor illusion [Pinna & Grossberg, 2005]. Purple undulated contours flanked by orange edges are perceived as undefined irregular curved shapes with a plain volumetric effect evenly colored by a light veil of orange tint spreading from the orange edges (left). When purple and orange lines are reversed, stars with a different number of points are now perceived (right).

though the shifted rectangles are horizontally aligned (Fig. 24).[11]

4.6. *Anomalous motion illusion*

Anomalous motion illusion [Kitaoka, 2007] is an illusion in which if you look around the static image it will appear to move, or part of a figure will appear to move in a direction different from the rest (Fig. 25).[12] These motion illusions all work differently for different people, and it often works best if the image is larger, and if you let your eye jump from one position to the next.

4.7. *Glare effect illusion*

Glare effect illusion is an illusion in which a region appears self-luminous when it is surrounded

[11]See also Kitaoka's illusion gallery: http://www.psy.ritsumei.ac.jp/~akitaoka/
[12]See also Kitaoka's illusion gallery: http://www.psy.ritsumei.ac.jp/~akitaoka/

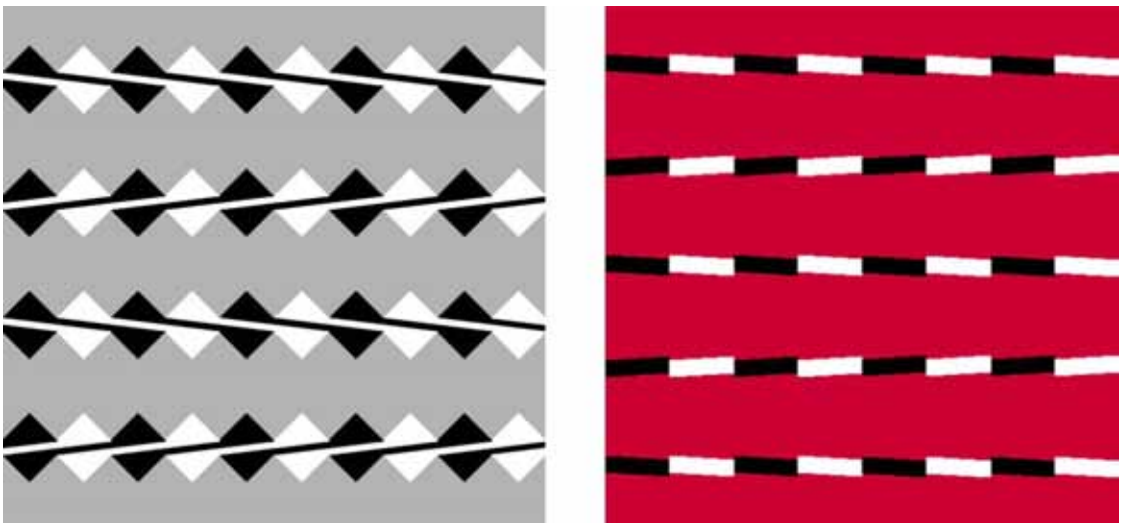

Fig. 24. Fraser illusion (left) and shifted edges illusion (right) [Kitaoka, 2007]. When slightly tilted line segments are aligned horizontally, the whole array appears to tilt toward the tilt of the line elements (left). Each row is aligned horizontally but appears to tilt counterclockwise or clockwise alternately (right).

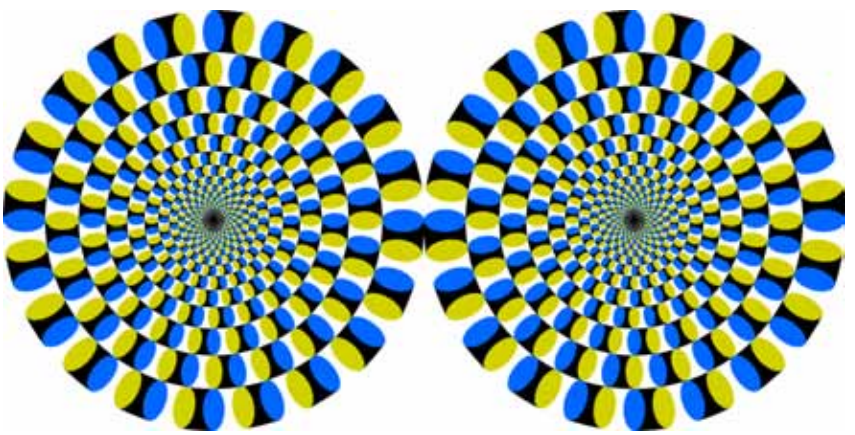

Fig. 25. Anomalous Motion Illusion [Kitaoka, 2007]. The left disk appears to rotate counterclockwise while the right one clockwise. *Anomalous motion illusions might make sensitive observers dizzy or sick. If you feel dizzy, you should leave this page immediately.*

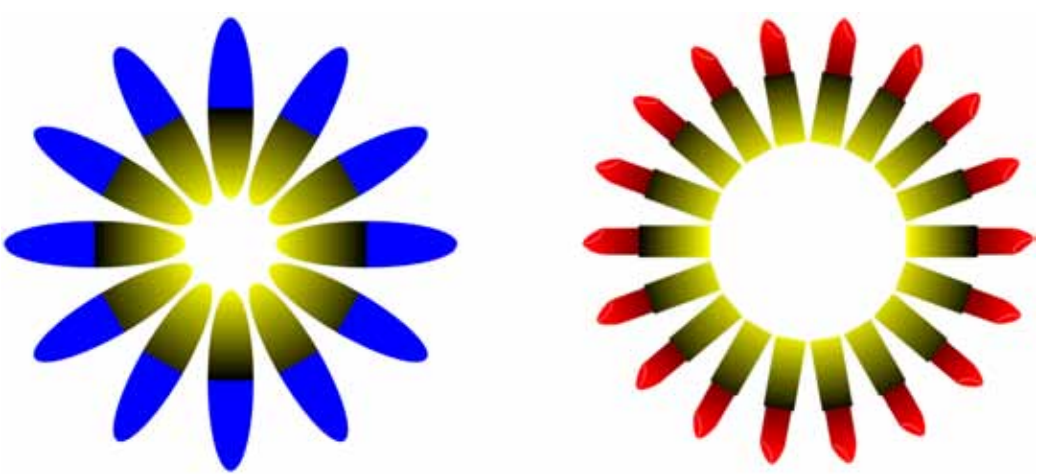

Fig. 26. Glare effect illusion [Kitaoka, 2007]. The white targets surrounded by luminance gradients appear self-luminous. The flower petals (left) and the ring of lipsticks (right) appear to glare.

Fig. 27. Hybrid image illusion [Oliva *et al.*, 2006]. Hybrid images change interpretation as a function of viewing distance. They combine the low-spatial frequencies of one picture with the high spatial frequencies of another picture producing an image with an interpretation that changes with viewing distance. An angry man or a thoughtful woman (left). On a close-up view of the left figure, we can see a stern woman, but if we step away from the picture, we will see the face of an angry man. We can switch the percept by watching the picture from a few meters. The house of the right figure is under construction. When we view the image at a short distance, the house is seen under construction, but if we step away from the picture, we will see its final state.

by gradients decreasing in luminance with distance from the region. For example, a white target surrounded by luminance gradients appears self-luminous, and the image appears to expand [Kitaoka, 2007] (Fig. 26).

4.8. *Hybrid image illusion*

Hybrid images [Oliva *et al.*, 2006] change interpretation as a function of viewing distance (Fig. 27).[13] Hybrids combine the low-spatial frequencies of one picture with the high spatial frequencies of another picture producing an image with an interpretation that changes with viewing distance. Hybrid images are based on the multiscale processing of images by the human visual system.

5. Imitation of visual illusions

Visual illusions seem to be caused by image processing occurring in the brain. In this section, we imitate the mechanism of Ehrenstein illusion, neon color spreading illusion, watercolor illusion, Kanizsa illusion, shifted edges illusion, and hybrid image illusion using OpenCV's image processing functions and CNN templates. Note that some illusion images have been simplified or deformed in order to enable their recognition by the OpenCV, and the CNN.

5.1. *Ehrenstein illusion*

1. The OpenCV can imitate the Ehrenstein illusion by using the programming functions *cvHoughCircles* and *cvCircle*. The function cvHoughCircles finds circles in a grayscale image[14] using the Hough transform. A detected circle can be transformed into a set of three parameters, representing its center and radius. The function cvCircle can draw a simple or filled circle with given center and radius. In order to imitate the mechanism of Ehrenstein illusion, we assume that the "*OpenCV's eye*" constructs a multilayered image[15] from the detected objects, and emphasizes a top layer object by brightening it.

[13]See also the gallery of the Computational Visual Cognition Laboratory: http://cvcl.mit.edu/hybridimage.htm

[14]The function *cvCVtColor* can convert an input image from one color space to a grayscale image.

[15]A layer is an individual level of an image. Think of it as a transparent sheet. A multilayered image is formed by stacking multiple images together. Layer can be added, deleted, and pixels may be blended in a variety of ways.

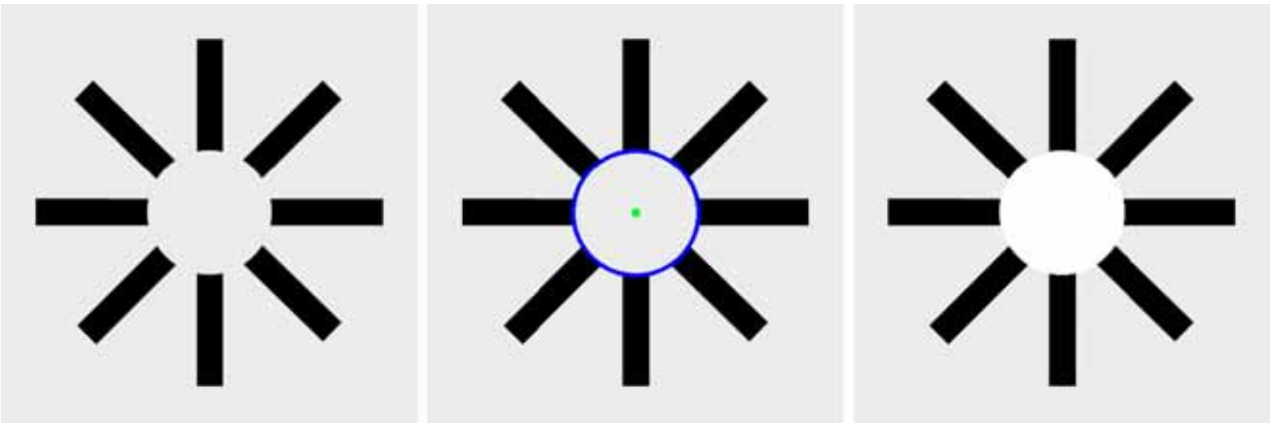

Fig. 28. Ehrenstein illusion. The cvHoughCircles detect an illusory circle from the illusory image (left). Its center and circle are painted in green and blue, respectively (center). The cvCircle fills this circle with a whiter color than the background (right).

In our computer simulations, the cvHoughCircles detect an illusory circle from a series of radial lines, and the cvCircle fills the top layer circle in a brighter color than the background (Figs. 28 and 29).

In the case of the Ehrenstein illusion with a colored ring (Fig. 30), consider the multilayered image of Fig. 31. In our computer simulations, the cvHoughCircles detects two illusory circles from a series of radial lines. By the above assumption, the cvCircle fills the circle on the top-layer in a brighter color than the background (Fig. 30).

2. The CNN can also imitate the Ehrenstein illusion with the help of OpenCV (Figs. 32–35). The *watershed segmentation* CNN fills the circle which the cvHoughCircles found in the grayscale image. Then, the *marker* CNN adds an illusory white disk to the Ehrenstein illusion image. Thus, the illusory image can be realized by the equilibrium state of the *marker* CNN. Note that there are several ways to imitate the illusion as shown in Figs. 32–35.

   In the case of the Ehrenstein illusion with a colored ring, the *watershed segmentation* CNN fills the inner circle which the cvHoughCircles found in the illusory image. The negative image CNN transforms a black disk into a white disk. Then, the *marker* CNN adds an illusory white disk to the Ehrenstein illusion image. Thus, the illusory image can be realized by the equilibrium state of the *marker* CNN (Fig. 36).
3. These imitation mechanisms are summarized as follows:

Imitation of Ehrenstein Illusion					
OpenCV	ccvHoughCircles	⇒	OpenCV's drawing functions	⇒	illusory image
CNN & OpenCV	cvHoughCircles	⇒	watershed segmentation and marker CNNs	⇒	equilibrium state

5.2. *Neon color spreading illusion*

1. The OpenCV can imitate the neon color spreading illusion by using the programming functions *cvHoughCircles* and *cvCircle*. In order to imitate the illusion mechanism, we assume that the OpenCV's eye constructs a multilayered image from the detected objects. In our computer simulations, the cvHoughCircles detects an illusory circle from a series of cross lines (Fig. 37). By reconstructing a multilayered image from a detected circle, the cvCircle fills the top-layer circle in red (Figs. 37 and 38). If the cross has any gap, the illusory disk disappears (Fig. 39). In this case, consider the multilayered image of Fig. 40. The cvHoughCircles detects two illusory circles

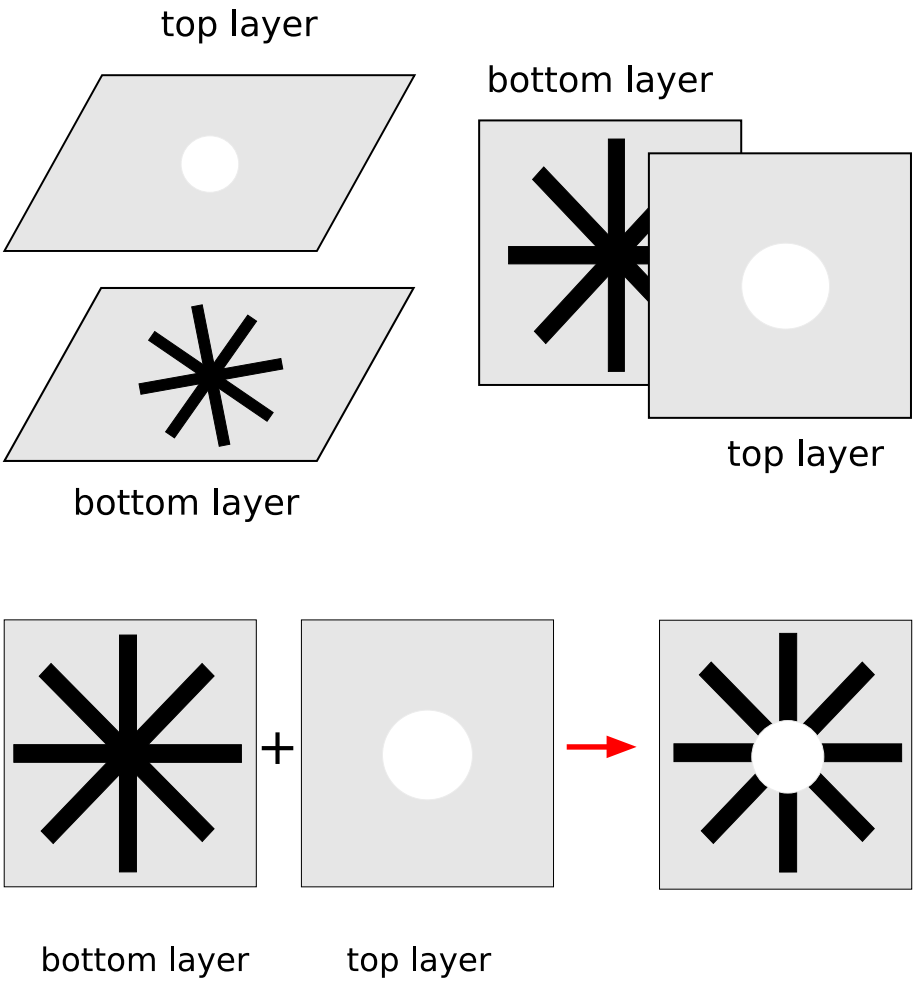

Fig. 29. Two-layered image of Ehrenstein illusion, which is formed by stacking two images together (upper figure). Ehrenstein illusion is obtained by combining (adding) these two-layered images (lower figure).

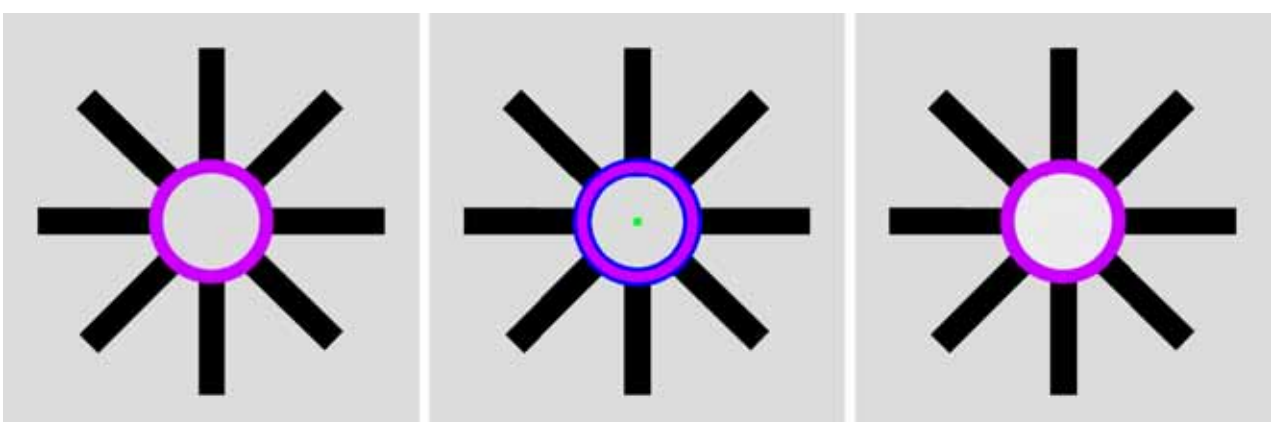

Fig. 30. Ehrenstein illusion image with a colored ring. The cvHoughCircles detect two illusory circles from the illusory image (left). Their centers and circles are painted in green and blue, respectively (center). The cvCircle fills the top circle with a whiter color than the background (right).

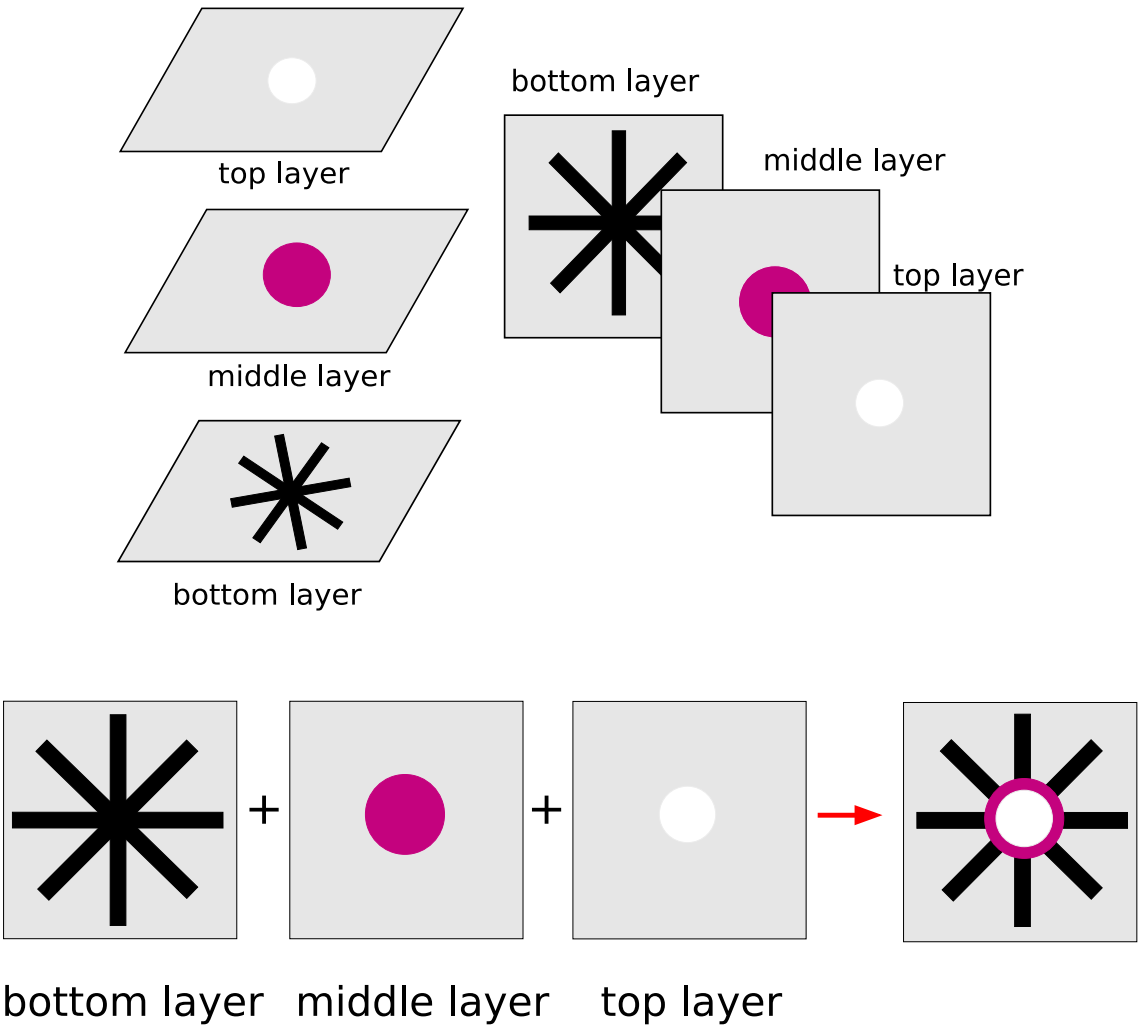

Fig. 31. Multilayered image of Ehrenstein illusion with a colored ring, which is formed by stacking three images together (upper figure). Ehrenstein illusion image with a colored ring is obtained by combining (adding) these three-layered images (lower figure).

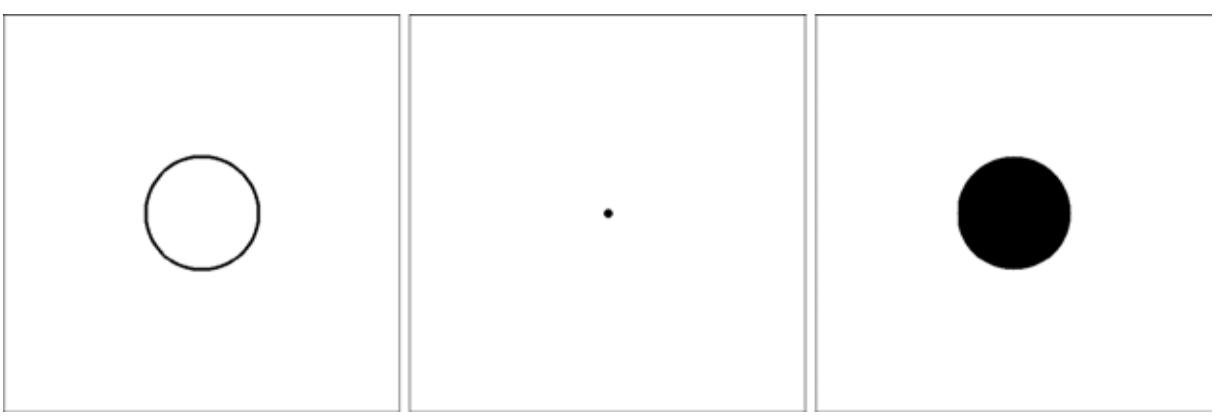

Fig. 32. Illusory disk can be generated by the *water segmentation* CNN, whose input image u_{ij}, initial state $x_{ij}(0)$, and output image y_{ij} are illustrated from left to right.

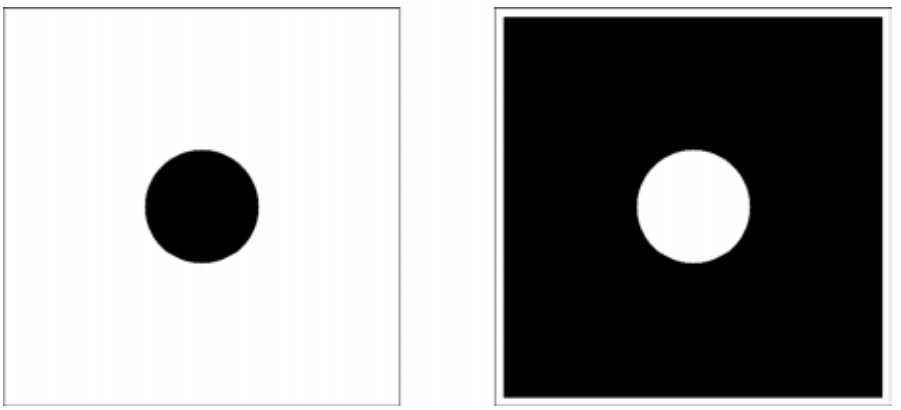

Fig. 33. Illusory white disk can be generated by the *negative image* CNN. A black disk image (left) is transformed into a negative image (right) by this CNN.

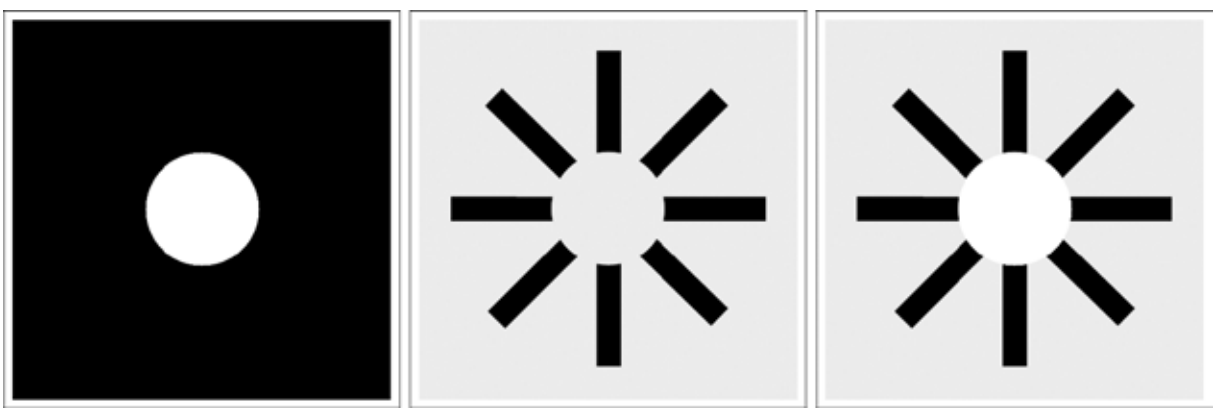

Fig. 34. Imitation of Ehrenstein illusion via CNN (32). Illusory white disk is added to the original illusion image by the *marker* CNN, whose input image u_{ij}, initial state $x_{ij}(0)$, and output image y_{ij} are illustrated from left to right. Note that the *marker* CNN (32) adds white markers using the "white" objects of a binary input image.

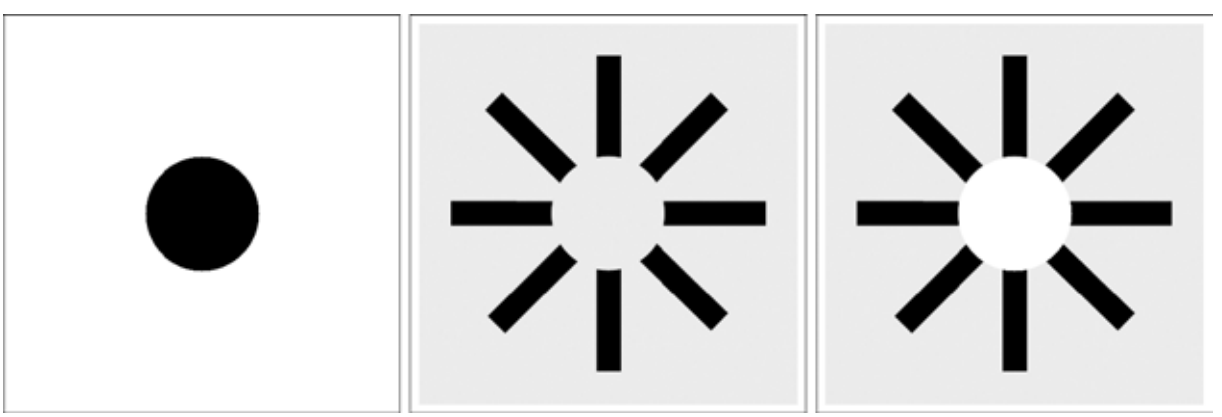

Fig. 35. Imitation of Ehrenstein illusion via CNN. Illusory white disk can be also generated by the *marker* CNN (31), whose input image u_{ij}, initial state $x_{ij}(0)$, and output image y_{ij} are illustrated from left to right. Note that the *marker* CNN (31) adds white markers using the "black" objects of a binary input image.

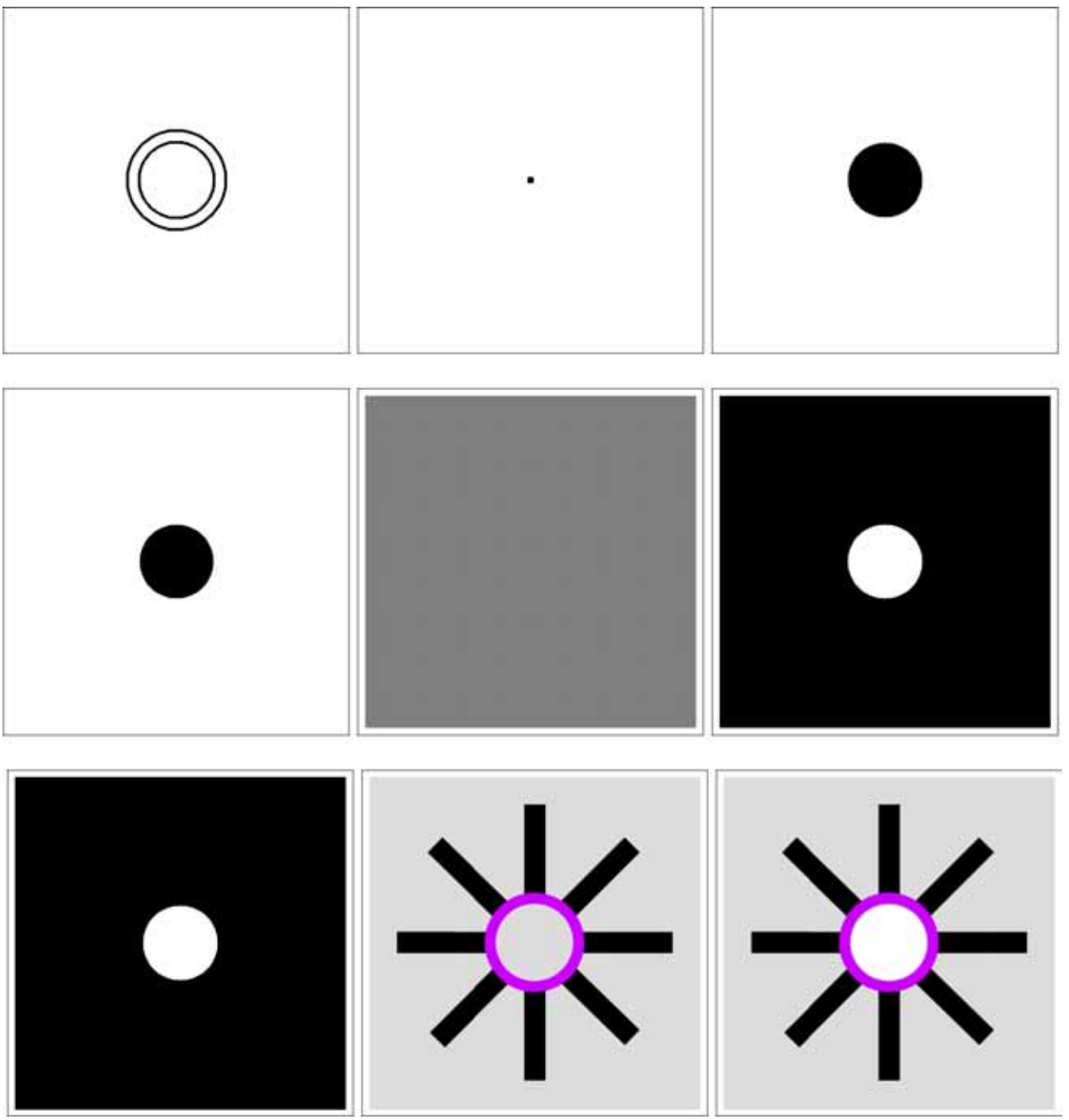

Fig. 36. Imitation of Ehrenstein illusion with a colored ring via CNN. The *water segmentation* CNN generates a black disk (top). The *negative image* CNN transforms a black disk into a white disk (middle). The *marker* CNN adds an illusory white disk to Ehrenstein illusion image. Note that the center disk of the right figure is whiter than that of the center figure. Input images, initial states and output images of these CNNs are illustrated from left to right.

Fig. 37. Neon color spreading illusion image (left). The cvHoughCircles detect an illusory circle from the illusory image (left). Its center and circle are painted in green and red, respectively (center). The cvCircle fills the top-layer circle in dark red (right).

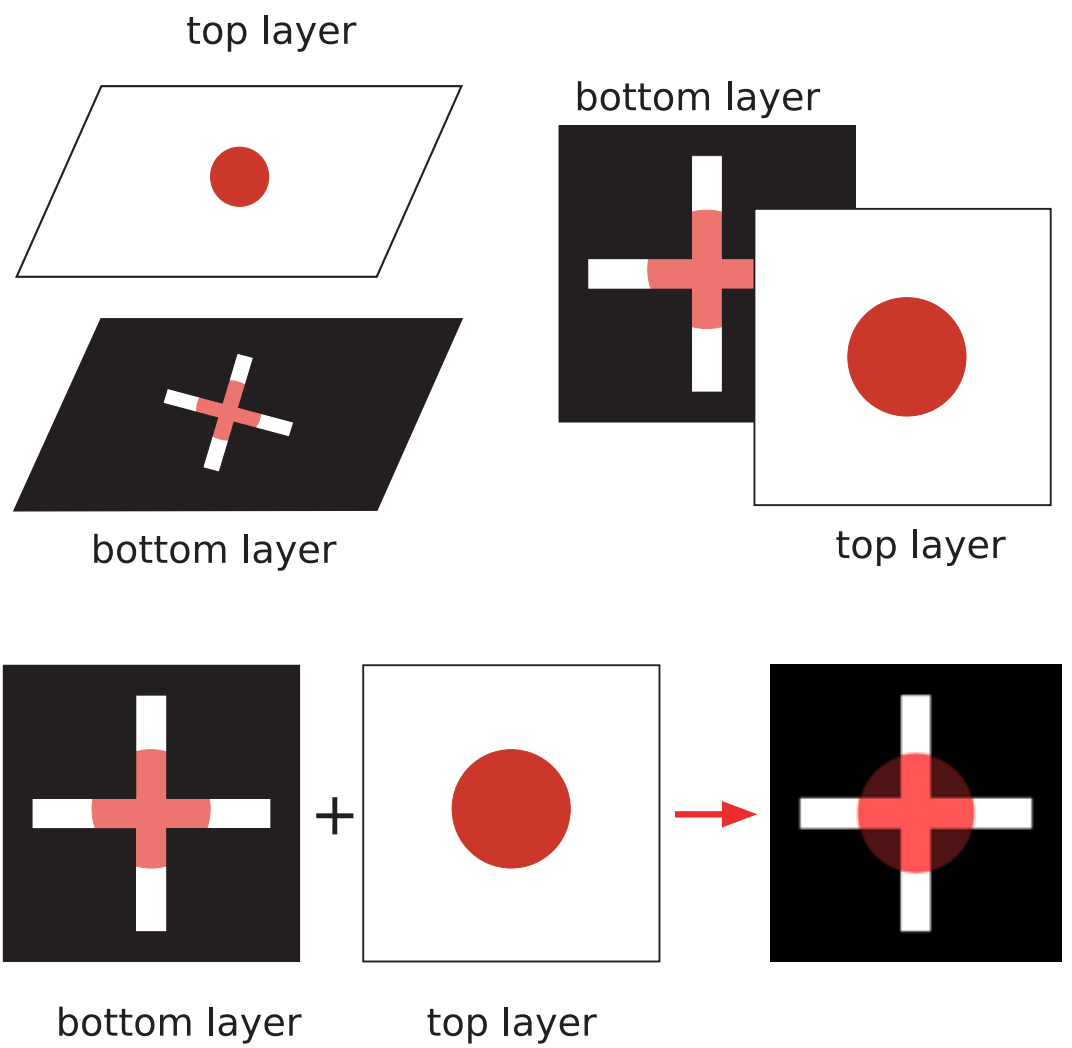

Fig. 38. Two-layered image of neon color spreading (upper figure). Neon color spreading illusion image is obtained by combining these two-layered images (lower figure).

Fig. 39. Neon color spreading illusion image with a black ring. The cvHoughCircles detect two illusory circles from the illusory image (left). Their centers and circles are painted in green and red, respectively (center). The cvCircle emphasizes the black ring (right) by gray lines.

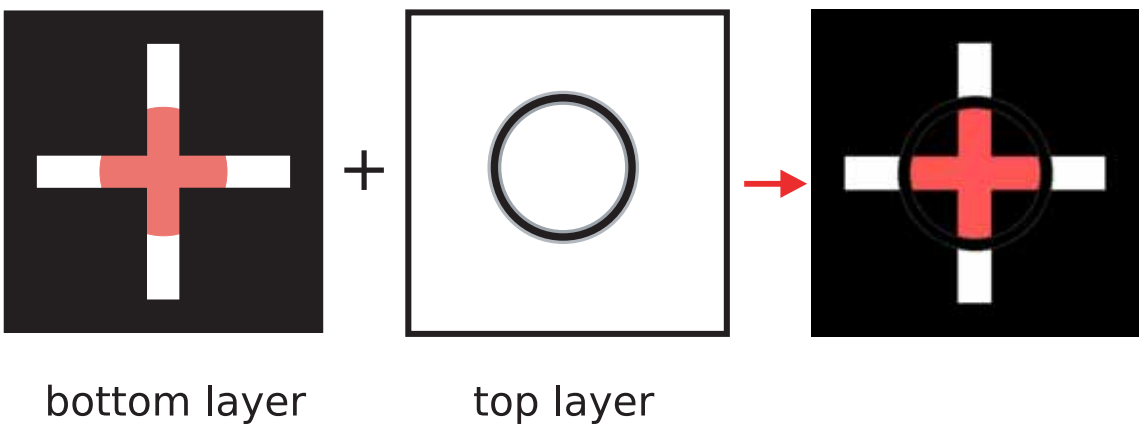

Fig. 40. Neon color spreading illusion image with a black ring is obtained by combining (adding) the two-layered images.

from a given image. The cvCircle emphasizes the black ring by constructing the multilayer image (Fig. 40).

2. The CNN can imitate the neon color spreading illusion with the help of OpenCV (Fig. 41). The *watershed segmentation* CNN fills the circle which the cvHoughCircles found in the illusory image. The *painting* CNN colors a black disk with dark red. The *marker* CNN adds the dark red disk to the illusory image. Thus, the illusory image can be realized by the equilibrium state of the *marker* CNN. When the cross has any gap, the cvHoughCircles finds two circles and their centers in an illusory image. The *painting* CNN colors them with dark gray. The *marker* CNN adds dark gray rings to the illusory image (Fig. 42). Thus, the illusory image can be realized by the equilibrium state of the *marker* CNN.
3. The imitation mechanisms are summarized as follows:

Imitation of Neon Color Spreading Illusion					
OpenCV	ccvHoughCircles	⇒	OpenCV's drawing functions	⇒	illusory image
CNN & OpenCV	cvHoughCircles	⇒	watershed segmentation and marker CNNs	⇒	equilibrium state

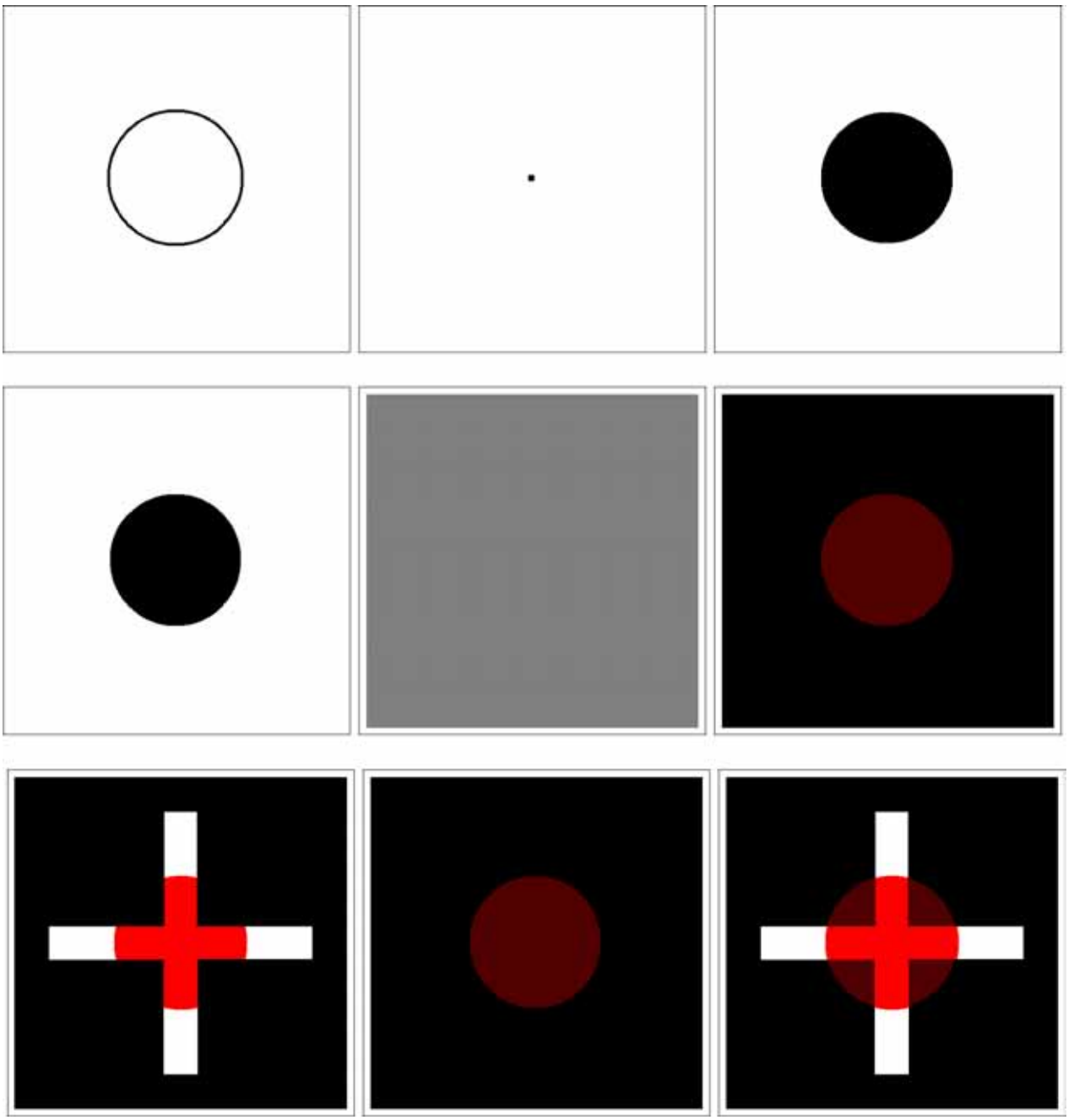

Fig. 41. Imitation of neon color spreading illusion via CNN. An illusory disk is generated by the *water segmentation* CNN (top). It is colored with dark red by the *painting* CNN (middle). The *marker* CNN adds dark gray rings to the illusory image (bottom). Input images, initial states, and output images of these CNNs are illustrated from left to right.

5.3. *Kanizsa illusion*

1. The OpenCV can imitate the Kanizsa illusion by using the programming functions *cvHoughLines2* and *Square Detector*.[16] The function cvHough-Lines2 finds lines in a grayscale image using a standard Hough transform, or a probabilistic Hough transform. The Square Detector finds squares from a given image. In order to imitate the illusion mechanism, we assume that the OpenCV's eye constructs a multilayered image from the detected objects.

   In our computer simulations, the cvHough-Lines2 finds four long lines from the illusory image if the standard Hough transform is used,

[16] "Square Detector" program is included in the OpenCV sample programs.

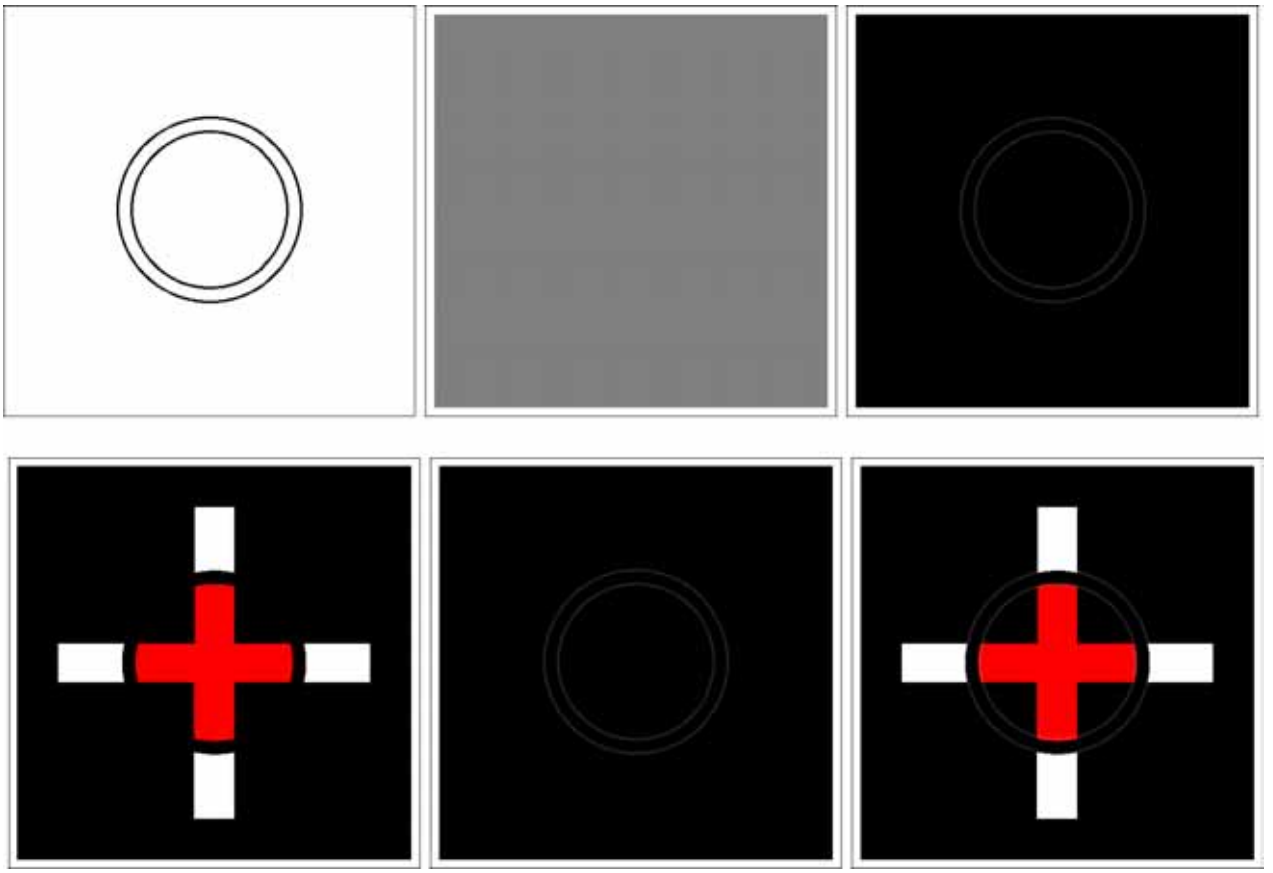

Fig. 42. Imitation of neon color spreading illusion via CNN. Two circles are colored with dark gray by the *painting* CNN. Illusory black ring can be generated by the *marker* CNN. Input images, initial states, and output images of these CNNs are illustrated from left to right.

and it finds four short lines if the probabilistic Hough transform is used (Fig. 43). The Square Detector finds a square from the image which is marked by the cvHoughLines2, and fills a square in blue by constructing the multilayer image (Figs. 43 and 44).

2. The CNN can imitate Kanizsa illusion with the help of OpenCV (Fig. 45). The *watershed segmentation* CNN fills the square, which the cvHoughLines2 found from the illusory image. Then, the *painting* CNN colors it with dark blue. The *marker* CNN next adds the dark square to the original illusory image. Thus, the illusory image can be realized by the equilibrium state of the *marker* CNN.
3. The imitation mechanisms are summarized as follows:

Imitation of Kanizsa Illusion					
OpenCV	cvHoughLines2	⇒	Square Detector	⇒	illusory image
CNN & OpenCV	cvHoughLines2	⇒	watershed segmentation and marker CNNs	⇒	equilibrium state

5.4. *Shifted edges illusion*

1. The OpenCV can imitate the shifted edges illusion by using the programming functions *cvCanny* and *cvHoughLines2*. The function cvCanny finds the edges on the input image and marks them in the output image using the Canny algorithm.

   In our computer simulations, the cvHoughLines2 finds many long lines from the output

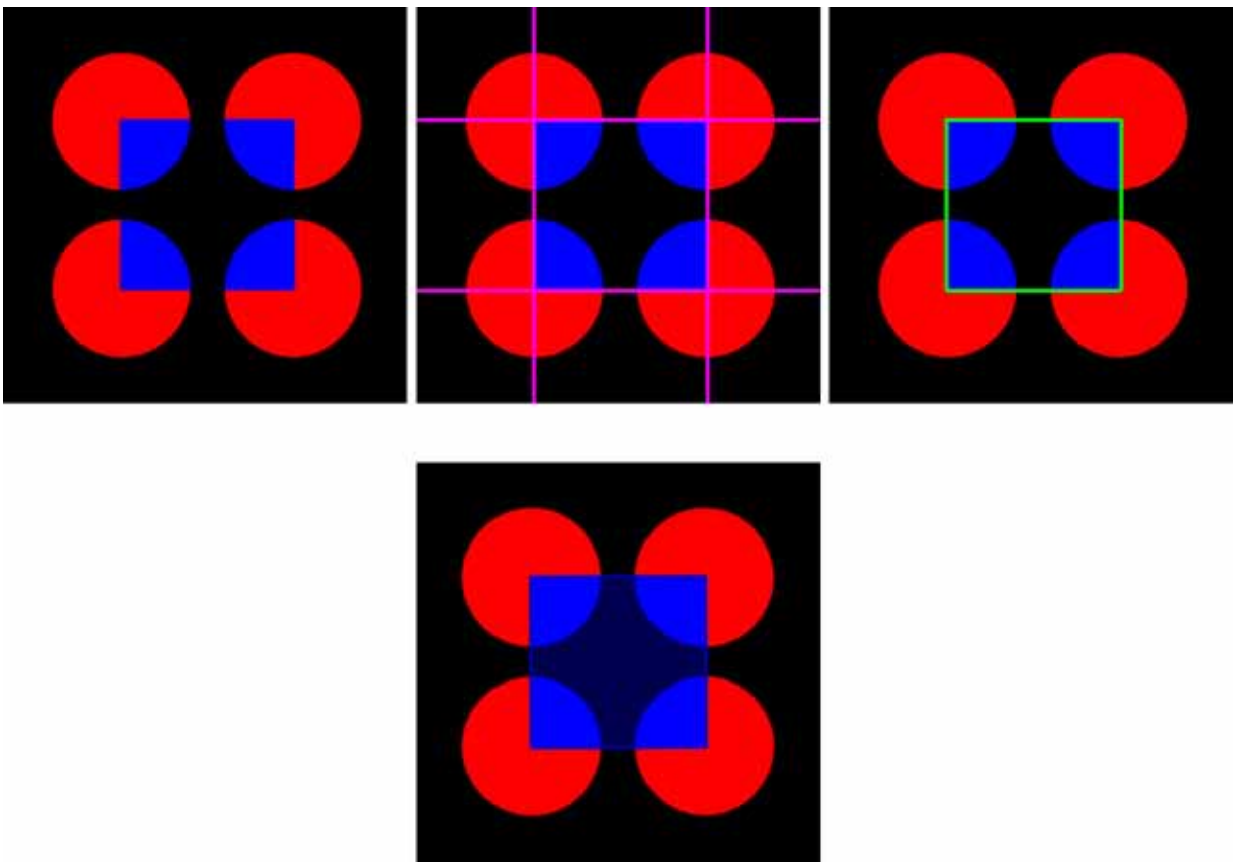

Fig. 43. Kanizsa illusion. The cvHoughLines2 finds four long lines from the illusory image (top left) if the standard Hough transform (top center) is used, and finds four short lines if the probabilistic Hough transform is used (top right). The Square Detector finds a square (bottom center) from the image which is marked by the cvHoughLines2 (top right), and fill a square in dark blue (bottom).

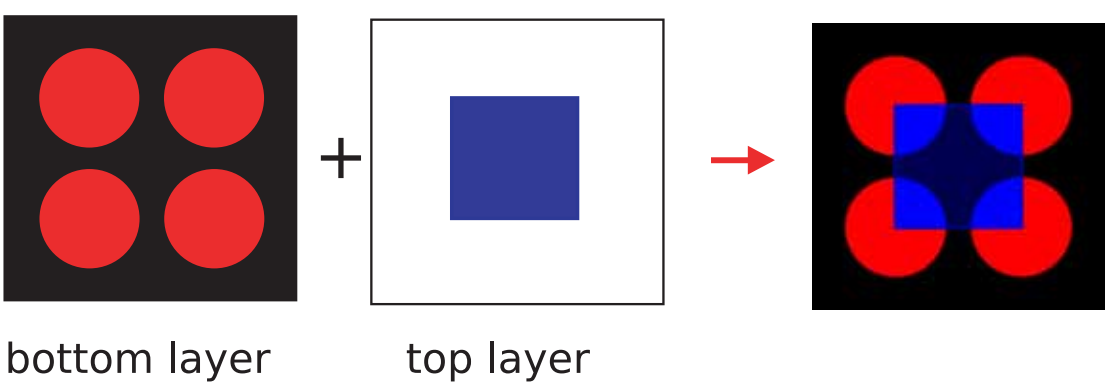

Fig. 44. Kanizsa illusion image is obtained by combining (adding) the two-layered images.

image of the cvCanny if the standard Hough transform is used, and finds many short lines if the probabilistic Hough transform is used (Figs. 46 and 47). Observe that the cvHough-Lines2 detects *many tilted horizontal lines* if the standard Hough transform is used. Thus the entire row appears to tilt though shifted rectangles are horizontally aligned. Note that each row is aligned horizontally and does not tilt in the output image of the cvCanny.

2. The CNN can imitate the shifted edges illusion with the help of OpenCV (Figs. 48–51). The cvHoughLines2 detects many tilted lines. The *marker* CNN adds illusory lines to the input image. Thus, the illusory image can be realized by the equilibrium state of the *marker* CNN.

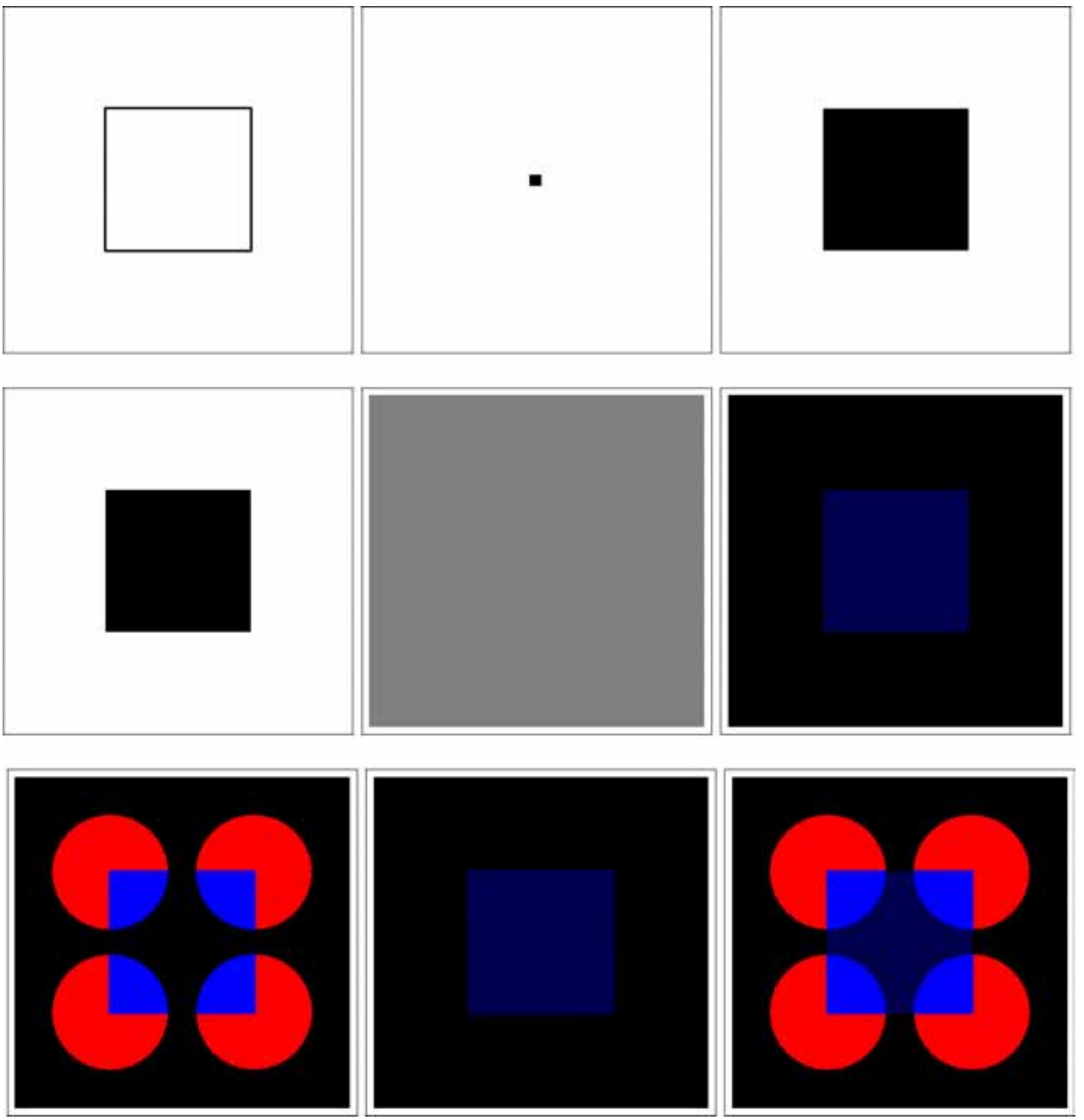

Fig. 45. Imitation of Kanizsa illusion via CNN. An illusory black square is generated by the *water segmentation* CNN (top). It is colored with dark blue by the *painting* CNN (middle). Illusory square is added to the input image by the *marker* CNN (bottom). Input images, initial states, and output images of these CNNs are illustrated from left to right.

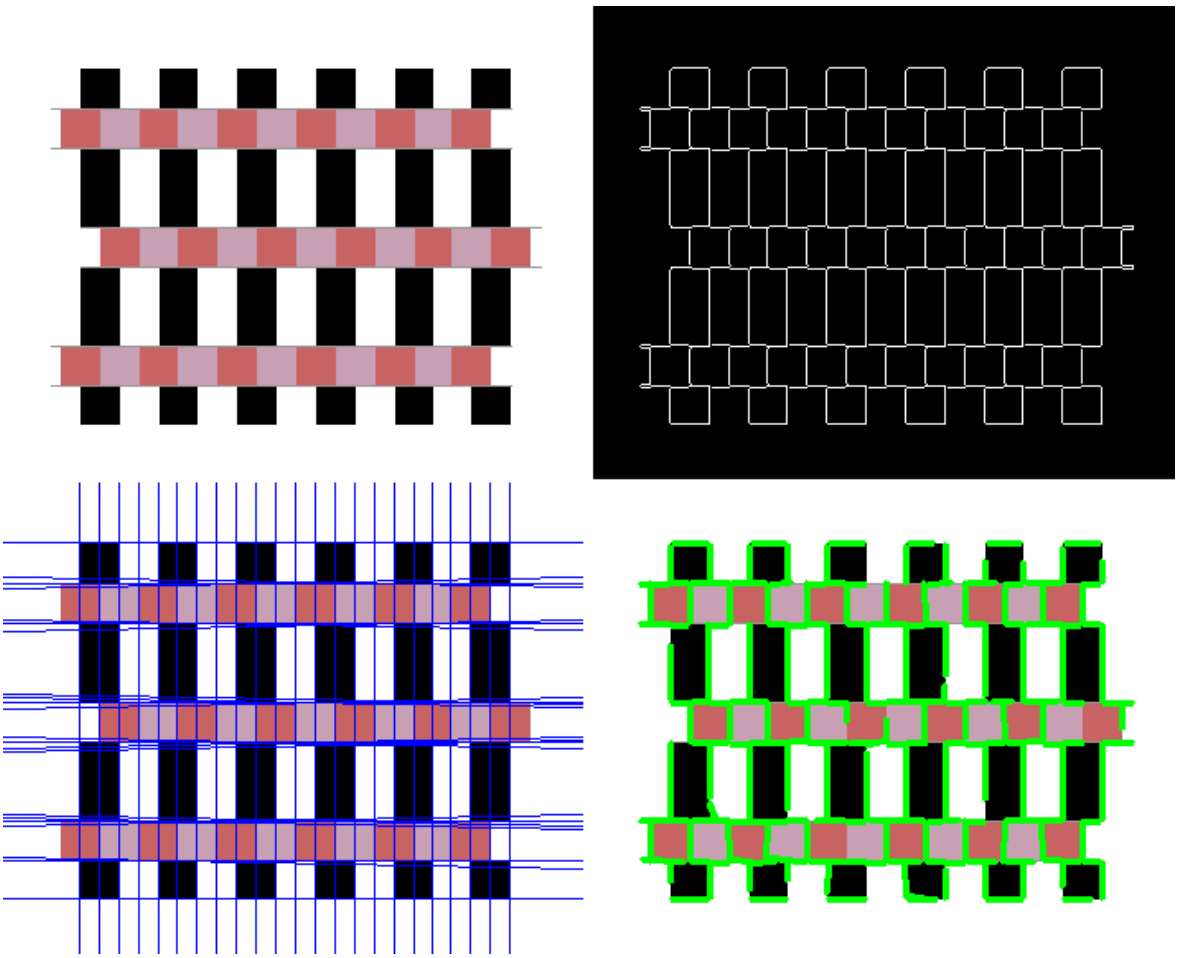

Fig. 46. Shifted edges illusion [Kitaoka, 2007] (top left). Observe that each row is aligned horizontally, and it does not tilt in the cvCanny output image (top right). The cvHoughLines2 finds long lines from the output image of the cvCanny if the standard Hough transform is applied (bottom left), and finds short lines if the probabilistic Hough transform is applied (bottom right). Observe that the cvHoughLines2 detects many tilted horizontal lines if the standard Hough transform is used. Thus, the entire row appears to tilt horizontally.

3. The imitation mechanisms are summarized as follows:

Imitation of Shifted Edges Illusion					
OpenCV	cvCanny and cvHoughLines2	⇒	OpenCV's drawing functions	⇒	illusory image
CNN & OpenCV	cvCanny and cvHoughLines2	⇒	marker CNN	⇒	equilibrium state

5.5. *Watercolor illusion*

1. The OpenCV can imitate the watercolor illusion by using the *watershed program*.[17] This program splits an image into areas, based on the topology of the image. Before passing the image to the watershed algorithm, the desired regions have to be marked roughly.

   In our computer simulations, the watershed program tints the interior of the figure with the similar hue of the lighter colored line (Fig. 52).

[17]The watershed program is included in the OpenCV sample programs. The watershed segmentation is performed by the function *cvWatershed* in this program.

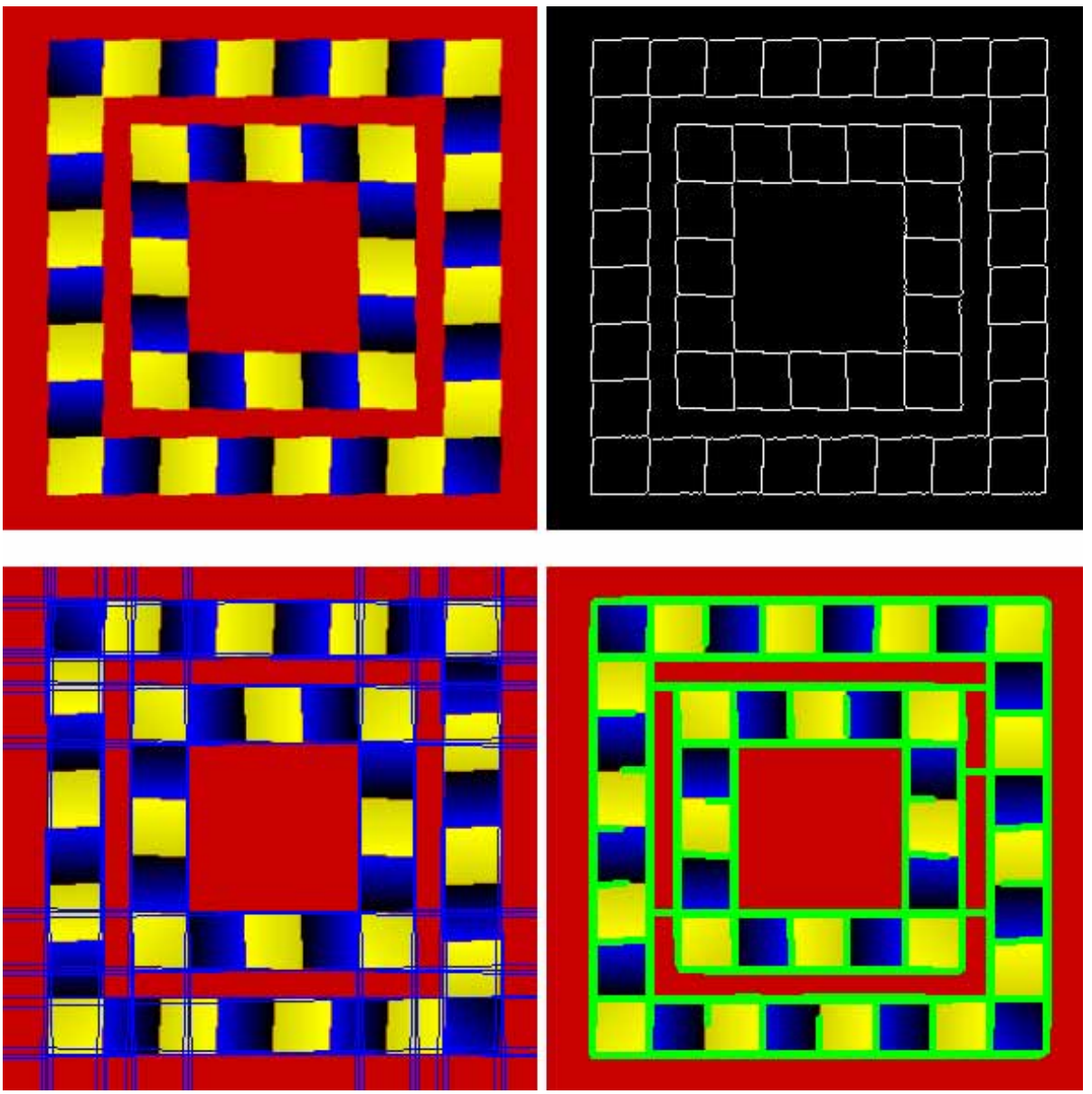

Fig. 47. Shifted edges illusion [Kitaoka, 2007] (top left). Observe that each row tilts neither horizontally nor vertically in the cvCanny output image (top right). The cvHoughLines2 finds long lines from the output image of the cvCanny if the standard Hough transform is applied (bottom left), and finds short lines if the probabilistic Hough transform is applied (bottom right). Observe that the cvHoughLines2 detects many tilted horizontal and vertical lines if the standard Hough transform is used. Thus, the entire row appears to tilt horizontally and vertically.

2. The CNN can imitate the watercolor illusion with the help of OpenCV (Fig. 53). The cvThreshold function (or *Thresholding* CNN) converts an illusory image into a binary image. The *marker* CNN tints the interior of the binary image with yellow.
3. The imitation mechanisms are summarized as follows:

Imitation of Watercolor Illusion					
OpenCV	cvWatershed	⇒	OpenCV's drawing functions	⇒	illusory image
CNN & OpenCV	cvThreshold or thresholding CNN	⇒	marker CNN	⇒	equilibrium state

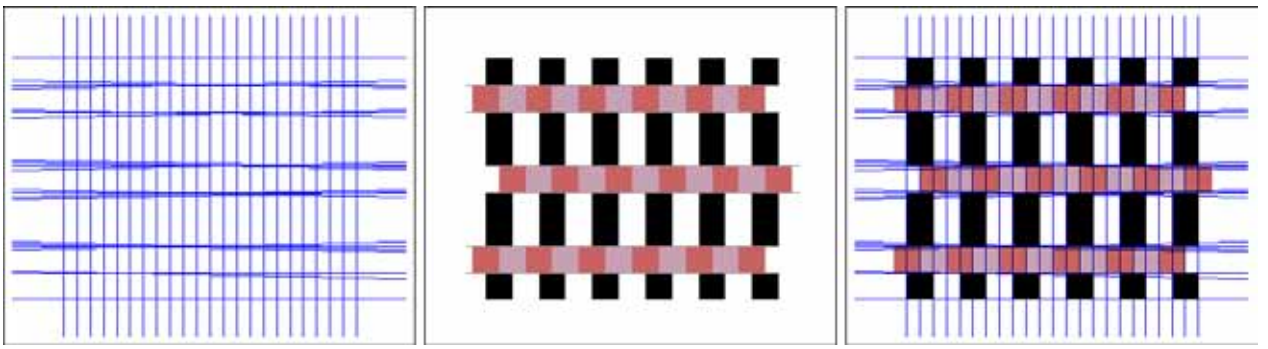

Fig. 48. Imitation of shifted edges illusion via CNN. Many illusory lines detected by cvHoughLines2 are added to the input image by the *marker* CNN, whose input image u_{ij}, initial state $x_{ij}(0)$, and output image y_{ij} are illustrated from left to right.

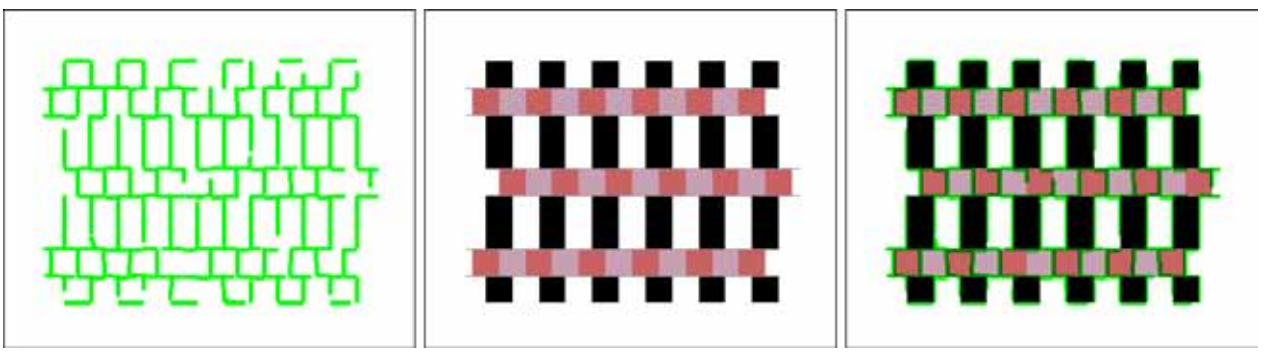

Fig. 49. Imitation of shifted edges illusion via CNN. Short lines detected by cvHoughLines2 are added to the input image by the *marker* CNN, whose input image u_{ij}, initial state $x_{ij}(0)$, and output image y_{ij} are illustrated from left to right.

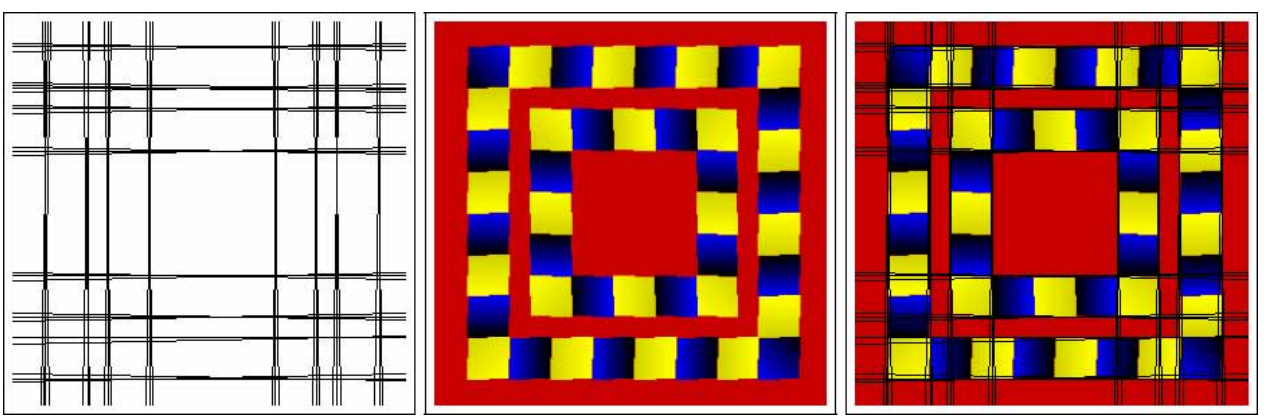

Fig. 50. Imitation of shifted edges illusion via CNN. Many tilted horizontal and vertical lines detected by cvHoughLines2 are added to the input image by the *marker* CNN, whose input image u_{ij}, initial state $x_{ij}(0)$, and output image y_{ij} are illustrated from left to right.

5.6. *Hybrid image illusion*

1. The OpenCV can imitate the hybrid images by using the programming functions *cvThreshold* and *cvAdaptiveThreshold*, which convert the input image to binary or black-and-white image, and apply the fixed-level threshold and adaptive threshold to input image, respectively.

In our computer simulations, the cvThreshold and cvAdaptiveThreshold extract two kinds of pictures from the hybrid image. One is the picture with the low-spatial frequencies and the other is the picture with the high spatial frequencies. Observe that the cvThreshold and cvAdaptiveThreshold can extract a *motorcycle*

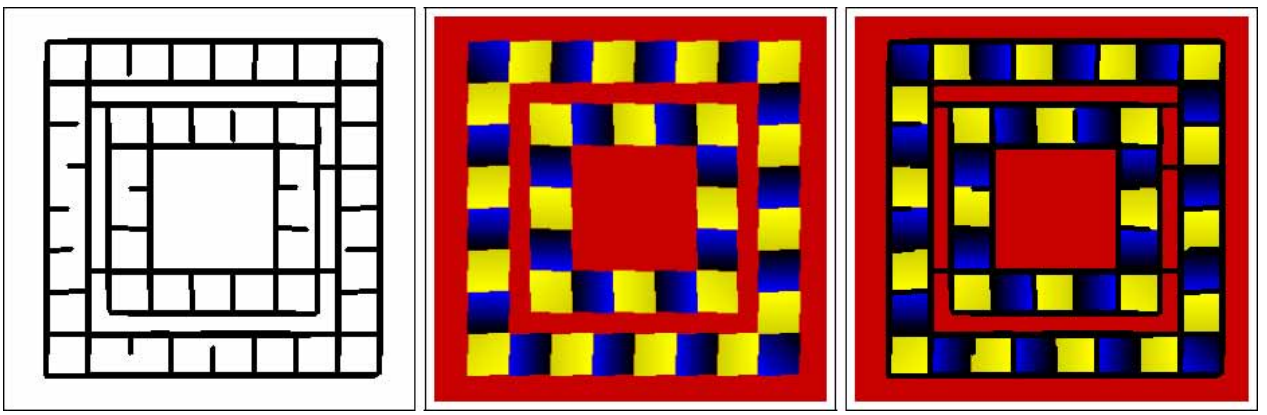

Fig. 51. Imitation of shifted edges illusion via CNN. Short horizontal and vertical lines detected by cvHoughLines2 are added to the input image by the *marker* CNN, whose input image u_{ij}, initial state $x_{ij}(0)$, and output image y_{ij} are illustrated from left to right.

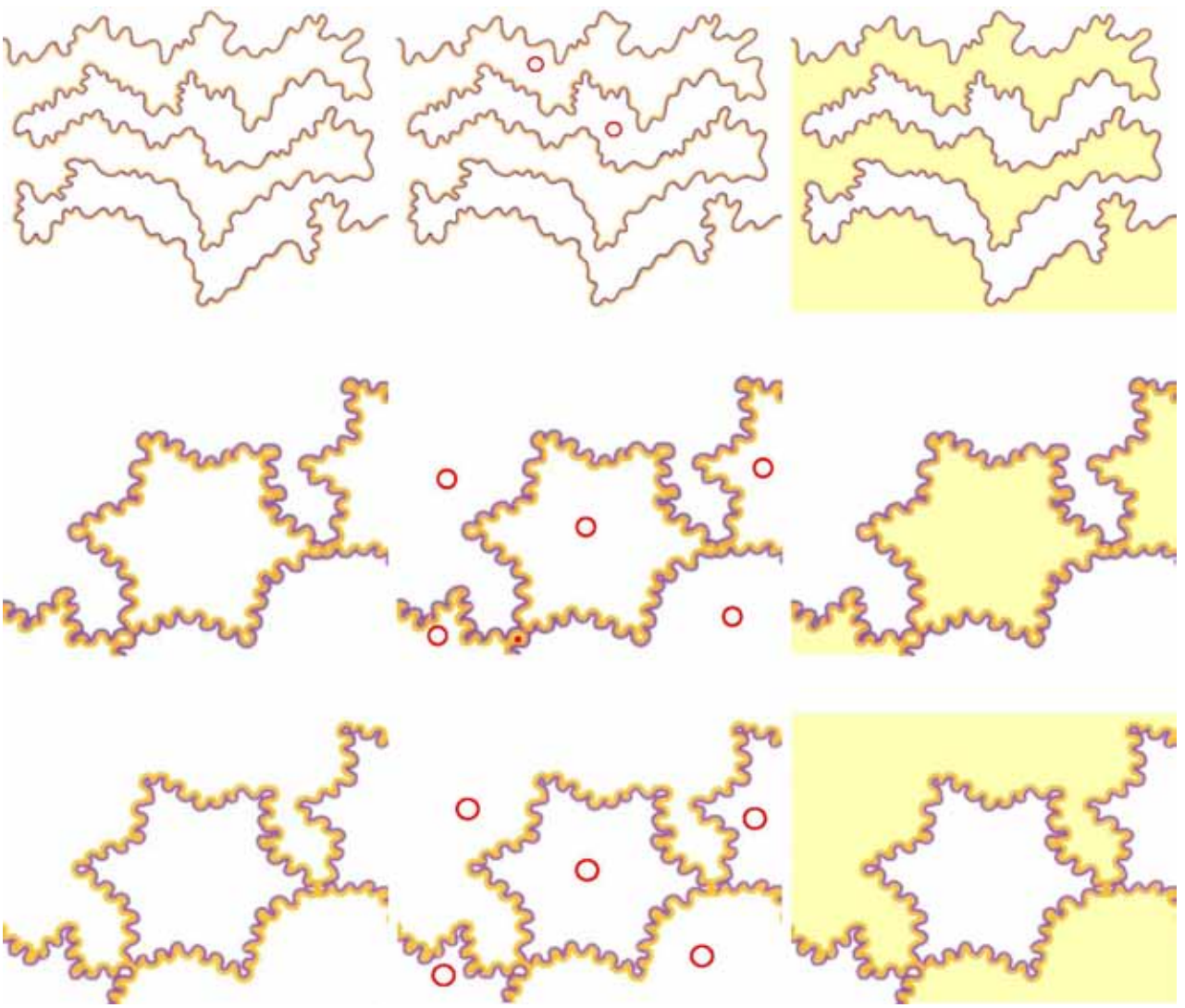

Fig. 52. Watercolor illusion [Pinna & Grossberg, 2005]. The watershed program splits an image into areas, which are marked by red circles. Observe that the interior of the figure is tinted with the similar hue of the lighter colored line.

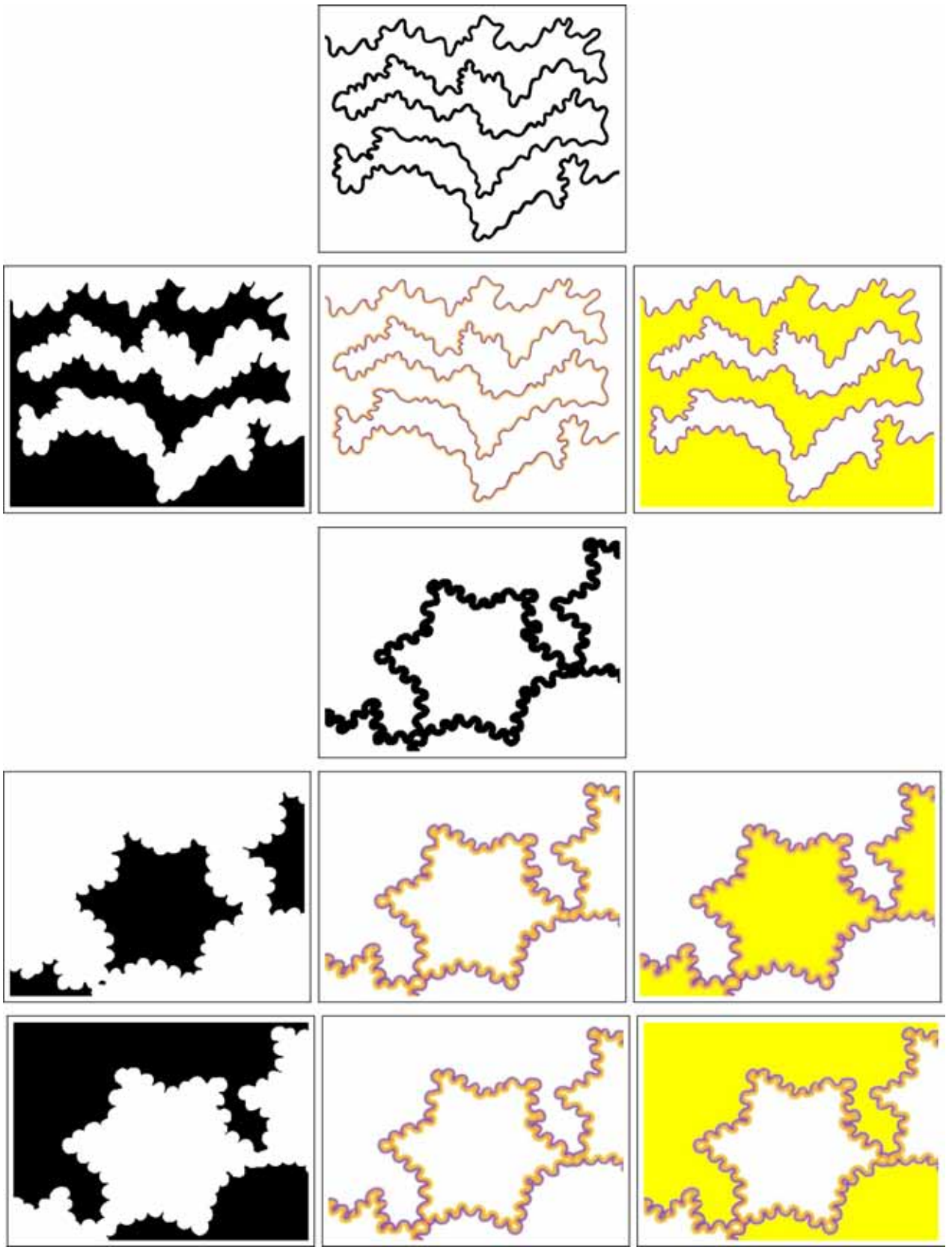

Fig. 53. Imitation of watercolor illusion via CNN. The cvThreshold function converts an illusory image into a binary image (top and third rows). The *marker* CNN tints the interior of the binary image with yellow. Input images, initial states, and output images of the *marker* CNN are illustrated from left to right (the second, fourth, and fifth rows).

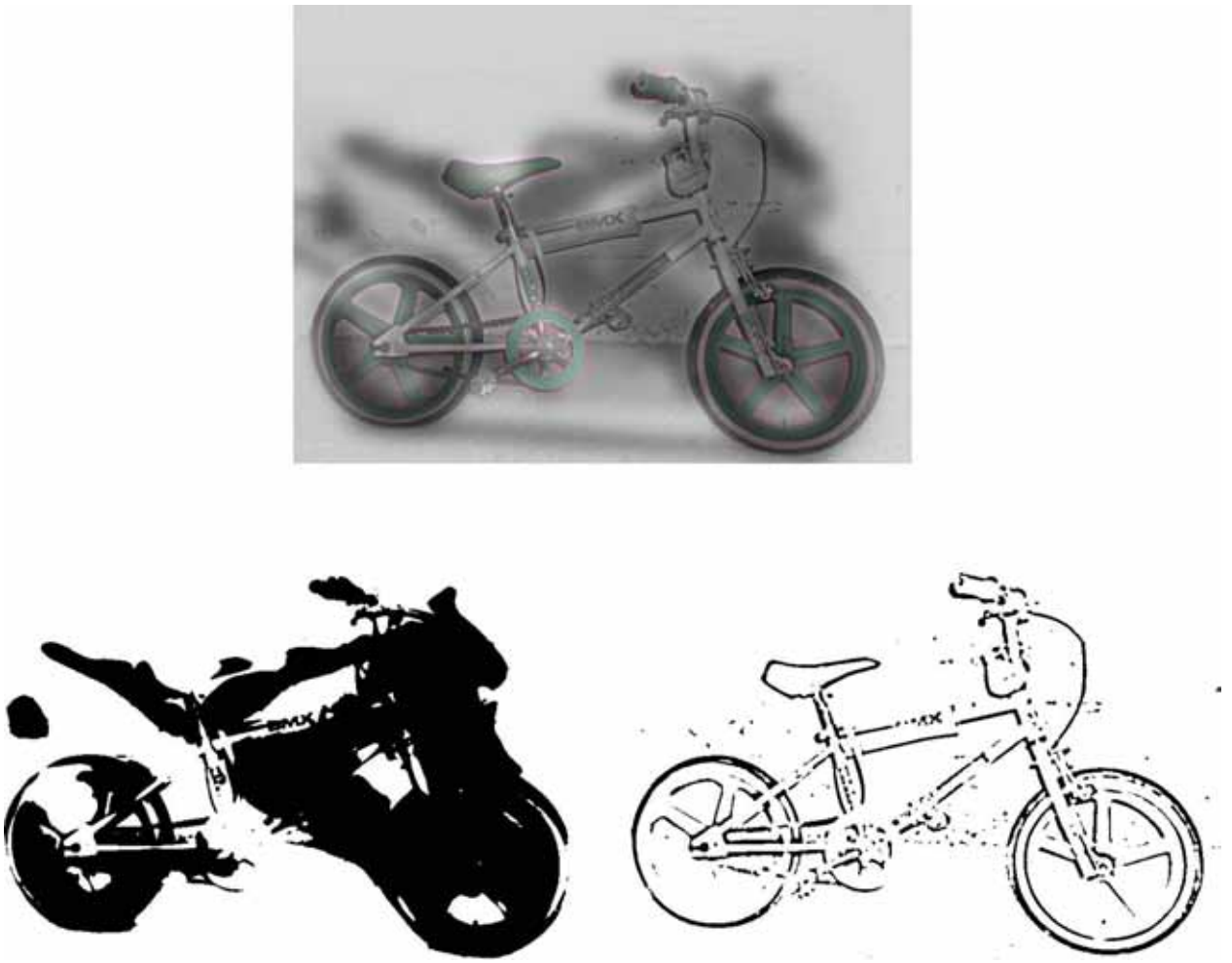

Fig. 54. Motorbicycle (hybrid image of a motorcycle and a bicycle [Oliva *et al.*, 2006]). At close perceptual range, the parts of the motorcycle appear to belong to the shadow of the bicycle. Once the viewer steps back from the image, what appears to be cast shadows from the bicycle will regroup to form a motorcycle (top). Observe that the cvThreshold and cvAdaptiveThreshold can extract a motorcycle (bottom left) and a bicycle (bottom right) from the hybrid image, respectively.

and a *bicycle* from the hybrid image, respectively (Fig. 54). In the case of catwoman, the cvAdaptiveThreshold extracts both *woman's face* and *cat's face* by using different block size (namely, the size of a pixel neighborhood), which is used to calculate a threshold value for the pixel (Fig. 55).

2. The CNN can imitate the hybrid image using the *thresholding* CNN. The *thresholding* CNNs for $z^* = 0.15$ and $z^* = -0.3$ can extract a motorcycle and a bicycle from the hybrid image, respectively (Fig. 56). In the case of catwoman, the *thresholding* CNNs for $z^* = 0.35$ and $z^* = 0$ can extract cat's face and woman's face from the hybrid image, respectively (Fig. 57). The *marker* CNN can add the extracted image to the illusory image.
3. The imitation mechanisms are summarized as follows:

Imitation of Hybrid Image Illusion					
OpenCV	cvThreshold and cvAdaptiveThreshold	⇒	OpenCV's drawing functions	⇒	illusory image
CNN	Thresholding CNN	⇒	marker CNN	⇒	equilibrium state

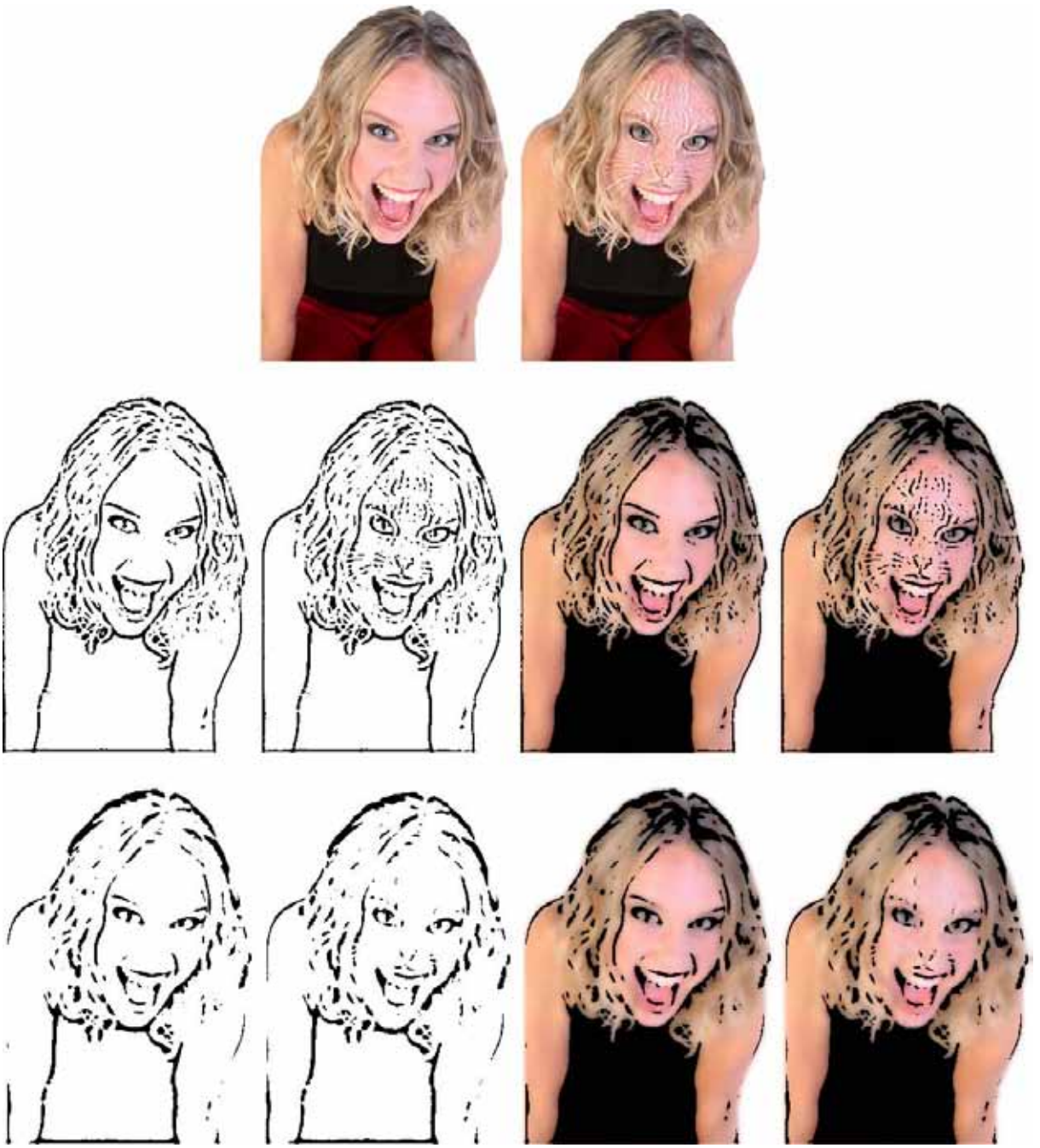

Fig. 55. Catwoman (hybrid image of woman's face and cats' face [Oliva *et al.*, 2006]). Woman's face that turns into a cat (right) at close distance (top). Observe that the cvAdaptiveThreshold can extract cat's face (middle) and woman's face (bottom) from the hybrid image by changing the block size to calculate a threshold value. The block sizes for cat's face and woman's face are 7×7 and 15×15, respectively. The processed binary images are superimposed to the original color figure (right).

5.7. *Anomalous motion illusion*

Applying the *morphological gradient* to the anomalous motion illusions (Figs. 32–41), we obtained three kinds of output images:

- Pale colored image (Figs. 58–61). Illusory motion is obtained by the vector originating from the dark blue part of the object and terminating at the bright blue or white part.
- Single colored image (Figs. 62 and 63). Illusory motion is not obtained from the morphological gradient output images. However, it is directly obtained by the original single colored image. That is, the illusory motion is defined by

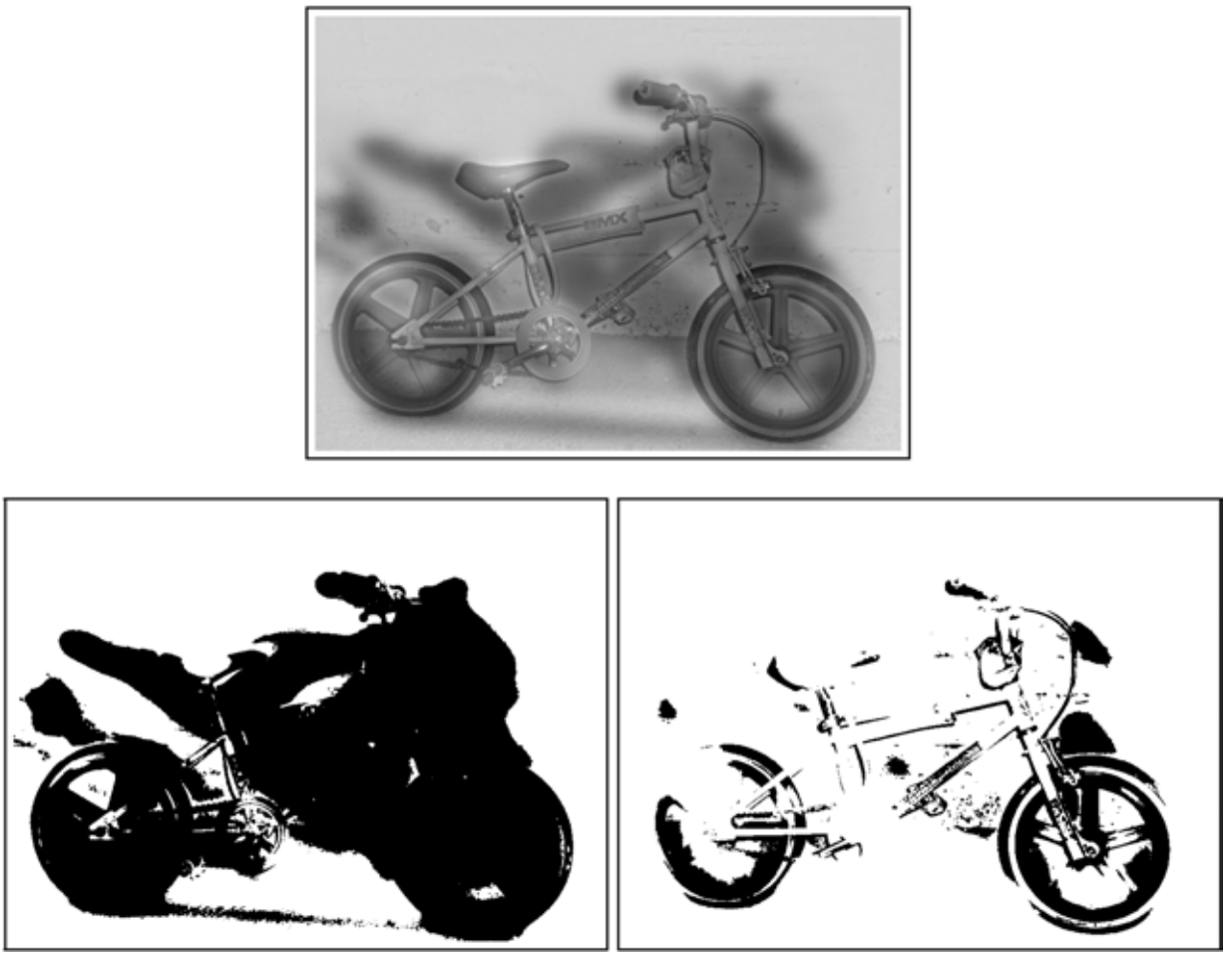

Fig. 56. Imitation of hybrid image via *thresholding* CNNs. The *thresholding* CNNs for $z^* = 0.15$ and $z^* = -0.3$ can extract a motorcycle (bottom left) and a bicycle (bottom right) from the hybrid image, respectively.

the vector originating from the dark part of the object and terminating to the bright or white part.

- Bright colored image (Figs. 64–67). Illusory motion is obtained by the vector originating from the nonpink part of the object and terminating at the pink part.

In our computer simulations, the *central drift illusion* [Kitaoka, 2007]) (which occur in the *central* vision) has a pale or a single colored output image. However, the *peripheral drift illusion* [Kitaoka, 2007]) (which occurs in the *peripheral*) has a bright colored output image. This suggests that the color contrast plays an important role in the anomalous motion illusion (Figs. 64–67).

1. The OpenCV can imitate the anomalous motion illusion by using the programing functions *morphological gradient* and *cvCalcOpticalFlowPyrLK*.[18] The *cvCalcOpticalFlowPyrLK* can generate the optical flows as shown in Figs. 68 and 69, though we have to prepare two input images.
2. The CNN can imitate the central drift illusion using the *shift translation* templates. The *shift translation* CNN rotates a ring of carrots clockwise by using the transformation between the (x, y)-plane and the (r, θ)-plane (Fig. 70). If an image is displayed on the computer screen then quickly replaced by a new image that is similar to the previous image, but shifted slightly, then illusory movement is created. Thus, the rotated carrots can create the illusory motion.

   In the case of the peripheral drift illusion, the *shift translation* CNN moves lines of beanjam cakes to left or right (Fig. 71). These shifted

[18]The programming function CalcOpticalFlowPyrLK calculates optical flow for two images using iterative Lucas-Kanade method in pyramids (for more details, see the OpenCV Reference Manual).

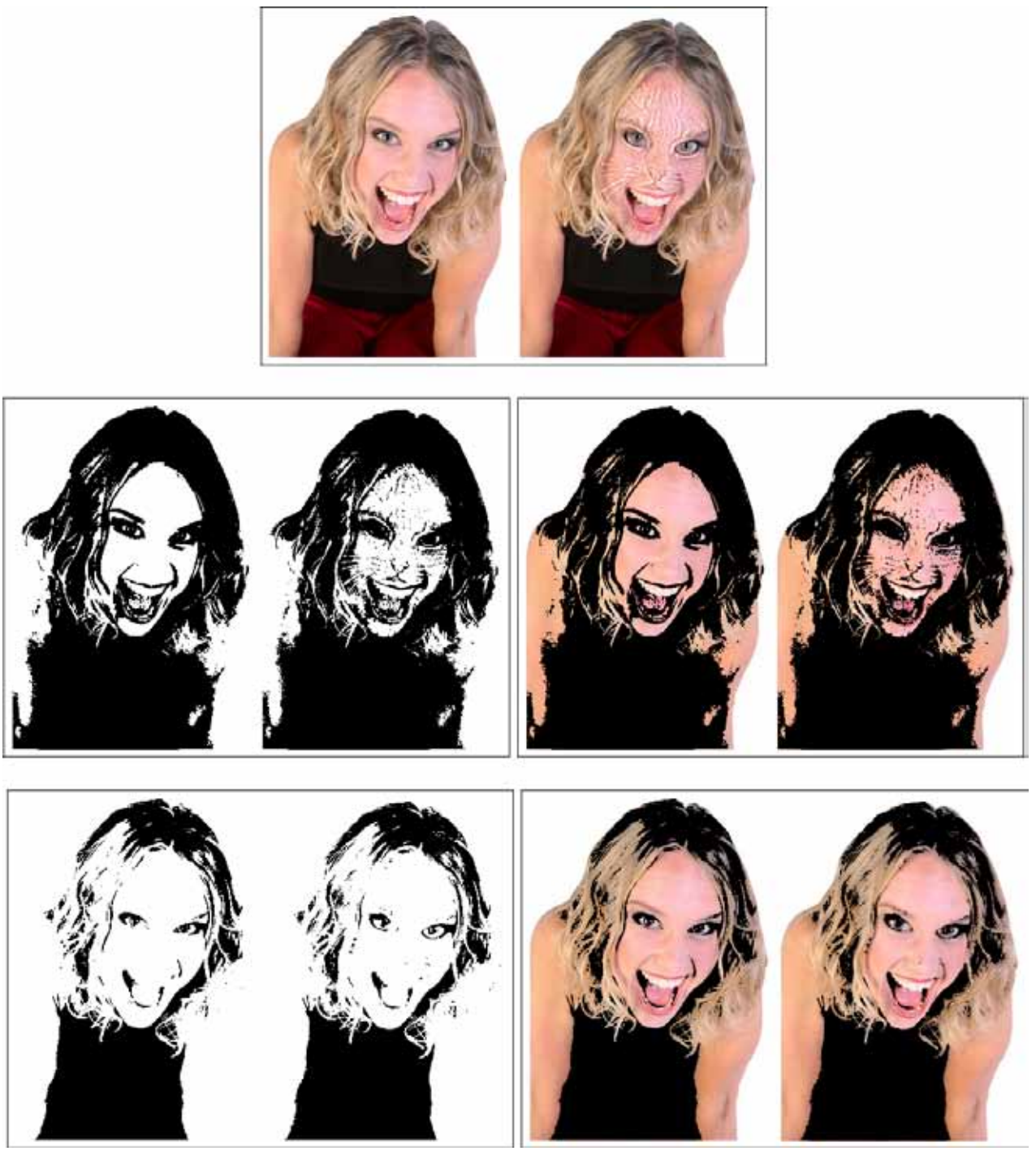

Fig. 57. Imitation of hybrid image illusion via CNNs. The *thresholding* CNNs for $z^* = 0.35$ and $z^* = 0$ can extract cat's face (middle)and woman's face (bottom) from the hybrid image, respectively. The processed binary images are superimposed to the color figure by using the *marker* CNN (right).

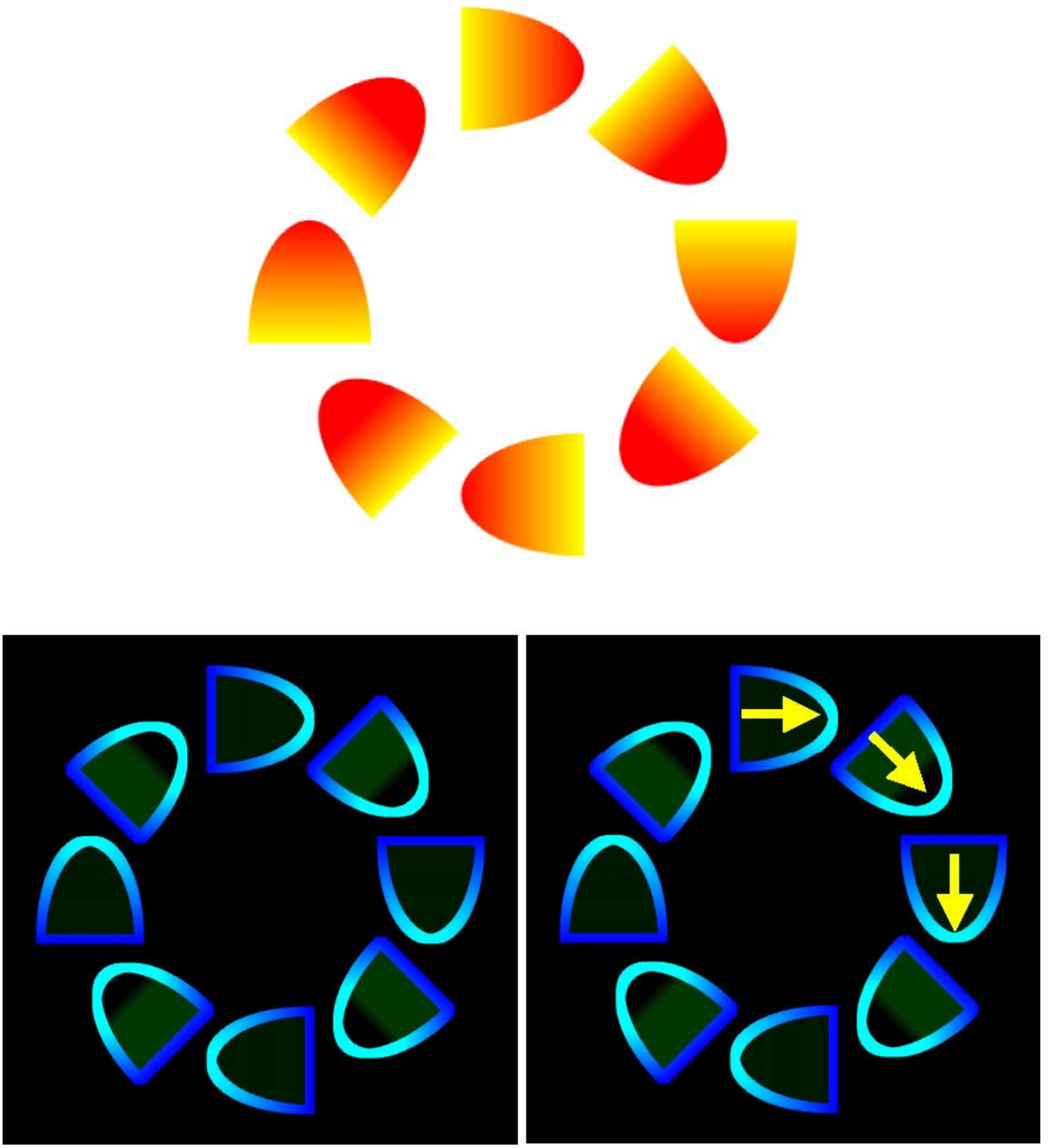

Fig. 58. Rotating carrots [Kitaoka, 2007]. A ring of carrots appear to rotate slowly (top). Applying the morphological gradient to this illusory image, we obtain the pale image (bottom) and the illusory motion of carrots (bottom right).

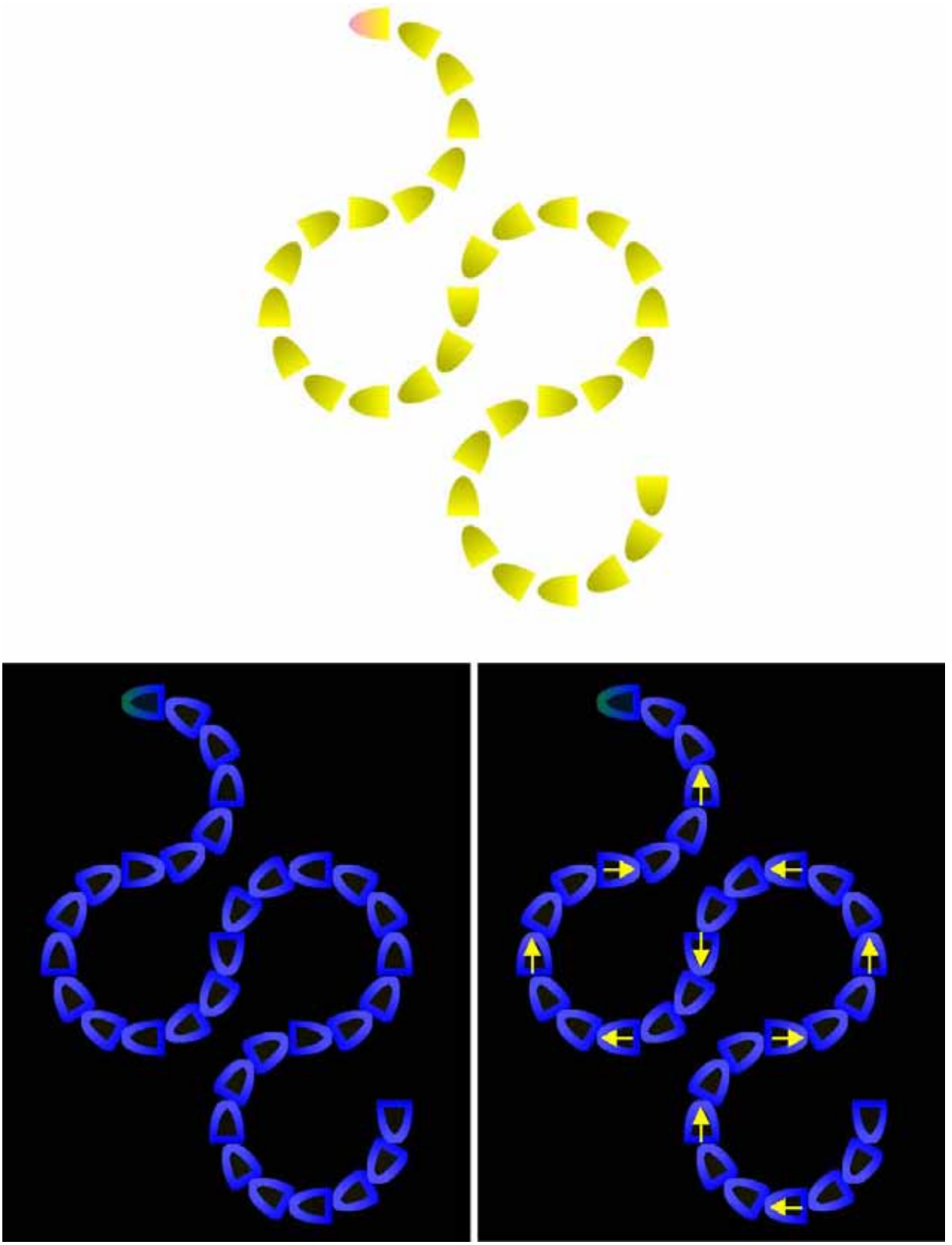

Fig. 59. Slow snake [Kitaoka, 2007]. The row appears to move toward the "head" (top). Applying the morphological gradient to this illusory image, we obtain the pale image (bottom left) and the illusory motion of slow snake (bottom right).

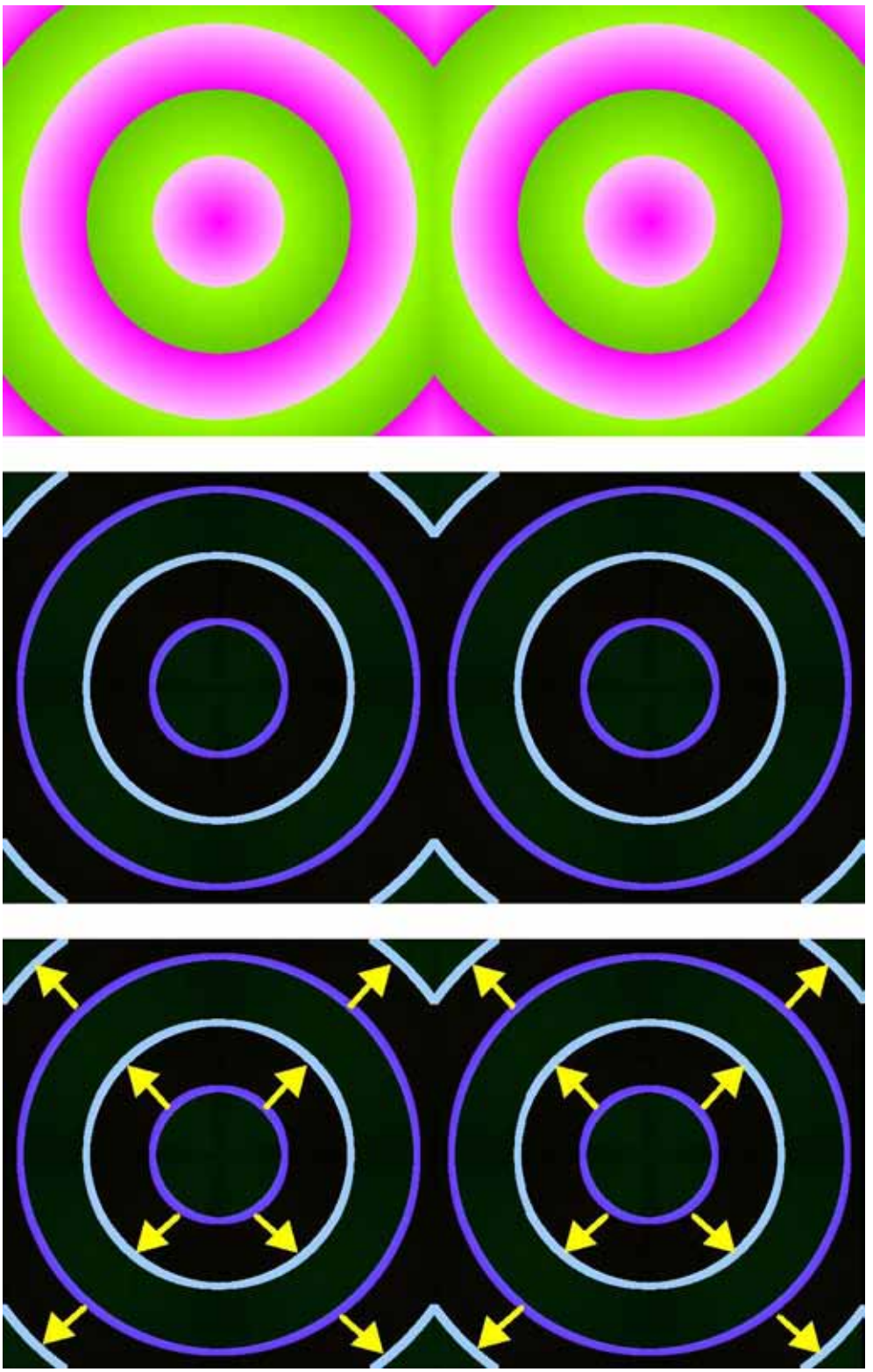

Fig. 60. Pink eyes [Kitaoka, 2007]. Pink eyes appear to expand a little (top). Applying the morphological gradient to this illusory image, we obtain the pale image (middle), and the illusory motion of pink eyes (bottom).

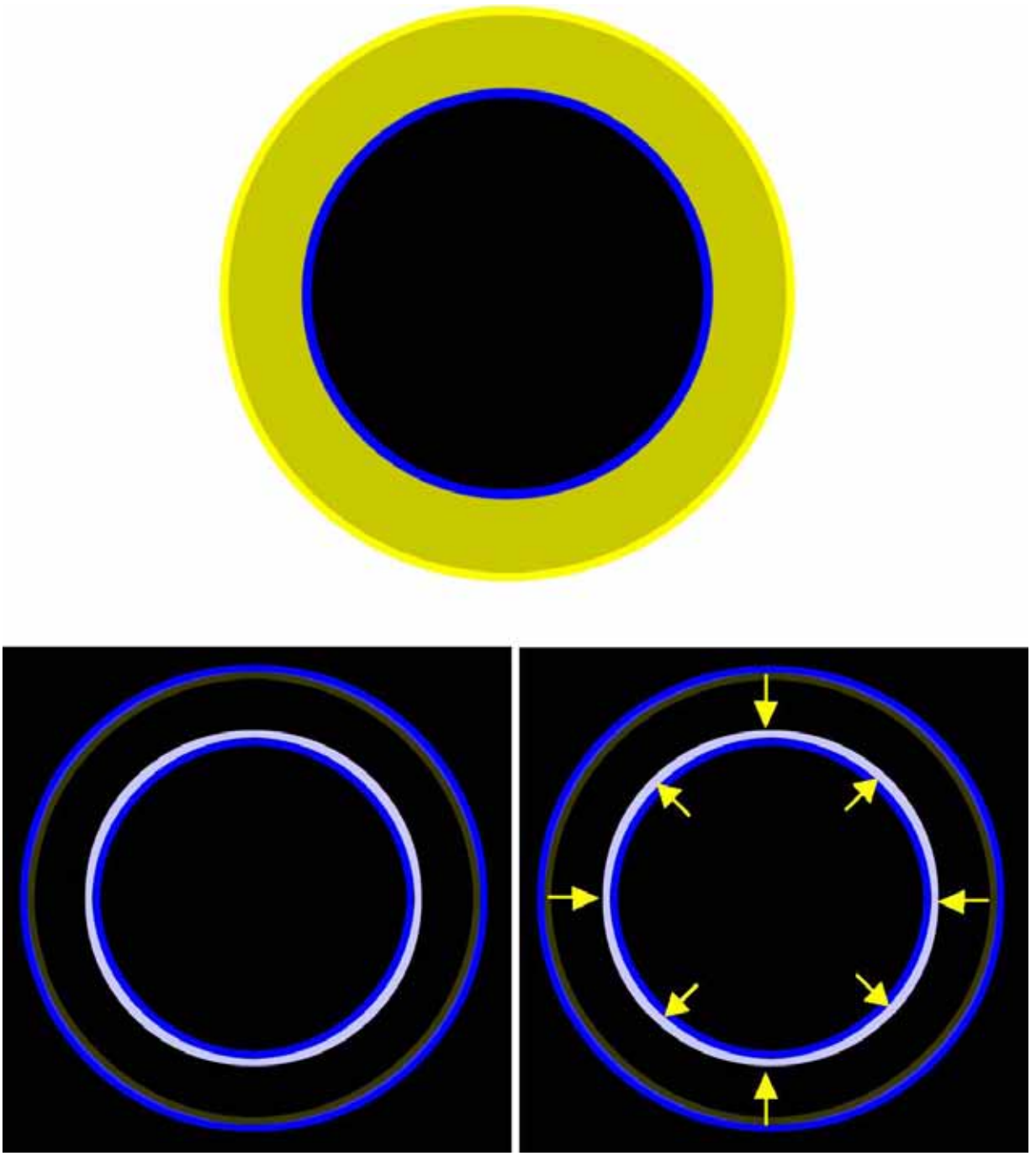

Fig. 61. Expanding pupil [Kitaoka, 2007]. The black circle appears to expand while the outer yellow circle appears to contract (top). Applying the morphological gradient to this illusory image, we obtain the pale image (bottom left) and the illusory motion of pupils (bottom right).

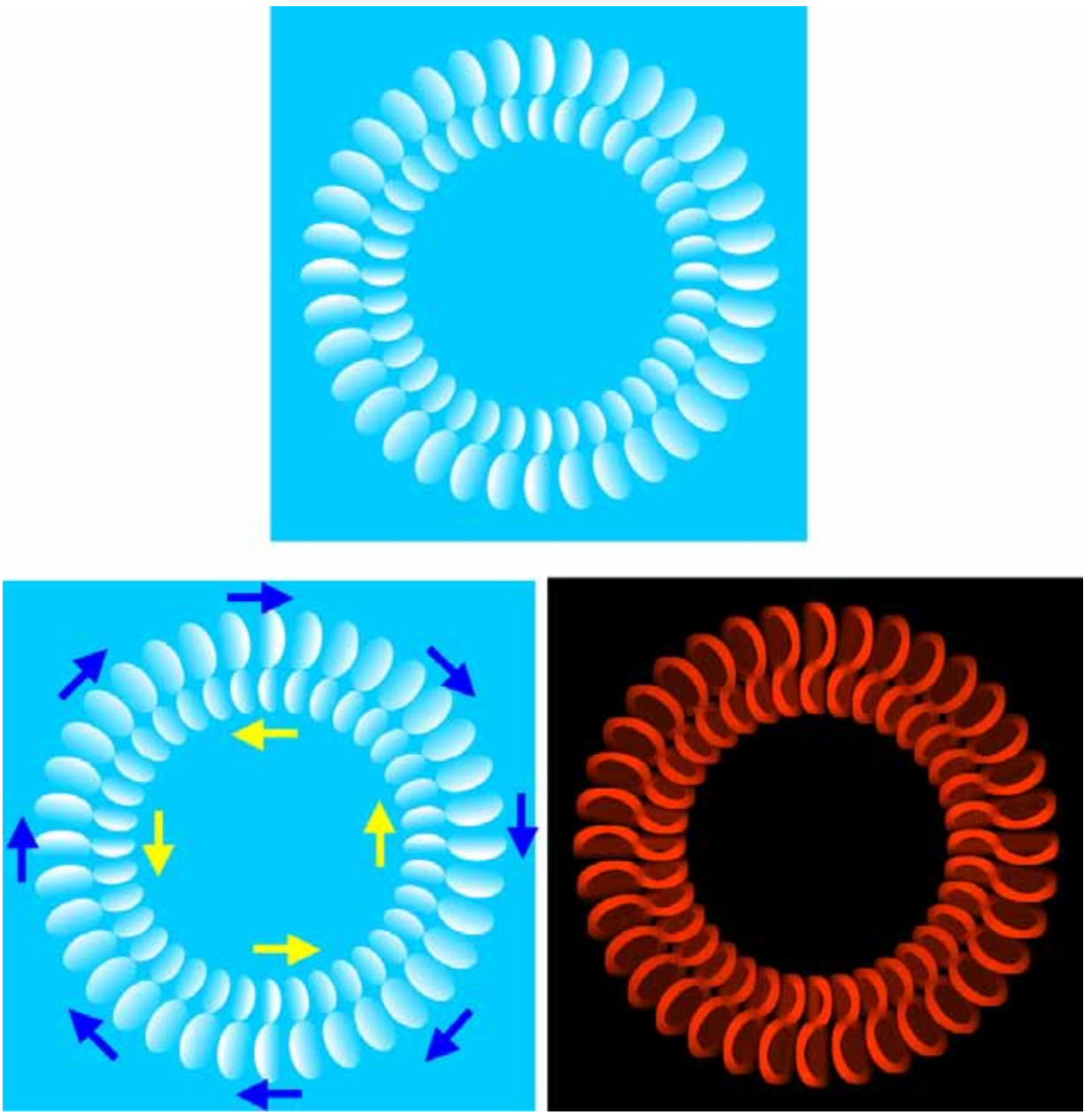

Fig. 62. Rotating shells [Kitaoka, 2007]. The inner ring appears to rotate counterclockwise while the outer one appears to rotate clockwise (top and bottom left). Applying the morphological gradient to this illusory image, we obtain the red ring (bottom right).

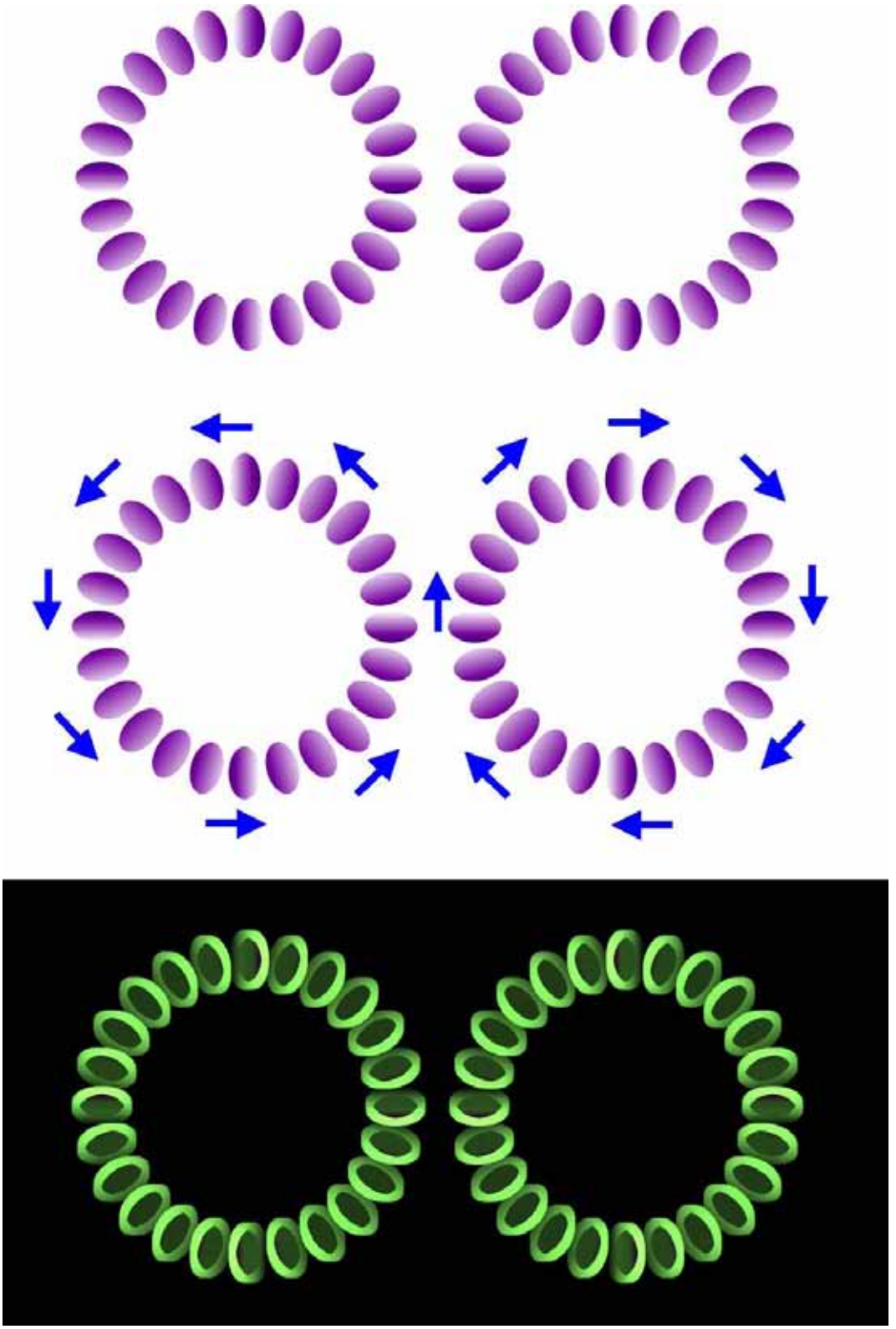

Fig. 63. Rotating purple almonds [Kitaoka, 2007]. The left ring appears to rotate counterclockwise while the right clockwise (top and middle). Applying the morphological gradient to this illusory image, we obtain the green ring (bottom).

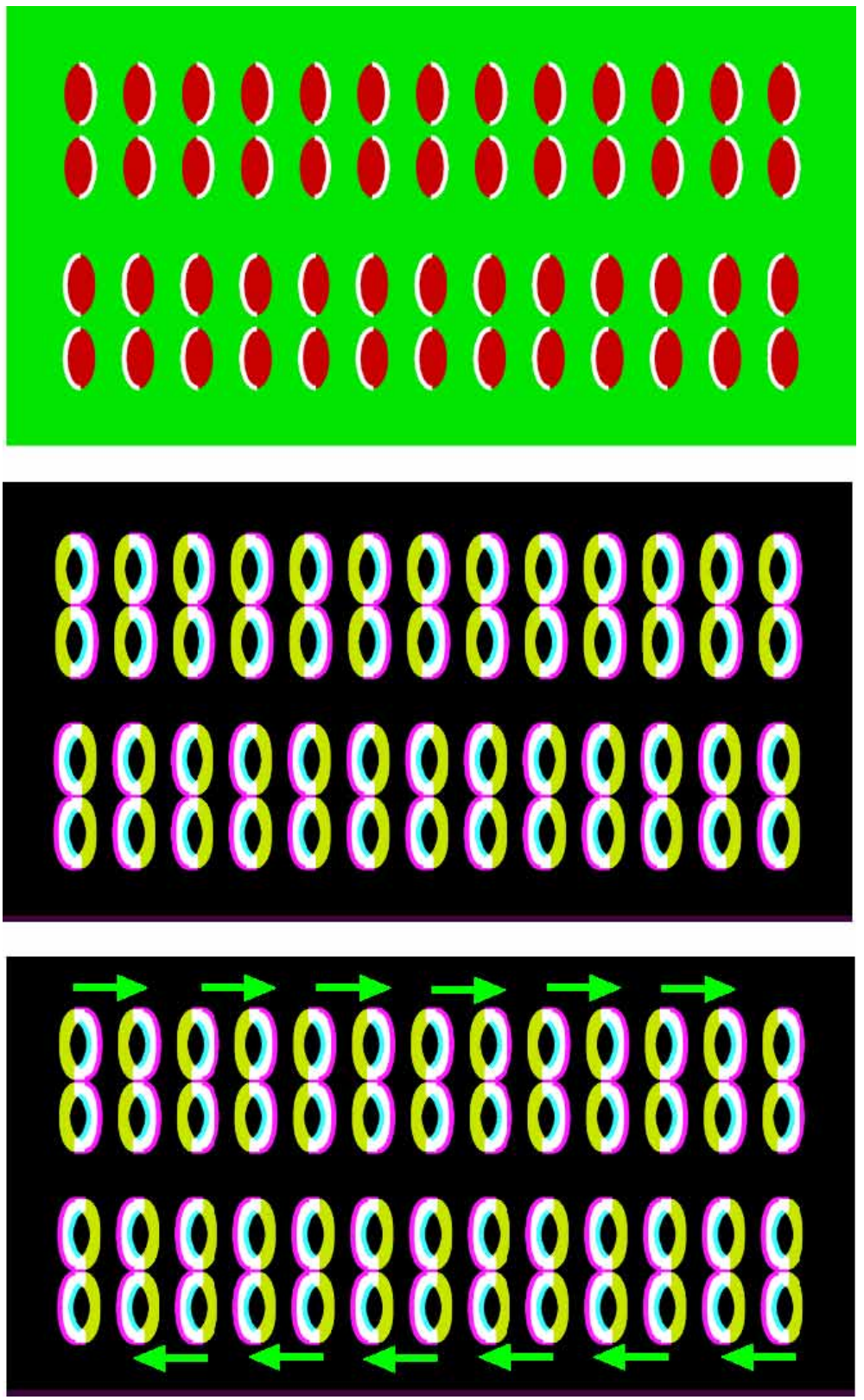

Fig. 64. Bean-jam pancakes [Kitaoka, 2007]. Rows of bean-jam pancakes in the top figure appear to move (top). Applying the morphological gradient to this illusory image, we obtain the bright image (middle) and the illusory motion of bean-jam cakes (bottom).

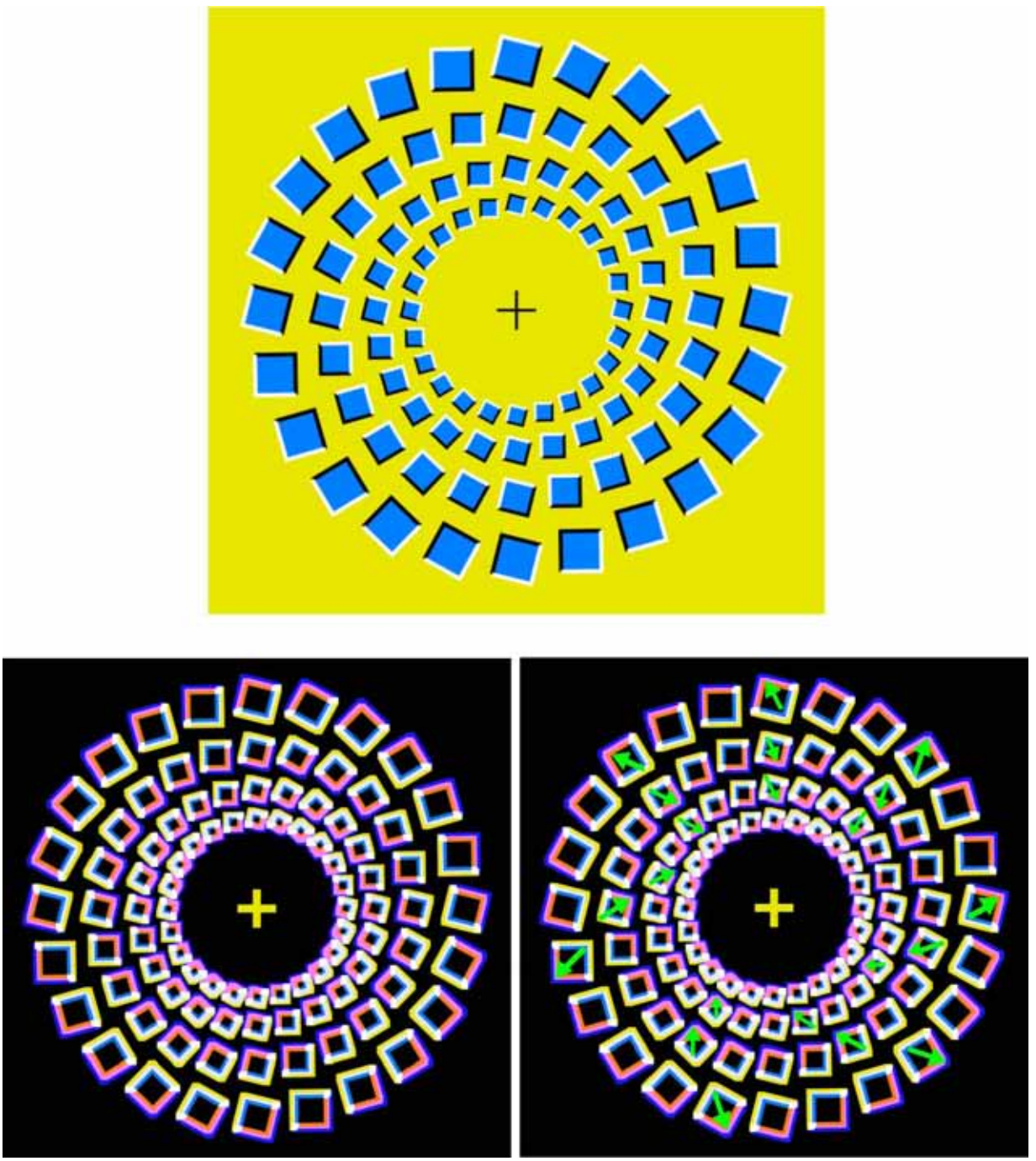

Fig. 65. Falling tornado [Kitaoka, 2007]. The inner three rings appear to contract while the outermost ring appears to expand (top). Applying the morphological gradient to this illusory image, we obtain the bright image (bottom left) and the illusory motion of tornado (bottom right).

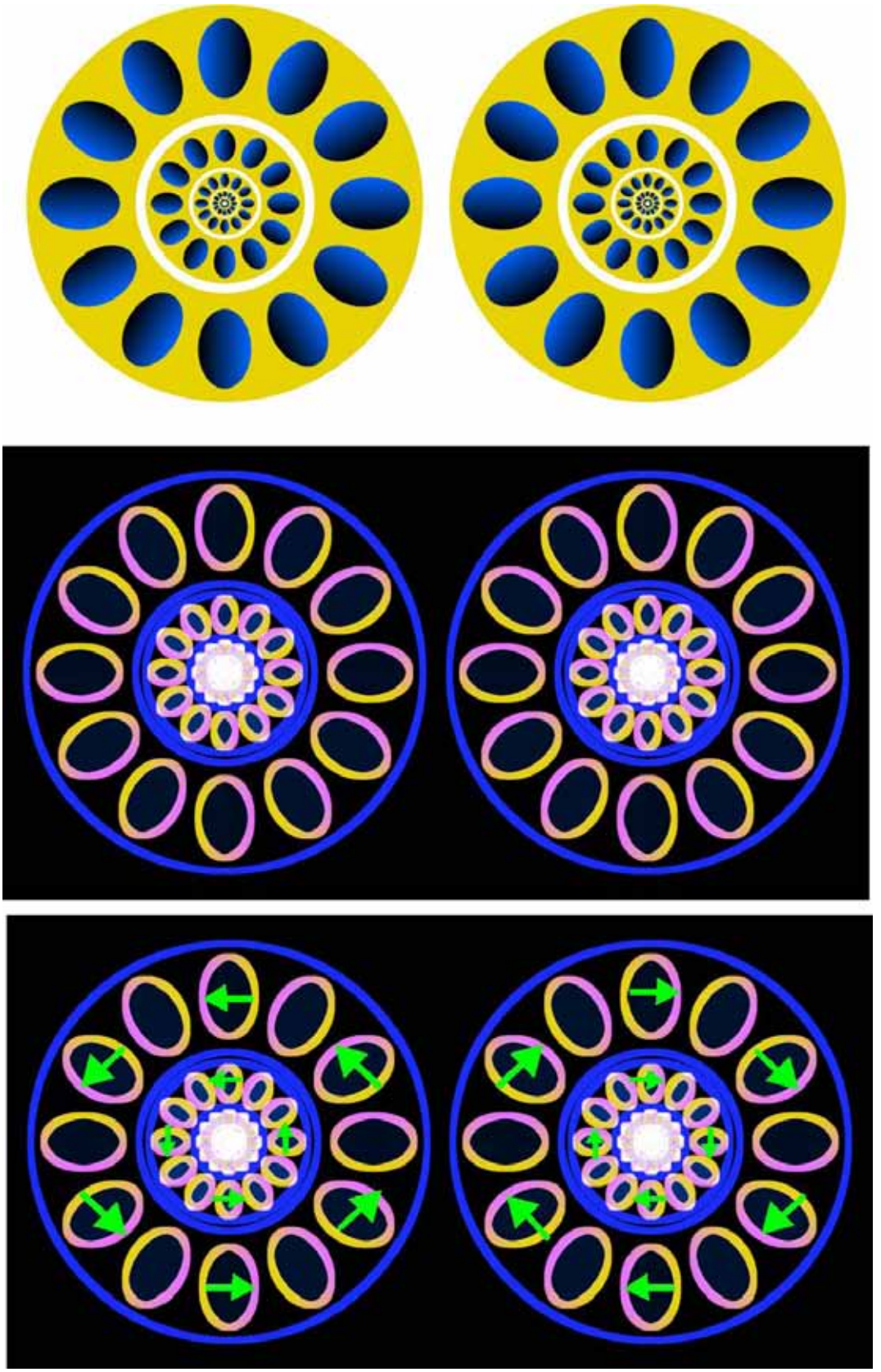

Fig. 66. Hubs [Kitaoka, 2007]. The left rings appear to rotate counterclockwise while the right ones clockwise (top). Applying the morphological gradient to this illusory image, we obtain the bright image (middle) and the illusory motion of hubs (bottom).

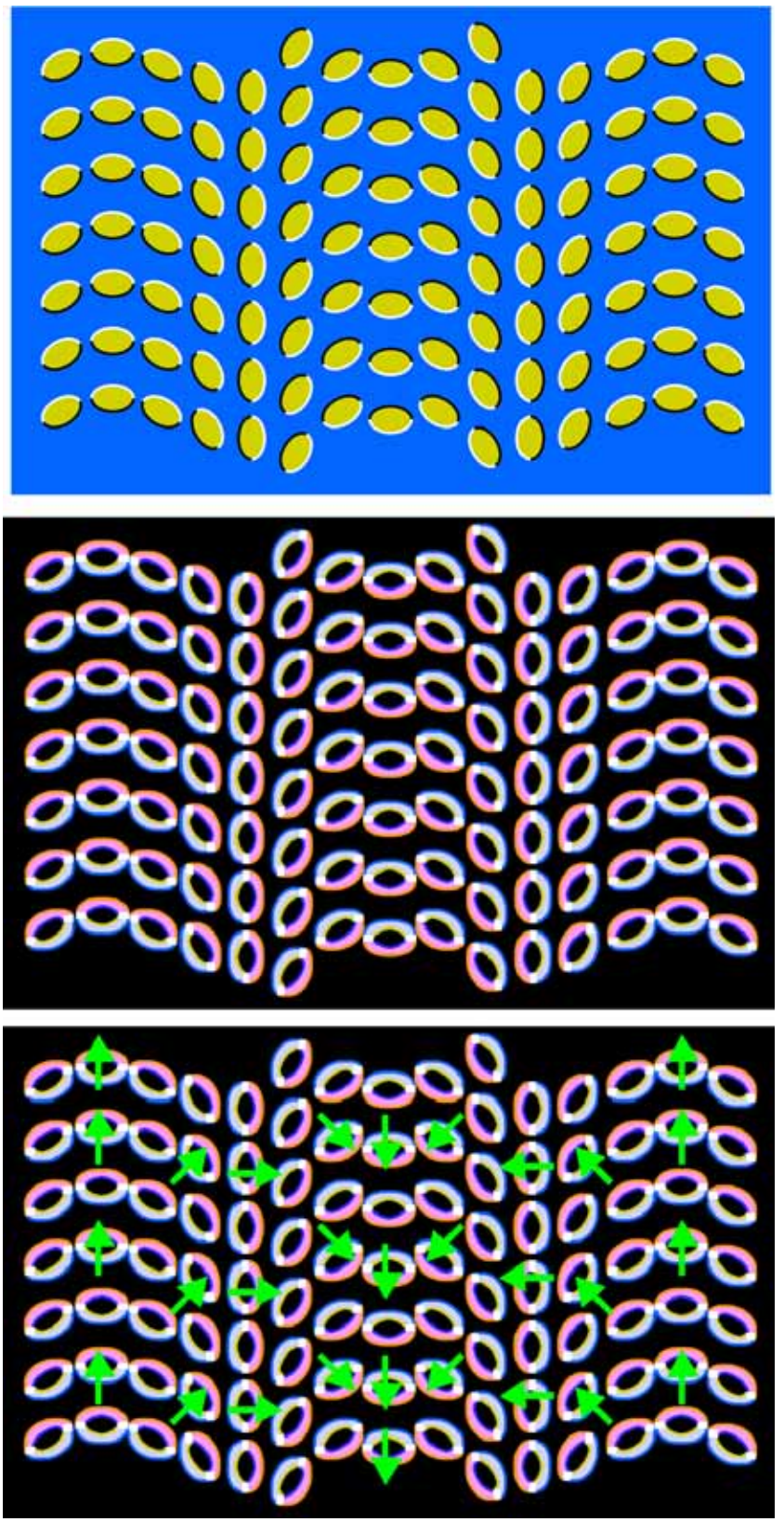

Fig. 67. Cicada-crab goblin [Kitaoka, 2007]. A pattern in the top figure appears to move (top). Applying the morphological gradient to this illusory image, we obtain the bright image (middle) and the illusory motion of a cicada-crab goblin (bottom).

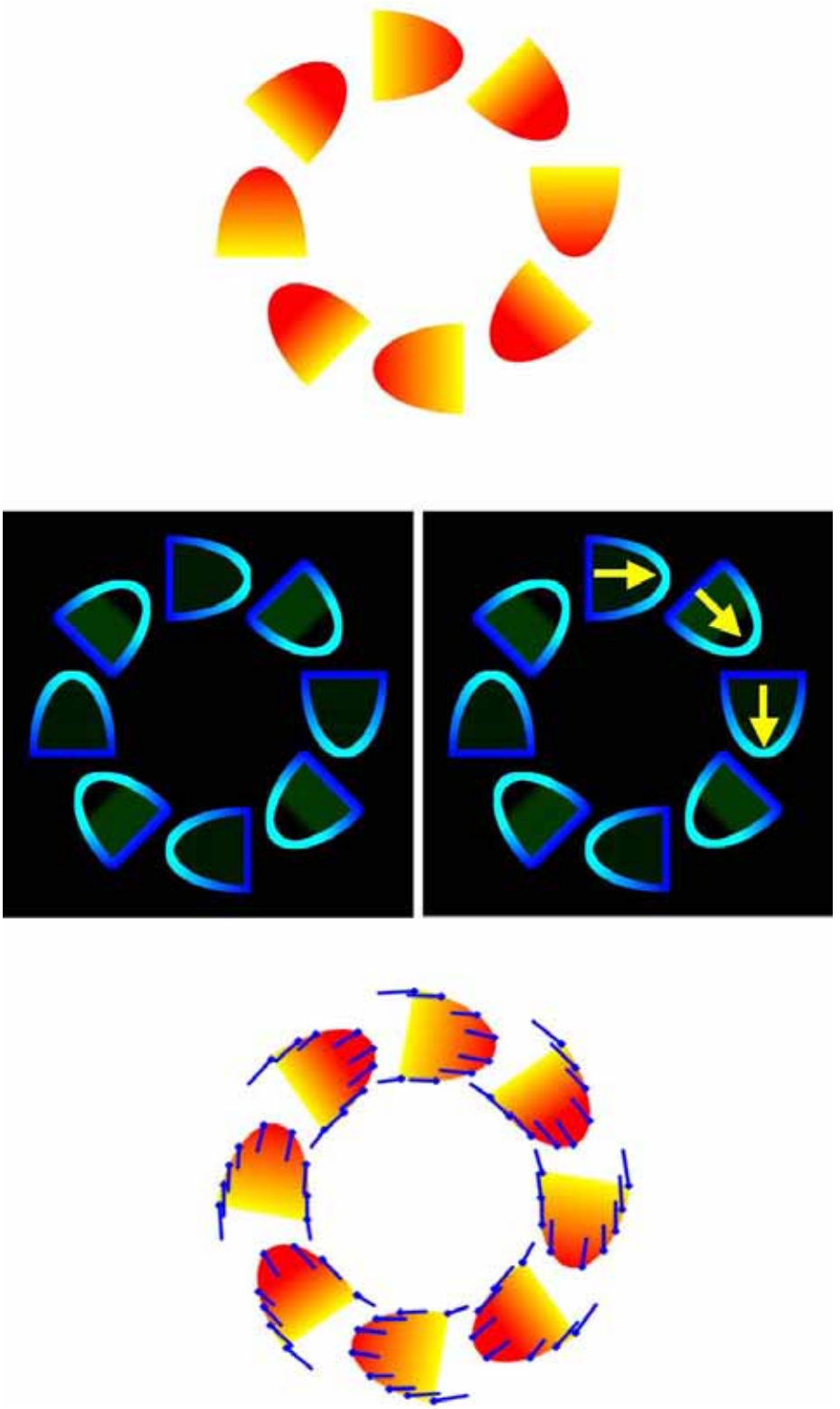

Fig. 68. Optical flow of the central drift illusion. The yellow arrows indicate the illusory motion of carrots (middle right). The cvCalcOpticalFlowPyrLK generates the optical flow of the illusion image (bottom).

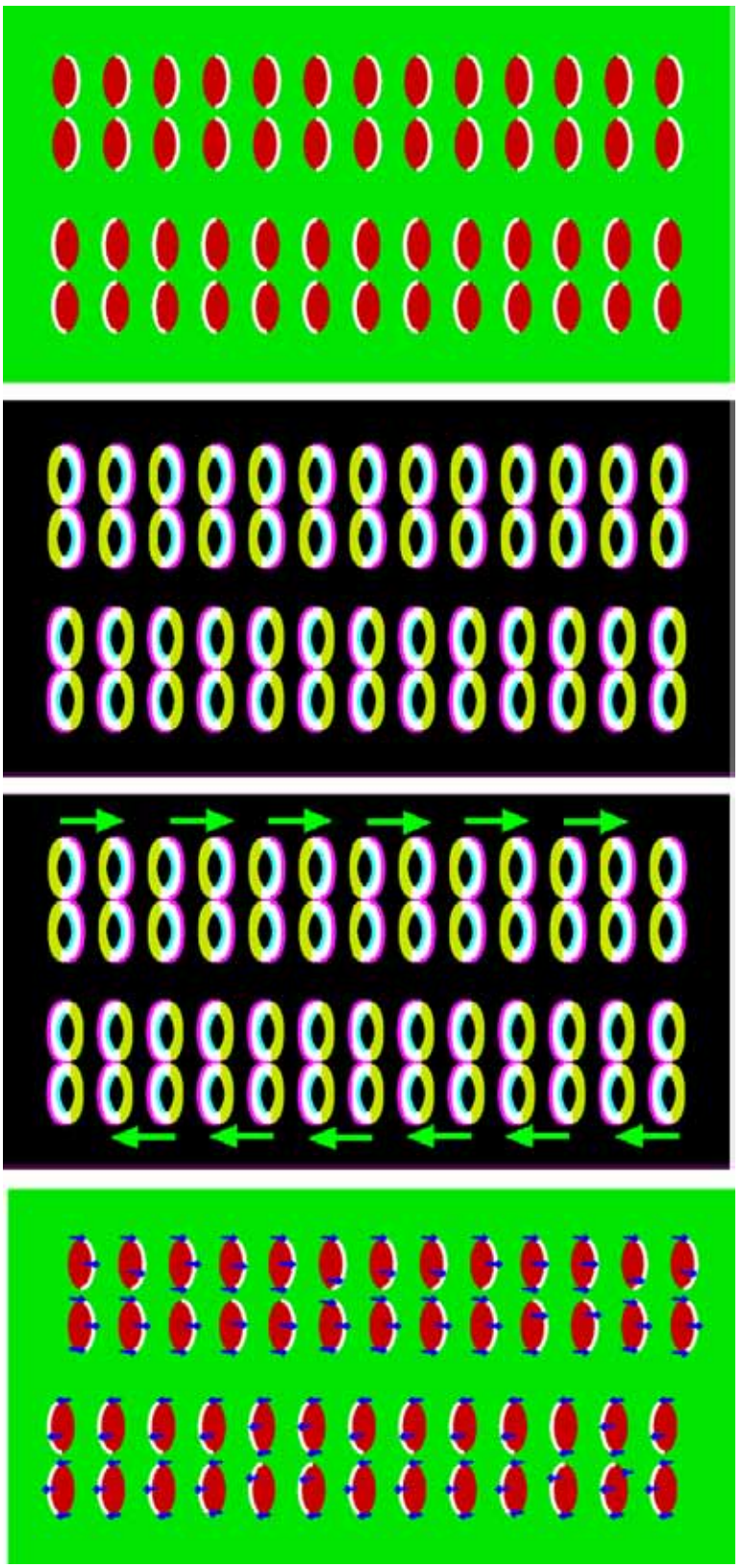

Fig. 69. Optical flow of the peripheral drift illusion. The green arrows indicate the illusory motion of bean-jam cakes (third), which are obtained from the morphological gradient image (second). The cvCalcOpticalFlowPyrLK generates the optical flow of the illusion image (bottom).

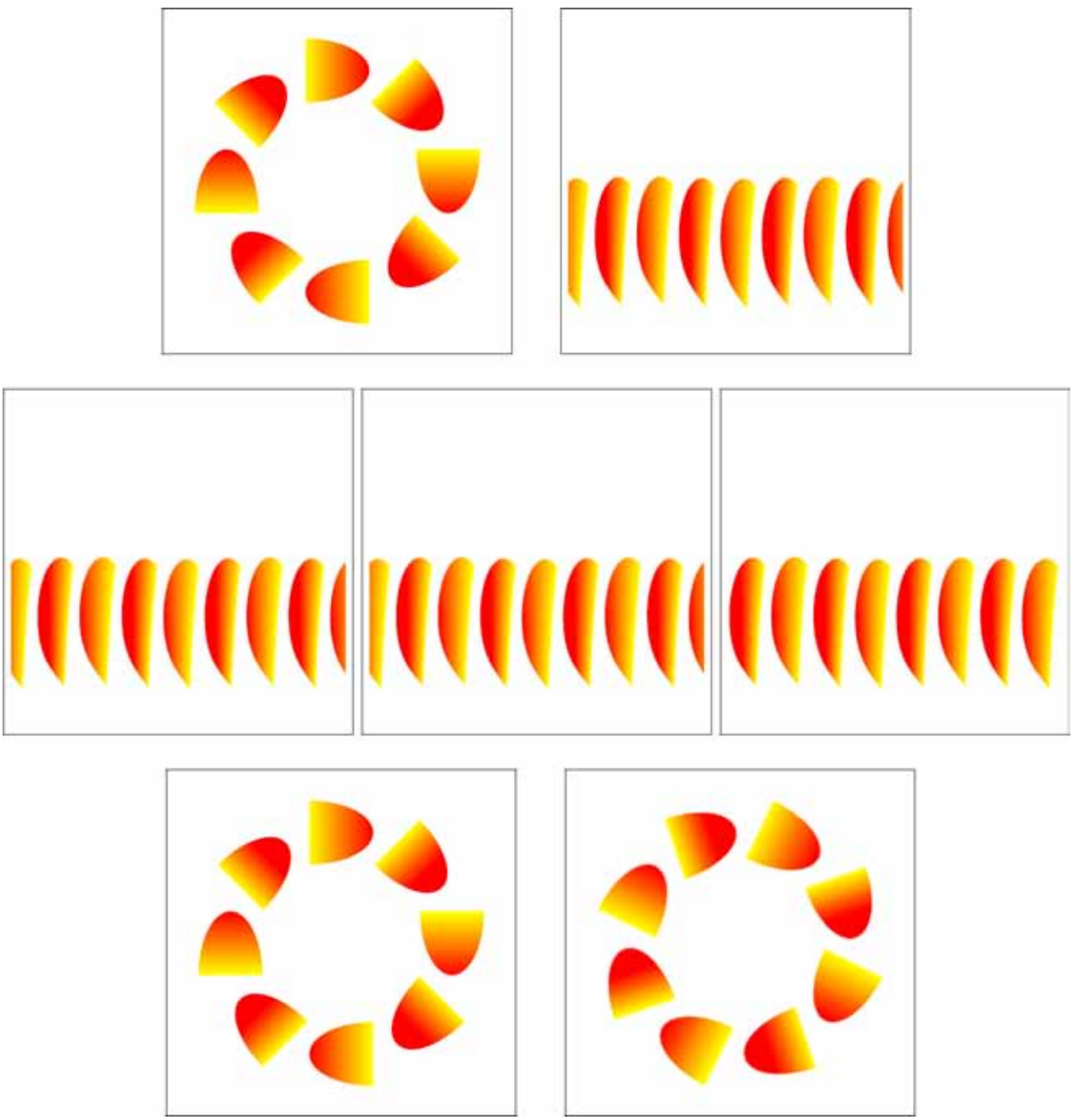

Fig. 70. Imitation of the central drift illusion via the *shift translation* CNN. A ring of carrots is transformed into the (r, θ)-plane (top). Then, the *shift translation* CNN moves a line of carrots to the left, whose input image u_{ij}, initial state $x_{ij}(0)$, and output image y_{ij} are illustrated from left to right (middle). Next, a line of carrots is transformed into the (x, y)-plane (bottom). Observe that the ring of carrots rotates clockwise.

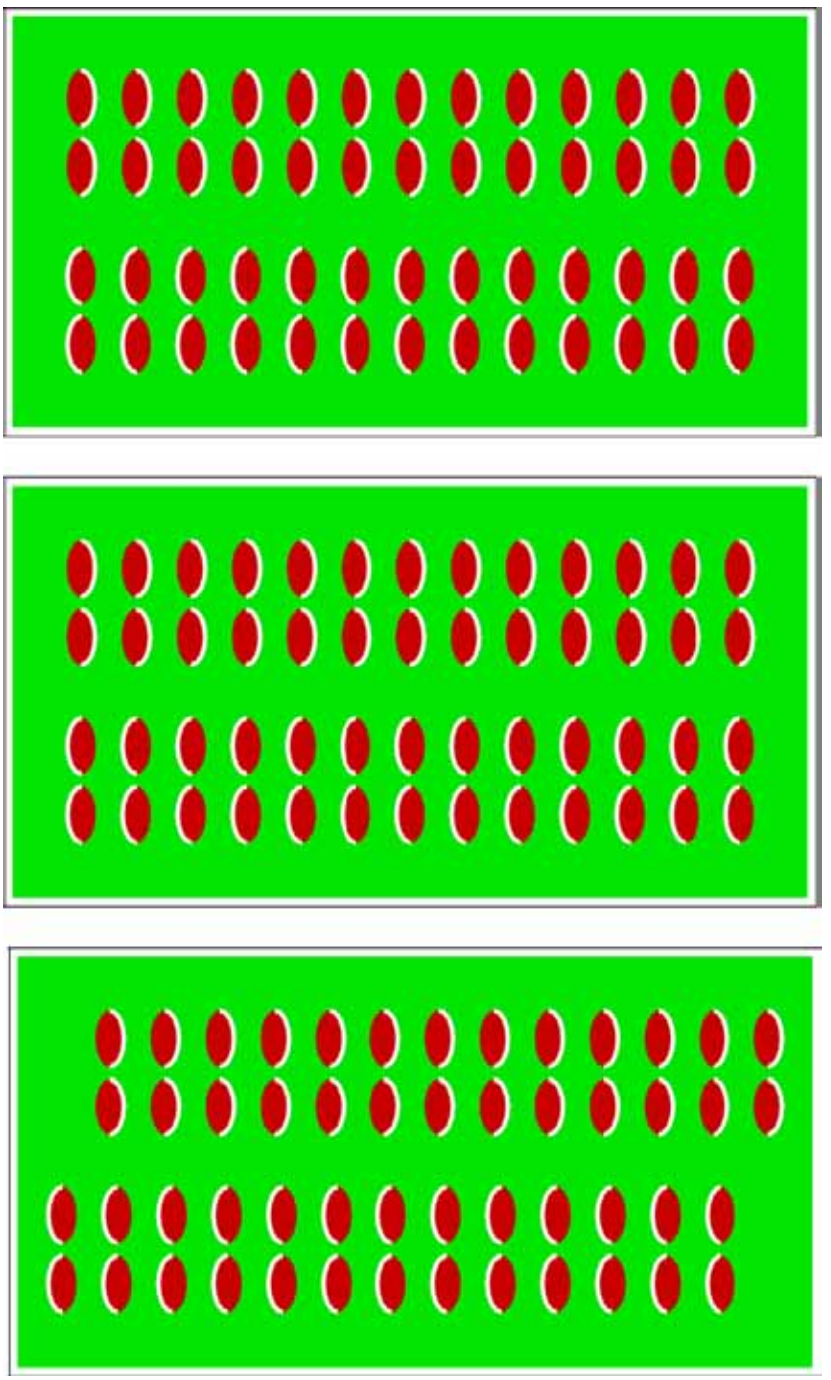

Fig. 71. Imitation of the peripheral drift illusion via the *shift translation* CNN. The *shift translation* CNN moves lines of bean-jam cakes to the right (upper line) and left (lower line), whose input image u_{ij}, initial state $x_{ij}(0)$, and output image y_{ij} are illustrated from top to bottom.

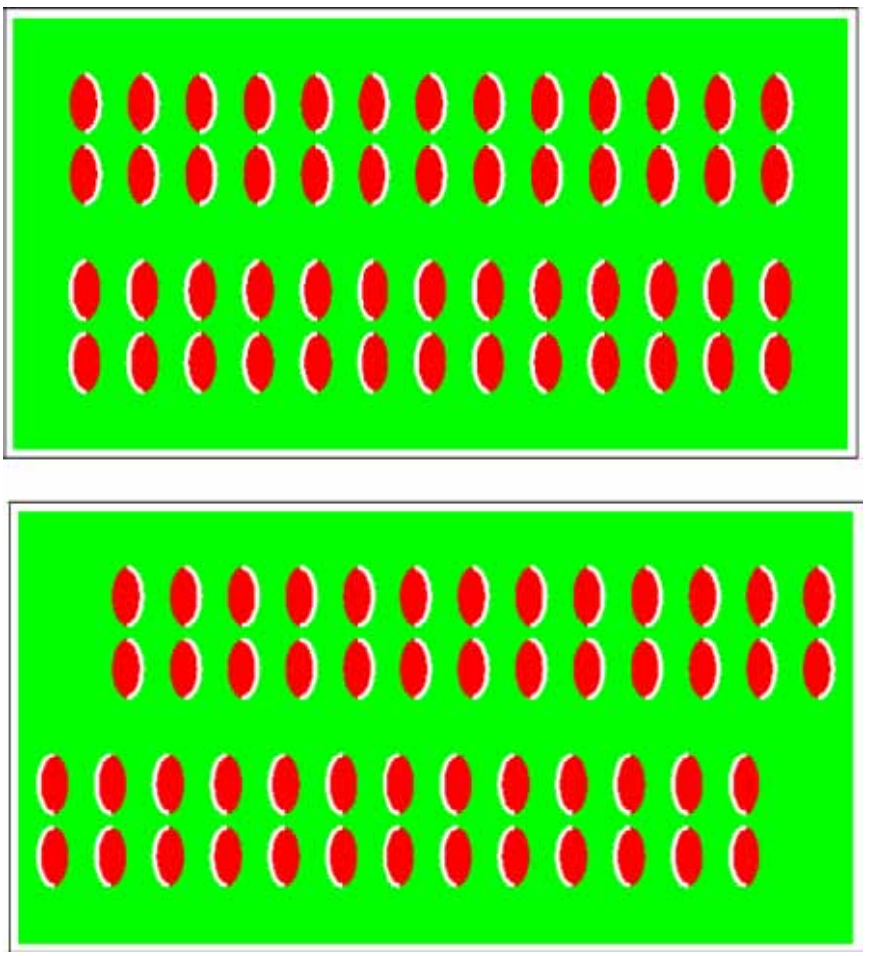

Fig. 72. Imitation of the peripheral drift illusion via the *shift motion* CNN. The *shift motion* CNN can move bean-jam cakes to the right (upper line) and left (lower line) continuously, whose initial state $x_{ij}(0)$ and output image y_{ij} at $t = 50$ are illustrated from top to bottom.

images can also create illusory motion. Furthermore, if each color component of illusory image consists of a binary image, then the *shift motion* CNN can move the bean-jam cakes *continuously* (Fig. 72).

3. The imitation mechanisms are summarized as follows:

Imitation of Anomalous Motion Illusion			
OpenCV	cvCalcOpticalFlowPyrLK	$\Rightarrow$	illusory image
CNN	shift translation or shift motion CNN	$\Rightarrow$	equilibrium state or moving state

5.8. *Glare effect illusion*

In the glare effect illusion, a white target surrounded by luminance gradients appears self-luminous and expands a little. Applying the *morphological gradient* to the glare effect illusion image, we can obtain the direction of glare, which is defined by the vector originating from the dark blue part of the object and terminating to the bright blue or white part (Fig. 73).

1. The cvCalcOpticalFlowPyrLK in the OpenCV can generate the optical flow of glare effect image (Fig. 74).

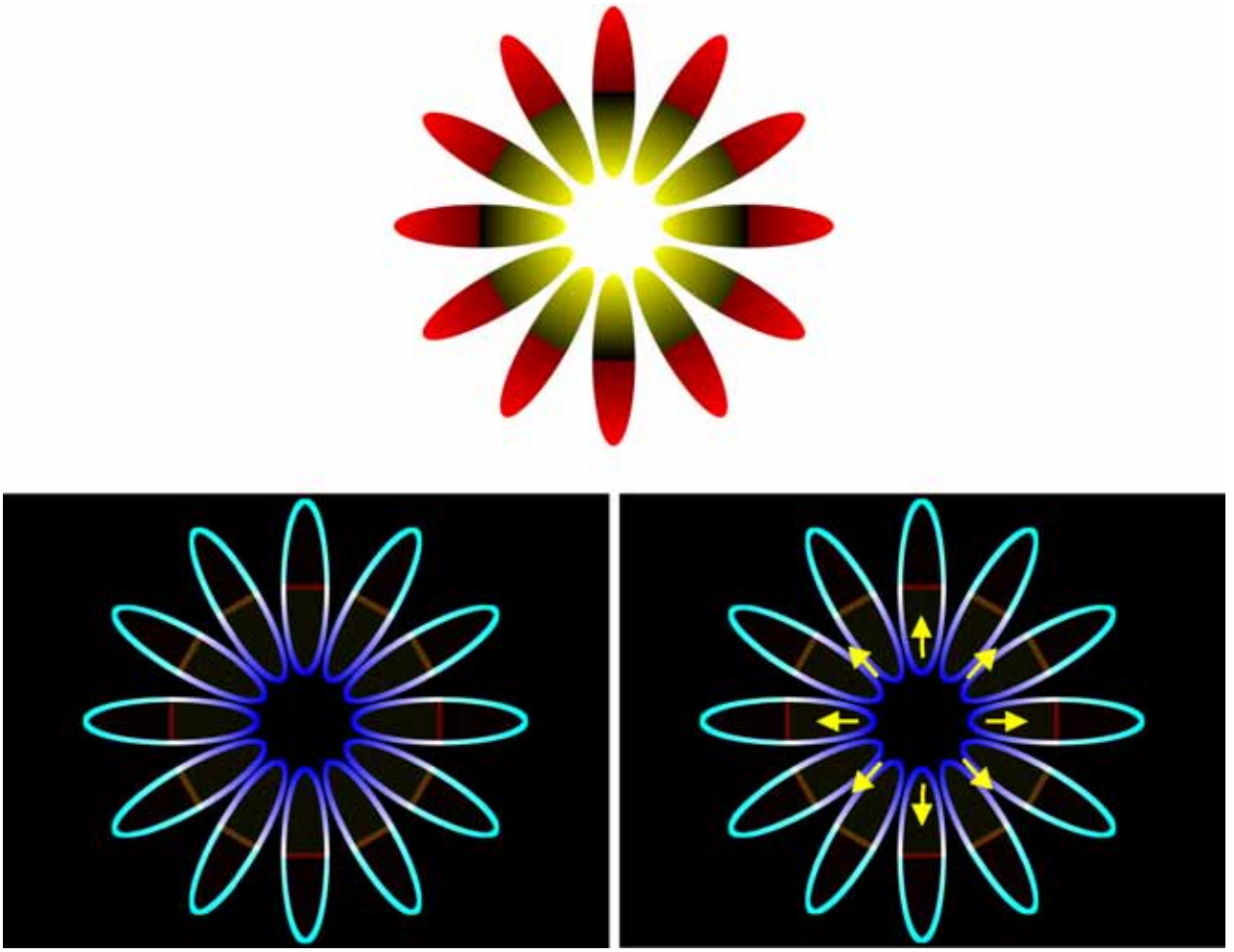

Fig. 73. Glare effect illusion [Kitaoka, 2007]. The white targets surrounded by luminance gradients appear self-luminous and expand a little. Applying the morphological gradient to this illusory image, we obtain the pale image (bottom left) and the glare direction (bottom right).

2. The shift translation CNN can expand a flower a little by using the transformation between the (x, y)-plane and the (r, θ)-plane (Fig. 75).
3. The imitation mechanisms are summarized as follows:

Imitation of Glare Effect Illusion			
OpenCV	cvCalcOpticalFlowPyrLK	$\Rightarrow$	illusory image
CNN	shift translation CNN	$\Rightarrow$	equilibrium state

5.9. *Advantage of using CNN*

The image processing of CNNs is dynamic, however, that of the OpenCV is static, since the CNN is defined by a system of differential equations, and the OpenCV is usually defined by nonlinear or linear functions. By integrating the OpenCV into the CNN image processing, we can use both dynamic and static properties. The advantage of using CNN can be described as follows:

- The transition between an initial image and an illusory image can be observed *visually* by using the CNN simulator.

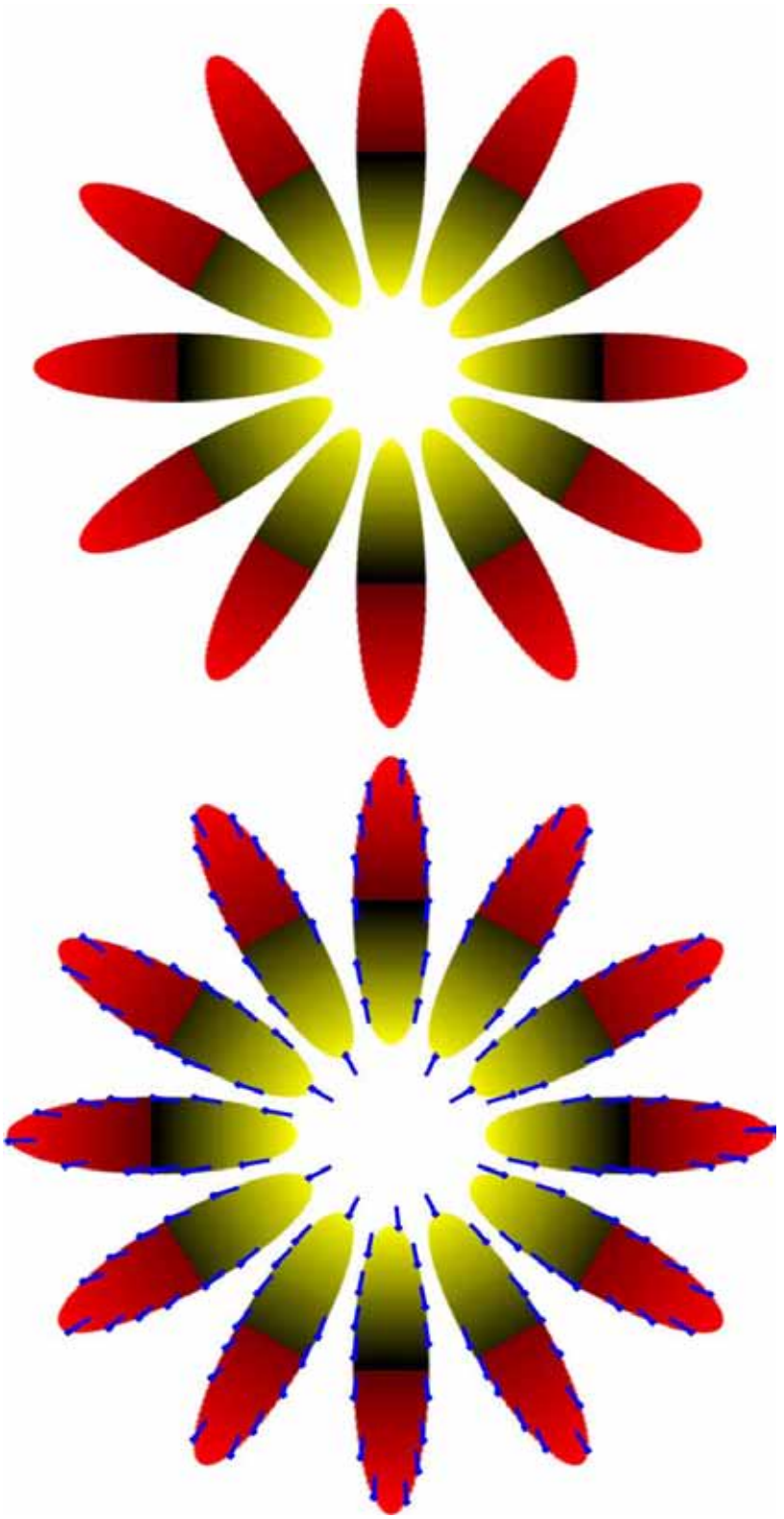

Fig. 74. Optical flow of the glare effect illusion. The cvCalcOpticalFlowPyrLK generates the optical flow of glare effect image. The blue arrows indicate the direction of glare and expansion.

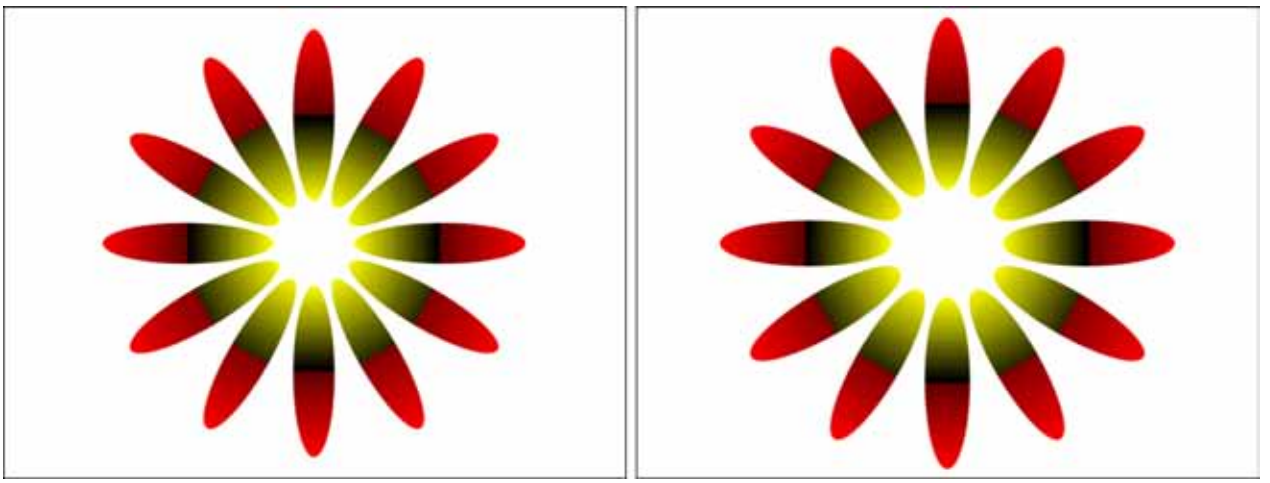

Fig. 75. Imitation the glare effect illusion via the *shift motion* CNN. A flower is expanded by the *shift motion* CNN.

- The CNN can move the illusory image *continuously.*
- The CNN uses only input and initial state images, and multilayered image is not necessary to obtain illusory images.

6. Conclusion

We have imitated the mechanism of Ehrenstein illusion, neon color spreading illusion, watercolor illusion, Kanizsa illusion, shifted edges illusion, hybrid image illusion, anomalous motion illusion, and glare effect illusion using the OpenCV programming functions and the CNN templates.

In this paper, we do not suggest that human illusions are generated by the mechanism of OpenCV and CNNs. However, those imitations suggest that some color illusions are processed by high-level brain functions, and the contrast of the colors plays an important role in the visual illusion. Furthermore, we suppose that some visual illusions may be processed by the illusory movement of the animation.

References

Canny, J. [1986] "A computational approach to edge detection," *IEEE Trans. Patt. Anal. Mach. Intell.* **8**, 679–698.

Chua, L. O. [1998] *CNN: A Paradigm for Complexity* (World Scientific, Singapore).

Chua, L. O. & Roska, T. [2002] *Cellular Neural Networks and Visual Computing* (Cambridge University Press, Cambridge).

Itoh, M. & Chua, L. O. [2003] "Designing CNN genes," *Int. J. Bifurcation and Chaos* **13**, 2739–2824.

Itoh, M. & Chua, L. O. [2007] "Advanced image processing cellular neural networks," *Int. J. Bifurcation and Chaos* **17**, 1109–1150.

Kitaoka, A. [2007] *How is the Brain Deceived? Perfect Demonstration of Visual Illusions* (Newton Press, Tokyo) (in Japanese).

Meyer, F. [1992] "Color image segmentation," *Proc. Int. Conf. Image Processing and its Applications*, pp. 303–306.

Oliva, A., Torralba, A. & Schyns, P. G. [2006] "Hybrid images," *ACM Trans. Graph.* (*Siggraph*) **25**, 527–532.

Pinna, B. & Grossberg, S. [2005] "The watercolor illusion and neon color spreading: A unified analysis of new cases and neural mechanisms," *J. Opt. Soc. Am. A* **22**, 2207–2221.

Werner, J. S., Pinna, B. & Spillmann, L. [2007] "Illusory color and the brain," *Sci. Amer.* **296**, 70–75.