# Application of String Matching Algorithms and Regular Expressions Using Visual Pattern Representation for AI-Generated and Real Image Analysis

Matthew Sebastian Kurniawan - 18223096
Information Systems and Technology Study Program
School of Electrical Engineering and Informatics
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: matthewsebastian1107@gmail.com , 18223096@std.stei.itb.ac.id

***Abstract*—The rapid development of generative artificial intelligence has made AI-generated images more realistic and accessible. This condition creates a need for simple and explainable methods to analyze differences between AI-generated images and real photographs. Many existing approaches rely on learned models that require training data and computational resources. Therefore, this paper explores a lightweight non-machine-learning alternative based on visual pattern representation, string matching algorithms, and regular expressions. Each image is resized to 256 x 256 pixels, divided into 64 blocks, and encoded into a 64-symbol visual string using block features such as texture, edge density, contrast, brightness, saturation, glow, and line score. The string is analyzed using Brute Force, Knuth-Morris-Pratt, Boyer-Moore, and regular expressions. A rule-based scoring mechanism produces AI and real scores, confidence, and tendency results. Experiments on 995 images show balanced accuracy of around 80 percent. The result is an algorithmic demonstration, not forensic authentication. It shows that image features can be effectively represented as searchable symbol sequences.**

***Keywords—String Matching; Regular Expression; Visual Pattern Representation; AI-Generated Image; Brute Force; Knuth-Morris-Pratt; Boyer-Moore; Rule Based Analysis;***

## I. Introduction

Generative artificial intelligence has changed the way digital images are produced. Modern image generation systems can synthesize realistic objects, scenes, lighting effects, and artistic compositions from text prompts or other conditioning inputs. The availability of these systems makes image generation more accessible, but it also creates a practical challenge because generated images can be difficult to distinguish from real photographs through visual observation alone. This challenge motivates the exploration of methods that can analyze visual differences between AI-generated images and real images.


Fig. 1. Comparison between original photographs and AI-generated versions. Adapted from [1].

Most existing approaches for AI-generated image detection rely on machine learning or deep neural networks. These methods can achieve strong performance, but they usually require large datasets, model training, learned parameters, and higher computational resources during experimentation or deployment. Such requirements may be unnecessary when the goal is not forensic detection, but an explainable analysis of visual tendencies. In contrast, the objective of this paper is not to build a learned image detector. Instead, this paper investigates whether a lightweight rule-based representation can transform image characteristics into strings and make them analyzable using string matching algorithms and regular expressions.

The proposed approach converts each image into a visual pattern string. An input image is resized, divided into fixed size blocks, and each block is encoded as one symbol according to its dominant visual characteristics. After this transformation, the problem becomes a string analysis problem. Exact patterns are searched using Brute Force, Knuth-Morris-Pratt, and Boyer-Moore algorithms, while flexible visual sequences are searched using regular expressions.

The main contribution of this paper is the application of string matching and regular expressions to a visual pattern representation of images as a lightweight alternative to model-based analysis. The system produces a tendency score rather than an absolute forensic decision. Therefore, the result should be interpreted as an indication of whether an image has more AI-like or real-like visual patterns, not as a final proof of authenticity.

## II. Theoretical Foundation

### A. Visual Pattern Representation

A digital image consists of pixels, where each pixel contains color or intensity information. Directly applying string matching algorithms to raw pixel values is impractical because a single image may contain thousands or millions of pixels. Therefore, this paper abstracts the image into a smaller sequence of symbols. Each symbol represents a block-level visual feature, such as smoothness, contrast, texture variation, or edge density.

In the proposed representation, a 256 × 256 image is divided into blocks of 32 × 32 pixels. This division produces an 8 × 8 grid, so one image produces 64 blocks. Each block is then encoded into one symbol, and the final representation is a 64-

character visual string. This abstraction reduces image complexity while preserving the local visual characteristics that can be analyzed using string algorithms.

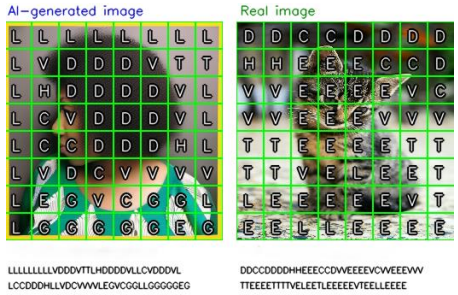



Fig. 2. Image-to-string visual representation.

## B. String Matching

String matching is the process of finding the occurrences of a pattern P inside a longer text T. If T has length n and P has length m, the algorithm reports all positions in T where P occurs. In this paper, T is not a natural language sentence. Instead, T is a visual string generated from an image, and P is a predefined visual pattern such as VVV, LLL, HHH, or EEE. This allows image analysis to be framed as a classical exact pattern matching problem.

Three classical exact string matching algorithms are used and compared, namely Brute Force, Knuth-Morris-Pratt (KMP), and Boyer-Moore. All three locate the same occurrences and therefore return identical match counts. However, they differ in how they perform comparisons and handle mismatches, which affects their execution time. The number of occurrences found for each pattern is later accumulated into the AI and real scores used for classification.

## C. Brute Force String Matching

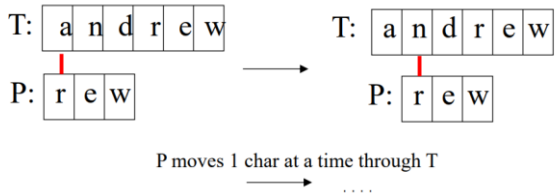



Fig. 3. Brute Force string matching. Adapted from [2].

The Brute Force string matching algorithm compares the pattern with the text at every possible alignment. At each position, the algorithm checks characters one by one until a mismatch occurs or the entire pattern matches. Its worst-case time complexity is O(nm), where n is the length of the text and m is the length of the pattern [10]. Although simple, this algorithm is useful as a baseline because it performs direct comparison without preprocessing.

## D. Knuth-Morris-Pratt Algorithm

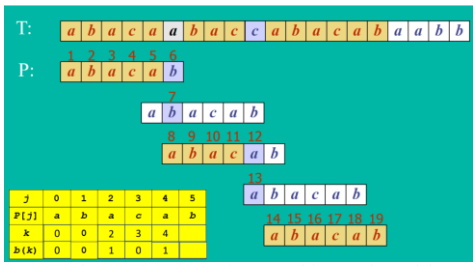



Fig. 4. Knuth-Morris-Pratt string matching. Adapted from [2].

The Knuth-Morris-Pratt algorithm improves string matching by preprocessing the pattern into a longest proper prefix that is also a suffix table [6]. This table allows the algorithm to skip unnecessary comparisons after a mismatch. Unlike the Brute Force approach, KMP does not move backward in the text, giving it a time complexity of O(n + m). In this program, KMP is used as the main exact matching method for scoring. Since Brute Force, KMP, and Boyer-Moore return identical match counts, using only the KMP counts avoids counting the same occurrence three times. Brute Force and Boyer-Moore are retained for running-time comparison.

## E. Boyer-Moore Algorithm

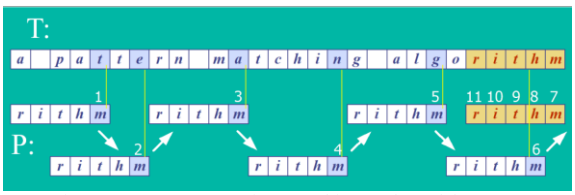
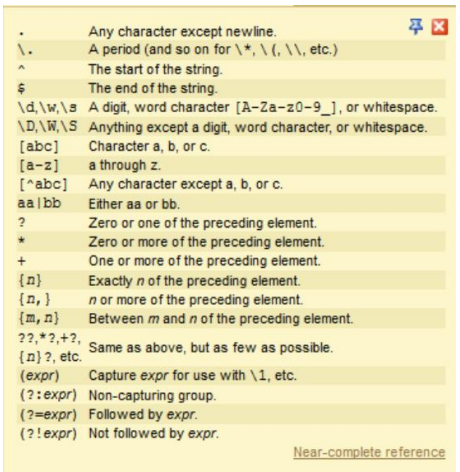



Fig. 5. Boyer-Moore string matching. Adapted from [2].

The Boyer-Moore algorithm compares the pattern from right to left and uses shift rules to skip positions when a mismatch occurs [7]. The implementation in this work uses the bad character rule, which records the last occurrence of each character in the pattern. Boyer-Moore is often efficient for longer texts because it can skip several positions at once. However, in this paper the visual string length is only 64 symbols, so its theoretical advantage may not be visible in the measured running time.

## F. Regular Expression



| | |
|---|---|
| . | Any character except newline. |
| \. | A period (and so on for *, \(, \\, etc.) |
| ^ | The start of the string. |
| $ | The end of the string. |
| \d,\w,\s | A digit, word character [A-Za-z0-9_], or whitespace. |
| \D,\W,\S | Anything except a digit, word character, or whitespace. |
| [abc] | Character a, b, or c. |
| [a-z] | a through z. |
| [^abc] | Any character except a, b, or c. |
| aa\|bb | Either aa or bb. |
| ? | Zero or one of the preceding element. |
| * | Zero or more of the preceding element. |
| + | One or more of the preceding element. |
| {n} | Exactly *n* of the preceding element. |
| {n,} | *n* or more of the preceding element. |
| {m,n} | Between *m* and *n* of the preceding element. |
| ??,*?,+?, {n}?, etc. | Same as above, but as few as possible. |
| (*expr*) | Capture *expr* for use with \1, etc. |
| (?:*expr*) | Non-capturing group. |
| (?=*expr*) | Followed by *expr*. |
| (?!*expr*) | Not followed by *expr*. |

Near-complete reference

Fig. 6. Regular expression matching. Adapted from [3].

Regular expressions are formal notations for describing search patterns. Unlike exact string matching, regular expressions can represent flexible patterns, such as repeated symbols or variable-length symbol groups. For example, V{3,} represents three or more consecutive V symbols, while H+S+ represents one or more H symbols followed by one or more S symbols. In this work, regular expressions are evaluated using Python's built-in re module [8]. Regular expressions are used to detect repeated or clustered visual characteristics that may not be captured by fixed exact patterns.

## G. Algorithmic Complexity Consideration

The complexity comparison is an important part of this work because the same visual string is processed by three exact matching algorithms. Brute Force has simple control flow but

may compare the same characters repeatedly. KMP adds a preprocessing step to build the LPS table and guarantees linear matching time. Boyer-Moore also preprocesses the pattern and can skip several text positions after a mismatch. Nevertheless, all three algorithms are applied to very short visual strings in this paper. Since each image is represented by only 64 symbols, preprocessing overhead can be more visible than asymptotic advantage. This is why the experiment reports both match counts and running time.

## III. Proposed Method

The proposed method consists of five main stages, namely image preprocessing, block feature extraction, visual symbol encoding, string-based pattern analysis, and rule-based scoring. The first part of the method transforms an image into a visual string, while the second part applies exact string matching and regular expressions to compute AI-like and real-like pattern scores. The selection and validation of several configuration values, including symbol grouping, rule order, thresholds, and pattern lists, are discussed later in the Experiments and Evaluation section. Fig. 7 illustrates the overall pipeline of the proposed method.

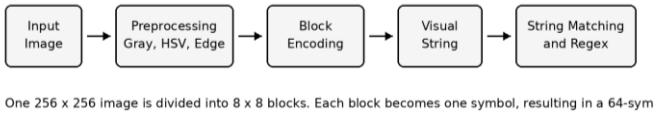



Fig. 7. Proposed pipeline for visual pattern string analysis.

### A. Image Preprocessing and Block Division

Each input image is resized to 256 x 256 pixels to ensure a uniform representation. The image is then converted into grayscale and HSV color space [4]. Grayscale values are used to compute brightness, texture variation, and contrast, while HSV values are used to measure saturation and support glow detection. Canny edge detection is also applied to identify local edge structures [5]. After preprocessing, the image is divided into 32 x 32 pixel blocks. Since the resized image has a size of 256 x 256 pixels, this division produces an 8 x 8 grid with 64 blocks in total. All of these preprocessing operations, including resizing, color conversion, and edge detection, are implemented using the OpenCV library [9].

### B. Block Feature Extraction

For each block, seven numerical features are computed. Brightness is calculated as the mean grayscale intensity. Texture variation is calculated using the standard deviation of grayscale values. Edge density is the proportion of edge pixels obtained from the Canny edge result. Contrast is calculated as the difference between the maximum and minimum grayscale values. Saturation is obtained from the HSV saturation channel. Glow score is computed using normalized brightness and saturation, while line score approximates the presence of long horizontal or vertical edge structures inside the block. The features used for symbol encoding are shown in Table I.

TABLE I. Block Features Used for Symbol Encoding

| Feature | Description | Purpose |
|---|---|---|
| Brightness | Mean grayscale value | Bright or dark area |
| Texture | Standard deviation | Smooth or varied texture |
| Edge density | Edge pixel ratio | Dense local edges |
| Contrast | Max minus min | Strong local difference |
| Saturation | Mean HSV saturation | Color intensity |
| Glow score | Normalized brightness and saturation | Glow-like area |
| Line score | Longest edge row or column | Digital line pattern |

### C. Symbol Encoding

Each block is converted into one symbol using a priority-based rule. The rules are checked from more specific visual properties to more general properties. This priority is important because one block may satisfy more than one condition, but only the first matching rule is selected. Glow and line patterns are checked first because they represent more distinctive visual characteristics, while general smooth blocks are checked later so that they do not overwrite more specific conditions.

The complete rule priority used in the final configuration is shown in Table II. This table defines how numerical block features are converted into symbols. The calibration process used to determine and validate the final rule configuration is discussed in the Experiments and Evaluation section.

TABLE II. Symbol Encoding Rule Priority

| Order | Condition | Symbol | Meaning |
|---|---|---|---|
| 1 | Glow score > 0.30 | G | Glow or saturated block |
| 2 | Line score > 0.70 | L | Digital line pattern |
| 3 | Edge density > 0.12 and contrast > 120 | E | High edge density |
| 4 | Texture > 32 | V | High texture variation |
| 5 | Contrast > 120 | C | High contrast |
| 6 | Brightness > 150 and texture < 32 | T | Bright smooth block |
| 7 | Brightness < 90 and texture < 32 | D | Dark smooth block |
| 8 | Texture < 8 | S | Uniform block |
| 9 | Otherwise | H | Smooth block |

The meaning of each symbol is defined by the encoding rule that produces it, while its scoring group is based on the final configuration used in this paper. In the final configuration, V, C, and L are treated as AI-like symbols, H, S, T, D, and E are treated as real-like symbols, and G is treated as a neutral symbol. The neutral symbol does not directly contribute to either the AI score or the real score. The symbol meanings and groups are shown in Table III.

TABLE III. Symbol Meaning in Visual String

| Symbol | Meaning | Group |
|---|---|---|
| V | High texture variation | AI-like |
| C | High contrast | AI-like |
| L | Digital line pattern | AI-like |
| H | Smooth block | Real-like |
| S | Uniform block | Real-like |
| T | Bright smooth block | Real-like |
| D | Dark smooth block | Real-like |
| E | High edge density | Real-like |
| G | Glow or saturated block | Neutral |

After all blocks are encoded, the image is represented as a 64-symbol visual string. This representation reduces the complexity of the image while preserving selected local visual characteristics that can be analyzed using string-based methods.

### D. Exact Pattern Matching

After the visual string is generated, the method searches for predefined AI-like and real-like exact patterns. AI-like exact patterns include VVV, LLL, VCV, VCL, CVL, LCV, VLC, VVC, VLV, LVL, CLC, and CVV. Real-like exact patterns include HHH, SSS, EEE, TTT, DDD, HSH, SHS, EHE, HEE, EEH, SSE, and TDH. These patterns represent repeated or structured symbol sequences that are used as evidence for AI-like or real-like visual tendencies.

The exact patterns are searched using Brute Force, Knuth-Morris-Pratt, and Boyer-Moore algorithms. All three algorithms return the same matching positions and match counts because they solve the same exact pattern matching problem. However, only KMP counts are used in the scoring stage to avoid counting the same evidence three times. Brute Force and Boyer-Moore are retained for running-time comparison.

### *E. Regex Pattern Analysis*

Regular expressions are used to detect flexible symbol sequences. Unlike exact pattern matching, regular expressions can detect repeated or variable-length patterns. AI-like regex patterns include V{3,}, L{3,}, V{2,}, V+C+, C+V+, V+L+, L+V+, C+L+, and L+C+. Real-like regex patterns include H{3,}, S{3,}, E{3,}, (HS){2,}, (SH){2,}, H+S+, S+H+, E+H+, H+E+, D+H+, and T+H+.

These expressions are useful because visual patterns may appear in clusters with variable length rather than as fixed exact strings. For example, V{3,} detects three or more consecutive V symbols, while H+S+ detects one or more H symbols followed by one or more S symbols. The regex results are accumulated together with exact pattern matches to form the final AI-like and real-like scores.

### *F. Rule Based Scoring*

The final score is calculated from exact KMP matches and regex matches. The AI score is the sum of AI-like exact pattern matches and AI-like regex matches. The real score is the sum of real-like exact pattern matches and real-like regex matches. The confidence value is calculated using the following formula.

$$confidence = \frac{|AI\ score - Real\ score|}{(AI\ score + Real\ score)} \qquad (1)$$

If the total score is zero, the confidence is set to zero and the result is inconclusive. If the AI score is higher than the real score and the confidence is at least 0.20, the image tends to have AI-generated visual characteristics. If the real score is higher than the AI score and the confidence threshold is satisfied, the image tends to have real image visual characteristics. Otherwise, the result is marked as inconclusive.

The confidence threshold is included to prevent forced decisions when the evidence is weak. This is important because several images may contain mixed characteristics. For example, an AI-generated nature image may contain many smooth regions, while a real city image may contain many strong edges and high-contrast structures. Without the inconclusive option, the system would always produce a binary decision even when the visual string does not provide sufficient separation.

## IV. IMPLEMENTATION

The implementation focuses on converting images into visual strings, applying exact string matching and regular expression analysis, and producing a tendency result. The system is implemented as a rule-based program without machine learning training or learned model parameters. Experimental calibration scripts are separated from the runtime implementation because they are used only for offline validation and are discussed later in the Experiments and Evaluation section.

### *A. Program Structure*

The program is organized into several modules so that each part of the analysis process has a clear responsibility. The main program controls the execution mode, the imaging module converts images into visual strings, the string matching module performs exact pattern matching, the regex module detects flexible patterns, and the scoring module produces the final tendency result. The configuration file stores the threshold values, symbol groups, exact pattern lists, regex pattern lists, and decision threshold used by the system.

TABLE IV. Main Program Modules

| Module | Function |
|---|---|
| main.py | Runs the command-line program and provides single, compare, and batch modes |
| gui_app.py | Launches the graphical user interface |
| config.py | Stores thresholds, symbol groups, exact patterns, regex patterns, and decision rules |
| image_to_string.py | Converts an image into a 64-symbol visual string |
| string_matching.py | Implements Brute Force, KMP, and Boyer-Moore string matching |
| regex_analyzer.py | Applies regular expressions to the visual string |
| scoring.py | Computes AI score, real score, confidence, and final tendency result |
| csv_writer.py | Saves batch analysis results into CSV files |
| gui | Implements the GUI tabs, image preview, and user interface components |

### *B. Image Analysis Flow*

The runtime analysis flow begins when an image is loaded by the system. The image is resized to 256 x 256 pixels and processed to extract grayscale, HSV, and edge information. The image is then divided into an 8 x 8 grid of blocks. For each block, the program computes brightness, texture variation, edge density, contrast, saturation, glow score, and line score. These values are evaluated using the priority-based encoding rules stored in config.py. Each block produces one symbol, so the complete image produces a 64-symbol visual string.

After the visual string is generated, the program searches for predefined exact patterns. Brute Force, KMP, and Boyer-Moore are executed on the same visual string and pattern lists. All three algorithms return matching positions and match counts, but only KMP counts are used for scoring. This prevents the same exact pattern occurrence from being counted multiple times. Brute Force and Boyer-Moore are still executed so their running time can be compared with KMP.

Regular expressions are then applied to the same visual string. The regex analysis detects repeated and variable-length symbol sequences that may not be captured by fixed exact patterns. The exact KMP match counts and regex match counts are combined into the AI score and real score. The scoring module then calculates confidence and determines whether the image tends to have AI-generated visual characteristics, real image visual characteristics, or an inconclusive result.

### *C. Runtime Modes*

The program provides three runtime modes, namely single mode, compare mode, and batch mode. The single mode analyzes one image and returns its visual string, AI score, real score, confidence value, and tendency result. The compare mode analyzes two images independently and displays both results so the user can compare their visual pattern tendencies. The batch mode processes multiple images from folders and saves the results into a CSV file. The batch mode is useful for evaluation

because it can process many images consistently using the same configuration.

TABLE V. Runtime Modes

| Mode | Purpose |
|---|---|
| Single | Analyze one image and show its tendency result |
| Compare | Analyze two images independently for comparison |
| Batch | Analyze multiple images and save the results into a CSV file |

### D. Graphical User Interface

In addition to the command-line interface, the system also provides a graphical user interface. The GUI is designed to make the analysis easier to use without manually running commands. It provides two main tabs, namely a single image analysis tab and a comparison tab. The single image analysis tab is used to analyze one image, while the comparison tab is used to analyze two images independently and compare their visual tendency results. The GUI uses the same analysis functions as the command-line program, so both interfaces produce consistent results for the same input image and configuration.

The graphical interface displays the selected image, the generated 64-symbol visual string, the AI score and real score together with their exact and regex components, the confidence value, and the final tendency result. This helps users observe how the image is transformed into a visual string and how the resulting symbols contribute to the final decision. The single image analysis interface is shown in Fig. 8, while the comparison interface is shown in Fig. 9.

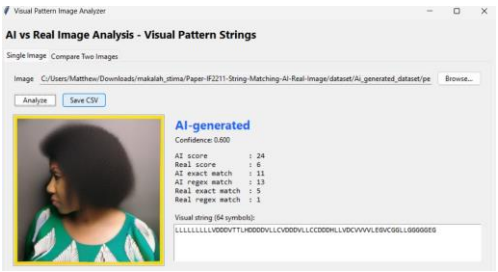



Fig. 8. Single image analysis interface of the developed program.

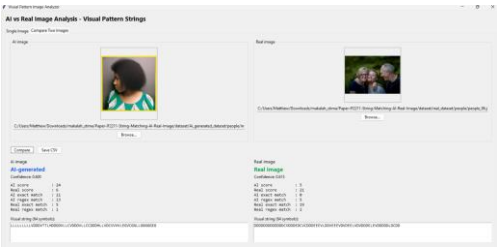



Fig. 9. Comparison interface of the developed program.

The screenshots in Fig. 8 and Fig. 9 show that the program does not only return a final label. It also presents intermediate information, such as the generated visual string and the score components. This makes the output more explainable because users can inspect which visual patterns contribute to the final tendency result. In the comparison interface, two images are analyzed independently, so the system does not directly compare pixels between the images. Instead, it compares the tendency results produced from each image analysis.

### E. Implementation Notes

The implementation does not use machine learning training or learned model parameters. All decisions are produced using fixed rules, predefined patterns, regular expressions, and a rule-based scoring mechanism. This makes the system easier to inspect because each result can be traced back to the visual string, exact pattern matches, regex matches, and final score calculation.

The main purpose of the implementation is not to provide a forensic image authentication system. Instead, it demonstrates how visual information can be abstracted into a string representation and analyzed using classical string matching algorithms and regular expressions.

## V. Experiments and Evaluation

The experiments were conducted to calibrate and evaluate the fixed rule-based configuration used by the system. The experimental scripts are offline tools and are not part of the normal runtime program. These scripts do not perform machine learning training because no model parameters are learned automatically. Instead, they are used to compare candidate configurations, validate rule choices, and measure the final performance of the proposed method.

### A. Dataset and Evaluation Metric

The experiments used 995 images consisting of 250 AI-generated images and 745 real images. The dataset covers five categories, namely animals, city, food, nature, and people. Since the dataset is imbalanced, balanced accuracy is used as the main evaluation metric. Balanced accuracy is calculated as the average of AI recall and real recall, so both classes receive equal importance during evaluation.

In addition to balanced accuracy, the evaluation also reports precision, recall, F1-score, inconclusive rate, coverage, accuracy on decided images, and average execution time for exact string matching algorithms.

### B. Symbol Calibration Experiment

The first experiment was conducted to analyze block-level visual features and determine the symbol grouping used in the scoring process. Fig. 10 shows the average block features for AI-generated and real images. The result shows that AI-generated images have higher texture variation and contrast than real images. Based on the calibration result, symbols V, C, and L are treated as AI-like symbols, while H, S, T, D, and E are treated as real-like symbols. The symbol G is treated as neutral because it does not consistently indicate either class. The train and test validation result also shows that the selected grouping gives stable balanced accuracy.

```
Average block features per class
feature              AI      REAL      diff
brightness      110.483   116.364    -5.881
texture          51.569    30.190    21.379
edge_density      0.107     0.121    -0.014
contrast        179.958   136.799    43.159
saturation       97.552    90.907     6.645
glow_score        0.148     0.133     0.015
line_score        0.524     0.346     0.179

Symbol grouping with highest balanced accuracy on full data
  AI-like symbols   : V, C, L
  real-like symbols : H, S, T, D, E
  AI recall         : 60.0%
  real recall       : 92.5%
  balanced accuracy : 76.2%

Train/test validation (grouping chosen on train, evaluated on test)
  grouping from train : AI=VCL REAL=HSTDE
  balanced on train   : 78.5%
  balanced on test    : 74.0%
```

Fig. 10. Output of the symbol calibration experiment.

### C. Rule Order Experiment

The second experiment tested the priority order of the symbol encoding rules. This experiment is important because

one block may satisfy more than one condition, while only the first matching rule is applied. The experiment_order.py script re-encodes all images using different candidate rule orders and evaluates the resulting balanced accuracy. Fig. 11 shows the comparison among the tested orders.

The baseline order achieved a balanced accuracy of 80.0 percent, essentially tied with the best-performing variant in this run, which swaps the glow and line checks (80.2 percent). This 0.2 percentage-point difference is negligible and within run-to-run variation, indicating that the method is not sensitive to small reorderings among the most distinctive features, the baseline specific-before-general order is therefore retained. In contrast, orders that move the texture or contrast checks to the front reduced balanced accuracy sharply by collapsing real recall, because too many blocks were then labeled as AI-like.

```
images: 995

baseline              G L E V C T D S  balanced =  80.2%  (AI 73.6%, real 86.7%)
swap V and C          G L E C V T D S  balanced =  75.5%  (AI 56.8%, real 94.2%)
E after V             G L V E C T D S  balanced =  61.2%  (AI 90.8%, real 31.5%)
texture first         V C E G L T D S  balanced =  57.5%  (AI 93.2%, real 21.7%)
glow and line last    E V C T D S G L  balanced =  67.0%  (AI 39.6%, real 94.4%)
swap G and L          L G E V C T D S  balanced =  80.2%  (AI 74.6%, real 86.4%)
contrast first        C V E G L T D S  balanced =  58.0%  (AI 21.6%, real 94.5%)

best order: swap G and L (L G E V C T D S) -> 80.2%
```

Fig. 11. Output of the encoding rule order experiment.

## D. *Threshold Sensitivity Experiment*

The threshold sensitivity experiment was conducted to observe whether the method depends too heavily on specific threshold values. Fig. 12 shows the threshold experiment for edge density, line score, brightness, texture, contrast, and glow score. The result shows that performance remains relatively stable across a range of threshold values. Some thresholds affect the balance between AI recall and real recall, but the overall performance does not collapse sharply when the values are changed within the tested range.

The edge density and line score thresholds have a visible effect on the balance between AI and real recall. The current edge density threshold of 0.12 and line score threshold of 0.70 are among the best-performing values in the tested range. The brightness, texture, contrast, and glow score thresholds also show relatively stable performance. This suggests that the proposed rule-based representation is not fully dependent on one exact threshold value.

```
images: 995
baseline balanced accuracy: 80.2% (AI 73.6%, real 86.7%)

TEXTURE_THRESHOLD (current = 32)
  TEXTURE_THRESHOLD = 24    balanced =  80.7%  (AI 78.4%, real 83.0%)
  TEXTURE_THRESHOLD = 28    balanced =  80.3%  (AI 74.8%, real 85.8%)
  TEXTURE_THRESHOLD = 32    balanced =  80.2%  (AI 73.6%, real 86.7%) <= current
  TEXTURE_THRESHOLD = 36    balanced =  80.0%  (AI 72.4%, real 87.5%)
  TEXTURE_THRESHOLD = 40    balanced =  80.1%  (AI 71.2%, real 89.0%)
  best: TEXTURE_THRESHOLD = 24 -> 80.7%

CONTRAST_THRESHOLD (current = 120)
  CONTRAST_THRESHOLD = 100    balanced =  79.9%  (AI 76.4%, real 83.4%)
  CONTRAST_THRESHOLD = 110    balanced =  80.2%  (AI 74.8%, real 85.5%)
  CONTRAST_THRESHOLD = 120    balanced =  80.2%  (AI 73.6%, real 86.7%) <= current
  CONTRAST_THRESHOLD = 130    balanced =  80.0%  (AI 72.0%, real 88.1%)
  CONTRAST_THRESHOLD = 140    balanced =  79.0%  (AI 69.2%, real 88.7%)
  best: CONTRAST_THRESHOLD = 120 -> 80.2%

GLOW_THRESHOLD (current = 0.3)
  GLOW_THRESHOLD = 0.25   balanced =  78.3%  (AI 71.2%, real 85.4%)
  GLOW_THRESHOLD = 0.3    balanced =  80.2%  (AI 73.6%, real 86.7%) <= current
  GLOW_THRESHOLD = 0.35   balanced =  80.8%  (AI 73.2%, real 88.3%)
  GLOW_THRESHOLD = 0.4    balanced =  80.8%  (AI 73.2%, real 88.5%)
  best: GLOW_THRESHOLD = 0.4 -> 80.8%
```

```
EDGE_DENSITY_THRESHOLD (current = 0.12)
  EDGE_DENSITY_THRESHOLD = 0.08   balanced =  74.3%  (AI 53.6%, real 95.0%)
  EDGE_DENSITY_THRESHOLD = 0.1    balanced =  78.2%  (AI 64.8%, real 91.7%)
  EDGE_DENSITY_THRESHOLD = 0.12   balanced =  80.2%  (AI 73.6%, real 86.7%) <= current
  EDGE_DENSITY_THRESHOLD = 0.15   balanced =  79.5%  (AI 81.6%, real 77.4%)
  best: EDGE_DENSITY_THRESHOLD = 0.12 -> 80.2%

LINE_SCORE_THRESHOLD (current = 0.7)
  LINE_SCORE_THRESHOLD = 0.5    balanced =  65.8%  (AI 85.6%, real 46.0%)
  LINE_SCORE_THRESHOLD = 0.6    balanced =  78.2%  (AI 78.0%, real 78.4%)
  LINE_SCORE_THRESHOLD = 0.7    balanced =  80.2%  (AI 73.6%, real 86.7%) <= current
  best: LINE_SCORE_THRESHOLD = 0.7 -> 80.2%

BRIGHTNESS_THRESHOLD (current = 150)
  BRIGHTNESS_THRESHOLD = 130    balanced =  80.1%  (AI 74.0%, real 86.2%)
  BRIGHTNESS_THRESHOLD = 140    balanced =  80.3%  (AI 74.0%, real 86.6%)
  BRIGHTNESS_THRESHOLD = 150    balanced =  80.2%  (AI 73.6%, real 86.7%) <= current
  BRIGHTNESS_THRESHOLD = 160    balanced =  80.0%  (AI 73.2%, real 86.7%)
  best: BRIGHTNESS_THRESHOLD = 140 -> 80.3%
```

Fig. 12. Output of the threshold sensitivity experiment.

## E. *Pattern Mining Experiment*

The pattern mining experiment was conducted to find symbol sequences that appear more frequently in one class than the other. The script generates length-2 and length-3 symbol sequences, counts their average occurrences in AI-generated and real images using KMP, and ranks them based on the difference between the two classes. Fig. 13 shows the most AI-leaning and real-leaning patterns found by the experiment.

The result indicates that L and V based sequences are strong AI-like indicators. These patterns are related to digital line structures and high texture variation. On the other hand, E, T, and D based sequences appear more frequently in real images. These patterns are related to edge-dense, bright smooth, and dark smooth regions. The pattern mining result helps refine the final exact pattern lists and regex pattern lists used by the scoring system.

```
AI images: 250  real images: 745
alphabet: HSEVTDCGL  candidate lengths: 2 and 3

Top AI-leaning patterns (most frequent in AI relative to real)
  pattern      avg in AI   avg in real   difference
  LL             13.108         1.761       11.347
  LLL             9.500         0.828        8.672
  VV              3.804         1.713        2.091
  LV              1.540         0.281        1.259
  VL              1.540         0.282        1.258
  GL              1.220         0.078        1.142
  LG              1.204         0.102        1.102
  VVV             1.708         0.643        1.065
  VLL             0.868         0.074        0.794
  LLV             0.860         0.097        0.763
  GLL             0.712         0.020        0.692
  LLG             0.660         0.019        0.641
  LLE             1.204         0.576        0.628
  EV              1.892         1.282        0.610
  ELL             1.184         0.576        0.608
  VE              1.976         1.392        0.584
```

```
Top real-leaning patterns (most frequent in real relative to AI)
  pattern      avg in AI   avg in real   difference
  EE               4.988        14.236       -9.248
  EEE              2.200         9.928       -7.728
  TT               1.772         7.081       -5.309
  TTT              0.932         5.358       -4.426
  DD               5.100         6.565       -1.465
  DDD              3.460         4.663       -1.203
  CC               0.400         1.103       -0.703
  CE               0.648         1.333       -0.685
  EC               0.716         1.338       -0.622
  HH               0.420         0.976       -0.556
  CEE              0.276         0.809       -0.533
  ELE              0.380         0.831       -0.451
  GGG              1.552         1.993       -0.441
  EEC              0.372         0.809       -0.437
  TC               0.200         0.612       -0.412
  DE               0.532         0.929       -0.397

Suggested config lists derived from the data:
AI_EXACT_PATTERNS = ['LL', 'LLL', 'VV', 'LV', 'VL', 'GL', 'LG', 'VVV', 'VLL', 'LLV', 'GLL', 'LLG', 'LLE', 'EV', 'ELL', 'VE']
REAL_EXACT_PATTERNS = ['EE', 'EEE', 'TT', 'TTT', 'DD', 'DDD', 'CC', 'CE', 'EC', 'HH', 'CEE', 'ELE', 'GGG', 'EEC', 'TC', 'DE']
```

Fig. 13. Output of the pattern mining experiment.

## F. *Final Evaluation*

The final evaluation was performed using the output from batch mode. The evaluation computes the confusion matrix, precision, recall, F1-score, balanced accuracy, inconclusive rate, coverage, accuracy on decided images, and per-category accuracy. The final configuration achieved a balanced accuracy of 80.6 percent and a simple accuracy of 83.6 percent. The AI-generated class obtained 65.0 percent precision, 74.8 percent recall, and 69.6 percent F1-score. The real class obtained 91.1 percent precision, 86.5 percent recall, and 88.7 percent F1-score.

The evaluation also shows that 12.7 percent of the images were marked as inconclusive. This result is important because the system is not designed to force every image into a binary decision. Instead, the inconclusive option allows the system to abstain when the AI score and real score are too close. The accuracy on decided images reached 86.3 percent, which shows that the confidence threshold helps reduce weak decisions. The final evaluation summary is shown in Table VI.

TABLE VI. Final Evaluation Summary

| Metric | Result |
|---|---|
| Total images | 995 |
| AI-generated images | 250 |
| Real images | 745 |
| Balanced accuracy | 80.6% |
| Simple accuracy | 83.6% |
| AI precision | 65% |
| AI recall | 74.8% |
| AI F1-score | 69.6% |
| Real precision | 91.1% |
| Real recall | 86.5% |
| Real F1-score | 88.7% |
| Inconclusive rate | 12.7% |
| Coverage | 87.3% |
| Accuracy on decided images | 86.3% |

### G. Runtime Comparison

The exact string matching algorithms were also compared based on execution time. Brute Force, KMP, and Boyer-Moore produced identical match counts because they solve the same exact pattern matching problem. However, their execution times were measured to observe the practical effect of algorithmic differences. The runtime comparison is shown in Table VII.

TABLE VII. Average Exact-Match Execution Time per Image

| Algorithm | Average Runtime |
|---|---|
| Brute Force | 0.00015531 seconds |
| Knuth-Morris-Pratt | 0.00016243 seconds |
| Boyer-Moore | 0.00016163 seconds |

The average running times of the three algorithms are very close. This result occurs because each image is represented by only 64 symbols. On such short strings, the preprocessing overhead of KMP and Boyer-Moore can reduce their theoretical advantage over Brute Force. The measured running times are hardware-dependent and may vary between runs, but the result consistently shows that all three algorithms have very close execution times on 64-symbol visual strings.

### H. Discussion

The evaluation shows that the proposed visual string representation can capture several useful differences between AI-generated and real images. The method performs better when visual patterns contain structured lines, high texture variation, or strong local differences. However, some images remain difficult to separate because AI-generated images can contain smooth natural regions, while real images can also contain strong edges and high-contrast structures.

The results should not be interpreted as forensic authentication. The system only produces a visual tendency based on the designed symbol representation, exact pattern matches, regex matches, and rule-based scoring. Nevertheless, the experiments show that classical string matching algorithms and regular expressions can be applied to image analysis after an image is transformed into a compact visual string.

## VI. Conclusion

This paper proposed a lightweight rule-based approach for analyzing AI-generated and real images using visual pattern representation, exact string matching algorithms, and regular expressions. Each image is transformed into a 64-symbol visual string by resizing the image, dividing it into an 8 × 8 block grid, extracting block-level features, and encoding each block into a visual symbol using thresholds and symbol groups that were empirically calibrated and validated on a held-out split. After the image is represented as a string, Brute Force, Knuth-Morris-Pratt, Boyer-Moore, and regular expressions are used to detect AI-like and real-like visual patterns.

The experimental results show that the proposed method can capture several visual tendencies between AI-generated and real images. The final configuration achieved a balanced accuracy of 80.6% and a simple accuracy of 83.6% on 995 images. The system also marked 12.7% of images as inconclusive, which helps avoid forced decisions when the visual evidence is weak. The runtime comparison shows that Brute Force, KMP, and Boyer-Moore have very close execution times in this system because each image is represented by only 64 symbols. Since all three algorithms return identical match counts, only KMP is used for scoring to avoid counting the same evidence multiple times. Their nearly identical running times indicate that on such short strings, the preprocessing overhead of KMP and Boyer-Moore reduces their theoretical skipping advantage over the simpler Brute Force approach.

The method is not intended to be a forensic authentication system. Its result should be interpreted as a tendency based on the designed symbol representation and rule-based scoring mechanism. The performance is also limited by the chosen visual features, thresholds, symbol rules, and pattern lists. Some images remain difficult to separate because AI-generated images may contain smooth natural regions, while real images may contain strong edges, high contrast, or structured patterns.

Future work can improve the method by exploring richer visual features, larger datasets, more adaptive symbol encoding rules, and additional pattern selection strategies. The visual string representation can also be extended with different block sizes or multi-scale analysis so that both local and global image characteristics can be captured more effectively.

## Appendix

The implementation consists of command-line and graphical interface modes. The command-line interface supports single image analysis, two-image comparison, and batch experiment evaluation. The graphical interface supports single image and compare modes for user-facing demonstrations. The source code is available at: https://github.com/Matthew12-t/Paper-IF2211-String-Matching-AI-Real-Image

The batch experiment output can also be used to reproduce the tables in this paper. The output contains the image path, label, visual string, exact scores, regex scores, confidence, predicted tendency, and execution time of Brute Force, KMP, and Boyer-Moore.

## Video Link at Youtube

Further explanations of the implementation in this paper are available to watch in the following YouTube video link: https://youtu.be/gc9XSuhAbpQ?si=XsPXwjj5n6WTOTEz

## ACKNOWLEDGMENT

The author expresses gratitude to God Almighty for His blessings during the completion of this paper. The author also thanks the lecturers and teaching assistants of IF2211 Algorithm Strategies for providing the theoretical foundation of algorithm analysis, string matching, and regular expressions. The author also thanks family and friends for their support during the development and writing process.

## REFERENCES

[1] C. Bush, "How to spot AI-generated photos: real vs. fake," Charles Bush Photography, Nov. 3, 2025. [Online]. Available: https://charlesbushphoto.com/charles-bush-photography-news/2025/11/ai-generated-vs-real-photos. Accessed: Jun. 9, 2026.

[2] R. Munir, "Pencocokan string," IF2211 Strategi Algoritma, Inst. Teknologi Bandung, 2026. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/23-Pencocokan-string-(2026).pdf. Accessed: Jun. 10, 2026.

[3] R. Munir, "String matching dengan regex," IF2211 Strategi Algoritma, Inst. Teknologi Bandung, 2026. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/24-String-Matching-dengan-Regex-(2026).pdf. Accessed: Jun. 10, 2026.

[4] R. C. Gonzalez and R. E. Woods, Digital Image Processing, 4th ed. New York, NY, USA: Pearson, 2018.

[5] J. Canny, "A computational approach to edge detection," IEEE Trans. Pattern Anal. Mach. Intell., vol. PAMI-8, no. 6, pp. 679-698, Nov. 1986.

[6] D. E. Knuth, J. H. Morris, Jr., and V. R. Pratt, "Fast pattern matching in strings," SIAM J. Comput., vol. 6, no. 2, pp. 323-350, 1977.

[7] R. S. Boyer and J. S. Moore, "A fast string searching algorithm," Commun. ACM, vol. 20, no. 10, pp. 762-772, Oct. 1977.

[8] J. E. F. Friedl, Mastering Regular Expressions, 3rd ed. Sebastopol, CA, USA: O'Reilly Media, 2006.

[9] G. Bradski, "The OpenCV library," Dr. Dobb's J. Software Tools, vol. 25, no. 11, pp. 120-125, 2000.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 3rd ed. Cambridge, MA, USA: MIT Press, 2009.

## STATEMENT

In this statement, I declare that this paper I have written is my own work, not a duplication or translation of someone else's paper and is not plagiarized.

Bandung, 19 June 2026

Matthew Sebastian Kurniawan
18223096