Chapter 4

Convolution

Searching for patterns in time and space makes the pattern recognition task you studied in Chapter 1 more challenging. Maybe you, for instance, would like the Tsetlin machine to recognize objects inside an image. Before the Tsetlin machine can learn their appearance, it must locate them. But without knowing their appearance in the first place, how can they be found? In this chapter, you discover how the Tsetlin machine can solve this two-sided problem using *convolution* with rules.

In Section 4.1, you study two illustrative tasks within health and image analysis. They capture the dual nature of the problem and why you need to perform localization and learning together.

Then, you learn how to divide an image into multiple patches in Section 4.2. The division allows the Tsetlin machine to focus on one image piece at a time, giving it a way to direct its attention.

Multiple image pieces require a new approach to evaluating and learning rules. When each input image turns into several parts, you need a strategy for selecting which part to focus on and which to ignore. We cover the new form of rule evaluation in Section 4.3, while Section 4.4 addresses learning.

Finally, Section 4.5 teaches how to use a patch's position inside the image to create more precise rules. The purpose is to narrow down the pattern matching to relevant image regions.

After reading this chapter, you will know how to build a convolutional Tsetlin machine that can recognize patterns in time and space.

4.1 Recognizing Patterns in Time and Space

Recognizing patterns in time and space can be useful in many applications.

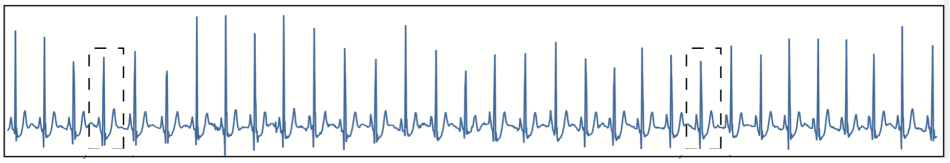


Figure 4.1: An electrocardiogram (ECG) for detecting heart disease (from Zhang et al., 2023).

Detecting Heart Disease. Figure 4.1 illustrates one example. It depicts an electrocardiogram (ECG) that captures the heart's electrical activity, recorded by electrodes on the limbs and chest of the patient. Notice the dotted rectangles in the figure. Inside each rectangle, you see an ECG waveform. Some of the waveforms may reveal heart disease. However, you do not know which ones. Accordingly, the task of the Tsetlin machine is to scan through the ECG, simultaneously locating and learning the waveform shapes that indicate disease.

Image Analysis. Recognizing flowers in an image is another example (Figure 4.2). Using the entire image as input makes the task unnecessarily difficult. The flowers can appear in different places, and each position gives different results. The Tsetlin machine from Chapter 1 must then learn how the flowers appear in all possible locations. Now, consider instead the image content inside the yellow rectangle. By moving the rectangle around, the Tsetlin machine can scan for flowers. It must then learn to skip the background while recognizing when the rectangle encloses flowers. In this manner, it can capture their appearance, regardless of location.

Figure 4.2: A flower in a field.

4.2 Input and Patterns

The key to overcoming the above challenge is to use *convolution* with rules, which you will learn to do here.

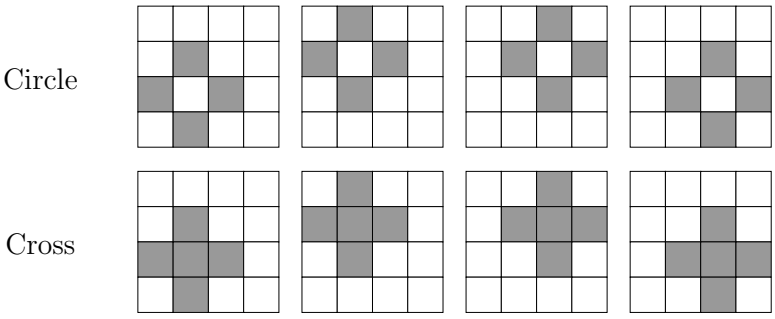


Table 4.1: Eight black-and-white images from the classes Circle and Cross.

Example Images. The eight black-and-white images in Table 4.1 are instructive for understanding rule-based convolution. Those in the first row contain circles. The second row of images contains crosses. So, you have two classes: Circle and Cross. Notice how the two shapes appear in various

locations. Accordingly, Table 4.1 demonstrates the essence of convolution — *the need to scan for smaller patterns within a larger space.*

Boolean Features. Your first step is to make Boolean features out of the images. To do that, give each image pixel a truth value, *True* for black and *False* for white. You then get one feature per pixel. Table 4.2 organizes these into four rows and four columns according to the location of the pixels. In this manner, you end up with $4 \times 4 = 16$ features. Name each $X_{i,j}$ where i denotes the pixel row and j the column. This structure is convenient later when we dive into convolution.

Brute Force Solution. A brute force solution could learn the locations of Circle and Cross separately, giving eight rules. An example of this is rule **R1** below. It recognizes Circle in the upper top left corner of the image.

R1: if $X_{1,2}$ **and** $X_{2,1}$ **and not** $X_{2,2}$ **and** $X_{2,3}$ **and** $X_{3,2}$ **then** *Circle.*

$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	$X_{1,4}$
$X_{2,1}$	$X_{2,2}$	$X_{2,3}$	$X_{2,4}$
$X_{3,1}$	$X_{3,2}$	$X_{3,3}$	$X_{3,4}$
$X_{4,1}$	$X_{4,2}$	$X_{4,3}$	$X_{4,4}$

Table 4.2: Black-and-white image described with 16 Boolean features.

	$X_{1,2}$	
$X_{2,1}$	$\overline{X}_{2,2}$	$X_{2,3}$
	$X_{3,2}$	

R1

	$X_{1,3}$	
$X_{2,2}$	$\overline{X}_{2,3}$	$X_{2,4}$
	$X_{3,3}$	

R2

	$X_{2,2}$	
$X_{3,1}$	$\overline{X}_{3,2}$	$X_{3,3}$
	$X_{4,2}$	

R3

	$X_{2,3}$	
$X_{3,2}$	$\overline{X}_{3,3}$	$X_{3,4}$
	$X_{4,3}$	

R4

Table 4.3: Condition part of rules R1, R2, R3, and R4. Together, the rules captures the Circle pattern at each corner of the image.

$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	$X_{1,4}$
$X_{2,1}$	$X_{2,2}$	$X_{2,3}$	$X_{2,4}$
$X_{3,1}$	$X_{3,2}$	$X_{3,3}$	$X_{3,4}$
$X_{4,1}$	$X_{4,2}$	$X_{4,3}$	$X_{4,4}$

$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	$X_{1,4}$
$X_{2,1}$	$X_{2,2}$	$X_{2,3}$	$X_{2,4}$
$X_{3,1}$	$X_{3,2}$	$X_{3,3}$	$X_{3,4}$
$X_{4,1}$	$X_{4,2}$	$X_{4,3}$	$X_{4,4}$

$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	$X_{1,4}$
$X_{2,1}$	$X_{2,2}$	$X_{2,3}$	$X_{2,4}$
$X_{3,1}$	$X_{3,2}$	$X_{3,3}$	$X_{3,4}$
$X_{4,1}$	$X_{4,2}$	$X_{4,3}$	$X_{4,4}$

$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	$X_{1,4}$
$X_{2,1}$	$X_{2,2}$	$X_{2,3}$	$X_{2,4}$
$X_{3,1}$	$X_{3,2}$	$X_{3,3}$	$X_{3,4}$
$X_{4,1}$	$X_{4,2}$	$X_{4,3}$	$X_{4,4}$

Table 4.4: A rectangular 3×3 convolution window at four different locations.

You find a visualization of the rule in the upper left part of Table 4.3. The features $X_{1,2}$, $X_{2,1}$, $X_{2,3}$, and $X_{3,2}$ trace four black pixels in a circle. At the centre of the outlined circle, $\overline{X}_{2,2}$ means feature $X_{2,2}$ negated with **not**. The negation tells you that the middle pixel must be white to match the rule. In the same way, each other Circle location requires its own pattern, again visualized in Table 4.3:

R2: if $X_{1,3}$ **and** $X_{2,2}$ **and not** $X_{2,3}$ **and** $X_{2,4}$ **and** $X_{3,3}$ **then** *Circle.*

R3: if $X_{2,2}$ **and** $X_{3,1}$ **and not** $X_{3,2}$ **and** $X_{3,3}$ **and** $X_{4,2}$ **then** *Circle.*

R4: if $X_{2,3}$ **and** $X_{3,2}$ **and not** $X_{3,3}$ **and** $X_{3,4}$ **and** $X_{4,3}$ **then** *Circle.*

Finally, you also need four rules for Cross. The formulation of these is left as an exercise.

Convolution Window. You solve the above task more efficiently by looking at the image through a rectangular window — the *convolution window*. The convolution window encloses a *patch* of pixels. By moving the window around, you can investigate the pixels it contains, looking for Circle or Cross. Table 4.4 illustrates this approach. The window is outlined in red and organizes nine pixels in three rows and three columns. Here, you have four possible window locations: (1) upper left, (2) upper right, (3)

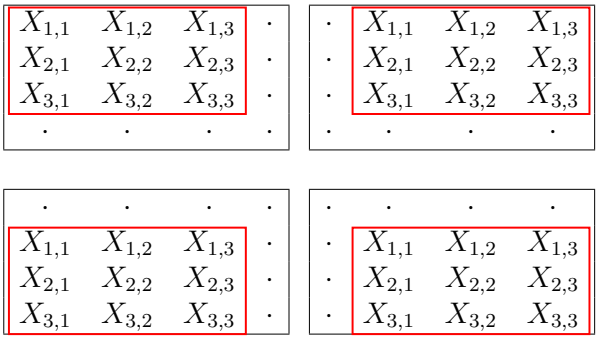


Table 4.5: Location-independent local view of 3×3 patch features.

lower left, and (4) lower right. Each placement allows the Tsetlin machine to detect Circle or Cross in that location.

Local View. We give the Tsetlin machine a local view by using the pixels inside the convolution window as features, disregarding the pixels on the outside. We refer to the enclosed pixels as *patch features* because they represent the image patch delineated by the window. This approach separates the representation from the placement of the window within the image, as shown in Table 4.5. You get the same nine features in every position. Accordingly, the Tsetlin machine can learn the Circle and Cross patterns independently of window location. You get away with two rules instead of eight:

R5: if $X_{1,2}$ **and** $X_{2,1}$ **and not** $X_{2,2}$ **and** $X_{2,3}$ **and** $X_{3,2}$ **then** *Circle*.

R6: if $X_{1,2}$ **and** $X_{2,1}$ **and** $X_{2,2}$ **and** $X_{2,3}$ **and** $X_{3,2}$ **then** *Cross*.

Furthermore, increasing the size of the image does not affect the rules. A larger image gives additional window locations, but the rules remain the same.

Remark – 1D Convolution. Recall the ECG analysis in Figure 4.1. There your data was waveforms, and not a 2D image. For such cases, you

$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	$X_{1,4}$
$X_{2,1}$	$X_{2,2}$	$X_{2,3}$	$X_{2,4}$
$X_{3,1}$	$X_{3,2}$	$X_{3,3}$	$X_{3,4}$

$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	$X_{1,4}$
$X_{2,1}$	$X_{2,2}$	$X_{2,3}$	$X_{2,4}$
$X_{3,1}$	$X_{3,2}$	$X_{3,3}$	$X_{3,4}$

$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	$X_{1,4}$
$X_{2,1}$	$X_{2,2}$	$X_{2,3}$	$X_{2,4}$
$X_{3,1}$	$X_{3,2}$	$X_{3,3}$	$X_{3,4}$

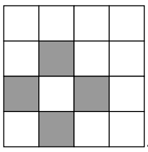
$X_{1,1}$	$X_{1,2}$	$X_{1,3}$	$X_{1,4}$
$X_{2,1}$	$X_{2,2}$	$X_{2,3}$	$X_{2,4}$
$X_{3,1}$	$X_{3,2}$	$X_{3,3}$	$X_{3,4}$

Table 4.6: 1D convolution with three features per step in the sequence.

can perform 1D convolution. You see an example of this in Table 4.6. The only difference from 2D convolution is that the convolution window is moving in the horizontal direction only. Step-by-step, it passes over the features that describe each point in time. Here, you have three features and four steps. We will return to such time-series in Chapter 10.

4.3 Rule Evaluation With Convolution

Classification with convolution follows the procedure from Chapter 1, except for the evaluation of rules. You will now discover how rule evaluation changes to accommodate for convolution. Start by considering the following input image of size 4×4:



Use a convolution window of size 3×3 and rule **R5** above as an example.

Stepwise Procedure. Figure 4.3 shows the procedure for rule evaluation under convolution, step-by-step. This procedure replaces the rule evaluation of Section 1.2. Each step is described below:

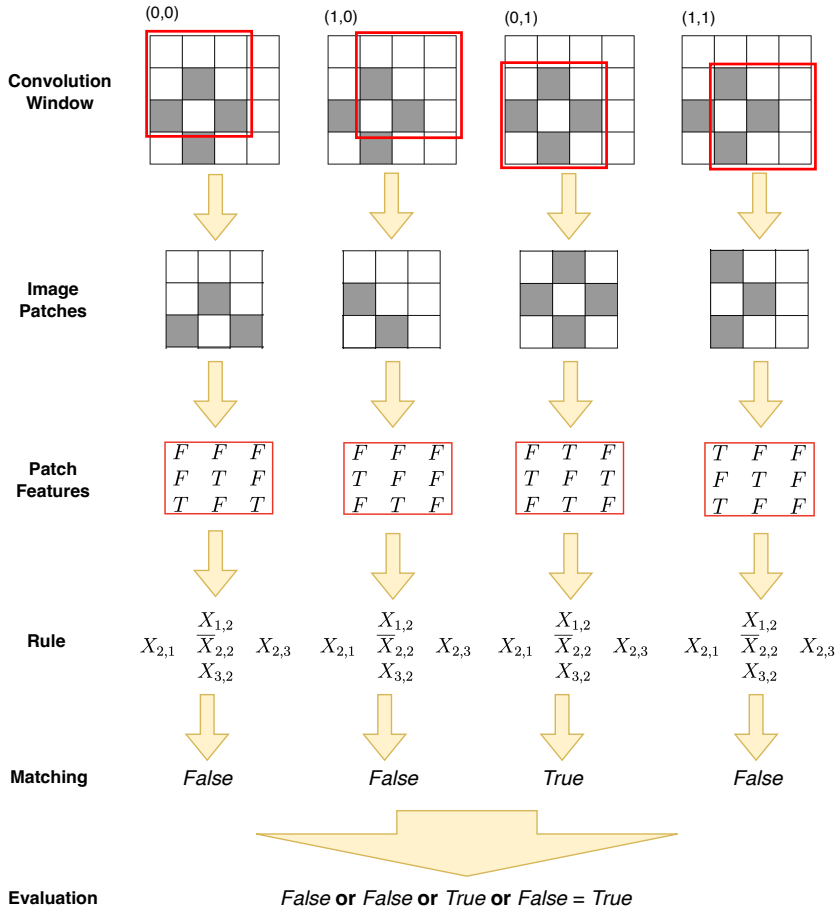
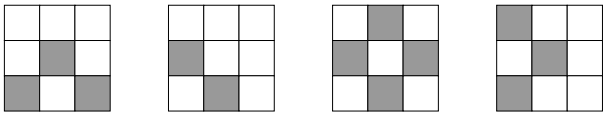


Figure 4.3: Rule-based convolution step-by-step.

1. **Convolution Window.** Place a *convolution window* on each possible position (x, y) in the image. Here, x refers to a pixel column and y to a pixel row, counting from left to right and top to bottom. Align the upper left corner of the window with the image pixel in each position. There are four possible convolution window positions: $(0, 0)$, $(1, 0)$, $(0, 1)$, and $(1, 1)$, shown from left to right in the figure.

2. **Image Patches.** Each convolution window position produces an *image patch*. The image patch consists of the pixels that the convolution

window delineates in the image. This procedure gives the following four patches:



.

3. **Patch Features.** Produce the Boolean features

$$\begin{array}{ccc} X_{1,1} & X_{1,2} & X_{1,3} \\ X_{2,1} & X_{2,2} & X_{2,3} \\ X_{3,1} & X_{3,2} & X_{3,3} \end{array}$$

per image patch. A black pixel is *True* while a white pixel is *False*:

$$\begin{array}{ccc} F & F & F \\ F & T & F \\ T & F & T \end{array} \quad \begin{array}{ccc} F & F & F \\ T & F & F \\ F & T & F \end{array} \quad \begin{array}{ccc} F & T & F \\ T & F & T \\ F & T & F \end{array} \quad \begin{array}{ccc} T & F & F \\ F & T & F \\ T & F & F \end{array} .$$

4. **Matching.** Match the rule under evaluation against the features of each patch. Our example rule is: **if** $X_{1,2}$ **and** $X_{2,1}$ **and not** $X_{2,2}$ **and** $X_{2,3}$ **and** $X_{3,2}$ **then** *Circle*. The matching then gives four outcomes:

$$\textit{False}, \textit{False}, \textit{True}, \text{ and } \textit{False}.$$

5. **Evaluation.** Finally, evaluate the rule for the image itself by doing **or** across the patch matches:

$$\textit{False} \textbf{ or } \textit{False} \textbf{ or } \textit{True} \textbf{ or } \textit{False} = \textit{True}.$$

A crucial property of the above evaluation steps is the following. You get only one output per rule for the given input image. In other words, from this point, you follow the same classification procedure as the one you saw in Chapter 1.

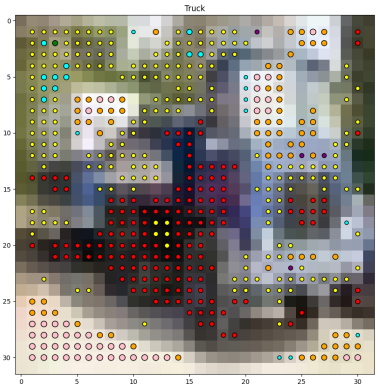


Figure 4.4: A truck with matching rules shown (by Vojtech Halenka, 2023).

Remark – Rule Interaction in Convolution. Section 1.5 of Chapter 1 taught you how multiple rules coordinate to classify the input. Now, the rules specialize in different kinds of image patches. Together, they can then give an overall assessment of the entire input. You see an example of this in Figure 4.4, which depicts a truck. Each circle overlapping a pixel refers to a matching rule. The circle's color points out which one. Accordingly, six different rules match the patches in the image. The red circle rule, for instance, captures the black parts of the truck. The pink circle rule, on the other hand, captures the ground beneath it.

4.4 Rule Learning With Convolution

Like the standard Tsetlin machine in Chapter 1, learning with convolution uses Recognize-, Erase-, and Reject Feedback. Recall how you updated the rules based on the input features. Each feedback type had its own way of using the features' truth values to achieve its effect.

However, convolution is different in one crucial way. A single input image turns into multiple image patches. And a single rule can match many of them at the same time. Multiple matches require a strategy for selecting which patch to update with. Our strategy is *random* selection, and you find the updated algorithm for learning below.

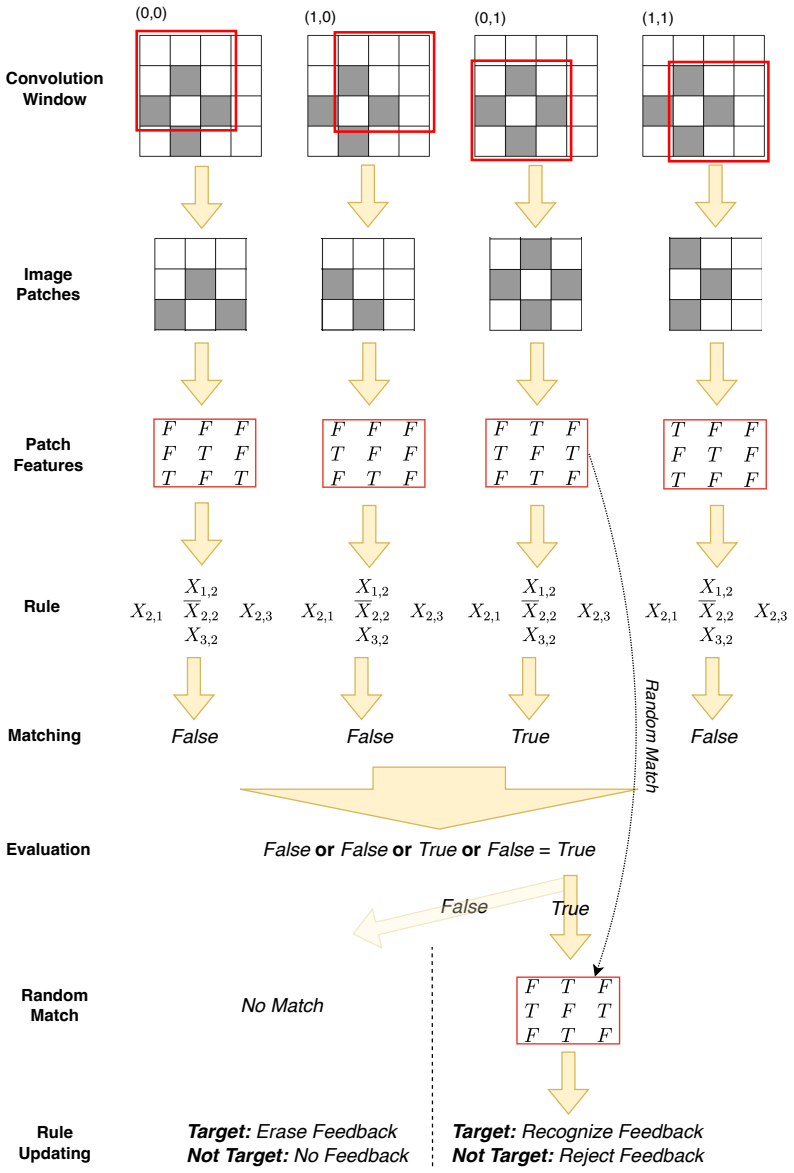


Figure 4.5: Rule-based convolution step-by-step – learning.

Complete Learning Algorithm. The convolutional Tsetlin machine learns complementary rules as follows (Figure 4.5 illustrates the steps for a single rule):

1. *Observe a new image along with its class, and execute the following convolution steps (see Section 4.2):*
    a) *Move the convolution window across the image to produce the image patches.*
    b) *Booleanize each image patch into features.*
2. *For each rule, perform the these evaluation steps (see Section 4.3):*
    a) *Match its condition against each image patch using the patch features.*
    b) *Obtain the truth value of the rule's condition per patch.*
    c) *Or these together to produce a single truth value for the entire input image.*
3. *Calculate the vote sum (see Section 1.5):*
    a) *Identify the rules that evaluated to* True. *Use these for voting.*
    b) *Add up the votes in favour of the image's class.*
    c) *Subtract the votes in favour of the other class.*
    d) *Refer to the summation outcome as v.*
    e) *Set v to the Vote Margin T if larger than T and to $-T$ if smaller than $-T$.*
4. *Go through each rule and give it feedback if Rand() $\leq \frac{T-v}{2T}$, drawn randomly per rule:*
    a) *Give the rule Recognize or Erase Feedback if the rule belongs to the image's class (see Section 1.3):*
        i. *If the rule's condition is* True, *give it Recognize Feedback. Use one of the image patches that matches the rule for the updating. If there are multiple matching patches, pick one of them randomly.*

ii. When the rule condition is False*, apply Erase Feedback. Notice that Erase Feedback does not require the truth values of the patch features.*

b) Give the rule Reject Feedback if its condition is True *and it belongs to another class (see Section 1.4). For updating, randomly pick an image patch from those matching the rule.*

5. Goto 1.

Remark. Notice the new vote margin parameter T above. In Chapter 1, you used a vote margin of two. By increasing T beyond two, more rules will collaborate to classify each input image. The reason is that the vote sum v must approach T to satisfy the vote margin, which drives the learning process to include more rules in the summation of votes. Recall the example of six rules coordinating to classify the image in Figure 4.4. That was possible due to a higher vote margin.

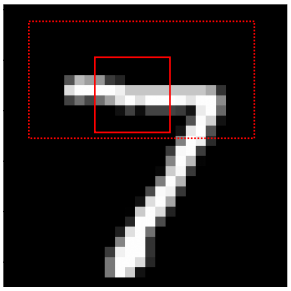


Figure 4.6: Hand-drawn digit '7' with the solid red rectangle marking an upper horizontal curve of the digit. The dotted red rectangle delineate possible appearances of such a pattern for other hand-drawn sevens.

4.5 Position Encoding

Sometimes, a patch's position inside the image can be crucial for correctly classifying the image. Figure 4.6 showcases one such example. Here, the task is to recognize a handwritten digit. The solid red rectangle shows a patch that captures the horizontal curve in the top part of the number '7'. Constraining this pattern to the upper area of the image, marked by the

dotted red rectangle, increases precision. You can then eliminate erroneous matches against the lower part of digit '2'. Next, you learn to incorporate such position information with so-called thermometer encoding.

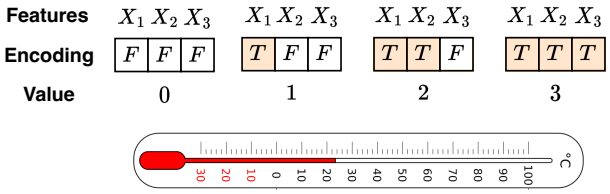


Figure 4.7: Thermometer encoding.

Thermometer Encoding. Thermometer encoding is a way of encoding integer numbers so that you take into account their natural order. Like a thermometer measures temperature, the encoding measures the size of a number. From Figure 4.7, observe how you use the same number of features as the maximum integer value. You get the integer value by counting how many of the features are *True*. The encoding captures the order of numbers by giving you the comparison operator, exemplified as follows:

- When the feature X_1 is *True*, it matches values greater than or equal to *one*, in other words, values *one*, *two*, and *three*. It does not match value *zero* because in its encoding, X_1 is *False*.
- Similarly, you can specify less than *three* with **not** X_3. Then, the encoding of the values *zero*, *one*, and *two* matches.
- You are now empowered to create a range: X_1 **and not** X_3. This condition means greater than or equal to *one*, while less than *three*. You have enclosed the values *one* and *two* in the range.

You can, of course, encode the four numbers with only two bits by using standard binary encoding of integers. That will give you a different kind of expression power. For instance, you can specify even numbers by matching against the least significant bit being zero. However, you then lose the ability to conveniently create ranges with a single rule, which is more appropriate for defining rectangular bounding boxes in image analysis.

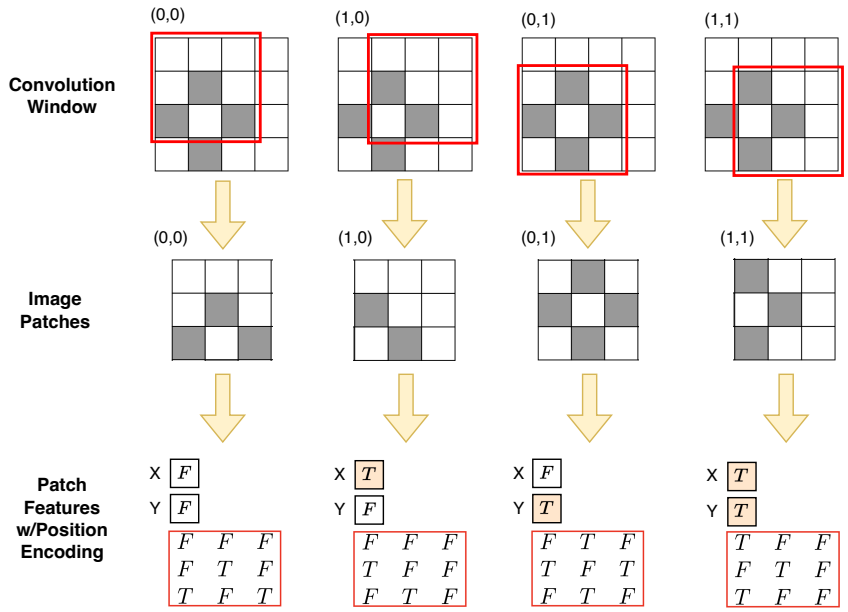


Figure 4.8: Patch features including position encoding.

Patch Positioning. You are now ready to use thermometer encoding to capture the position of each patch inside the image. Figure 4.8 shows the procedure. Notice the two thermometers. One is for the X-dimension and one is for the Y-dimension. Since the four possible patch positions are $(0, 0)$, $(1, 0)$, $(0, 1)$, and $(1, 1)$, you only need one feature per dimension. With these new features, the rules can also learn in which rectangular region the patch features applies. For instance, if you require that Y is *True*, you get the two lower patches. If Y is *False*, you obtain the upper ones.

4.6 Summary

Here are the main points from this chapter:

- A Tsetlin machine does *rule-based* convolution.
- Instead of matching against the complete image, each rule in a con-

volutional Tsetlin machine examines *rectangular patches* inside the image.

- The only difference between standard classification and convolution-based classification is the evaluation of the rule.
- In convolution, a rule is matched against many image patches, each image patch giving a truth value. The evaluation of the rule becomes an **or** over these truth values. In other words, the rule condition is *True* if it matches at least one of the image patches. Otherwise, it is *False*.
- Learning in convolution is also similar to learning in a standard Tsetlin machine, apart from one crucial difference. The patch features of one of the matching image patches is randomly selected for updating the rule.
- Tsetlin machine convolution can incorporate information on the position of the image patches inside the image. This is done by using thermometer encoding to encode the (X, Y)-coordinates of the image patches.