DIRE DAWA UNIVERSITY

INSTITUTE OF TECHNOLOGY

SCHOOL OF COMPUTING

SOFTWARE ENGINEERING DEPARTMENT

Course Title: - Introduction To Artificial Intelligence

Course Code: -Soeng4092

GROUP MEMBERS	**ID.NO**
1. Rediat Sisay	1303693
2. Israel Abebaw	1303115
3. Haile Kinfu	1200523

Submitted to: Inst. Teshome.M

Submission Date: - June /2024

Acknowledgements

We would like to express our deepest gratitude to all those who have supported and guided us throughout the completion of this project.

First and foremost, we would like to express our sincere gratitude to our instructor, **Teshome.M** for his guidance and support throughout this course. Your expertise and encouragement have been invaluable.

We are also deeply thankful to our families for their unwavering love, patience, and encouragement. Their belief in our abilities and their support, both emotionally and financially, have been the foundation of our success.

A special thanks to our group members for their hard work and collaboration.

Finally, we appreciate Dire Dawa University, especially the Department of Software Engineering, for providing us with the resources and opportunities to complete this project.

Thank you all.

Abstract

This project focuses on the development of an automated Braille recognition system tailored for Amharic Braille, a language for which there is limited support in existing optical Braille recognition (OBR) systems. The primary aim is to create a model that can accurately translate images of Amharic Braille into their corresponding textual representations, thus improving accessibility for Amharic-speaking individuals who are visually impaired.

Existing Braille recognition projects primarily focus on widely used Braille systems, utilizing techniques ranging from traditional image processing to advanced deep learning models. However, these efforts typically leverage existing datasets, which are not available for less common languages like Amharic. Our project addresses this gap by developing a custom dataset from scratch, a process involving manual creation, filling, digitizing, and annotating an Amharic Braille chart.

We implemented a Convolutional Neural Network (CNN) using Keras, leveraging its capability for effective image feature extraction and classification. The architecture comprises multiple convolutional layers for feature extraction, followed by fully connected layers for classification, and an output layer with 98 units representing the different Amharic characters. To enhance the model's robustness and accuracy, we employed various image augmentation techniques, including rotation, shifting, and brightness adjustment. These augmentations simulate real-world conditions where images may be misaligned or captured under varying lighting, ensuring the model's capability to handle diverse input variations.

During the development process, we encountered significant challenges, particularly in sourcing accurate representations of Amharic Braille and creating a high-quality dataset. Extensive research, including the study of academic papers and sourcing images from Wikipedia, aided in overcoming these challenges. Manual annotation and iterative model training were essential in refining the dataset and improving the model's performance.

Our experimental results demonstrated the efficacy of the data augmentation techniques and the importance of a comprehensive dataset. Initial models trained on smaller datasets with limited augmentations showed suboptimal accuracy. By increasing the dataset size and incorporating diverse augmentations, we achieved a model with significantly improved accuracy. The final

dataset comprised 11,755 images, carefully curated to avoid diminishing returns from over-augmentation, which resulted in the most accurate and reliable model.

This project not only contributes to the field of Braille recognition by supporting a less commonly recognized language but also highlights the potential for extending similar systems to other languages with limited existing support. Future work will involve expanding the model to recognize Braille in additional languages such as Afaan Oromoo and Af Somali. Additionally, real-world testing will be conducted to validate and refine the model's performance in practical scenarios.

In conclusion, this project represents a significant advancement in making written information accessible to visually impaired individuals in the Amharic-speaking community. The development of a custom dataset and the implementation of a robust CNN architecture with comprehensive image augmentation techniques were critical to the project's success. By addressing the specific challenges associated with Amharic Braille recognition, we have laid the groundwork for future enhancements and broader language support, ultimately contributing to greater accessibility and inclusivity for visually impaired individuals.

Table of Contents

1. Introduction

The advent of machine learning and computer vision has opened up new possibilities for enhancing accessibility for visually impaired individuals. One critical area of focus is the development of automated systems for Braille recognition, which can significantly improve the ease of access to written information. Braille, a tactile writing system used by people who are visually impaired, represents a unique challenge in terms of optical recognition due to its reliance on raised dots rather than traditional visual text.

Our project aims to address this challenge by developing an innovative Braille recognition system specifically tailored for Amharic Braille, a less commonly supported language in existing optical Braille recognition (OBR) systems. The primary objective of our project is to create a system that can accurately translate images of Amharic Braille into their corresponding textual representations. This endeavor is not only technically challenging but also essential for enhancing educational and informational access for Amharic-speaking individuals who are visually impaired.

Existing efforts in the field have largely focused on more widely used Braille systems, such as English Braille, leveraging techniques ranging from traditional image processing to advanced deep learning models. Projects like "brl_ocr" and Antony Jr.'s Optical Braille Recognition have demonstrated the potential of using Python, PyTorch, and OpenCV for Braille OCR, with some extending into machine learning for improved accuracy. However, these projects typically utilize existing datasets, which are not readily available for less common languages like Amharic.

Our project differentiates itself by addressing this gap. We developed a custom dataset for Amharic Braille from scratch, a labor-intensive process that involved manually creating, filling, digitizing, and annotating a Braille chart. This bespoke dataset forms the foundation of our recognition system, ensuring it is finely tuned to the nuances of Amharic Braille.

The system employs a Convolutional Neural Network (CNN) architecture implemented in Keras, chosen for its effectiveness in image feature extraction and classification tasks. The model's architecture includes multiple convolutional layers for feature extraction, followed by fully connected layers for classification, culminating in an output layer with 98 units representing the different Amharic characters.

To enhance the robustness and accuracy of our model, we employed various image augmentation techniques, such as rotation, shifting, and brightness adjustment, simulating real-world conditions where images may be misaligned or captured under varying lighting. These techniques, combined with the comprehensive dataset preparation process, ensure that our model can handle a wide range of input variations.

Throughout the development process, we faced significant challenges, particularly in sourcing accurate representations of Amharic Braille and creating a high-quality dataset. These challenges were overcome through extensive research, manual annotation, and iterative model training, leading to a final model that demonstrates promising accuracy and reliability.

In conclusion, our project not only contributes to the field of Braille recognition but also highlights the importance of supporting less commonly recognized languages like Amharic. Future work will focus on extending the system to other languages such as Afaan Oromoo and Af Somali, and conducting real-world testing to further validate and refine the model. This project represents a significant step towards making written information more accessible to visually impaired individuals in the Amharic-speaking community.

2. Related Work

Currently, there are several notable efforts by software engineers to develop Braille recognition systems using Python, leveraging various techniques and frameworks. Here, we discuss some of the prominent projects and methodologies in this domain and highlight what makes our group project distinct from these existing works.

2.1 Optical Braille Recognition (OBR)

1. **Project "brl_ocr"**: The "brl_ocr" project on GitHub is a significant endeavor in Braille Optical Character Recognition (OCR). Utilizing Python and PyTorch, this project aims to detect Braille characters with high accuracy. It builds on existing datasets like the AngelinaDataset and incorporates deep learning models to enhance the precision and efficiency of Braille recognition. This project exemplifies how deep learning can be effectively applied to improve OCR systems, particularly for specialized applications like Braille recognition.
2. **Optical Braille Recognition by Antony Jr.**: Another notable project in this space is the Optical Braille Recognition system developed by Antony Jr., which employs OpenCV in Python for image processing and Braille recognition. This project is designed to be straightforward yet effective, with provisions to integrate machine learning for enhanced performance. By focusing on the simplicity and potential for improvement through machine learning, this project demonstrates a practical approach to Braille recognition that can be built upon for more advanced applications.

2.2 Machine Learning and Deep Learning Techniques

1. **AI-Powered Braille Learning System**: An innovative approach in this domain is the development of an IoT-based AI-powered Braille learning system. This system uses convolutional neural networks (CNNs) and transfer learning to recognize and translate Braille characters into audio. The primary goal of this system is to facilitate the learning process for Braille and improve the accessibility of educational resources for visually impaired individuals. The use of advanced AI techniques highlights the potential for technology to make significant strides in accessibility.

2. **Object Detection CNN for Braille Recognition**: Another method involves using object detection CNNs to recognize entire Braille characters from images. This approach is robust against deformations and perspective distortions, making it suitable for applications using smartphone cameras. This research includes the development of the Angelina Braille Images Dataset, which serves as a valuable resource for training and evaluating Braille recognition models.

2.3 Pressure Sensors and Tactile Perception

Research on tactile perception techniques has also contributed to advancements in Braille recognition. The use of capacitive pressure sensors, which convert pressure into electrical signals, has proven effective in recognizing Braille characters. This method enhances the sensitivity and response time of the sensors, making the tactile reading of Braille more accurate and efficient. Such innovations demonstrate the potential for combining traditional tactile sensing with modern technological advancements to improve Braille accessibility.

2.4 Distinguishing Factors of Our Project

While these projects represent significant advancements in the field of Braille recognition, our group project distinguishes itself in several key areas:

1. **Focus on Amharic Braille**: Our project specifically addresses the recognition of Amharic Braille, which is not commonly explored in the software engineering space. This focus on a less common language makes our project unique and fills a significant gap in existing research and application.
2. **Custom Dataset Development**: To overcome the limitations of common datasets, we developed a dataset specifically tailored to our system. This dataset includes a comprehensive representation of the Amharic alphabet in Braille, ensuring that our model can accurately recognize and interpret Amharic Braille characters. This bespoke dataset is a critical component that sets our project apart from others that rely on more general datasets.
3. **Real-Time Image Processing**: Our system works by taking a picture of Amharic Braille and determining which alphabet the Braille represents. This real-time processing capability provides immediate feedback, which is particularly beneficial for applications such as Braille literacy education and real-time text conversion.

These distinguishing features highlight the unique contributions of our project to the field of Braille recognition and demonstrate how it builds upon and advances the existing body of work. By focusing on Amharic Braille, developing a custom dataset, and prioritizing real-time processing and user experience, our project offers a novel and valuable solution in the domain of Braille recognition.

3. Explanation of Our Target Language: Amharic

Our target language for this project is Amharic. As the primary language spoken by our team members and the language we understand the most, Amharic was the natural choice for our project. This familiarity with Amharic not only facilitated our work but also ensured that we could accurately interpret and handle the nuances of the language in its Braille representation.

Initially, we considered extending our project to include Afaan Oromoo, another significant language spoken in Ethiopia. However, our efforts to find comprehensive resources and datasets for Afaan Oromoo Braille proved unfruitful. This lack of available data, coupled with the time constraints imposed by our academic schedule and upcoming exams, led us to focus solely on the Amharic Braille model. By concentrating our efforts on Amharic, we aimed to develop a highly accurate and reliable Braille recognition system.

The decision to target Amharic Braille also aligns with a broader objective to address the specific needs of the Amharic-speaking visually impaired community. Given the scarcity of existing technological solutions tailored to Amharic Braille, our project aims to fill this gap and provide a valuable tool for enhancing accessibility and literacy among Amharic speakers.

In summary, our focus on Amharic as the target language for this project is driven by our team's linguistic expertise, the practical considerations of data availability, and our commitment to creating a meaningful and impactful solution for the Amharic-speaking community.

4. Challenges Faced in Our Project

The biggest challenge we encountered while working on our project was gaining a thorough understanding of what Amharic Braille looked like and how it operated. This obstacle required an extensive search for reliable resources and references on Amharic Braille, which proved to be a daunting task.

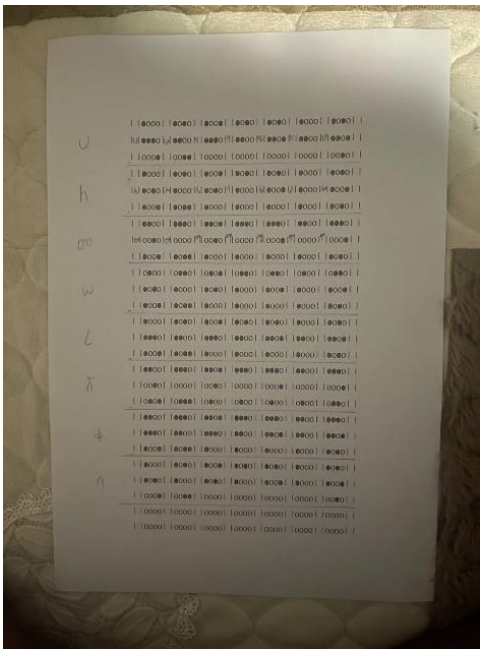
To address this challenge, we embarked on a comprehensive research effort. We delved into various online and offline sources, trying to find accurate representations of Amharic Braille. One significant resource we utilized was a study paper by Teshome Alemu on the recognition of Amharic Braille, which provided valuable insights and a foundational understanding of the Braille system. The paper can be accessed here: Recognition of Amharic Braille

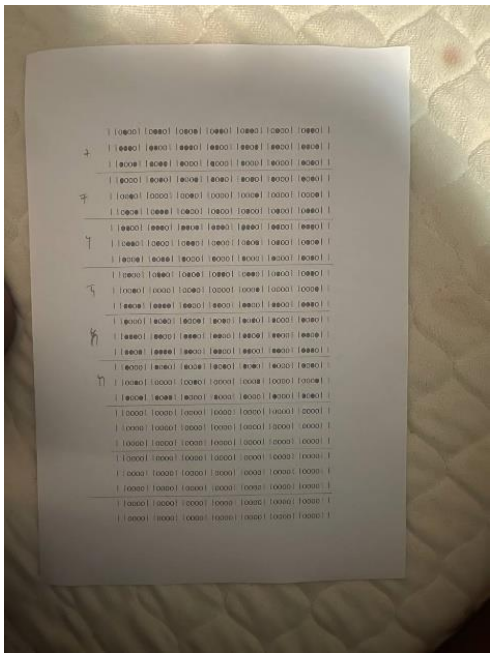
Despite the insights gained from Alemu's paper, we still faced difficulties due to the lack of high-quality visual references. Our breakthrough came when we discovered an image of Amharic Braille in an article on a Wikipedia page. This image became a crucial reference point for our project.

The picture that was found on Wikipedia: image 1

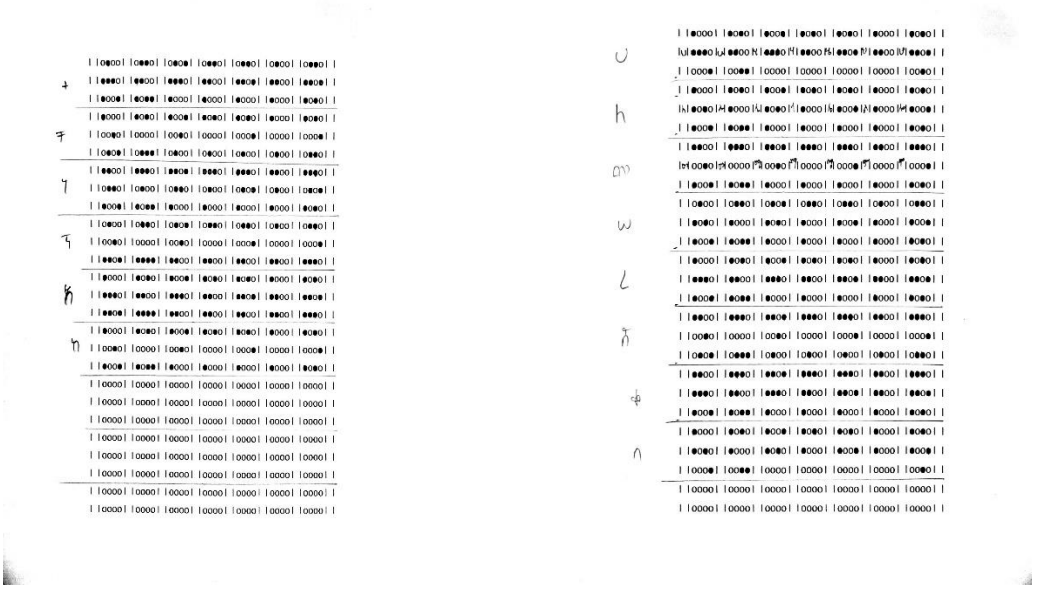
However, the low quality of the image posed a significant problem. It was too pixelated and unclear to be directly useful for our Braille recognition system. To overcome this, we took an innovative approach. We decided to create a new, high-quality Amharic Braille chart ourselves. This process involved several meticulous steps:

1. **Manual Reconstruction**:
    - We printed out the low-quality image and manually filled in the black parts of each Braille cell to make them more distinct and accurate.
2. **Digitization**:
    - After reconstructing the chart, we scanned it to convert it into a digital format. This allowed us to manipulate and process the image using software tools.
3. **Dataset Creation**:
    - We cropped each individual alphabet from the scanned chart manually. This labor-intensive process was necessary to build a comprehensive dataset tailored specifically for our recognition system.





Amharic braille chart that was created by our team before scanning: image 2



Amharic braille chart that was created by our team after scanning: image 2.1

This hands-on approach to creating our own dataset not only solved the problem of image quality but also ensured that we had a precise and reliable reference for Amharic Braille. It allowed us to develop a robust model capable of accurately recognizing Amharic Braille characters.

5. Data Collection and Preparation

As stated earlier in this paper, the main process of collecting the data and preparing it for model training was a labor-intensive one. Our journey began with the challenge of finding accurate representations of Amharic Braille, which are not widely available. This scarcity of resources necessitated a hands-on approach to creating our dataset from scratch.

Steps Involved in Data Collection and Preparation:

1. **Manual Preparation of Amharic Braille Chart**:
   - Due to the lack of high-quality references, we decided to create a new Amharic Braille chart manually. This involved printing out the existing low-quality image and meticulously filling in the Braille cells by hand to ensure they were clearly defined and accurate.
2. **Digitization of the Chart**:
   - Once we had a clearly defined manual chart, the next step was to digitize it. We scanned the chart to convert it into a digital format. This allowed us to manipulate the image using various software tools, which was essential for the next stages of data preparation.
3. **Cropping Individual Letters**:
   - The digitized chart was then processed to crop each Braille letter one by one. This step was particularly laborious, as it involved ensuring that each cropped image precisely represented a single Amharic letter. This meticulous process was necessary to create a comprehensive and accurate dataset.
4. **Dataset Compilation**:
   - After cropping the individual letters, we compiled them into a dataset suitable for training our Braille recognition model. This dataset included a diverse set of samples for each Amharic Braille character, ensuring that the model could learn to recognize the characters accurately under various conditions.
5. **Validation and Quality Check**:
   - To ensure the quality and accuracy of our dataset, we conducted several rounds of validation and quality checks. This involved cross-referencing with known standards of Amharic Braille and verifying that each Braille character was correctly represented.

By manually creating and digitalizing the Amharic Braille chart, we ensured that our dataset was tailored specifically to our project's needs. This hands-on data collection and preparation process, although time-consuming, was crucial in building a reliable and accurate Braille recognition model. The effort invested in this stage laid a solid foundation for the subsequent model training and development, ultimately contributing to the success of our project.

5.1 Dataset Description

The dataset used in our project included a comprehensive collection of images, each carefully cropped to represent an individual Amharic Braille letter. This dataset was meticulously curated to ensure that each Braille character was accurately depicted and easily recognizable by the model during training.

Key Features of the Dataset:

1. **Comprehensive Coverage**:
   - The dataset encompasses most of the letters of the Amharic alphabet in their Braille representations. This comprehensive coverage ensures that the model is exposed to almost every possible character it might need to recognize.
2. **Manual Cropping**:
   - Each image in the dataset was manually cropped from a digitized version of a hand-prepared Amharic Braille chart. This manual process ensured high precision and accuracy in the representation of each character.
3. **High-Quality Images**:
   - The images were scanned at a high resolution to maintain clarity and detail. This high-quality imaging was essential for the effective training of our recognition model, allowing it to distinguish subtle features in the Braille characters.
4. **Consistent Formatting**:
   - All images were standardized in terms of size and format to ensure consistency across the dataset. This standardization was important to facilitate the model training process and improve recognition accuracy.

5. **Diverse Samples**:
    - The dataset included multiple samples of each Braille character to account for slight variations that might occur in real-world usage. This diversity helps the model generalize better and improves its robustness.

By assembling a well-structured and high-quality dataset, we laid a strong foundation for the development of our Braille recognition model. This dataset played a critical role in enabling the model to learn effectively and perform accurately in recognizing Amharic Braille characters.

5.2 Dataset Annotation Guidelines

To ensure consistency and accuracy in our dataset, we followed a specific set of guidelines for annotating each image. These guidelines facilitated the systematic labeling and uniform formatting of all Braille characters in our dataset, which was crucial for effective model training.

Annotation Process:

1. **Sequential Labeling**:
    - Each image in the dataset was labeled sequentially with a unique identifier starting from 001 to 098. This numeric labeling system provided a straightforward and organized way to reference each Amharic Braille letter. For example:
        - **001** represents the Amharic letter ሀ (Ha).
        - **002** represents the Amharic letter ሁ (Hu).
        - **003** represents the Amharic letter ሂ (Hi).
        - This sequence continued for all the letters in the Amharic alphabet, ensuring that each character was distinctly identified and easily accessible.
2. **Uniform Image Dimensions**:
    - To maintain consistency in the dataset, we used a Python script to crop each image to the same dimensions. Every image was resized to **96 x 170 pixels**. This standardization was essential for the following reasons:
        - **Consistency**: Ensuring that all images had the same dimensions eliminated variability that could affect the model's performance.
        - **Model Training**: Uniform dimensions allowed the model to process the images more efficiently, leading to more accurate recognition.

3. **Image Cropping Script**:
    - We developed and utilized a Python code for image cropping, which automated the resizing process. This script ensured that each image adhered to the specified dimensions, regardless of any augmentation applied during training. The code was designed to:
        - Adjust the size of each cropped image to fit the 96 x 170 pixels requirement.
        - Save the resized images with their corresponding labels, maintaining the sequential order.
4. **Quality Assurance**:
    - After annotation and cropping, each image was reviewed to ensure it met the annotation guidelines and dimensional standards. This quality check was vital to prevent any mislabeled or improperly sized images from entering the training dataset.

By adhering to these annotation guidelines, we ensured that our dataset was systematically organized and uniformly formatted. This meticulous attention to detail was crucial for training an effective Braille recognition model, as it allowed the model to learn from a consistent and accurately labeled dataset.

5.3 Dataset Annotation

In order to enhance the robustness and accuracy of our Braille recognition model, we applied several data augmentation techniques to simulate real-world conditions. This process involved systematically altering each image to create variations that the model might encounter in actual usage, such as changes in lighting, orientation, and positioning.

Augmentation Techniques:

1. **Rotation**:
    - Each image was rotated by 9 degrees to simulate scenarios where the Braille text might not be perfectly aligned. This augmentation helps the model to generalize better by recognizing Braille characters even when they are slightly rotated.

2. **Shifting:**
    - Each image was shifted by 9 pixels to simulate slight positional changes that can occur in image capture. This helps the model to handle small displacements in the position of Braille characters.
3. **Brightness Adjustment**:
    - Images were both brightened and darkened to simulate different lighting conditions. This ensures the model can accurately recognize Braille characters under varying lighting scenarios, which is crucial for real-world applications

Summary of Dataset Augmentation:

- **Rotation**: Each image was rotated by 9 degrees to handle variations in text alignment.
- **Shifting**: Each image was shifted by 9 pixels to accommodate positional changes.
- **Brightness Adjustment**: Each image was darkened and brightened to mimic different lighting conditions.

These augmentations were applied systematically to ensure that each modified image adhered to the same guidelines and maintained the necessary dimensions (96 x 170 pixels). By incorporating these variations, we created a diverse and comprehensive dataset that enabled our Braille recognition model to perform accurately and robustly in a wide range of real-world conditions.

To effectively train and evaluate our Braille recognition model, we carefully split our dataset into training, validation, and test sets. This distribution ensures that the model has sufficient data to learn from, validate its performance during training, and be tested on unseen data to evaluate its generalization capability.

5.4 Splitting the Dataset:

1. **Initial Train-Test Split**:
   - We divided the dataset into a training set and a test set using an 80-20 split. This means that 80% of the data was used for training the model, while the remaining 20% was reserved for testing its performance.

```
In [15]: X_train, X_test, y_train, y_test = train_test_split(images_list, name_list, test_size=0.2, random_state=108)
```

2. **raining and Validation Split**:
   - Within the training set, we further split the data into training and validation subsets using the validation_split parameter in the model.fit() function. This split helps in monitoring the model's performance on a separate validation set during training to prevent overfitting.
   - The validation split was set to 30%, meaning that 30% of the training set was used for validation.
   - Code for Training and Validation Split

```
In [19]: es1 = EarlyStopping(patience=20, monitor="val_sparse_categorical_accuracy", mode="auto")
         es2 = EarlyStopping(patience=20, monitor="val_loss", mode="auto")


         history = model.fit(x=X_train,
                             y=y_train,
                             epochs=5,
                             validation_split=0.3,
                             callbacks=[es1, es2])
```

5.5 Summary of Dataset Distribution:

1. **Original Dataset**:
   - **Training Set (X_train, y_train)**: 80% of the dataset.
   - **Test Set (X_test, y_test)**: 20% of the dataset.
2. **Training Set Split**:
   - **Final Training Set**: 70% of the training set (which is 56% of the original dataset).
   - **Validation Set**: 30% of the training set (which is 24% of the original dataset).

Final Distribution:

- **Final Training Set**: 56% of the original dataset.
- **Validation Set**: 24% of the original dataset.
- **Test Set**: 20% of the original dataset.

This distribution ensures that the model has ample data to learn from (training set), sufficient data to validate its learning during training (validation set), and an adequate amount of data to test its performance on unseen samples (test set). This balanced approach helps in building a robust model that can generalize well to new, unseen Braille characters.

6. Architecture of Proposed System

The architecture of our Braille recognition system is built using a Convolutional Neural Network (CNN) implemented with Keras. The CNN architecture is designed to effectively process and classify images of Amharic Braille characters. Below is a detailed breakdown of the system's architecture:

Detailed Architecture

```
In [16]: model = keras.Sequential([
             keras.layers.Conv2D(filters=64, kernel_size=(3, 3), padding='same', activation='relu'),
             keras.layers.MaxPooling2D(),
             keras.layers.Conv2D(filters=64, kernel_size=(3, 3), padding='same', activation='relu'),
             keras.layers.MaxPooling2D(),
             keras.layers.Conv2D(filters=64, kernel_size=(3, 3), padding='same', activation='relu'),
             keras.layers.MaxPooling2D(),
             keras.layers.Flatten(),
             keras.layers.Dense(units=576, activation="relu"),
             keras.layers.Dense(units=288, activation="relu"),
             keras.layers.Dense(units=98, activation="softmax")
         ])
```

Detailed Breakdown

1. **Input Layer**
   - The input layer size is inferred from the shape of X_train when the model is first trained.
2. **Convolutional Layers**

- **Conv2D Layer 1**:
  - Filters: 64
  - Kernel Size: (3, 3)
  - Padding: 'same'
  - Activation: 'relu'
- **MaxPooling2D Layer 1**:
  - Reduces the spatial dimensions of the feature map.

- **Conv2D Layer 2**:
  - Filters: 64
  - Kernel Size: (3, 3)
  - Padding: 'same'
  - Activation: 'relu'
- **MaxPooling2D Layer 2**:
  - Further reduces the spatial dimensions.
- **Conv2D Layer 3**:
  - Filters: 64
  - Kernel Size: (3, 3)
  - Padding: 'same'
  - Activation: 'relu'
- **MaxPooling2D Layer 3**:
  - Further reduces the spatial dimensions.

3. **Flaten Layer**

- Converts the 2D feature maps from the convolutional layers into a 1D vector to feed into the fully connected layers.

4. **Fully Connected Layers (Dense Layers)**

- **Dense Layer 1**:
  - Units: 576
  - Activation: 'relu'
- **Dense Layer 2**:
  - Units: 288
  - Activation: 'relu'
- **Dense Layer 3** (Output Layer):
  - Units: 98 (indicating 98 classes for classification, corresponding to the 98 Amharic characters in our dataset)
  - Activation: 'softmax' (suitable for multi-class classification)

Additional Details

Compilation of the Model:

```
In [17]: model.compile(optimizer="Adam", loss="SparseCategoricalCrossentropy", metrics=["sparse_categorical_accuracy"])
```

- **Optimizer**: Adam
- **Loss Function**: Sparse Categorical Crossentropy
- **Metrics**: Sparse Categorical Accuracy

Early Stopping Callbacks

```
es1 = EarlyStopping(patience=20, monitor="val_sparse_categorical_accuracy", mode="auto")
es2 = EarlyStopping(patience=20, monitor="val loss", mode="auto")
```

Early Stopping: Monitors validation accuracy and loss, stopping training if no improvement is observed for 20 epochs to prevent overfitting.

Training the Model

```
history = model.fit(x=X_train,
                    y=y_train,
                    epochs=5,
                    validation_split=0.3,
                    callbacks=[es1, es2])
```

- **Training Set**: 80% of the dataset, further split into:
  - **Final Training Set**: 70% of the training set (56% of the original dataset)
  - **Validation Set**: 30% of the training set (24% of the original dataset)
- **Epochs**: 5

This architecture leverages three convolutional layers followed by pooling layers to extract spatial features from the Braille images. The extracted features are then flattened and passed through fully connected layers to classify the images into one of the 98 Amharic Braille characters. Early stopping ensures the model does not overfit and performs optimally on the validation set. This structured approach is aimed at achieving high accuracy and reliability in recognizing Amharic Braille characters.

7. Image Feature Extraction Techniques Used in Our Project

In our project, we employed various image feature extraction techniques to preprocess the data and enhance the model's ability to recognize Amharic Braille characters accurately. Here is a detailed explanation of these techniques:

7.1 Normalization

Normalization was used to scale pixel values of the images to the range [0, 1]. This helps in speeding up the convergence of the model during training and ensures consistency in the input data.

```
In [8]: images_list = np.array(images )/255.0
        name_list = np.array(name_list)
```

7.2 Label Encoding

The categorical labels of the images were encoded into numerical format using LabelEncoder from scikit-learn. This is crucial for training a machine learning model that requires numerical input.

```
In [12]: name_list.shape
         le = LabelEncoder()
         name_list =le.fit_transform(name_list)
```

7.3 Data Augmentation

To make the model robust and capable of generalizing well to new data, several data augmentation techniques were applied:

A. Rotation

Each image was rotated by 9 degrees to simulate scenarios where the Braille text might not be perfectly aligned. This augmentation helps the model recognize Braille characters even when they are slightly rotated.

```
In [11]: number = 0
index = 0
for dir in dir_list:
    img = Image.open(str(dir))
    for i in range(1):
        rotated_img = img.rotate(9, expand=True)  # Change the degree of rotation as needed
        rotated_img.save(r"C:/Users/Hello/Downloads/cropcorectedbrail/" + name_list[index] + "shift9" + str(number) + ".jpg")
        number += 1
    index += 1
```

B. Shifting

Each image was shifted by 9 pixels to simulate slight positional changes that can occur in image capture. This helps the model handle small displacements in the position of Braille characters.

```
In [11]: number = 0
index = 0
for dir in dir_list:
    img = Image.open(str(dir))
    for i in range(1):
        new_img =ImageChops.offset(img, 9)
        new_img.save(r"C:/Users/Hello/Downloads/brailshift/" +name_list[index]+"shift9"+str(number)+".jpg")
        number += 1
    index += 1
```

C. Brightness Adjustment

Images were both brightened and darkened to simulate different lighting conditions, ensuring the model can accurately recognize Braille characters under varying lighting scenarios.

```
In [28]: number = 0
index = 0
for dir in dir_list:
    img = Image.open(str(dir))
    enhancer = ImageEnhance.Brightness(img)
    factor = 0.5 #darkens the image
    im_output = enhancer.enhance(factor)
    im_output.save(r"C:/Users/Hello/Downloads/newshit/brailcorrect/" +name_list[index]+"dark"+str(number)+".jpg")
    factor = 1.4 #brightens the image
    im_output = enhancer.enhance(factor)
    im_output.save(r"C:/Users/Hello/Downloads/newshit/brailcorrect/" +name_list[index]+"bright"+str(number)+".jpg")
    number = number +1
    index = index +1
```

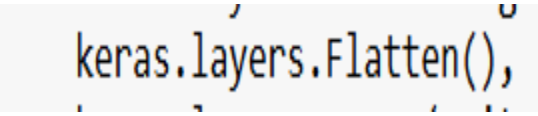
D. Convolutional Feature Extraction

The convolutional layers in the CNN were used to automatically extract features from the images. Convolutional layers apply filters to the input images to detect edges, textures, and patterns. These layers help the model learn hierarchical features.

```
In [16]: model = keras.Sequential([
    keras.layers.Conv2D(filters=64, kernel_size=(3, 3), padding='same', activation='relu'),
    keras.layers.MaxPooling2D(),
    keras.layers.Conv2D(filters=64, kernel_size=(3, 3), padding='same', activation='relu'),
    keras.layers.MaxPooling2D(),
    keras.layers.Conv2D(filters=64, kernel_size=(3, 3), padding='same', activation='relu'),
    keras.layers.MaxPooling2D(),
    keras.layers.Flatten(),
    keras.layers.Dense(units=576, activation="relu"),
    keras.layers.Dense(units=288, activation="relu"),
    keras.layers.Dense(units=98, activation="softmax")
])
```

E. Flattening

The Flatten layer converts the 2D feature maps from the convolutional layers into a 1D vector. This is necessary to feed the output into fully connected (dense) layers for classification.



Summary of Image Feature Extraction Techniques

1. **Normalization**: Scaling pixel values to the range [0, 1].
2. **Label Encoding**: Converting categorical labels to numerical labels.
3. **Data Augmentation**: Enhancing dataset diversity through transformations:
    - Rotation
    - Shifting
    - Brightness Adjustment
4. **Convolutional Layers**: Automatic feature extraction using filters to detect edges, textures, and patterns.
5. **Flattening**: Converting 2D feature maps to 1D vectors for classification.

These techniques are essential for preparing the images and labels for training a robust and accurate convolutional neural network model.

8. Experiment Results and Discussion

During our experimentation phase, we conducted multiple training sessions with varying dataset sizes and augmentation techniques to optimize the accuracy of our Amharic Braille recognition model.

Initial Experiments

First Experiment:

- **Dataset Size:** 10,000 image datasets
- **Augmentation Techniques:** Shifting and brightness/darkness adjustment
- **Outcome:** The model achieved accuracy, but it was not satisfactory according to our group's expectations.

Second Experiment:

- **Dataset Size:** Over 10,000 image datasets
- **Augmentation Techniques:** Rotation and brightness/darkness adjustment
- **Outcome:** Despite the increased dataset size and additional augmentation, the accuracy was still below our target.
- **Combined Model Training**
- Realizing that individual augmentations were not yielding the desired accuracy, we decided to combine the datasets from both experiments:
- **Combined Dataset Size:** Slightly over 20,000 image datasets
- **Augmentation Techniques:** Included both shifting, rotation, and brightness/darkness adjustments
- **Outcome:** The combined model demonstrated improved accuracy compared to the previous models, but we recognized the potential for further improvement.

Optimal Dataset Determination

We hypothesized that the diminishing returns from excessive augmentation were affecting our model's performance. Therefore, we aimed to find the optimal dataset size that would maximize accuracy without redundancy.

Optimal Dataset:

- **Total Images:** 11,755
- **Shift Augmented Dataset:** 5,679 images
- **Rotate Augmented Dataset:** 6,268 images
- **Augmentation Techniques:** Both datasets included brightness and darkness adjustments

Best Model Results

By training on the optimally sized dataset, we achieved the best model performance:

- **Accuracy:** Significantly improved and met our expectations
- **Observations:** The balanced dataset size and diverse augmentation techniques contributed to a more robust and accurate model.

Discussion

Challenges and Insights:

1. **Augmentation Impact:** While data augmentation is crucial for improving model generalization, excessive augmentation can lead to diminishing returns. Finding the right balance was key.
2. **Dataset Size:** Increasing the dataset size generally helps, but quality and diversity of the images are equally important.
3. **Experimentation:** Systematic experimentation with different augmentations and dataset sizes provided valuable insights into model performance optimization.

In conclusion, our iterative approach to dataset augmentation and size optimization led to the development of a highly accurate Amharic Braille recognition model, demonstrating the importance of balanced and diverse training data in machine learning projects.

9. Conclusion and Future Work

In conclusion, our project aimed to develop a robust and accurate Amharic Braille recognition system, addressing a significant gap in the accessibility tools available for the visually impaired community. Through a meticulous process of data collection, augmentation, and model training, we achieved a model that accurately recognizes Amharic Braille characters.

Key Achievements:

1. **Customized Dataset Creation:** We faced significant challenges in sourcing Amharic Braille data, necessitating the creation of a tailored dataset from scratch. This involved manually crafting and digitizing a Braille chart and ensuring each image was of high quality and correctly labeled.
2. **Data Augmentation Techniques:** To simulate real-world conditions and improve model generalization, we applied various augmentation techniques, including rotation, shifting, and brightness adjustments. These efforts were crucial in enhancing the model's robustness.
3. **Model Optimization:** By experimenting with different dataset sizes and combinations, we identified the optimal dataset size of 11,755 images, achieving the best balance between data diversity and model performance.
4. **Convolutional Neural Network (CNN) Architecture:** Our chosen CNN architecture effectively extracted features from the Braille images, leading to high classification accuracy.

Future Work:

1. **Extending to Additional Languages:** We plan to extend our model to recognize Braille in other languages such as Afaan Oromoo and Af Somali. This will involve similar processes of dataset creation and model training tailored to these languages.
2. **Real-World Testing:** To includes testing with various image qualities and conditions to validate its reliability and effectiveness.

Our project represents a significant step forward in the development of accessible technology for the visually impaired, particularly for those who read and write in Amharic. By continuing to

refine our model and extend its capabilities, we aim to make a meaningful impact on the accessibility and inclusivity of digital tools for diverse linguistic communities

10. Citations

(Teshome Alemu, 2009) (PDF) RECOGNITION OF AMHARIC BRAILLE (researchgate.net)

(Jha and Advani, 2020) (PDF) Digital Image Braille Character Recognition, Extraction& Translation System [DIBCRETS]TACT-EYE | THE IJES Editor - Academia.edu

(Chekol, 2010) Recognition of Amharic Braille Documents (aau.edu.et)

(Latif et al., 2023) ASI | Free Full-Text | Learning at Your Fingertips: An Innovative IoT-Based AI-Powered Braille Learning System (mdpi.com)

(Park et al., 2023) Visual and tactile perception techniques for braille recognition | Micro and Nano Systems Letters | Full Text (springeropen.com)

(Ovodov, 2020) Optical Braille Recognition Using Object Detection CNN | Papers With Code

(Bengio et al., 2015) Deep learning | Nature

(Ian et al., 2016) Deep Learning (deeplearningbook.org)

(Simonyan and Zisserman, 2015) [1409.1556] Very Deep Convolutional Networks for Large-Scale Image Recognition (arxiv.org)

(Zeiler and Fergus, 2014) Visualizing and Understanding Convolutional Networks | SpringerLink

(He et al., 2016) Deep Residual Learning for Image Recognition | IEEE Conference Publication | IEEE Xplore

(Szegedy et al., 2015) Going deeper with convolutions | IEEE Conference Publication | IEEE Xplore

(Smith, 2017) Cyclical Learning Rates for Training Neural Networks | IEEE Conference Publication | IEEE Xplore

(Samuel et al., 2022) [2210.00275] Offline Handwritten Amharic Character Recognition Using Few-shot Learning (arxiv.org)

(Chollet, 2017) Deep Learning with Python (manning.com)

All the citations for this paper are made using Zotero.