**Task Prescription:** Extract from a binary image fed to CNN initial states, those regions which contain the boundary and which completely filled the interior of all closed curves of a binary image loaded to CNN inputs.

**Cloning Template**

<table>
<thead>
<tr>
<th>z:</th>
<th>-5.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>B:</td>
<td>0 0 0</td>
</tr>
<tr>
<td></td>
<td>0 2 0</td>
</tr>
<tr>
<td></td>
<td>0 0 0</td>
</tr>
<tr>
<td>A:</td>
<td>0 1 0</td>
</tr>
<tr>
<td></td>
<td>1 5 1</td>
</tr>
<tr>
<td></td>
<td>0 1 0</td>
</tr>
</tbody>
</table>

**Boundary Condition**

Fixed: \( x_{i*j*} = 0, \ u_{i*j*} = 0 \)

\((i*j* denotes boundary cells)\)

**Initial State**

\(x_{ij}(0) = a given binary image\)

**Example 1:** Array Size = 16 x 32

**Input Image**

**Initial State**

**Output Image**

**Example 2:** Array Size = 300 x 300
GLOBAL CONNECTIVITY DETECTION CNN

Determine whether a given geometric pattern is “globally” connected in one contiguous piece, or is it composed of two or more disconnected components. An object or component is any connected set in red pseudo color. Deletes marked objects in binary image. An object is marked by changing at least one red pixel into blue in the initial state. The output contains the unmarked objects only.

**Cloning Template**

\[ \begin{array}{ccc}
  z : & -4.5 & \\
  B : & -0.5 & 3 & -0.5 \\
  A : & 0 & 0.5 & 0 \\
  0 & -0.5 & 0 \\
\end{array} \]

**Boundary Condition**

Fixed: \( x_{i*j*} = -1, \ u_{i*j*} = -1 \)

\( (i*j* \text{ denotes boundary cells}) \)

**Initial State**

\( x_{ij}(0) = \text{marked objects} \)

---

**Example 1:** Array Size = 100 x 200

- **Input Image**
- **Initial State**
- **Output Image**

---

**Example 2:** Array Size = 200 x 500
**HOLE-FILLING CNN**

*Fill the interior of all closed contours in a binary image.*

**Task Prescription:**

Fill the interior of all closed contours in a binary image.

**Cloning Template**

<table>
<thead>
<tr>
<th>z</th>
<th>B</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>0 0 0</td>
<td>0 1 0</td>
</tr>
<tr>
<td>0 4 0</td>
<td>1 3 1</td>
<td></td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 1 0</td>
<td></td>
</tr>
</tbody>
</table>

**Boundary Condition**

Fixed: \( x_{i*j*} = 0, \ u_{i*j*} = 0 \)

\((i*j*) \text{ denotes boundary cells}\)

**Initial State**

\(x_{ij}(0) = 1 \) (red in pseudo-color)

**Example 1:** Array Size = 30 x 48

**Input Image**

[Image of input image]

**Initial State**

[Image of initial state]

**Output Image**

[Image of output image]

**Example 2:** Array Size = 300 x 300

[Image of example 2]
**Name:** FACE-VASE ILLUSION CNN

**Task** Simulate the well-known visual illusion where the following input image is perceived either as two **symmetric faces**, or as a **vase**, depending on the initial thought or attention, which is simulated by specifying a small patch of black (red in pseudo-color) pixels inside the object to be picked out. As usual, “red” corresponds to object and “blue” corresponds to background.

---

**Cloning Template**

<table>
<thead>
<tr>
<th>z :</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>B :</td>
<td>0 0 0</td>
</tr>
<tr>
<td></td>
<td>0 -5 0</td>
</tr>
<tr>
<td></td>
<td>0 0 0</td>
</tr>
</tbody>
</table>

| A : | 0 1 0|
|     | 1 2 1|
|     | 0 1 0|

**Boundary Condition**

- Fixed: $x_{i^*j^*} = -1$, $u_{i^*j^*} = -1$  
  (i*j* denotes boundary cells)

**Initial State**

$x_{ij}(0) = a$ given binary image

---

**Example 1:** Array Size = 200 x 400

- **Input Image**
- **Output Image**

**Example 2:** Array Size = 200 x 400

- **Initial State**
HALF-TONING CNN

Transform a gray-scale image into a “half-tone” binary image. The binary image preserves the main features of the gray-scale image.

**Cloning Template**

<table>
<thead>
<tr>
<th>z :</th>
<th>B :</th>
<th>A :</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.1</td>
<td>0.1</td>
</tr>
<tr>
<td>0.1</td>
<td>0.32</td>
<td>0.1</td>
</tr>
<tr>
<td>0.07</td>
<td>0.1</td>
<td>0.07</td>
</tr>
</tbody>
</table>

**Boundary Condition**

Fixed: $x_{i*j*} = 0$, $u_{i*j*} = 0$

(i*j* denotes boundary cells)

**Initial State**

$x_{ij}(0) = a$ given image

---

**Example 1: Array Size = 30 x 48**

**Input Image**

**Initial State**

**Output Image**

---

**Example 2: Array Size = 200 x 200**
**INVERSE HALF-TONING CNN**

(Convert binary images into gray-scale images while preserving their average gray-levels.)

### Cloning Template

<table>
<thead>
<tr>
<th>z :</th>
<th>B :</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.1</td>
</tr>
<tr>
<td>0.1</td>
<td>0.32</td>
</tr>
<tr>
<td>0.07</td>
<td>0.1</td>
</tr>
<tr>
<td>0.07</td>
<td>0.1</td>
</tr>
</tbody>
</table>

### Boundary Condition

Fixed: \( x_{i^*j^*} = 0, \quad u_{i^*j^*} = 0 \)  
(i\(^*j^*\) denotes boundary cells)

### Initial State

\( x_{ij}(0) = 0 \) (white in pseudo-color)

### Example 1: Array Size = 30 x 48

**Input Image**

**Initial State**

**Output Image**

### Example 2: Array Size = 200 x 200
TEXTURE DISCRIMINATION CNN

Name:
Task
Discriminate between textures which have the same flat gray-level histogram.
Prescription: The output is a binary image where different average gray-levels identify different textures.

Cloning Template

<table>
<thead>
<tr>
<th>z</th>
<th>1.8</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>0.16 -1.56 1.25</td>
</tr>
<tr>
<td></td>
<td>-2.89 1.09 -3.2</td>
</tr>
<tr>
<td></td>
<td>4.06 4.69 3.75</td>
</tr>
</tbody>
</table>

| A | 0.86 0.94 3.75 |
|   | 2.11 -2.81 3.75 |
|   | -1.33 -2.58 -1.02 |

Boundary Condition

Fixed: \( x_{i*j*} = 0, \ u_{i*j*} = 0 \)
(i*j* denotes boundary cells)

Initial State

\( x_{ij}(0) = u_{ij} = \) input image

Example 1: Array Size = 64 x 128

Input Image

Initial State

Output Image

Example 2: Array Size = 200 x 256
**Generate a histogram (by projecting all pixels to the left) for each row of a given binary image.**

### Boundary Condition

- **Fixed:** $x_{i*j*} = 0$, $u_{i*j*} = 0$
- *(i*j*) denotes boundary cells*

### Initial State

- $x_{ij}(0) = a$ given binary image

---

**Example 1:** Array Size = 30 x 48

| G | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | a | 1 | b | 0 | 0 | 0 |

- $a = -1.5 \text{sgn}(y_{ij} - y_{kl} - 1.5) - 1.5$
- $b = -1.5 \text{sgn}(y_{ij} - y_{kl} + 1.5) + 1.5$

**Input Image: arbitrary**

**Initial State**

**Output Image**

---

**Example 2:** Array Size = 256 x 256
Determine whether the number of black (red in pseudo-color) neighbors of every pixel $C_{i,j}$, including pixel $C_{i,j}$ itself, is odd, or even. Color $C_{i,j}$ black (red in pseudo-color) if the parity is odd, and white (blue in pseudo-color) if even.

**Input Image**

**Initial State**

**Output Image**

**Example 1:** Array Size = 7 x 7

**Example 2:** Array Size = 101 x 101
**Name:** ROW-WISE PARITY DETECTION CNN

**Task:** Determine whether the number of black pixels in a row is even or odd. If the parity of all pixels located to the right of any pixel “c_{ij}” (including pixel “c_{ij}” itself) is even, then pixel “c_{ij}” is coded white (blue in pseudo-color), otherwise it is coded black (red in pseudo-color).

**G:**

\[
\begin{array}{cccccccccccc}
0 & 0 & 0 & 0 & 0 & c & 0 & 0 & 0 & 0 & 0 & 0 & a & b & 0 & 0 & 0 \\
\end{array}
\]

\[
a = 1.1 \text{sgn}(y_{ij} + 0.7) - 1.1 \text{sgn}(y_{ij} + 0.1) + 2 \text{sgn}(y_{ij} - 0.1) - 2 \text{sgn}(y_{ij} -0.3) + 2 \text{sgn}(y_{ij} - 0.7)
\]

\[
b = - 0.35 \text{sgn}(y_{kl} + 0.7) + 0.3 \text{sgn}(y_{kl} + 0.1) + 0.1 \text{sgn}(y_{kl} - 0.1) - 0.1 \text{sgn}(y_{kl} - 0.3) + 0.05 \text{sgn}(y_{kl} - 0.7)
\]

\[
c = 0.1 \text{sgn}(u_{ij} -0.1)
\]

**Boundary Condition**

Fixed: \(x_{i*j*} = 0, \ u_{i*j*} = 0\)  
(\(i*j*\) denotes boundary cells)

**Initial State**

\(x_{ij}(0) = -0.5\) (light blue in pseudo-color)

**Example 1:** Array Size = 30 x 48

**Example 2:** Array Size = 64 x 64