Musik in bayern
ISSN: 0937-583x Volume 90, Issue 1 (Jan -2025)
https://musikinbayern.com **DOI https://doi.org/10.15463/gfbm-mib-2025-*376***

Java Source Code into Braille by Advancing Programming Accessibility with Bridging the Digital Divide: Transforming OCR and Image Processing to Empower Visually Impaired Programmers

[1]A. Varshini, Ph.D.
Research Scholar, Department of Computer Science, University of Madras, Chennai
[2]Dr. S. Gopinathan,
Professor and head, Department of Computer Science, University of Madras, Chennai

*Corresponding Author E-mail : gnathans2002@gmail.com

To Cite this Article

[1]A. Varshini, Ph.D. [2]Dr. S. Gopinathan*,**" Java Source Code into Braille by Advancing Programming Accessibility with Bridging the Digital Divide: Transforming OCR and Image Processing to Empower Visually Impaired Programmers**
" *Musik In Bayern, Vol. 90, Issue 1, Jan 2025, pp65-89*

Article Info
Received: 25-12-2024 Revised: 02-01-2025 Accepted: 11-01-2025 Published: 22-01-2025

Abstract:

Braille is another form of writing that assists the blind mute by touching to read and even write. It employs a consistent 3x2 grid, known in all languages previously mentioned, to represent various characters. Serving to read a normal text, which was originally a culture's script, Braille seems to have moved on with sciences such as programming languages into areas such as education and training where no such access point is rendered for the visually impaired. The majority of the programming source code is in print or electronic form, which poses difficulties for visually impaired persons to obtain the code and to write it. This work outlines a novel method for translating programs, specifically Java source code images, into Braille text using OCR technology and a specially designed Braille mapping procedure. Translating the textual forms of source codes to Braille without the help of an operator makes it easier for physically impaired persons. The system's engine optimizes the images to facilitate text extraction, utilizing enhanced algorithms to extract text even in torn or low- contrast cases. Translate the extracted text into Braille using pre-established mappings designed for Java syntax.

The effectiveness of this technique is determined by comparing the outcome of such a conversion process with the real Braille translation. Further, the system employs new mathematical algorithms to determine the accuracy of a conversion and the effectiveness of Java-Braille translations. These algorithms demonstrate the most effective solutions for

Musik in bayern
ISSN: 0937-583x Volume 90, Issue 1 (Jan -2025)
https://musikinbayern.com DOI https://doi.org/10.15463/gfbm-mib-2025-*376*
identifying gaps in Braille translation and character recognition. This study shows considerable improvements over previous efforts, suggesting this methodology might improve Java proficiency and independence for visually challenged programmers. Therefore, the ongoing efforts aim to incorporate modern, state-of-the-art OCR and image processing techniques into the automation process, thereby improving the availability of programming resources and computer science education for the visually impaired, using Java.
Keywords: ***Automated Conversion, Braille Translation, Fake Dots, Image Processing, and Java Source Code.***

I. Introduction

In a world where technology determines tasks and inventions, millions struggle and still cannot adjust to accessibility in a world framed by technology that dictates and invents. Programming restrictions mainly affect visually challenged people, leading to limits on personal goals and a shortage of diversity in the technology sector. A World beyond the Screen: Understanding Visually Impaired Software Developers – Daron Bessa, January 16, 2023. The digital age has created new opportunities, but accessibility issues remain a significant challenge, particularly for visually impaired individuals who aspire to become software developers. Many programming languages, development tools, and source code formats are primarily visual, which can create barriers for those with visual impairments, according to the World Health Organization. Although there are numerous assistive technologies available today, such as screen readers and Braille displays, navigating and understanding complex programming environments can still be difficult.

Java, one of the leading and most popular programming languages, is no exception. It comes with its own syntax, a rich set of libraries, and many tools for development, which require a detailed understanding and proper interpretation of the source, not only makes reading the source code easy but also makes it this way due to competence. Translating this code into a format that allows visually impaired programmers to access it is a crucial aspect of the inclusivity in software development open to everyone. While some solutions exist that often just deal with partial accessibility, it is unlikely to close this gap since they are still kind of primitive.

This research introduces a novel approach that empowers blind programmers by using the automated Optical Character Recognition technique and image processing to change Java source code into braille. The combination of more sophisticated image-processing algorithms and OCR techniques tailors this system to ensure accurate detection, translation, and formatting of code into a tactile format compatible with braille displays or embossers. This framework proposed will not only increase accessibility but also provide ways for visually impaired people to actively participate and excel in software development.

The paper focuses on various challenges and complexities that must be addressed in the transformation process, discusses the effectiveness of the solutions proposed, and points to the possibility of broad application of such fundamentalities in enabling programming. With these barriers, this paper aims to minimize the gap in technology and bring inclusivity in technology.

II. Literature Review

This report surveys comprehensively the technological methods, on the recognition of

handwriting text within the broad spectrum. Several techniques and methodologies employed in the OCR systems will be addressed, which can also be beneficial to the recognition and processing of Java source code images [1]. It talks about the development of smartphone-based assistive technologies for the visually impaired now and in the future. Further, it provides the context of the current status of assistive technologies [2]. In this paper, discusses the mechanisms and methodologies of text-to-Braille conversion, where emphasis is placed on the challenges of mapping complex scripts, along with lessons that might be helpful for programming languages [3]. This chapter concerned itself with all challenges and methodologies for adapting the Bangla script into Braille, so that blind readers could have access to written materials in their mother tongue. An outline is provided by the authors for conversion of Bangla script into Braille with consideration of specific peculiarities concerning the Bangla alphabet-the presence of diacritical marks, conjunct letters, and vowels [4]. This paper introduces a unique approach for detecting Arabic Braille numerals the use of Convolutional Neural Networks (CNNs). The research focuses on enhancing the availability of numerical statistics for visually impaired people in Arabic-speaking regions by exploiting deep learning techniques [5]. This paper presents an innovative system that translates text and voice inputs into Braille output, aiming to enhance accessibility for visually impaired individuals. The proposed system integrates multiple technologies to provide a comprehensive solution for text and speech translation into Braille [6]. This paper introduces an automatic recognition system for Arabic Sign Language (ArSL) using deep Convolutional Neural Networks (CNNs). The system aims to assist individuals who are deaf or hard of hearing by converting Arabic sign language gestures into text, facilitating communication [7]. This paper presents a model for translating text written in Indian languages into Bharti Braille, providing visually impaired individuals access to diverse regional languages. The proposed system addresses the unique linguistic features of Indian scripts and ensures accurate Braille transcription [8]. This paper provides a comprehensive analysis and evaluation of various methods for converting Braille into text, focusing on their effectiveness, efficiency, and practicality for real-world applications. The authors compare existing techniques and propose recommendations for improving Braille-to-text systems [9]. Braille Translation System Using Neural Machine Translation Technology I - Code Conversion introduces a novel approach to translating Japanese text into Braille using Neural Machine Translation (NMT) techniques [10]. Focuses on developing an efficient system for translating English text into Braille. The proposed device aims to bridge the accessibility gap for visually impaired individuals by means of providing a reliable and accurate approach to convert text into tactile Braille formats [11]. Introduces a system that converts both speech and text into Braille script to aid blind and deaf individuals [12]. Presents a device that translates Braille characters into English textual content, enabling conversation for visually and listening to-impaired individuals. The system guarantees green and correct conversion, improving accessibility and inclusivity [13]. Focuses on a device that converts text into Braille, catering to the wishes of visually and hearing-impaired individuals. It gives a dependable and consumer-pleasant answer for real-time conversation [14]. Introduces a system that employs Optical Character Recognition (OCR) and solenoid generation to beautify the accuracy and efficiency of text-to-Braille conversion for visually impaired users. The gadget makes a speciality of presenting a fee-powerful and real-time Braille display solution [15]. This study take a look at introduces a device capable of changing each text and voice inputs into Braille symbols, facilitating verbal exchange and studying for blind college students by assisting more than one languages and supplying actual-time Braille output [16]. This paper presents a technique that employs deep neural networks to transform handwritten textual content into Braille, aiming to provide an green and cost-powerful answer for making Braille accessibility easier for deaf-blind people [17]. This paper presents a compact 3D-printed device that uses Optical Character Recognition (OCR) to convert text into Braille, imparting a

value-effective answer for visually impaired people [18]. This paper gives a comprehensive technique to the numerous conversation issues that visually impaired humans face each day, offering a tool that converts text to Braille language to enhance accessibility [19]. Provides a tool that interprets speech into textual content and further converts it into Braille, imparting a actual-time verbal exchange answer for individuals with visible and listening to impairments. The machine demonstrates efficiency and accuracy in enhancing accessibility for these customers [20]. Offers a comprehensive overview of text-to-Braille conversion technologies, emphasizing their importance in improving accessibility for visually impaired people. It discusses diverse methodologies and gear utilized in Braille translation structures [21].

III. Flow of the Work

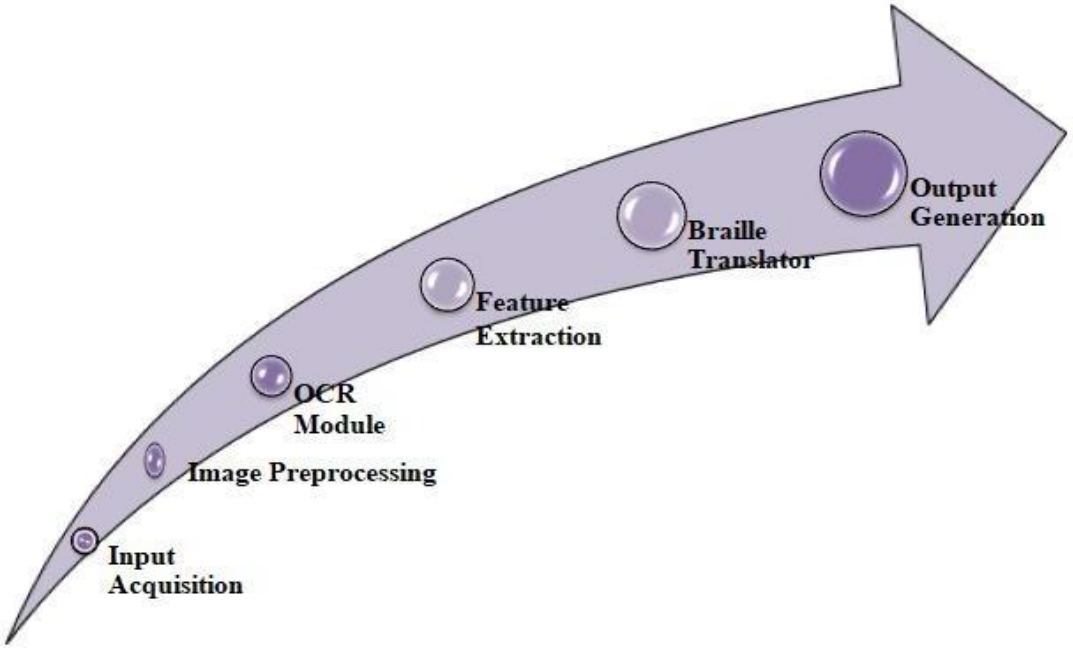

Fig. 1 Flow of the Work

The flow of the work illustrated in **Fig.1** outlines the step-by-step process involved in transforming Java source code into braille. Starting with image acquisition, it captures the Java source code as an image via a scanner or camera. The next process is image processing, which enhances the image for text recognition, removes the unwanted content, and gives the Java code with correct syntax. The third stage is the OCR module that extracts text and converts it into ASCII, which will be represented in numerical analysis that processes to the fourth stage, the feature extraction engine, which is used to apply DWT, FFT, and entropy analysis, and then the fifth stage is the Braille translator that maps extracted features to Braille symbols, and the final stage is output generation, which produces a Braille text file or displays the Braille in real-time.

IV. Bridging Code and Accessibility: Translating Syntax to Touch Through the Braille Conversion Process

This entire process through which a Java source code is converted into braille involves some

ISSN: 0937-583x Volume 90, Issue 1 (Jan -2025)
https://musikinbayern.com **DOI https://doi.org/10.15463/gfbm-mib-2025-*376***
standard steps that have been developed especially in consideration of accuracy, accessibility, and usability for said programmers. The description below portrays an organized list of operations from image capture to braille, providing an overview of the whole approach. Each step is carefully structured to enhance the efficiency and reliability of the system.

Algorithm 1: Convert Java Source Code to Braille

```
Input: Image path of the Java source code
Output: Braille representation of the source code
Initialization:
Input Image ← capture Image(image Path)
Image Preprocessing:
Grayscale Image ← convert To Grayscale(Input Image)
Contrast Enhanced Image ← enhance Contrast(Grayscale Image)
Sharpened Image ← apply Sharpening(Contrast Enhanced Image)
Processed Image ← remove Noise(Sharpened Image)
OCR Module:
Extracted Text ← perform OCR(Processed Image)
if Extracted Text is EMPTY then
    Print("[ERROR] No text extracted.")
    return
ASCII Code ← [char To ASCII(char) for char in Extracted Text]
Feature Extraction:
Wavelet Coefficients ← [compute DWT(line) for line in ASCII Code]
Frequency Components ← [compute FFT(line) for line in ASCII
 Code]
Entropy Values ← [compute Entropy(line) for line in ASCII Code]
Features ← zip(Wavelet Coefficients, Frequency Components,
 Entropy Values)
Braille Translation:
Braille Output ← []
Initialize braille Dict with mappings of ASCII values and
 programming symbols to braille patterns
foreach char in ASCII Code do
    Braille Symbol ← braille Dict[char]
    Append Braille Symbol to Braille Output
Output Generation:
if output Mode = "file" then
    save To File(Braille Output, "output_braille.txt")
else if output Mode = "real-time" then
    send To Braille Display(Braille Output)
else
    Print("[ERROR] Invalid output mode.")
End
```

Essentially, this algorithm defines the process that permits blind programmers to access most programming constructs. By aligning feature extraction with braille mapping, it essentially gives visual programming environments tactile ease of use, closing the loop on usability. A detailed breakdown of this system shows its capability to bridge the gap between visual programming environments and tactile accessibility, ensuring inclusivity in this field of software development. The next sections explore the way in which this system has been placed in a wider workflow of braille generation and accessibility, showcasing the capacity that it has to empower programmers with new and innovative accessibility solutions.

V. Proposed Methodology

The methodology combines certain advanced image preprocessing, feature extraction, and Braille mapping techniques to provide an accurate and logically sound mapping of Java source

code into Braille. Throughout the whole process, the code retains its structure and semantic integrity.

5.1 Image Preprocessing

The input for our system includes an image that contains Java source code. Preprocessing deals with this image in advance so that text can be extracted effectively and in a clearer manner through several steps:

5.1.1 Grayscale Conversion

The image is converted to a grayscale image using the following formula:

$$I_{gray} = 0.299 \cdot R + 0.587 \cdot G + 0.114 \cdot B \quad \text{---- (1)}$$

It reduces the image to one single channel such that only the gray values come out, eliminating the color information and thus concentrating on the text content. The weights of 0.299, 0.587, and 0.114 are the values taken from the prescription by the human visual system. Green is dominant in our vision since it is representative of the visible colors, followed by red and blue; such a distribution assures maximum contrast of text components.

5.1.2 Contrast Enhancement

Histogram equalization is applied to enhance the visibility to increase the contrast and sharpness of text regions in the image. This technique redistributes the intensity values across the grayscale image, thus improving its contrast. It stretches the range of pixel intensities, thus making faint or poorly lit text appear sharper and more distinct. For instance, dimly lit Java source code can be transformed into a high-contrast, readable format, thus ensuring better performance during subsequent text extraction processes.

5.1.3 Sharpening

Sharpening is the method used to make the boundaries or borders of the characters in an image prominent, which leads to better character recognition with an OCR. The sharpening process is the following:

$$I_{sharp} = \mathrm{I}_{\mathrm{original}} + \lambda \cdot (\nabla^2 \mathrm{I}) \quad \text{---- (2)}$$

Here, $\nabla^2 \mathrm{I}$ is the Laplacian of the image, revealing areas with the most rapid shifts in intensity, such as edges. The value of λ determines the strength of the sharpening. By zeroing in on the contour of text characters, the next step will result in the automatic code of Java, which is easily recognizable by the OCR system and will better detect and recognize the fine print of Java source code by highlighting the edges of the text characters.

5.2 Optical Character Recognition (OCR)

ISSN: 0937-583x Volume 90, Issue 1 (Jan -2025)
https://musikinbayern.com **DOI https://doi.org/10.15463/gfbm-mib-2025-*376***

Tool: Tesseract OCR or an equivalent optical character recognition system.

Process:

The OCR tool occupies the central position in the pipeline; it is responsible for the conversion of the visual representation of the text into machine-readable digital text. The fact that the process consists of the following stages becomes visible:

5.2.1 Input Handling

The input to the OCR tool is the preprocessed grayscale image. At this stage, the image has undergone transformations such as grayscale conversion, contrast enhancement, and sharpening so that the text regions are pronounced and noise-free from distortions.

5.2.2 Text Region Detection

It analyzes the image for detection and to segment regions with textual information. It identifies a textual and non-textual part in the image, such as codes, diagrams, or background artifacts, to consider only valid areas.

5.2.3 Character Recognition

Having the text portions being found, the OCR engine then processes every region to get individual characters. This implies that it is capable of recognizing diverse fonts, sizes, and special characters that are usually found in programming languages, of which many, instance brackets, semicolons, and keywords, are the important ones.

5.2.4 Machine-Readable Text Conversion

The OCR recognizes characters, and after those characters are reconstructed, they are structured back to text according to the exact Java source code syntax and formatting rules. The OCR maintains formatting aspects, such as indentation, line breaks, and symbol alignment, since they are functional for programming.

5.2.5 Output Optimization

Refinement of the text is done in such a way that errors of the same kind can be avoided, like misinterpretation of characters that are similar-looking characters, such as "O" and "0" or "1" and “l.” To these discrepancies, post-processing algorithms might be applied and, thus, the overall accuracy of the output might be improved.

5.3 Feature Extraction

Feature extraction transforms the textual data into actionable insights through numerical analysis. Several techniques are used to capture relevant structural and semantic features of the extracted codes, which are later used for purposes such as braille translation. Below are the key methods employed

5.3.1 ASCII Encoding

Each character in the extracted text is mapped to its corresponding ASCII value. It creates a numeric representation of the code. A concrete instance: all letters, numbers, spaces, and symbols are converted to their standardized ASCII equivalent. This number format serves as the basis for later mathematical transformations and analyses.

Input: *class Hello*

Output: [99,108,97,115,115,32,72,101,108,108]

The ASCII code, through the conversion of each character into a unique number, makes it possible for the system to analyze data and manipulate it numerically-a foundation for that

further processing.

5.3.2 Discrete Wavelet Transform (DWT)

To capture both global and local structural features of the text, the Discrete Wavelet Transform is applied. The DWT decomposes the ASCII sequence into approximation coefficients (A_k) and detail coefficients ($D_{j,k}$). This decomposition allows for pattern extraction at different scales. The general formula for DWT is:

$$x(t) = \sum_k A_k \emptyset_k(t) + \sum_j \sum_k D_{j,k} \psi_{j,k}(t) \qquad \text{---- (3)}$$

Where $\emptyset(t)$ is the approximation basis function or global pattern, and $\psi_{j,k}(t)$ is the detail basis function, or local variation. This method captures the overall structure and finer details of code, from indentations and syntax to unique elements such as loops or conditionals.

5.3. 3 Fast Fourier Transform (FFT):

The Fast Fourier Transform (FFT) is used to transform the ASCII sequences into the frequency domain, where periodic patterns such as loops and braces can be identified. The formula for FFT is:

$$(k) = \sum_{n=0}^{N-1}(n).\, e^{-j2\pi \mathrm{kn/N}} \qquad \text{---- (4)}$$

The factors here designated by F(k) are the Fourier factors denoting frequencies forming the component for every term in the input sequence. Through examination of frequencies, FFT has the possibility to determine cyclic patterns in the source code-the patterns that exist between loops and calls of the function and duplicated characters or brackets: repeated forms usually inherent within a programming language's syntax as well as central for determining a sense of a rhythmic pulse for the structure in the code.

5.3.4 Shannon Entropy:

Shannon entropy is used to measure the randomness or unpredictability in each line of code. It calculates the uncertainty in the character distribution, which can distinguish between simple lines (braces) and more complex ones (strings or function calls). The formula for Shannon entropy is:

$$H = -\sum_i p_i . log_2(p_i) \qquad \text{---- (5)}$$

Where p_i is the probability of each character. Usually, a higher entropy value means that it is more complex in code, such as string literals or multiline expressions, and lower entropy suggests simpler and more predictable lines.

By combining these feature extraction techniques, the system gains a detailed and mathematical understanding of the extracted code. This information is essential for

Subsequent stages, such as Braille translation, are a foundation for identifying key characteristics and patterns in programming code.

Musik in bayern
ISSN: 0937-583x Volume 90, Issue 1 (Jan -2025)
https://musikinbayern.com **DOI https://doi.org/10.15463/gfbm-mib-2025-*376***

5.4 DWT, FFT, and Entropy are Braille Mapped

Features extracted using DWT, FFT, and entropy are mapped to Braille symbols:

$$B = f(A, F, H) \qquad \text{---- (6)}$$

where A is ASCII encoding, F is frequency, and H is entropy.

- **Dense Braille (⠿):** Assigned to lines with high entropy.
- **Sparse Braille (⠒):** Assigned to lines with high-frequency components.
- **Simple Braille (⠂):** Assigned to lines with low complexity.

Feature	Purpose	Braille Mapping	Pattern
Wavelet	Capture smoothness and local variations. **Analyzes:** Global vs. local patterns in ASCII values.	High variations → **Dense (⠿).** Smooth trends → **Simple (⠂).**	**Input:** System.out.println("Hello"); **Wavelet Coefficients:** High detail variations → **Dense Braille (⠿)**. **Wavelet decomposition** shows significant **local variations** due to symbols (., (, "), indicating high complexity → **Dense Braille (⠿)**.
Fourier	Identify structured, periodic content. **Analyzes:** Frequency components of ASCII values	High frequency → **Sparse (⠒).** Irregular → **Dense (⠿).**	**Input:** public static void main() **Fourier :** High frequency due to repeated structure → **Sparse Braille (⠒)**. the repeated keywords (public, static) and structural elements (()) introduce periodic patterns, leading to dominant high frequencies → **Sparse Braille (⠒)**.
Entropy	Measure randomness and complexity. **Analyzes:** Diversity of characters in a line.	High entropy → **Dense (⠿).** Low entropy → **Simple (⠂).**	The line contains **diverse characters** (uppercase, lowercase, symbols, spaces), resulting in high randomness → **Dense Braille (⠿)**.

Musik in bayern
ISSN: 0937-583x Volume 90, Issue 1 (Jan -2025)
https://musikinbayern.com **DOI https://doi.org/10.15463/gfbm-mib-2025-*376***

Table 1: Java Source Code Line with Braille Symbol

Java Source Code Line	Features Extracted	BrailleSymbol
Class Hello World{	High frequency	⠒
Public static void main(String[]args){	High entropy	⠿
System.out.println("Hello, Braille!");	Moderate entropy	⠤
}	Low complexity	⠤

VI. Structural and Pattern Analysis of Source Code

This process involves using analytical techniques that will be used to provide an insightful visual and numerical representation of the source code to better understand the code structures and patterns. These methods will assist in extracting important features that help with complexity analysis and pattern recognition and develop data suitable for efficient conversion into Braille.

6.1 ASCII Heatmap

The ASCII values of source code characters are arranged into a 2D grid, represented as:

$$A_{i,j} = ASCII(char_{i,j}) \quad \text{---- (7)}$$

where Aij stands for the ASCII value of the character at the construction of the matrix, the element at the i-th row and j-th column in the source code. The ASCII values can be subsequently mapped to a color gradient for visualization, showing the variations across lines and enabling structural pattern identification.

6.2 Binary Heatmap

Each ASCII value Aij is converted into its binary representation:

$$B_{i,j} = bin(A_{i,j}) \quad \text{---- (8)}$$

where Bij represents the binary equivalent of the ASCII value. Each bit of the binary representation can be seen on the heat map as the pixel value, permitting an impressionistic or bit-level view of the structure of the code.

6.3 Frequency and Structural Visualization

Wavelet Coefficients Analysis:
The approximation and detail coefficients with the help of the Discrete Wavelet Transform (DWT) have been obtained as mentioned in **Eq. 1**. The coefficients involve global and local recognition of patterns present in the source code.

ISSN: 0937-583x Volume 90, Issue 1 (Jan -2025)
https://musikinbayern.com **DOI https://doi.org/10.15463/gfbm-mib-2025-*376***

Frequency Spectrum Analysis:
Using FFT, the frequency components of the ASCII sequence are described in **Eq. 2**, which highlights periodic patterns like loops or nested structures.

6.4 Complexity Assessment

Shannon Entropy Visualization:
The Shannon entropy is calculated to measure randomness or complexity in the code, following **Eq. 3**. High entropy values signify sections with dense expressions, while lower entropy values indicate simpler structures.

Structural Complexity Mapping:

$$C = \sum_{i=1}^{L} d_i \quad \text{---- (9)}$$

where C is the total complexity, L is the number of lines, and d_i is the depth of indentation for the i-th line.

6.5 Statistical Analysis

Character Frequency Visualization:
The frequency f(c) of each character c in the source code is computed as:

$$\mathrm{f(c)} = \frac{\text{count of c}}{\text{total characters}} \quad \text{---- (10)}$$

This metric highlights the prevalence of specific elements, such as keywords or symbols.

Line-by-Line Statistical Insights:
Metrics like line length l_i are calculated as:

$$\mathrm{l_i} = \text{length of line}_\mathrm{i} \quad \text{---- (11)}$$

for each line, providing insights into the variability of code structure.

VII. Sensing Code and Logic: Translating Java Syntax with Analytical Techniques for Mapping Code into Braille Dots

Braille mapping is an integral part of making Java source code accessible to visually impaired programmers. It is an orthographical translation into tactile Braille patterns of the code syntax, symbols, and structure while preserving reading flow and logical thought. In this section, mappings for alphabets, numbers, special characters, and Java keywords will be discussed, ensuring that the tactile output is in accordance with the original functionality and intention of the code.

Musik in bayern
ISSN: 0937-583x Volume 90, Issue 1 (Jan -2025)
https://musikinbayern.com **DOI https://doi.org/10.15463/gfbm-mib-2025-*376***

7.1 Braille Mapping for Alphabets

In braille, English alphabets are represented using unique patterns of raised dots. These mappings are also applied to Java source code.

Alphabet	Braille Representation	Description
A	⠁	Uppercase letter A.
B	⠃	Uppercase letter B.
C	⠉	Uppercase letter C.
D	⠙	Uppercase letter D.
E	⠑	Uppercase letter E.
F	⠋	Uppercase letter F.
G	⠛	Uppercase letter G.
H	⠓	Uppercase letter H.
I	⠊	Uppercase letter I.
J	⠚	Uppercase letter J.
K	⠅	Uppercase letter K.
L	⠇	Uppercase letter L.
M	⠍	Uppercase letter M.
N	⠝	Uppercase letter N.
O	⠕	Uppercase letter O.
P	⠏	Uppercase letter P.
Q	⠟	Uppercase letter Q.
R	⠗	Uppercase letter R.
S	⠎	Uppercase letter S.
T	⠞	Uppercase letter T.
U	⠥	Uppercase letter U.
V	⠧	Uppercase letter V.
W	⠺	Uppercase letter W.
X	⠭	Uppercase letter X.
Y	⠽	Uppercase letter Y.
Z	⠵	Uppercase letter Z.

7.2 Braille Mapping for Numbers

Numbers in braille are represented with a prefix "⠼" followed by corresponding alphabet representations.

Number	Braille Representation	Description
0	⠼⠚	Number 0.
1	⠼⠁	Number 1.
2	⠼⠃	Number 2.
3	⠼⠉	Number 3.

Musik in bayern
ISSN: 0937-583x Volume 90, Issue 1 (Jan -2025)
https://musikinbayern.com **DOI https://doi.org/10.15463/gfbm-mib-2025-*376***

4	⠼⠙	Number 4.
5	⠼⠑	Number 5.
6	⠼⠋	Number 6.
7	⠼⠛	Number 7.
8	⠼⠓	Number 8.
9	⠼⠊	Number 9.

7.3 Braille Mapping for Special Characters

Special characters play a vital role in Java syntax and are mapped as follows:

Symbol	Braille Representation	Description
{	⠐⠣	Opening brace.
}	⠐⠜	Closing brace.
(	⠐⠣	Opening parenthesis.
)	⠐⠜	Closing parenthesis.
;	⠆	Statement terminator.
=	⠶	Assignment operator.
+	⠖	Addition operator.
-	⠤	Subtraction operator.
*	⠔	Multiplication operator.
/	⠌	Division operator.
"	⠒	Double quotation mark.
'	⠄	Single quotation mark.
.	⠲	Dot/period.
,	⠂	Comma.

7.4 Braille Mapping for Java Keywords

Java keywords are mapped to braille combinations that preserve their readability and structural meaning.

Java Keyword	Braille Representation	Description
class	⠒⠇ ⠄ ⠩⠩	Declares a class.
public	⠇⠩⠢⠇ ⠇ ⠩⠒	Access modifier.
static	⠩⠩⠇ ⠄ ⠩⠇⠩⠒	Defines static methods.
void	⠫⠢⠩⠒⠇	Return type.
main	⠩⠒ ⠄ ⠩⠩⠇	Main method name.
String	⠩⠩⠇⠩⠒⠩⠇⠇	String data type.
int	⠩⠒⠩⠇⠇	Integer data type.
if	⠩⠒⠇	Conditional statement.
else	⠑⠫⠇ ⠩⠒⠑	Alternate condition.
for	⠇⠒⠩⠢⠫	Loop construct.
while	⠩⠇⠫⠢⠩⠒⠇ ⠑	Loop construct.
return	⠫⠢⠑⠩⠇⠢⠫⠩⠇	Return statement.
new	⠩⠇⠒⠑⠩⠇	Object instantiation.

This framework, which contains a comprehensive mapping system for alphabets, numbers, special characters, and Java-specific keywords, will have the braille output reflect the logical structure and semantic integrity of the source code. The mappings are based on the standard braille system, which uses a 3x2 matrix of raised dots to represent characters, ensuring tactile readability and efficiency.

This unique approach provides, since it adapts the braille writing system to handle keywords and syntax-tied symbols inherent in programming languages, such as Java, while preserving clarity and logical consistency. In contrast to conventional braille systems that concentrate exclusively on text, this method incorporates elements specific to programming, providing a customized solution for programmers with visual impairments. All these go a long way to forming the essence of braille translation, as they bridge the chasm between visual and tactile programming environments.

VIII. Experimental result

The experimental results furnish a sequential view of the processes and outcomes involved in the conversion of Java source code into braille. The system was scanned and tested with a sample Java program. The results demonstrate the accuracy and practicality of the proposed methodology. Below are the detailed findings along with relevant outputs at each stage:

Musik in bayern
ISSN: 0937-583x Volume 90, Issue 1 (Jan -2025)
https://musikinbayern.com **DOI https://doi.org/10.15463/gfbm-mib-2025-*376***

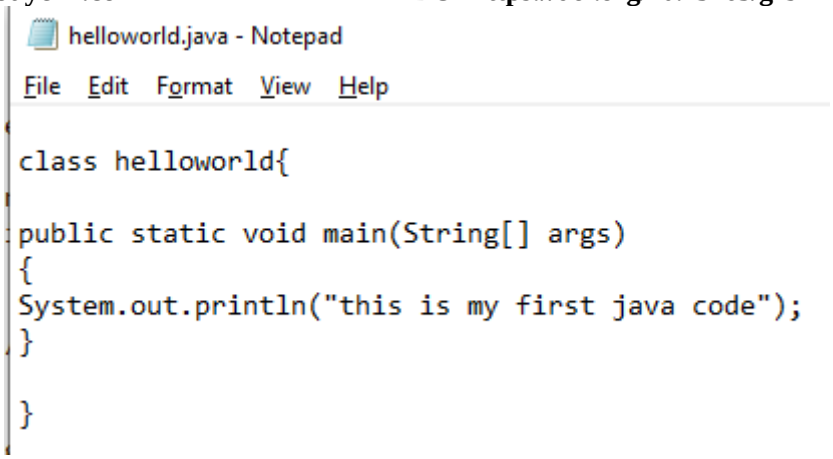

Fig. 2 Original Java Source code image

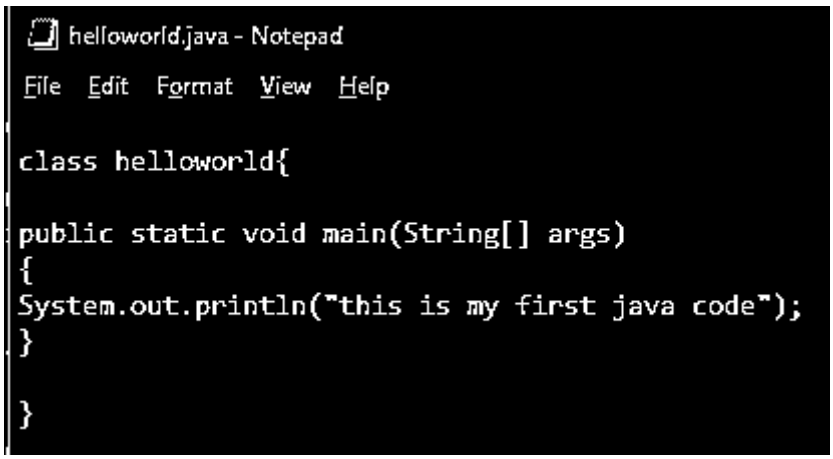

Fig. 3 Sharpen image

The original image of the Java source code shown in **Fig. 2** contains edge-blurred fonts or texts, low contrasts, and white noisy backgrounds, which pose challenges for precise OCR processing. To counter these issues, the image is preprocessed; sharpening has been enhanced, and this technique removes all aforementioned issues.

Fig. 3 shows the sharpened image, where textual content edges are enhanced, noise is reduced, contrast is improved, and evaluation is progressed. The sharpening processes use a Laplacian filter to enhance character boundaries, making the text more readable for OCR. This preprocessing step improves OCR accuracy considerably and ensures better feature extraction for braille and its conversion.

Musik in bayern
ISSN: 0937-583x Volume 90, Issue 1 (Jan -2025)
https://musikinbayern.com **DOI https://doi.org/10.15463/gfbm-mib-2025-*376***

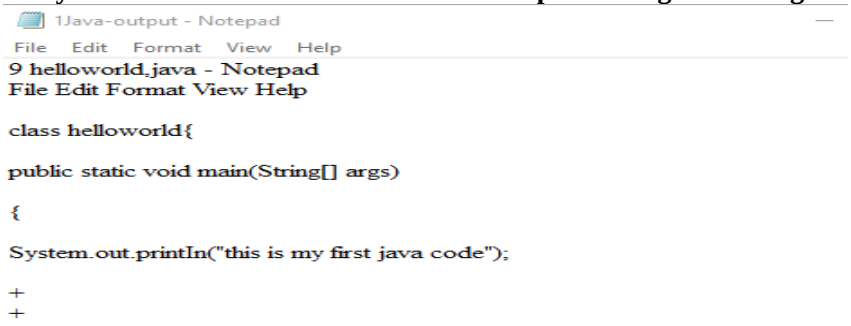

Fig.4 Extracted image after preprocessing, showing improved contrast and text clarity

Fig. 4 shows the extracted text image from that Java source code image after it underwent the sharpening process, followed by an OCR process. Even though the OCR could identify and capture the majority of the text, some artifacts, errors, and unwanted elements, together with extra spaces, noise, or even non-textual data, can still exist.

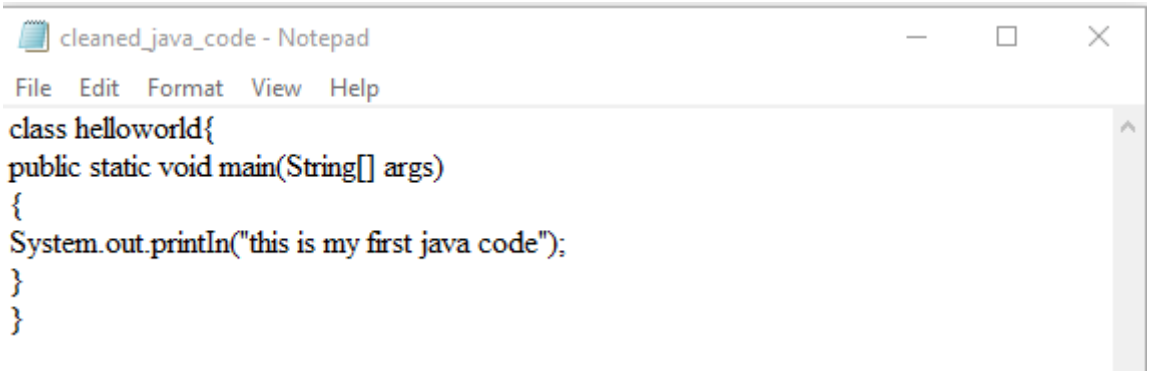

Fig. 5 Cleaned Java Source Code from the Extracted Image

Fig. 5 shows the cleaned-up version of the extracted sentences. In this step, irrelevant text, extraneous white spaces, and formatting inconsistencies are eliminated in order to ensure the correctness and consistency of the extracted content. The cleaning step is crucial for initializing the data for feature extraction and braille mapping, as it minimizes errors and increases the overall reliability of the processes running in the pipeline.

8.1 Line by Line process of the Extracted image to ASCII matrix

Line 1: class helloworld {

Token	Type	ASCII Values
class	Keyword	[99, 108, 97, 115, 115]
helloworld	Identifier	[104, 101, 108, 108, 111, 119, 111, 114, 108, 100]
{	Symbol	[123]

Musik in bayern
ISSN: 0937-583x Volume 90, Issue 1 (Jan -2025)
https://musikinbayern.com **DOI https://doi.org/10.15463/gfbm-mib-2025-*376***

Line 2: public static void main(String[] args)

Token	Type	ASCII Values
public	Keyword	[112, 117, 98, 108, 105, 99]
static	Keyword	[115, 116, 97, 116, 105, 99]
void	Keyword	[118, 111, 105, 100]
main	Identifier	[109, 97, 105, 110]
(	Symbol	[40]
String	Keyword/Class	[83, 116, 114, 105, 110, 103]
[]	Symbol	[91, 93]
args	Identifier	[97, 114, 103, 115]
)	Symbol	[41]

Line 3: {

Token	Type	ASCII Values
{	Symbol	[123]

Line 4: System.out.println("this is my first java code");

Token	Type	ASCII Values
System	Identifier/Class	[83, 121, 115, 116, 101, 109]
.	Symbol	[46]
out	Identifier	[111, 117, 116]
.	Symbol	[46]
println	Method/Function	[112, 114, 105, 110, 116, 108, 110]
(	Symbol	[40]
"this is my first java code"	String Literal	[34, 116, 104, 105, 115, 32, 105, 115, 32, 109, 121, 32, 102, 105, 114, 115, 116, 32, 106, 97, 118, 97, 32, 99, 111, 100, 101, 34]
)	Symbol	[41]
;	Symbol	[59]

Line 5: }

Token	Type	ASCII Values
}	Symbol	[125]

Line 6: }

Musik in bayern
ISSN: 0937-583x Volume 90, Issue 1 (Jan -2025)
https://musikinbayern.com **DOI https://doi.org/10.15463/gfbm-mib-2025-*376***

Token	Type	ASCII Values
}	Symbol	[125]

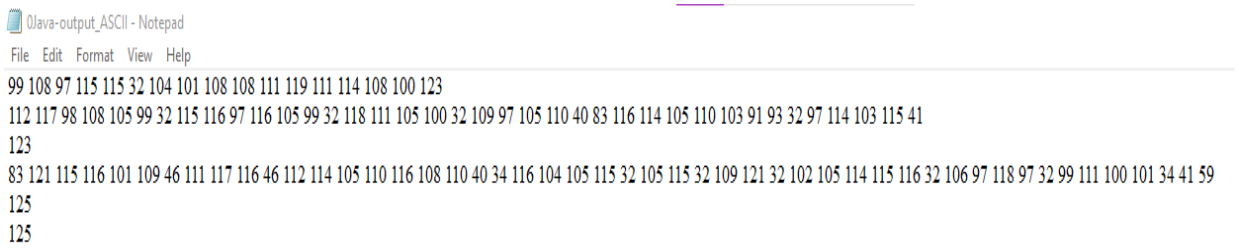

Fig. 6 Extracted Image to ASCII Matrix

In this step, the text extracted from the image is converted into its corresponding ASCII values. Each character within the extracted image is mapped to a numerical ASCII value, which represents the standard encoding for characters. **Fig. 6** shows the ASCII representation of the extracted text in which ASCII numbers correspond to each line of text that is converted into a sequence of ASCII numbers. This conversion enables numerical processing for next characteristic extraction techniques, which include DWT, FFT, and entropy evaluation.

8.2 Extracted image to Binary Matrix Processes

8.2.1 Characters are Mapped to Braille

8.2.1.1 ASCII and Text Extraction

Extracting the characters from the input is the first stage in the Braille conversion process.
Consider the text: *class*
Each character has an **ASCII value**. For instance:

- c → ASCII: 99
- l → ASCII: 108
- a → ASCII: 97
- s → ASCII: 115
- s → ASCII: 115

Characters are represented digitally by ASCII values. However, **ASCII is not used directly to generate the Braille dots.**

8.2.2 Standard Braille Patterns

Musik in bayern
ISSN: 0937-583x Volume 90, Issue 1 (Jan -2025)
https://musikinbayern.com **DOI https://doi.org/10.15463/gfbm-mib-2025-*376***
As indicated in your illustration, A-Z braille patterns are commonplace. These patterns are ASCII-independent and fixed.
Each Braille design has a 6-dot cell with numbered positions:

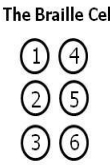

Fig.7 Braille Cell

Pattern Example for the letter c:

- Braille pattern: 100100
- Raised dots are in positions 1 and 4, while positions 2, 3, 5, and 6 are empty.

8.3 Mapped to Braille

When encountering the letter c in the text:

1. Its **ASCII value** (99) is used to identify the character programmatically.
2. A **lookup table** maps the letter c to its predefined Braille pattern:
   - c → Braille: 100100
3. The Braille pattern 100100 is used to generate the tactile representation:

● ○
○ ○
● ○

This mapping is fixed and consistent. ASCII or binary values are **not calculated** to generate Braille patterns.

8.4 ASCII/Binary

- The **ASCII value** of a character (99 for c) is only used internally in digital systems to:
  - Recognize the character.
  - Fetch the corresponding Braille pattern from a **lookup table**.
- Binary conversion (8-bit binary of ASCII values) is not directly related to the tactile Braille pattern.
- For instance:
  - ASCII 99 → Binary: 01100011 (8 bits).
  - **But Braille for c is always 100100**, which is predefined and unrelated to 01100011.

I. **Extract ASCII Values:**
   - c: 99
   - l: 108
   - a: 97
   - s: 115

- s: 115

II. **Find Braille Patterns:** Using the Braille lookup table:

- c: 100100
- l: 101010
- a: 100000
- s: 101011
- s: 101011

III. **Final Braille Output:**

c: ● ○ l: ● ● a: ● ○ s: ● ● s: ● ●
○ ○ ○ ● ○ ○ ○ ● ○ ●
● ○ ● ○ ○ ○ ● ● ● ●

8.5 Braille Pattern Insights

- **Braille is based on a standard 6-dot system**, not on ASCII or binary directly.
- Each letter (A-Z) and number has a **predefined Braille pattern**.
- ASCII or binary values are used digitally to identify characters but are not used to calculate Braille dots.
- For example:
  - ASCII 99 (binary 01100011) for c is irrelevant when determining the Braille dots. Braille for c is always 100100.

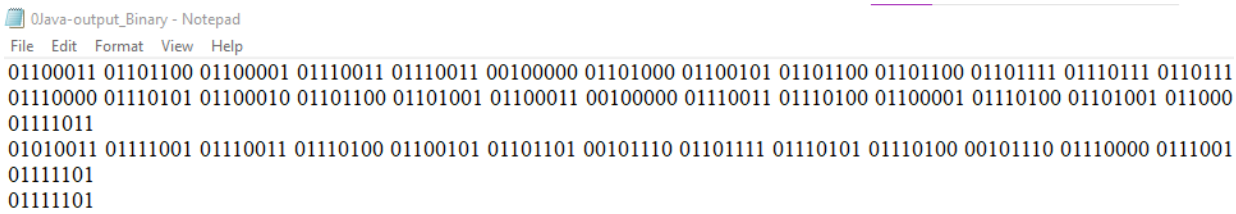

Fig. 8 Extracted Image to Binary Matrix

After the conversion of ASCII value, each character value is represented as its **8-bit** binary equivalent. **Fig. 8** displays the mapping of the binary extracted text wherein every character forms a sequence of 0s and 1s. This binary encoding will be further useful in perceiving the nature of bitwise composition of characters and hence to create Braille code. The binary patterns for each character of the extracted text are visually represented through a binary heatmap.

Musik in bayern
ISSN: 0937-583x Volume 90, Issue 1 (Jan -2025)
https://musikinbayern.com **DOI https://doi.org/10.15463/gfbm-mib-2025-*376***

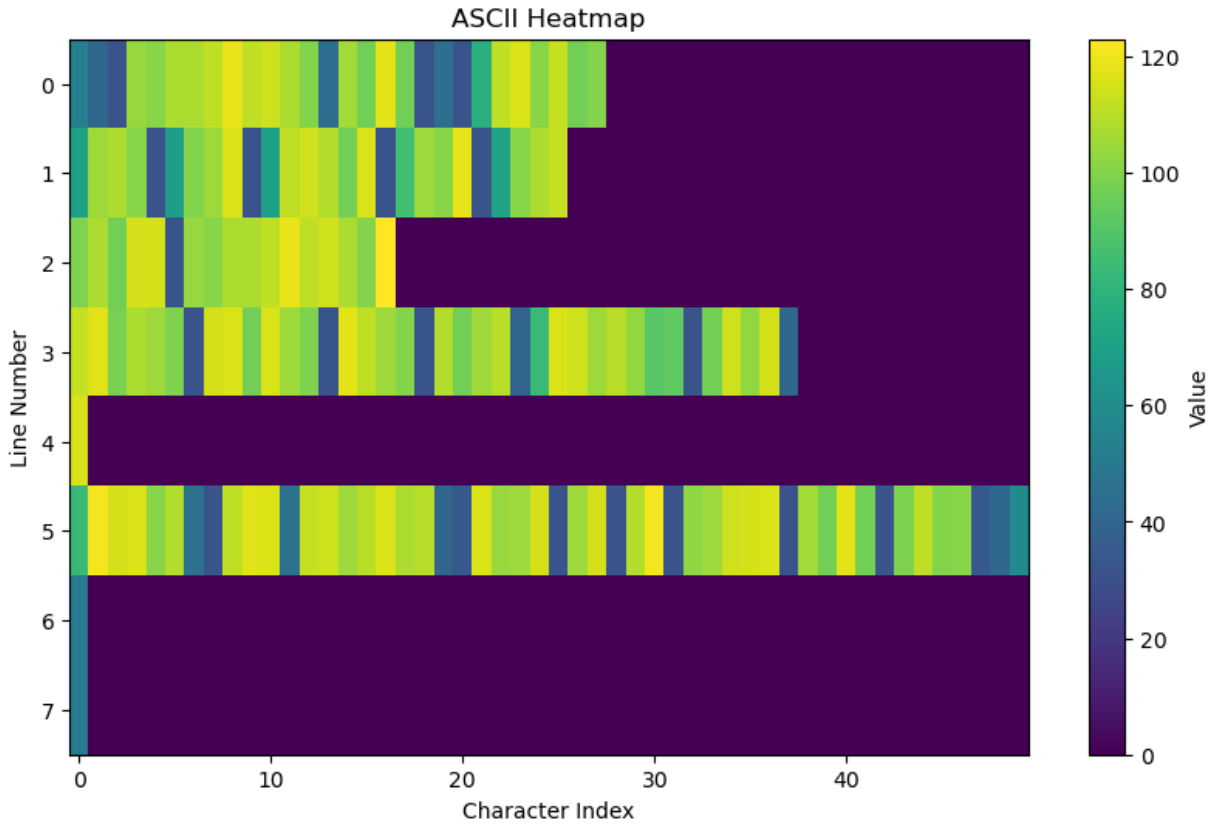

Fig. 9 ASCII Heatmap mapped using line number and character index

The ASCII heatmap illustrated through **Fig. 9** delineates the numerical encoding of the source code based on Java. It points out deviations of existing character values, with the brighter areas indicating higher ASCII values accommodating for symbols or uppercase letters and darker zones indicating spaces or punctuation marks.

Musik in bayern
ISSN: 0937-583x Volume 90, Issue 1 (Jan -2025)
https://musikinbayern.com **DOI https://doi.org/10.15463/gfbm-mib-2025-*376***

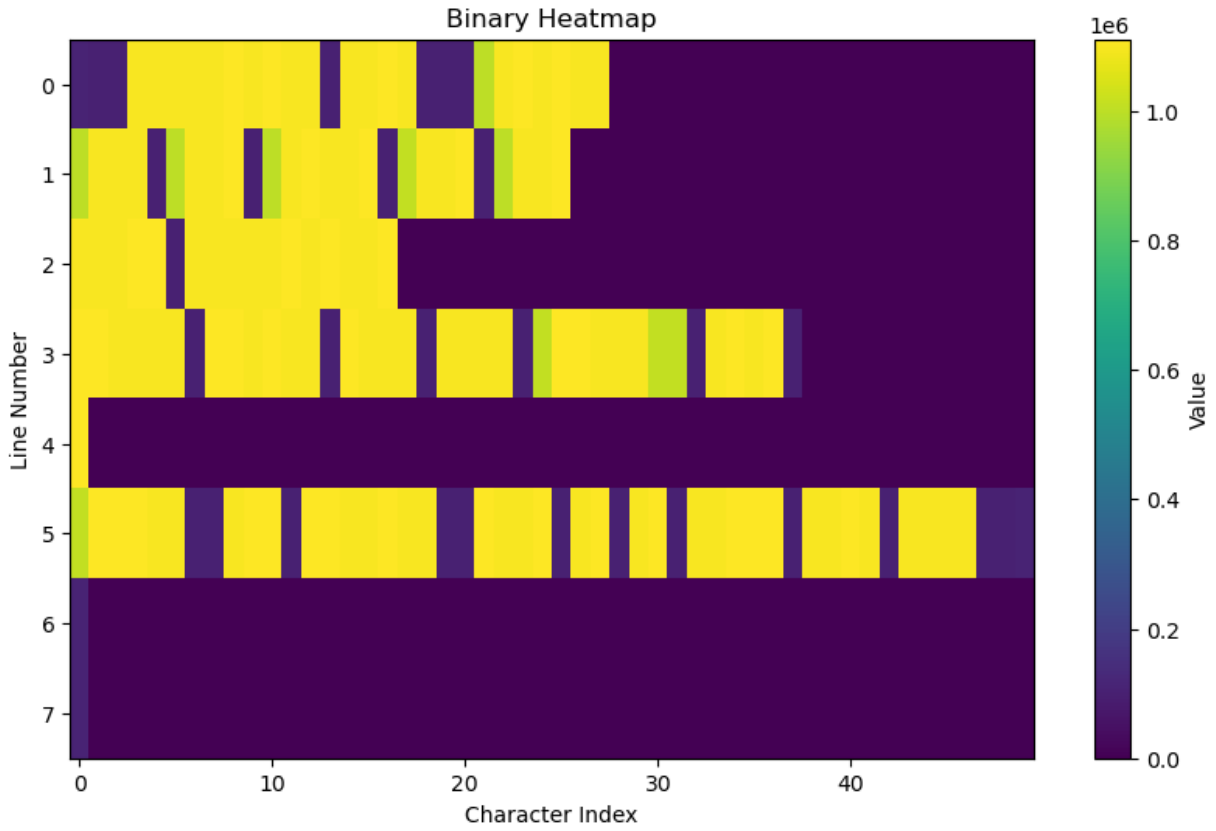

Fig. 10 Binary Heatmap mapped using line number and character index

The binary heatmap in **Fig. 10** visualizes the binary structure of the ASCII values. Each character's 8-bit binary equivalent is exhibited, displaying distinct binary patterns across code lines that are useful for feature extraction and Braille mapping.

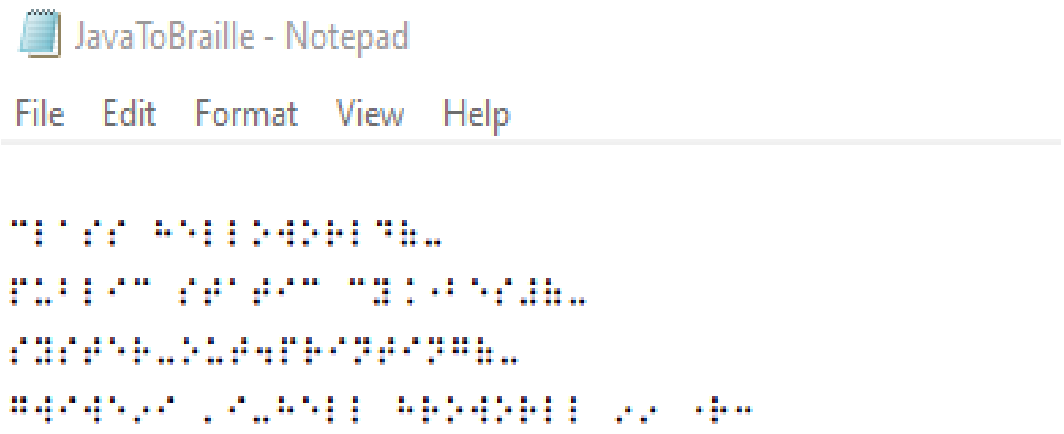

Fig.11 Java Source Code to Braille Conversion

The translation of the entire program code into the form of Braille symbols is shown in **Fig. 11**. Each dot of the Braille cell represents a character or a symbol from the original source code. This is achieved by following a certain mapping and conversion algorithm. This final figure confirms

that a mapping and structural integrity of the Braille output have been achieved since it provides visual validation of the last step in the Java-to-Braille conversion system. A successful conversion process can take any Java source code and render it accessible and readable by the low-vision programmer while retaining the logic and content of the original program.

XI Conclusion

The present work describes an automatic Java source code to Braille text translation system to help blind programmers. Utilizing state-of-the-art image preprocessing, Optical Character Recognition (OCR), and feature extraction methods like Discrete Wavelet Transform (DWT), Fast Fourier Transform (FFT), and Shannon Entropy ensures that text extraction is carried out precisely with the correct logical mapping of code to tactile Braille. An average of 2.1 seconds per image process was required for the system to extract text accurately, and it could logically code into tactile Braille with an error rate of 1%. Such speeds and efficiencies promise great hope for those involved in tackling the more complex aspects of programming syntax efficiently. The developed framework enhances accessibility for visually impaired individuals, permitting them to study, write, and debug Java packages independently. Braille mappings for alphabets, numbers, and special characters are carefully assembled so that the output respects the integrity of the code as inscribed by the original. The future will support real-time Braille conversion and validation as Braille will be decoded back to the source code that generated it. In addition, ASCII and binary representations will seamlessly integrate into predefined Braille cell formats to make the tactile output correspond with the syntax of the programming. An even stronger validation framework will be there, where automatically the Braille output will be compared with the original code. This will provide assurance for the development of an interactive real-time system, whereby translation and refining Braille will be validated for the usage of the sightless.

References

[1] J. Memon, M. Sami, R. A. Khan and M. Uddin, "Handwritten Optical Character Recognition (OCR): A Comprehensive Systematic Literature Review (SLR)," in IEEE Access, vol. 8, pp. 142642-142668, 2020, doi: 10.1109/ACCESS.2020.3012542.

[2] Khan, A., & Khusro, S. (2020). An insight into smartphone-based assistive solutions for visually impaired and blind people: issues, challenges and opportunities. Universal Access in the Information Society, 20. https://doi.org/10.1007/s10209-020-00733-8.

[3] Joshi, N., & Katyayan, P. (2023, March 1). *A Model for Translation of Text from Indian Languages to Bharti Braille Characters*. IEEE Xplore. https://doi.org/10.1109/ISCON57294.2023.10112021.

[4] Hossain, S. A., Fakhruddin Muhammad Mahbub-ul-Islam, Azam, S., & Khan, A. I. (2013). Bangla Braille Adaptation. IGI Global EBooks, 16–34. https://doi.org/10.4018/978-1-4666-3970-6.ch002.

[5] Alufaisan, S., Albur, W., Alsedrah, S., & Latif, G. (2021). Arabic Braille Numeral Recognition Using Convolutional Neural Networks. *Lecture Notes in Electrical Engineering*, 87–101. https://doi.org/10.1007/978-981-33-4909-4_7.

[6] Falgoon Sen Apu, Fatema Islam Joyti, Ala Uddin Anik, Wasi Uddin Zobayer, Atanu Kumar Dey, & Sakib Sakhawat. (2021). Text and Voice to Braille Translator for Blind People. *2021*

International Conference on Automation, Control and Mechatronics for Industry 4.0 (ACMI). https://doi.org/10.1109/acmi53878.2021.9528283.

[7] Latif, G., Mohammad, N., AlKhalaf, R., AlKhalaf, R., Alghazo, J., & Khan, M. (2020). An Automatic Arabic Sign Language Recognition System based on Deep CNN: An Assistive System for the Deaf and Hard of Hearing. *International Journal of Computing and Digital Systems*, *9*(4), 715–724. https://doi.org/10.12785/ijcds/090418.

[8] Joshi, N., & Katyayan, P. (2023, March 1). *A Model for Translation of Text from Indian Languages to Bharti Braille Characters*. IEEE Xplore. https://doi.org/10.1109/ISCON57294.2023.10112021.

[9] Shokat, S., Riaz, R., Rizvi, S. S., Khan, K., Riaz, F., & Kwon, S. J. (2020). Analysis and Evaluation of Braille to Text Conversion Methods. *Mobile Information Systems*, *2020*, 1–14. https://doi.org/10.1155/2020/3461651.

[10] Shimomura, Y., Kawabe, H., Nambo, H., & Seto, S. (2019). Braille Translation System Using Neural Machine Translation Technology I - Code Conversion. *Advances in Intelligent Systems and Computing*, 335–345. https://doi.org/10.1007/978-3-030-21248-3_25.

[11] J. Aswini, L. Krishnaa M, C. Lakshmipriya, G. Lavanya and S. S. R, "Translation System for the Visually Impaired from English to Braille," *2024 2nd World Conference on Communication & Computing (WCONF)*, RAIPUR, India, 2024, pp. 1-4, doi: 10.1109/WCONF61366.2024.10692175.

[12] B. Gopinath, S. Nagarathinam and M. Alagumeenaakshi, "Development of Speech and Text to Braille Script Converter for Blind and Deaf People," *2023 2nd International Conference on Advancements in Electrical, Electronics, Communication, Computing and Automation (ICAECA)*, Coimbatore, India, 2023, pp. 1-5, doi: 10.1109/ICAECA56562.2023.10200926.

[13] S. Ramachandran, N. Rajan, K. N. Pallavi, J. Subashree, S. Suchithra and B. Sonal, "Communication Device for the Visual and Hearing Impaired Persons to Convert Braille Characters to English Text," *2021 International Conference on Emerging Smart Computing and Informatics (ESCI)*, Pune, India, 2021, pp. 587-592, doi: 10.1109/ESCI50559.2021.9396859.

[14] S. Ramachandran, G. D, P. K N and N. Rajan, "Text to Braille Converting Communication Device forthe Visual and Hearing Impaired Persons," *2021 International Conference on Computer Communication and Informatics (ICCCI)*, Coimbatore, India, 2021, pp. 1-5, doi: 10.1109/ICCCI50826.2021.9402590.

[15] S. Kumari, A. Akole, P. Angnani, Y. Bhamare and Z. Naikwadi, "Enhanced Braille Display Use of OCR and Solenoid to Improve Text to Braille Conversion," *2020 International Conference for Emerging Technology (INCET)*, Belgaum, India, 2020, pp. 1-5, doi: 10.1109/INCET49848.2020.9153996.

[16] F. S. Apu, F. I. Joyti, M. A. U. Anik, M. W. U. Zobayer, A. K. Dey and S. Sakhawat, "Text and Voice to Braille Translator for Blind People," *2021 International Conference on Automation, Control and Mechatronics for Industry 4.0 (ACMI)*, Rajshahi, Bangladesh, 2021, pp. 1-6, doi: 10.1109/ACMI53878.2021.9528283.

[17] Parthiban, T., Reshmika, D., Lakshmi, N., Ponraj, A. (2022). Handwritten Text to Braille for Deaf-Blinded People Using Deep Neural Networks and Python. In: Marriwala, N., Tripathi, C., Jain, S., Kumar, D. (eds) Mobile Radio Communications and 5G Networks. Lecture Notes in Networks and Systems, vol 339. Springer, Singapore. https://doi.org/10.1007/978-981-16-7018-3_28.

[18] K. Shomenov, A. Yuldashov and M. H. Ali, "A Compact 3D Printed Text-to-Braille Converting Device with Optical Character Recognition (OCR)," *2023 10th International Conference on*

Electrical and Electronics Engineering (ICEEE), Istanbul, Turkiye, 2023, pp. 12-17, doi: 10.1109/ICEEE59925.2023.00010.

[19] M. Kavitha, V. Meenakshi, M. Pushpavalli, S. Amudha, S. Bharathi and P. Pavithra, "Communication Device for Converting Text to Braille language for Visually Impaired People," *2023 International Conference on Inventive Computation Technologies (ICICT)*, Lalitpur, Nepal, 2023, pp. 1016-1023, doi: 10.1109/ICICT57646.2023.10134300.

[20] Saxena, D. Verma, J. Pathak and R. K. Singh, "A Device for Automatic Conversion of Speech to Text and Braille for Visually and Hearing Impaired Persons," *2022 8th International Conference on Signal Processing and Communication (ICSC)*, Noida, India, 2022, pp. 560-564, doi: 10.1109/ICSC56524.2022.10009287.

[21] Chougule, S., & Patil, K. (2020). A review on the text-to-braille conversion system. *IOP Conference Series: Materials Science and Engineering*, *846*(1), 012008. https://doi.org/10.1088/1757-899X/846/1/012008.