Handwritten Syllable Classification: a Neglected Area of Research

A project report submitted in partial fulfillment of the requirements for B.Tech. Project

Integrated Post Graduate (IPG) Master of Technology

in

Information Technology

by

DEVANSHU PATIDAR (2020IMT-026)

Under the Supervision of

Prof. Aditya Trivedi

विश्वजीवनामृतं ज्ञानम्

ABV-INDIAN INSTITUTE OF INFORMATION TECHNOLOGY AND MANAGEMENT GWALIOR-474 015

2023

CANDIDATES DECLARATION

I hereby certify that the work, which is being presented in the report/thesis, entitled **Handwritten Syllable Classifcation: A Neglected Area of Research**, in fulfillment of the requirement for the **Bachelor's of Technology Project - 2023** and submitted to the institution is an authentic record of my work carried out during the period May 2023 to August 2023 under the supervision of **Prof. Aditya Trivedi**. I also cited the reference about the text(s)/figure(s)/table(s) from where they have been taken.

Date: Signature of the Candidate

This is to certify that the above statement made by the candidates is correct to the best of my knowledge.

Date: Signature of the Research Supervisor

ABSTRACT

Attaining human-level competence in Handwritten Character Recognition serves as the fundamental stepping stone to achieving Handwritten Text Recognition. While strides have been made in handwritten Txt Recognition for the English language, comparable progress has been limited for Indian regional languages. This disparity stems from the fact that many Indian languages utilize abugida scripts rather than alphabetic ones. Within abugida scripts, characters represent combinations of consonants and vowels, with diacritics or modifications indicating different vowel sounds.These are known as Syllables. The intricacy of this script often yields visually similar classes for classification, Which should pose a unique challenge in their classification. To the best of my knowledge , there are zero to no datasets related to handwritten syllables available publicly. There has been very limited research work done on handwritten syllable classification if any.

Addressing this gap necessitates an initial focus on syllable classification, laying the groundwork for attaining human-level competence in Handwritten Character Recognition within regional languages. This journey begins with the development of a dataset for syllable classification, a task to which I've tried to work on . Employing data augmentation techniques, I've effectively expanded the dataset's size. Moreover, to support my argument that syllable character classes exhibit visual similarities, I've employed t-SNE and UMAP visualizations comparing my Syllable dataset to the Tensorflow mnist dataset.

Continuing this exploration, I conducted transfer learning using pre-trained CNN layers as a fixed feature extractor, using three models :Xception, InceptionV3, and EfficientNetV2 recognized for their effectiveness in the base paper[1].Xception Model performs the best among the three in terms of testing accuracy 92.65 percent.Results show there is more room for improvement for Handwritten Syllable Classifcation. Notably, the global linguistic landscape highlights that our country predominantly employs abugida scripts. While other nations may lack incentives to delve into this area, we have the unique opportunity to lead the charge.

Keywords: Handwritten Character Recognition, Abugida Scripts, Handwritten Syllable Classification, Dataset Creation, data augmentation, t-SNE, UMAP, transfer learning using pre-Trained CNN layers as a fixed feature extractor

ACKNOWLEDGEMENTS

I am grateful to **Prof. Aditya Trivedi** for allowing me to function independently and explore with ideas. I would like to take this opportunity to express my heartfelt gratitude to him not only for his academic guidance but also for his personal interest in my project and constant support as well as confidence-boosting and motivating sessions that proved extremely beneficial and were instrumental in instilling self-assurance and trust in me. The current work has been nurtured and blossomed mostly as a result of his valuable direction, astute judgment, recommendations, constructive criticism and an eye for perfection. Only because of his tremendous enthusiasm and helpful attitude has the current effort progressed to this point.

Finally, I am grateful to the Institution and colleagues whose constant encouragement served to renew my spirit, refocus my attention and energy and helped me in carrying out this work

DEVANSHU PATIDAR
2020IMT-026

TABLE OF CONTENTS

ABSTRACT **ii**

LIST OF TABLES **v**

LIST OF FIGURES **vi**

1 INTRODUCTION **1**

1.1 Writing systems and Scripts of the world 1
1.2 Challenges in Handwritten Character Recognition in Abugida Scripts . 3
1.3 Motivation 4
1.4 Objectives 4
1.5 Report Layout 4

2 LITERATURE SURVEY **6**

2.1 Related Works 6
2.2 Looking into existing Dataset 8
2.3 Research Gaps 10
2.4 Looking into Dimensional Reduction Techniques 10
2.4.1 t-SNE 10
2.4.2 UMAP 11

3 METHODOLOGY **13**

3.1 Developing the Android app used to create the dataset 13
3.1.1 Major functionalities implemented 14
3.2 Creating Dataset 15
3.2.1 Data Organization 15
3.2.2 Data Preprocessing 16
3.2.3 Data Labelling 16
3.2.4 Data Augmentation 16
3.3 Transfer Learning 17

4 EXPERIMENT AND RESULTS 20
4.1 Performance Evaluation Metrics 20
4.2 Experimental Setup 21
4.3 Results 21
4.3.1 Results from EDA 21
4.3.2 Results from Experiment 23

5 CONCLUSION AND FUTURE SCOPE 26
5.1 Conclusion 26
5.2 Future Scope 27

REFERENCES 27

LIST OF TABLES

4.1 Training Results 24

LIST OF FIGURES

1.1 A world map depicting regions based on writing systems[2] 2
1.2 Variety of characters in Hindi, an abugida script[3] 2

2.1 An Example Demonstrating t-SNE[4] 11

3.1 Code Snippet showing UI setup 14
3.2 Clear Button Implementation 14
3.3 Save Button Implementation 15
3.4 The app interface 16
3.5 Image Displaying the random stroke width function 17
3.6 The different syllable classes for a consonant 18
3.7 Approach[1] 19

4.1 t-SNE plot for the Syllable Classes 22
4.2 t-SNE plot for the Mnist Digit Classes 22
4.3 UMAP plot for the Syllable Classes 23
4.4 UMAP plot for the Mnist Digit Classes 23
4.5 Accuracy vs. Epoch Graph for Xception Model 24
4.6 Loss vs. Epoch Graph for Xception Model 25
4.7 Confusion Matrix for Xception Model 25

ABBREVIATIONS

app	Application
CNN	Convolutional Neural Networks
CRF	Computer Readable Format
EDA	Exploratory Data Analysis
Mnist	Modified National Institute of Standards and Technology
OCR	Optical Character Recognition
PCA	Principle Component Analysis
PIL	Python Image Processing
RGB	Red Green Blue
t-SNE	t-distributed Stochastic Neighbor Embedding
UMAP	Uniform Manifold Approximation and Projection

CHAPTER 1

INTRODUCTION

This chapter offers an overview of the subject matter. In section 1.1, We talk about alphabetic scripts and abugida scripts. We talk about the challenges related to handwritten character recognition in abugida scripts in Section 1.2. The motivation for working in this area of research is discussed in section 1.3. The objectives of this project are presented in section 1.4. Section 1.5 presents the report layout.

1.1 Writing systems and Scripts of the world

There are 4 broad writing systems in the world:- Alphabetic Scripts, Abugida Scripts, Abjad Scripts, and Logographic Scripts.(Fig 1.1). Most of the world's languages are predominantly based on Alphabetical script, Whereas the languages in India are predominantly Based on Abugida script.

Alphabetic Script: An alphabetic script is a writing system where individual characters or symbols represent distinct phonemes (speech sounds) of a language. Each character usually corresponds to a single consonant or vowel sound. Alphabetic scripts are known for their ability to directly represent the sounds of a language, making them highly phonemic. Examples of languages that use alphabetic scripts include English (Latin script), Greek, and Cyrillic (used for Russian). In an alphabetic script, there is a one-to-one correspondence between characters and phonemes. For example, in English, the letter "A" represents the sound /æ/ as in "cat," the letter "B" represents the sound /b/ as in "bat," and so on[3].

Abugida Script: An abugida script, also known as a syllabary or segmental script, is a writing system where each character represents a consonant-vowel unit or, in some cases, a consonant-vowel-consonant unit. In Abugida scripts, consonants are typically represented by a base character, and vowels are indicated using diacritics or modifica-

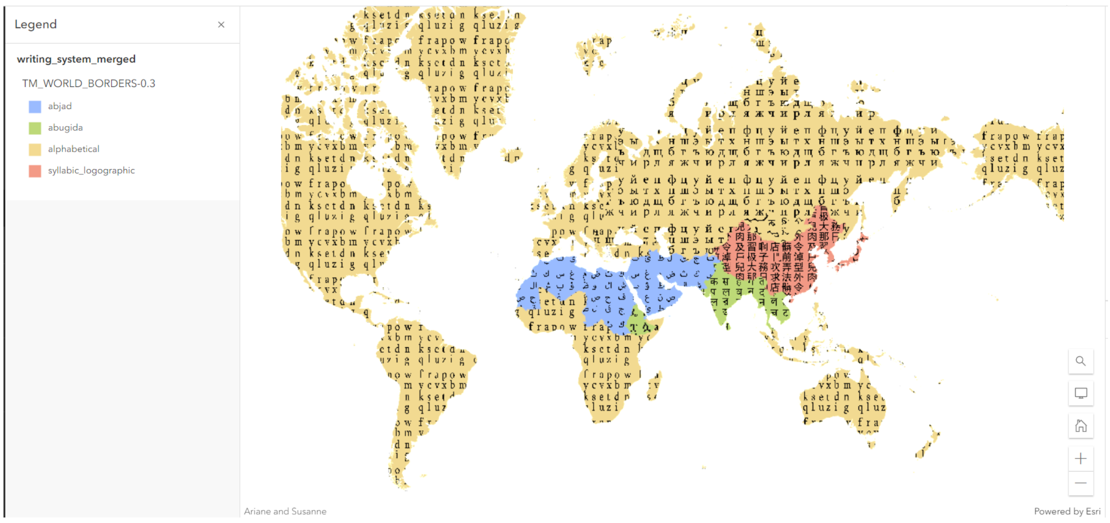


Figure 1.1: A world map depicting regions based on writing systems[2]

tions to the base character. This means that the basic shape of the character represents the consonant sound, and additional marks or modifications indicate the vowel sound associated with that consonant. Abugida scripts are often used in languages where syllables play a significant role. Examples of languages that use abugida scripts include Devanagari (used for Sanskrit, Hindi, and other Indian languages), Ethiopic (used for Amharic), and Brahmic scripts such as Thai and Tibetan[3].

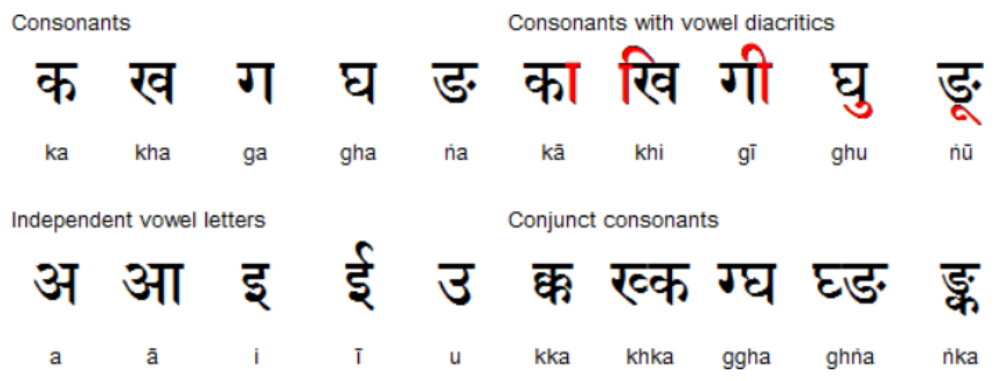


Figure 1.2: Variety of characters in Hindi, an abugida script[3]

Difference between Alphabetic and Abugida Scripts:

Basic Unit: In Alphabetic Scripts, Each character represents a single consonant or vowel sound, whereas, in Abugida scripts, each character represents a consonant-vowel unit, with modifications to indicate vowel sounds.

Representation of Vowels: In Alphabetic Script, Vowels are represented by separate characters. But in Abugida Script, Vowels are often represented through diacritics or modifications to the base character.

Syllabic Structure: Alphabetic Script doesn't inherently convey the structure of syllables. Abugida Script reflects syllabic structure more directly due to the consonant-vowel unit representation.
Examples: Alphabetic Script: Latin Script (used for English), Greek, Cyrillic.
Abugida Script: Devanagari (used for Hindi), Ethiopic, Brahmic scripts (e.g., Thai, Tibetan).

1.2 Challenges in Handwritten Character Recognition in Abugida Scripts

Handwritten syllable recognition in Abugida scripts presents a host of intricate challenges stemming from the unique characteristics of these writing systems. Unlike alphabetic scripts, where each symbol represents a sound (consonant or vowel), abugida scripts encode syllables by combining consonant-vowel pairs or modifications of basic characters. This inherent complexity poses specific difficulties for recognition systems.

Diacritics, modifiers, or ligatures can drastically alter the appearance of characters, making accurate segmentation and classification particularly demanding. These variations, often subtle yet critical, hinder the development of straightforward recognition algorithms that need help to capture these nuances.

The combination of all these leads to an expansive character set that necessitates robust data representation and management. Curating comprehensive and diverse datasets representative of the script's intricacies becomes essential for training recognition models. However, acquiring a balanced dataset with ample samples of each character variation proves challenging due to the potentially limited availability of annotated data.

The variability in writing styles further compounds the challenge. Individual handwriting styles, combined with script-specific modifications, result in a wide range of shape variations for the same syllable. This variation introduces ambiguity into the recognition process and necessitates algorithms capable of accommodating diverse styles and shapes[5].

1.3 Motivation

In today's context, organizations, both government and private sectors, strive for paperless operations. Vital documents such as birth certificates, marriage certificates, land deeds, and property records often employ regional languages. Much of the official work in various states of India is conducted in their respective official languages. Banks, post offices, and government or non-governmental offices regularly handle handwritten paper documents, predominantly in regional languages. The digital age necessitates the transformation of these paper records into CRF.[1]

This further motivates the necessity for intensified research efforts in the domain of handwritten characters and text recognition. Handwritten Syllable Classification is a stepping stone toward achieving these goals. But unfortunately, it has not received the proper research attention it deserves. To the best of my knowledge, there are only a few relevant works on it and there exists no handwritten syllable dataset.

In order to promote more research work in this area, this project provides some of the preliminary work that needs to be done.

1.4 Objectives

The main objectives of the project are:

- To create an Andriod app to generate a Handwrittten Syllable Classification dataset, the first of its kind.
- to perform EDA to show that the classes are visually similar and compare it with the Mnist dataset and perform relevant data augmentation techniques and their combinations to increase dataset size.
- Perform a transfer learning scenario and evaluate the performance of the models (based on base research paper[1]) using evaluation measures like training accuracy, testing accuracy, precision, recall, AUC score and Confusion Matrix.

1.5 Report Layout

In this paper, **Chapter 2** covers the related research concerning the problem. The primary purpose of **Chapter 3** is to describe the Android app created to generate the dataset, describe the data generation process, and the approach for handwritten syllable classification. **Chapter 4** discusses and demonstrates the results of the experiment performed. **Chapter 5** concludes the work done and talks about the future scope.

CHAPTER 2

LITERATURE SURVEY

Section 2.1 covers some of the recent research work done in the field of handwritten character recognition in some of the Indian Languages based on Abugida script. Section 2.2 highlights exploring existing datasets. Section 2.3 presents the research gaps. Section 2.4 looks into the Dimensional Reduction Techniques.

2.1 Related Works

The field of handwritten numeral recognition has experienced significant advancements recently, primarily due to the application of deep learning techniques. Various studies have explored strategies to enhance the accuracy of recognizing handwritten numerals, especially in Indian scripts.

One remarkable study, "Multistage Recognition of Mixed Numerals" [6], focuses on recognizing mixed handwritten numerals from prominent Indian scripts such as Devanagari, Bangla, and English. The authors employ a comprehensive framework involving wavelet-based multiresolution representations and multilayer perceptron classifiers. Their approach employs a multistage cascaded process, achieving near-human level accuracy: 99.04 percent for the Hindi Numeral dataset, 98.2 percent for the Bangla Numeral dataset, and 98.79 percent for the MNIST digits dataset.

Furthermore, the study titled "Handwritten Gujarati Numerals Classification Based on Deep Convolution Neural Networks Using Transfer Learning Scenarios" [1] addresses the challenge of recognizing Gujarati numerals using deep learning. The researchers investigate transfer learning as a solution to data scarcity. They explore three transfer learning scenarios involving various pre-trained CNN architectures. Their experimentation with different models and transfer learning strategies demonstrate the promising performance of the EfficientNetV2S architecture, achieving a testing accuracy of 97.92

percent and other favorable metrics.

In a different vein, "Bangla Handwritten Character Recognition Using Deep Convolutional Autoencoder Neural Network" [7] proposes a novel approach that combines an Autoencoder with a Deep CNN for recognizing Bangla Handwritten Characters. This approach achieves impressive accuracy across various character categories, reaching up to 95.53 percent accuracy for specific classes.

Moreover, the research titled "Identification of Handwritten Gujarati Alphanumeric Script by Integrating Transfer Learning and Convolutional Neural Networks" [8] concentrates on optical character recognition (OCR) for handwritten Gujarati script, a challenging task due to the diversity of patterns and languages in India. Creating a dataset of 75,600 images covering 54 Gujarati character classes addresses the lack of benchmark data. The study utilizes transfer learning with CNN and pre-trained models, resulting in around 97 percent accuracy in recognizing the 54 character classes.

The significance of deep learning techniques in handwritten character recognition is further underscored in "Handwritten Hindi Character Recognition Using Deep Learning Techniques" [9]. The authors introduce a system employing Convolutional Neural Networks and Deep Feed Forward Neural Networks for recognizing handwritten Hindi characters, achieving recognition rates of up to 97.33 percent. This success demonstrates the effectiveness of deep learning in character recognition tasks.

Transfer learning emerges as a valuable technique in this context, enabling models trained in one domain to be adapted for another. Such methodologies prove especially useful when data scarcity is a limiting factor, as exemplified by the Gujarati numerals classification study [1]. Nonetheless, it's crucial to consider the source and target domain nature when applying transfer learning, as highlighted in the literature [10].

In parallel, traditional handwriting recognition systems historically relied on manually designed features and substantial prior knowledge, posing challenges for training OCR systems. Contemporary research has shifted towards deep learning techniques, leading to notable advancements. Yet, the increasing volume of handwritten data and computational power availability demand further improvements in recognition accuracy. Convolutional Neural Networks (CNNs) excel in extracting features from handwritten characters, making them well-suited for handwriting recognition. This work aims to explore CNN design parameters for accurate handwritten digit recognition and evaluate SGD optimization algorithms. Our proposed CNN architecture surpasses ensemble methods in accuracy while reducing operational complexity and cost. We introduce optimal learning parameter combinations, achieving a new MNIST digit recognition record with a recognition accuracy of 99.87percent[6].

In conclusion, these studies collectively underline the potential of deep learning techniques, transfer learning, and cascaded recognition approaches in achieving high accuracy for handwritten numeral recognition tasks across diverse Indian scripts. Despite the advancements, there are still research avenues to explore, including recognizing handwritten syllables and more complex character structures.

2.2 Looking into existing Dataset

Devanagari Handwritten Character Dataset- an image database containing Handwritten Devanagari characters. This database encompasses a diverse collection of 46 distinct character classes, each comprising 2000 individual examples. To ensure a comprehensive evaluation, the Dataset has been meticulously divided into two subsets: a training set constituting 85 percent of the data and a separate testing set accounting for the remaining 15 percent[4].

This Dataset does not contain handwritten syllables.

The "BanglaLekha-Isolated" Dataset is a comprehensive collection of handwritten Bangla isolated character samples. This Dataset encompasses a diverse sample of 50 primary Bangla characters, samples of 10 Bangla numerals, and 24 selected compound characters, which are combinations of basic characters that represent specific sounds or syllables. The data collection process involved gathering 2000 handwriting samples for each of the 84 characters. These samples were digitized and subjected to pre-processing to ensure optimal quality and clarity. After careful scrutiny, mistakes and scribbles were removed, resulting in a final collection of 166,105 handwritten character images[11].

This Dataset contains only 2 handwritten syllable classes for 24 basic characters. This does not represent the Dataset we are looking for. We need a complete set of handwritten syllables for a basic character.

The MatriVasha dataset is a substantial collection of handwritten Bangla compound characters designed for research purposes in handwritten Bangla compound character recognition. Comprising a total of 306,464 images, the Dataset encompasses 120 distinct types of compound characters. The images are derived from male and female handwriting samples, with 152,950 images originating from male writers and 153,514 images from female writers[12].

This Dataset also does not contain handwritten syllables. But it has a good collection of other compound characters.

SHABD Dataset: The SHABD dataset addresses the limitation of existing datasets by including images of basic consonants, vowels, and their combinations. Generated using the TextRecognitionDataGenerator module, the Dataset contains 304,150 grayscale images. Each image is 32 pixels in height and 32 pixels in width, containing pixel values ranging from 0 (black) to 255 (white). The Dataset offers comprehensive coverage of the Hindi language's intricacies and serves as a valuable resource for research in character recognition. The images in the SHABD dataset were generated using varying blur and skew values, along with 35 font styles, resulting in 792 images for each alphabet. Each image in the Dataset is 32 pixels by 32 pixels, resulting in 1024 pixels per image. The pixel values indicate the level of greyness, ranging from black (0) to white (255).[13]

This Dataset has all different syllables. But the Dataset is synthetic. It contains computer Font generated syllables. Still, this Dataset provided a reference to a great existing tool that can be used to generate synthetic datasets. Further Analysis would be needed to do that.

Gujarati OCR / Typed Gujarati Characters - This Dataset comprises images representing the Gujarati Language, utilizing the "Shruti" font style with a font size of 12 and diverse style variations. These images are presented in RGB format. Within the Dataset, there are a total of 34 characters from the Gujarati language, each accompanied by 11 variations of modifiers (Maatra), resulting in 374 distinct character forms. Additionally, there are 11 separate vowels, contributing to a grand total of 385 unique characters. The images within the Dataset have been standardized to a size of 32x32 pixels, ensuring uniformity across the dataset[14].

This is the closest Dataset to what we are trying to accomplish. But again, the Dataset is synthetic. The reason for not including it as a part of the Dataset that I created is the huge difference in image quality and color schemes.

2.3 Research Gaps

Addressing the need for comprehensive datasets containing handwritten syllables for basic characters is a pivotal research gap. As many scripts and languages rely on syllables as the fundamental building blocks, creating datasets encompassing diverse syllable variations can enhance recognition accuracy. Exploring strategies to curate or generate such datasets, ensuring variability in handwriting styles and contextual nuances, holds the potential for training more accurate and versatile recognition models. These datasets could aid in capturing the intricacies of real-world handwriting and improve the performance of recognition systems.

The presence of synthetic datasets in certain languages raises intriguing questions about the impact of synthetic versus real handwritten data on recognition accuracy and generalization. Investigating how well models trained on synthetic data can adapt to real-world handwritten samples is a compelling avenue. This exploration could shed light on the trade-offs between using synthetically generated data, which offers control over dataset size and diversity and utilizing real data, which captures the complexities of genuine handwriting. Understanding how models perform across these domains has implications for developing more robust recognition systems.

In a linguistically diverse landscape like India, where multiple scripts and languages coexist, focusing on specific languages and scripts within research raises the potential for cross-language and cross-script recognition challenges. Exploring techniques that enable models to be trained on one script or language and applied to another could have a significant societal impact. This avenue could contribute to breaking down language barriers in digital applications and facilitate efficient information exchange in multilingual environments. Investigating the nuances of adapting models across scripts and languages can provide insights into the universality and limitations of recognition approaches.

2.4 Looking into Dimensional Reduction Techniques

2.4.1 t-SNE

t-SNE is an unsupervised technique used for visualizing high-dimensional data by reducing its dimensionality. Unlike linear methods, t-SNE is nonlinear, allowing it to separate data that can’t be linearly separated. It is employed to transform complex datasets into two or three dimensions, making underlying patterns and relationships more comprehensible. The process involves finding similarities between data points in higher and lower dimensions, then optimizing two similarity measures(Fig 2.1).

t-SNE calculates pairwise similarities using a Gaussian kernel in the high-dimensional space. Closer points have higher selection probabilities. It maps high-dimensional points to lower dimensions while maintaining their pairwise similarities(Fig 2.1). By minimizing divergence between probability distributions of both dimensions, it uses gradient descent for optimization.[15]

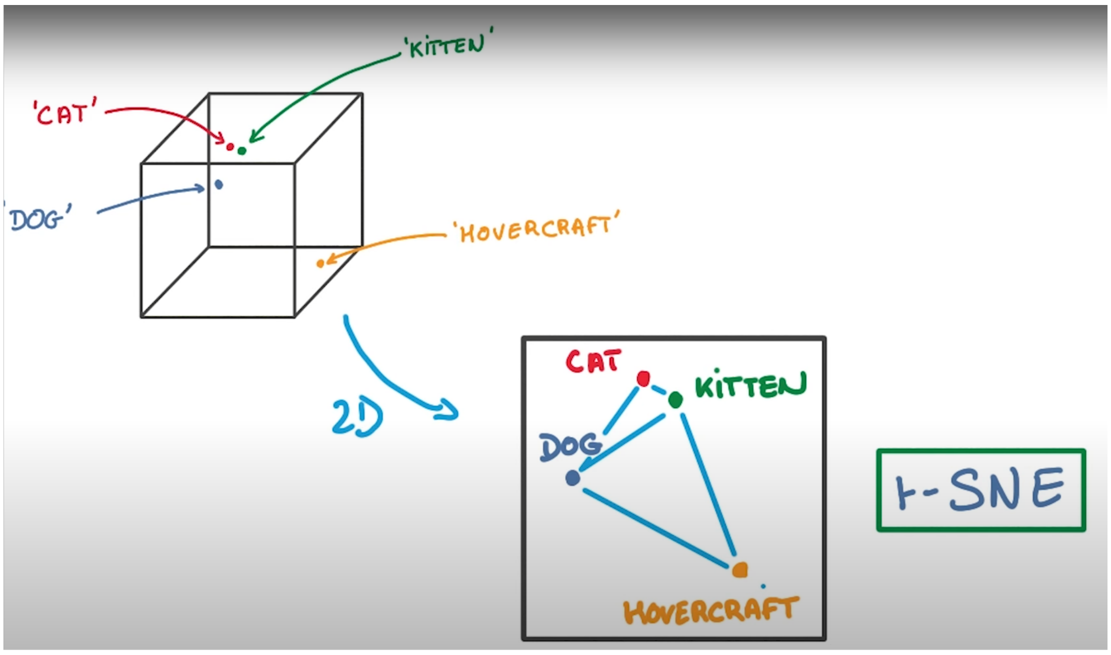


Figure 2.1: An Example Demonstrating t-SNE[4]

2.4.2 UMAP

UMAP is a powerful technique for reducing dimensionality and manifold learning. It's mainly used to visualize high-dimensional data in a lower-dimensional space, particularly for exploratory Analysis. UMAP excels at nonlinear dimensionality reduction, capturing intricate relationships between data points that linear methods like PCA might miss.

UMAP constructs a neighborhood graph by identifying the nearest neighbors among high-dimensional data points. It creates a fuzzy topological representation in the high-dimensional space, capturing data relationships based on distances. UMAP then creates a lower-dimensional representation while preserving local relationships. It minimizes the difference between pairwise distances in both spaces, considering the fuzzy topological weights.

UMAP's strength lies in retaining structural properties while isolating known classes. For data with known classes, supervised UMAP cleanly separates classes while maintaining meaningful point embeddings. UMAP is efficient even with large, high-dimensional datasets, surpassing many t-SNE implementations. UMAP often maintains a global data structure better than typical t-SNE implementations. It provides a comprehensive overview of data alongside local neighbor relationships[16].

CHAPTER 3

METHODOLOGY

This chapter provides an overview of the dataset generation process in Section 3.1. Section 3.2 discusses the EDA techniques used. Section 3.3 talks about the proposed approach used.

3.1 Developing the Android app used to create the dataset

I searched for existing datasets related to handwritten syllable classification on well-known sites. To the best of my knowledge, there are no publicly known datasets catering to our needs.

Initially, my strategy involved seeking an Android drawing board application that could facilitate the creation of my dataset. I aimed to identify an app equipped with features like adjustable stroke sizes, simplified fast Image saving to a designated folder, and effortless clearing of the drawing board with a single tap. Regrettably, my search on the Play Store yielded no applications with all these desired functionalities.

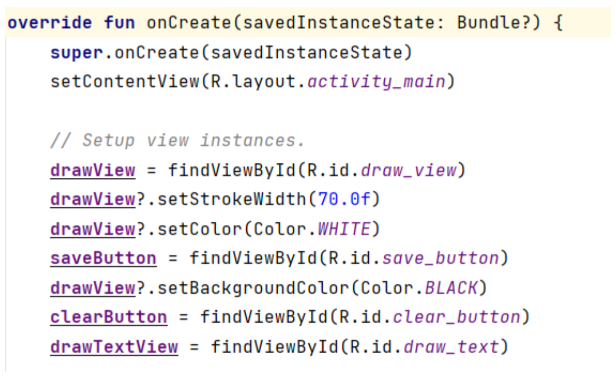
This led me to adopt an alternative approach: exploring existing apps/libraries that could be tailored to meet my specific requirements. During this pursuit, I stumbled upon a GitHub project called "AndroidDraw," which had a good drawing board to import. While this project served as a foundation, I undertook the task of enhancing it to align with my needs.

I implemented some additional functions to the drawing board application. I added buttons for both saving and clearing the canvas. These additions streamlined the image-saving process and simplified the process of clearing the drawing board for subsequent creations. Moreover, I introduced functionality that enabled the stroke size to be set randomly, injecting an element of diversity into the dataset creation process. I used the

DrawView library's inbuilt functionalities to achieve this.

3.1.1 Major functionalities implemented

UI Elements Setup: Within the onCreate method, the code initializes various UI elements such as drawView, clearButton, saveButton, and drawTextView by finding them using their respective resource IDs. Additionally, the stroke width and color of the drawing canvas (drawView) are set, and the background color of the canvas is set to black.(Fig 3.1)

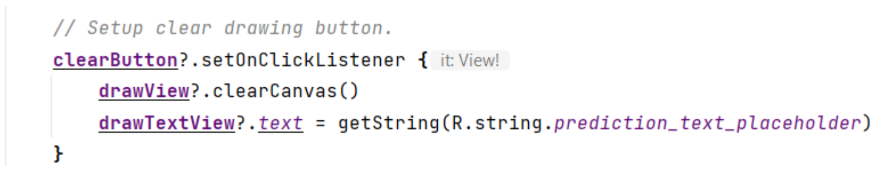


```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)

    // Setup view instances.
    drawView = findViewById(R.id.draw_view)
    drawView?.setStrokeWidth(70.0f)
    drawView?.setColor(Color.WHITE)
    saveButton = findViewById(R.id.save_button)
    drawView?.setBackgroundColor(Color.BLACK)
    clearButton = findViewById(R.id.clear_button)
    drawTextView = findViewById(R.id.draw_text)
```

Figure 3.1: Code Snippet showing UI setup

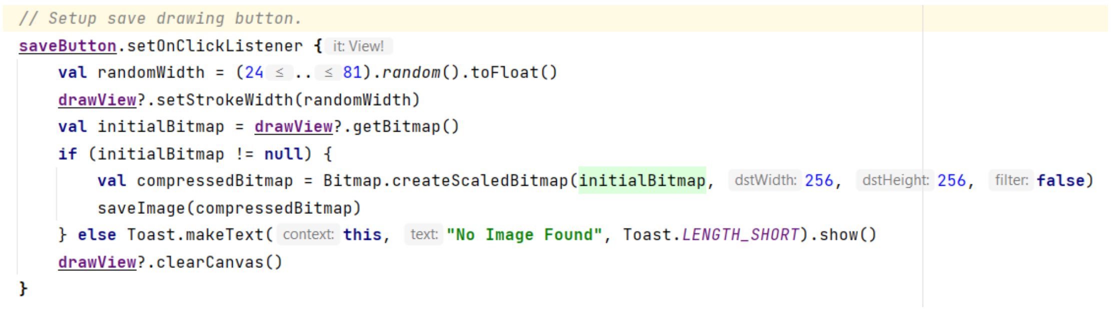
Clear Drawing Button: The code sets up a click listener for the clearButton. When the button is clicked, the clearCanvas() method of the drawView is called to clear the canvas, and the text of the drawTextView is set to a placeholder indicating a prediction text.(Fig 3.2)



```
// Setup clear drawing button.
clearButton?.setOnClickListener { it: View!
    drawView?.clearCanvas()
    drawTextView?.text = getString(R.string.prediction_text_placeholder)
}
```

Figure 3.2: Clear Button Implementation

Save Drawing Button: The code sets up a click listener for the saveButton. When the button is clicked, a random stroke width is generated for the drawing, and the getBitmap() method of the drawView is called to obtain the drawn Image. The Image is then compressed using Bitmap.createScaledBitmap() to a 256x256 size. The saveImage() function is called to save the compressed Bitmap as a JPEG image.(Fig 3.3
Drawing Interactions: The code uses the setOnTouchListener method to handle touch



```
// Setup save drawing button.
saveButton.setOnClickListener { it: View!
    val randomWidth = (24 ≤ .. ≤ 81).random().toFloat()
    drawView?.setStrokeWidth(randomWidth)
    val initialBitmap = drawView?.getBitmap()
    if (initialBitmap != null) {
        val compressedBitmap = Bitmap.createScaledBitmap(initialBitmap, dstWidth: 256, dstHeight: 256, filter: false)
        saveImage(compressedBitmap)
    } else Toast.makeText( context: this, text: "No Image Found", Toast.LENGTH_SHORT).show()
    drawView?.clearCanvas()
}
```

Figure 3.3: Save Button Implementation

interactions on the drawView. When a touch event occurs, it is passed through to the drawView to allow the drawing to show up.

saveImage Function: The saveImage() function is responsible for saving the provided bitmap image to the device's external storage. It creates a directory for saved images, generates a unique filename, compresses and saves the Image as a JPEG file. It also uses MediaScannerConnection to inform the media scanner about the new file for immediate availability.

Fig3.4 and Fig3.5 show the app interface and RandomStrokeWidth Set Feature in action.

3.2 Creating Dataset

The dataset was created manually using the Android application created mentioned in the previous section. To ensure uniformity across the dataset, I ensured that the size of the drawings remained nearly the same for each class. To add noise to the images, Less thought work was given while creating the images. There are 13 such classes(Fig3.6).

3.2.1 Data Organization

In terms of organization, I divided the dataset into three primary folders: "training", "validation and "testing". Within these folders, I further categorized the images into 13 subdirectories, each corresponding to a specific class. The division of the dataset into training and validation sets was carried out using a Python script.

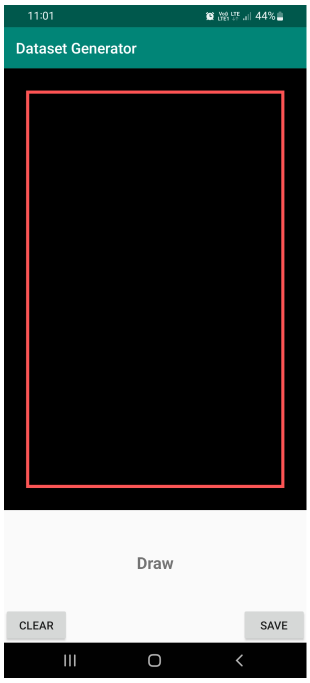


Figure 3.4: The app interface

3.2.2 Data Preprocessing

Before applying data augmentation techniques, the dataset required preprocessing. I added external padding of 40 pixels to all sides. This would safeguard important features during subsequent augmentation processes. This padding step was achieved using a Python script.

3.2.3 Data Labelling

For precise labeling, all images are appropriately categorized based on their respective classes. To maintain a balanced dataset, I steered clear of extreme deviations within classes. This approach aimed to prevent any imbalances from arising.

3.2.4 Data Augmentation

Subsequently, data augmentations like rotation, horizontal, vertical shifts,zoom-in,zoom-out, and shearing were employed.The Data augmentation parameter values were chosen

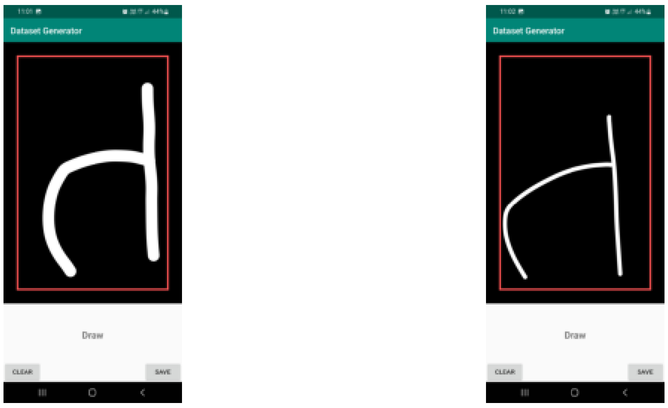


Figure 3.5: Image Displaying the random stroke width function

so as to avoid losing important features from the images. To maintain the integrity of the original class representation, I refrained from horizontal and vertical flipping, as they could excessively alter the images. Given that my dataset consists of grayscale images, channel shift was deemed redundant. Additionally, I avoided applying brightness changes, as they could introduce unnecessary noise, thereby degrading the quality of the dataset.

3.3 Transfer Learning

Transfer learning is a technique used to apply existing machine learning or deep learning models to new domains without starting from scratch. This saves time and enhances performance. When the source and target domains are similar, like both dealing with images, it's called homogeneous transfer learning. In our case, we're using a method based on Weiss et al.[17]. We're using inductive transfer learning as our source task (object classification in images) and target task (Gujarati handwritten digit classification) differ, following Pan and Yang [4].

Our Convolutional Neural Network (CNN) architecture consists of convolutional and max-pooling layers, ending with fully-connected layers. This setup learns complex patterns from data. The process involves transferring knowledge from pre-trained CNN models on the ImageNet dataset (source domain) to our model for Gujarati Handwritten syllables (target domain). ImageNet has 1000 categories, while our dataset has 13. The initial CNN layers learn low-level features like edges, and deeper layers capture intricate details. Transfer learning uses parameters from a well-trained source model.

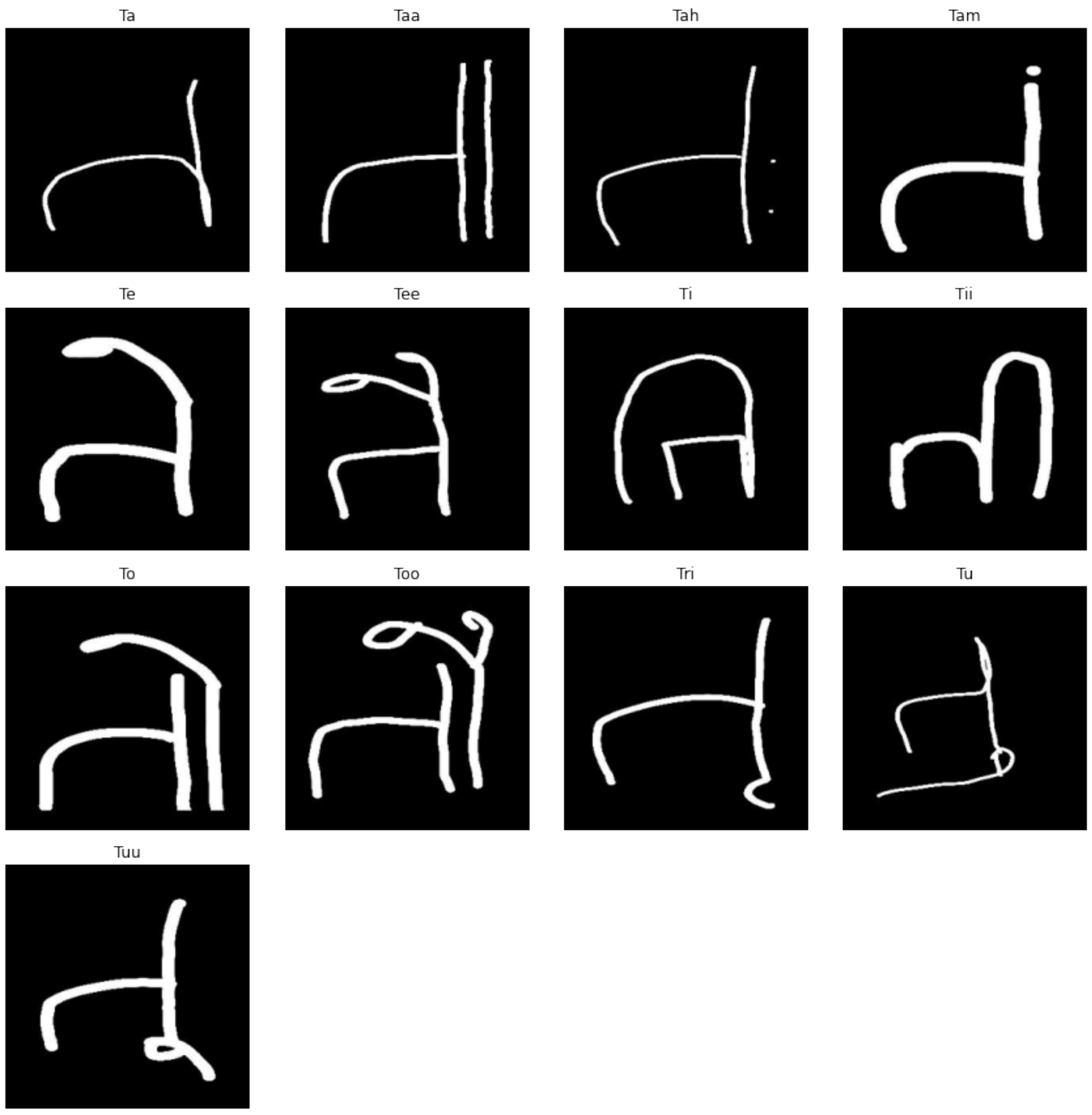


Figure 3.6: The different syllable classes for a consonant

If domains are related, low-level CNN features can be reused, and high-level ones can be fine-tuned or frozen using new layers using transfer learning strategies by adding new layers at the end of CNN.

CNNs consist of multiple layers, such as convolutional, max-pooling, and fully connected layers, each playing a role in learning complex data representations. Among these, trainable weight parameters exist in convolutional and fully connected layers. Convolutional layers extract foundational features, while fully connected layers translate these into object-specific attributes. Lower CNN layers grasp fundamental patterns, while higher ones capture object-specific details. In practice, pre-trained CNNs' convolutional layers act as feature extractors, kept unaltered during training. Original fully connected layers are removed, and replaced by two new ones (512 and 256 neurons). The final layer, with 13 neurons (matching Gujarati syllable classes), serves as the output. The ensuing two fully connected layers and output layer, equipped with a softmax classifier, are trained with the Gujarati handwritten syllable dataset. This process is repeated for three distinct pre-trained CNNs. I have purposefully avoided describing these model architectures because we are entirely freezing the models' layers and the choice of models was mostly based on the fact that these models performed really well in the base research paper for this transfer learning scenario[1].The core novel area of work this paper focuses on was creation of a new novel dataset that previously didnot existed. EfficientNetv2s model is designed to achieve high performance with relatively fewer parameters, making it efficient in terms of memory and computation. Xception model is particularly well-suited for image-related tasks like image classification, object detection, and image segmentation. InceptionV3 is a deep convolutional neural network architecture that was developed by Google's research team. It's a successor to the original Inception (GoogLeNet) architecture and is designed specifically for tasks like image classification.

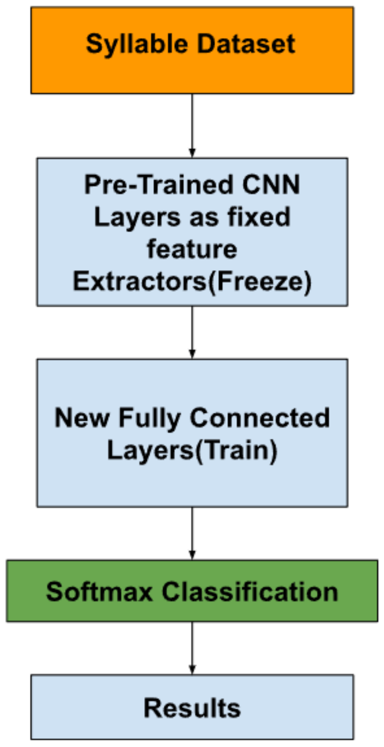

Syllable Dataset
Pre-Trained CNN Layers as fixed feature Extractors(Freeze)
New Fully Connected Layers(Train)
Softmax Classification
Results

Figure 3.7: Approach[1]

CHAPTER 4

EXPERIMENT AND RESULTS

This section discusses various experiments pertaining to the proposed hypothesis. Performance metrics used are discussed in Section 4.1. Section 4.2 discusses the experiment setup. And section 4.3 has the results obtained by the proposed approach using the pre-trained models.

4.1 Performance Evaluation Metrics

The performance assessment of the models involves various evaluation metrics, including training accuracy, testing accuracy, precision, recall, and AUC score. The calculations for accuracy, precision, recall, and AUC are based on equations (4.1) - (4.4). The definitions for key terms are as follows: True Positive (TP) denotes correctly classified syllables, False Positive (FP) signifies misclassified syllables, False Negative (FN) represents syllables incorrectly classified as correct ones, and True Negative (TN) indicates accurately classified negative syllables that belong to different classes.

The AUC (Area Under the Curve) metric measures the degree of separability, indicating the model's ability to differentiate between classes. It is determined by averaging the values of recall and specificity. In equation 4, the first term corresponds to recall, while the second term pertains to specificity. Training accuracy gauges the model's performance on the data used for training. On the other hand, testing accuracy assesses how well the trained model performs on unseen data with a consistent distribution.

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + FN + TN} \tag{4.1}$$

$$\text{Precision (P)} = \frac{TP}{TP + FP} \tag{4.2}$$

$$\text{Recall (R)} = \frac{TP}{TP + FN} \tag{4.3}$$

$$\text{AUC} = \frac{TP/(TP + FN) + TN/(TN + FP)}{2} \tag{4.4}$$

4.2 Experimental Setup

An Nvidia GTX 1650Ti (4GB) GPU, 16 GB of RAM, a 512 GB hard disk, an Intel i7 10th generation CPU running at 2.6GHz, and a 64-bit Windows operating system were the hardware specifications for conducting the experiments. The dataset underwent preprocessing using the PIL library. The proposed pre-trained models were developed using TensorFlow, Keras, and Scikit-Learn.

The dataset was divided into the predetermined train, validation, and test sets, with a split of 70 percent for training(84 images), 15 percent for validation(18 images), and 15 percent for testing(18 images). Prior to training, the images were resized to dimensions of 128×128 and data augmentation was done on training data to increase the number of training samples. During the training process, specific hyperparameters were utilized: a learning rate of 0.0001, 30 epochs, a momentum value of 0.9, and a batch size of 32.

To mitigate overfitting concerns, a dropout rate of 0.3 was applied. Additionally, the ReduceLROnPlateau method coupled with early stopping was implemented. This approach dynamically adjusts the learning rate by decreasing it when the model's progress plateaus and a minimum learning rate of 0.00001 is defined. A patience value of 3 was set, regulating the number of epochs the model could stall before making a decision[1].

For training the models, an Adam optimizer with backpropagation was employed, using categorical cross-entropy loss as the optimization metric. These strategies collectively aimed to enhance the training process and attain effective models.

4.3 Results

4.3.1 Results from EDA

A comparison of EDA results of the mnist dataset and Our custom-made handwritten Syllable dataset is made.

t-SNE plots: In the t-SNE plot for Syllable Classes, the data points are more scattered for the same classes. There is little cluster formation for datapoints of the same classes(Fig 4.1).

In the t-SNE plot for Mnists dataset classes, the data points are less scattered for the same classes. There is more cluster formation for data points of the same classes(Fig 4.2).

UMAP Plots: In the UMAP plot for the Syllable classes, the data points are more scattered for the same classes. There is less cluster formation for data points of the same classes. But the cluster formation is better than its t-SNE plot(Fig 4.3). In the UMAP plot for Mnist dataset classes, the data points are less scattered for the same classes. There is more cluster formation for datapoints of the same classes, and it is even more than its t-SNE plot(Fig 4.4).

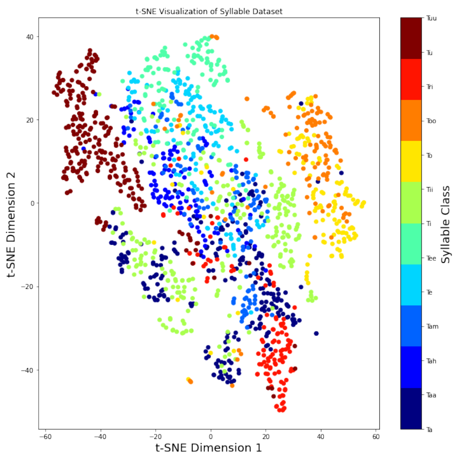


Figure 4.1: t-SNE plot for the Syllable Classes

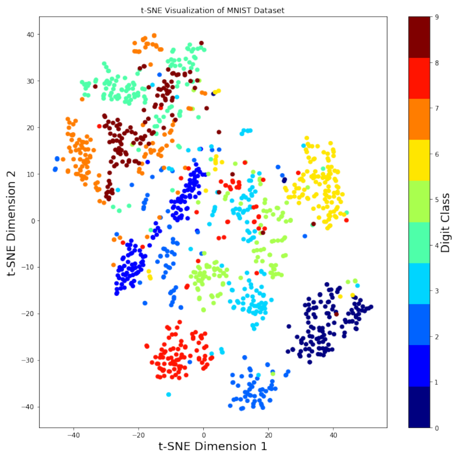


Figure 4.2: t-SNE plot for the Mnist Digit Classes

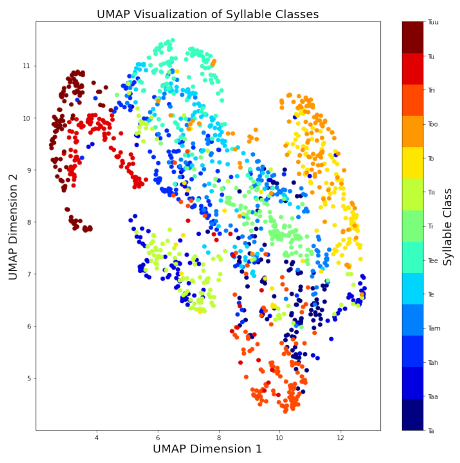


Figure 4.3: UMAP plot for the Syllable Classes

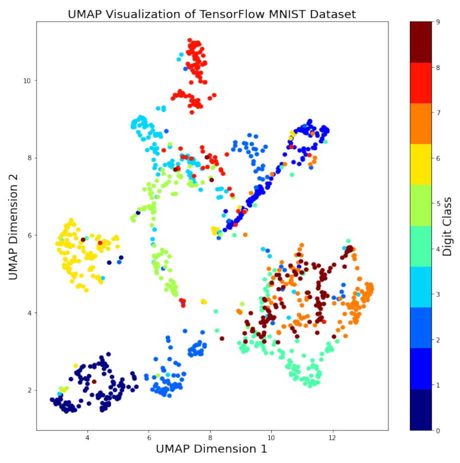


Figure 4.4: UMAP plot for the Mnist Digit Classes

4.3.2 Results from Experiment

We have used three pre-trained models: Xception, InceptionV3, and EfficientNetV2S model. Xception performs better than the other two models in testing accuracy and recall. It achieves Testing accuracy of 92.65 percent, 92.3 percent recall. InceptionV3 performs better than the other two models in precision and AUC score. The complete result is present in Table 4.1.

We have shown the results of Accuracy vs Epoch Graph, Loss vs. Epoch Graph, and Confusion Matrix for the Xception Model. Both Training and Testing Accuracy are increasing with the Number of Epochs, and both Training and Testing loss are decreasing with the number of Epochs(Fig 4.5, Fig4.6). Xception Model stops showing much improvement after 19 epochs. The confusion matrix shows that for the model Xcep-

Table 4.1: Training Results

Model Used	**Train Acc (%)**	**Test Acc(%)**	**Precision(%)**	**Recall(%)**	**AUC(%)**
Xception	96.52	92.65	92.57	92.3	99.71
InceptionV3	94.59	91.02	92.65	91.02	99.76
EfficientNetV2S	90.29	85.47	88.72	87.47	99.30

tion, the true label and the predicted label are the same most of the time for most of the classes.(Fig 4.7

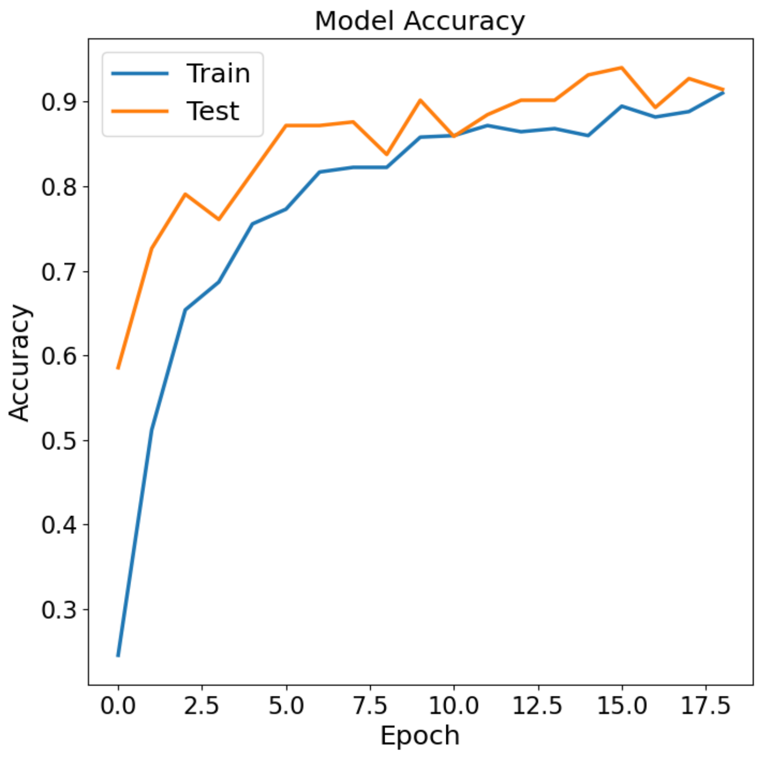


Figure 4.5: Accuracy vs. Epoch Graph for Xception Model

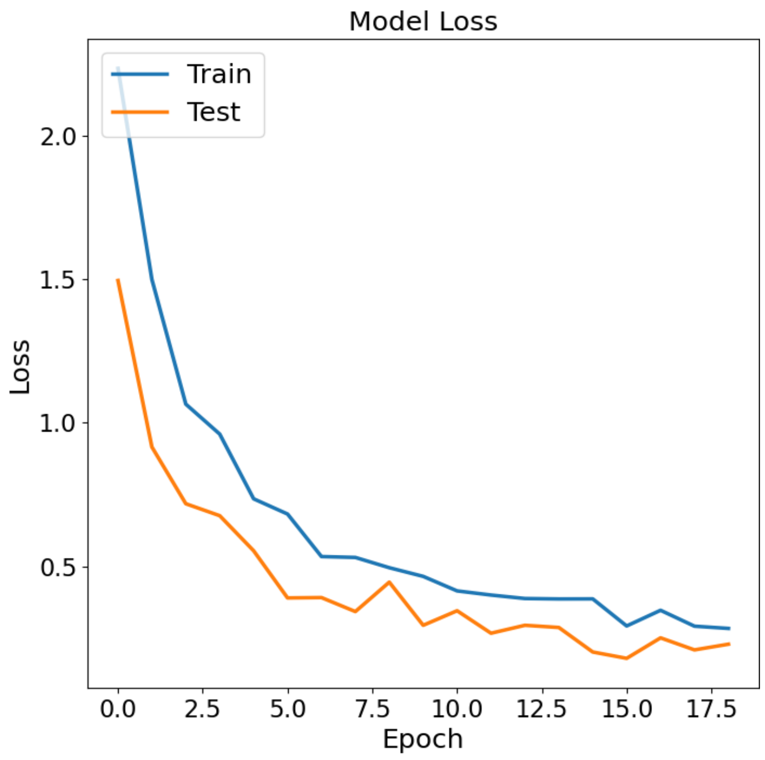
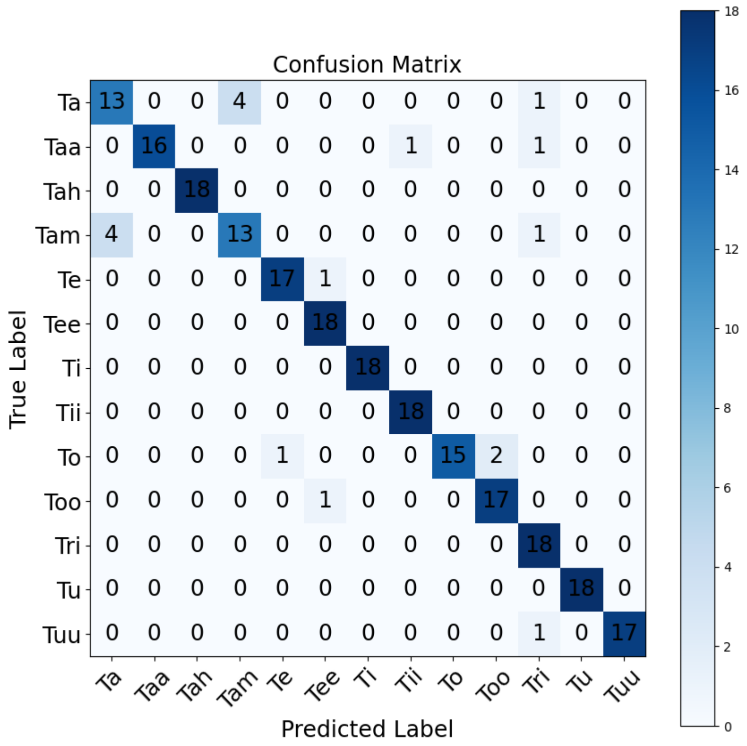


Figure 4.6: Loss vs. Epoch Graph for Xception Model



	Ta	Taa	Tah	Tam	Te	Tee	Ti	Tii	To	Too	Tri	Tu	Tuu
Ta	13	0	0	4	0	0	0	0	0	0	1	0	0
Taa	0	16	0	0	0	0	0	1	0	0	1	0	0
Tah	0	0	18	0	0	0	0	0	0	0	0	0	0
Tam	4	0	0	13	0	0	0	0	0	0	1	0	0
Te	0	0	0	0	17	1	0	0	0	0	0	0	0
Tee	0	0	0	0	0	18	0	0	0	0	0	0	0
Ti	0	0	0	0	0	0	18	0	0	0	0	0	0
Tii	0	0	0	0	0	0	0	18	0	0	0	0	0
To	0	0	0	0	1	0	0	0	15	2	0	0	0
Too	0	0	0	0	0	1	0	0	0	17	0	0	0
Tri	0	0	0	0	0	0	0	0	0	0	18	0	0
Tu	0	0	0	0	0	0	0	0	0	0	0	18	0
Tuu	0	0	0	0	0	0	0	0	0	0	1	0	17

Figure 4.7: Confusion Matrix for Xception Model

CHAPTER 5

CONCLUSION AND FUTURE SCOPE

In this chapter, the work is concluded. In Section 5.1, Conclusions are drawn from the results obtained in the previous chapter. In Section 5.2, Future scope is presented.

5.1 Conclusion

Results from t-SNE and UMAP plots show that classes in the Syllable dataset are more visually similar since they do not form a more well-defined cluster as compared to Mnist Digits dataset.

The high testing accuracy tells us about the model's generalization capacity. Despite the dataset's size constraints, the model's ability to make accurate predictions on new, previously unseen data is a positive sign of its reliability. It suggests that the model has successfully extracted meaningful features and patterns, enabling it to extrapolate its understanding to different instances.

Precision and recall scores complement accuracy by providing insights into the model's predictive power and ability to identify specific classes correctly. High precision signifies that the model's positive predictions are reliable, while high recall reflects its capacity to capture most instances of the true positive class. Moreover, a high AUC score, which measures the model's ability to distinguish between classes, confirms that it can effectively rank instances across different classes. The strong AUC score demonstrates that the model has grasped the underlying characteristics of the data, enabling it to discriminate between positive and negative cases with confidence. The high main diagonal value in the confusion matrix suggests that the model is performing well on our dataset. It suggests the model is correctly classifying instances, maintaining a low rate of false positives and false negatives.

However, it's essential to realize the model's performance may not necessarily generalize well to larger, more diverse datasets. As such, further validation on larger datasets is needed.

5.2 Future Scope

For future work, we should first start introducing more variability in the dataset. This includes manually collecting handwritten Syllable class text from different individuals. This will make the dataset more robust. Synthetic images may be added too.

Other areas of improvement include more thorough experimentation using transfer learning to find out the best hyperparameter values. And if a sufficiently large dataset is available, training from scratch should be considered.

Comparison with other popular models should also be made. More efforts are needed to be spent on creating a good quality complete handwritten character dataset for the regional languages of India.

REFERENCES

[1] P. Goel and A. Ganatra, "Handwritten gujarati numerals classification based on deep convolution neural networks using transfer learning scenarios," *IEEE Access*, vol. 11, pp. 20202–20215, 2023.

[2] A. Geograz, "Image showing writing systems and scripts of the world." https://www.arcgis.com/home/item.html?id=c12011864d984eabbac8d9d0136d1066. Last Accessed on: Aug. 21, 2023.

[3] S. Ager, "Writing systems and scripts by types of writing." https://www.omniglot.com/writing/types.htmalphabets. Last Accessed on: Aug. 21, 2023.

[4] S. Acharya and P. Gyawali, "Devanagari Handwritten Character Dataset." UCI Machine Learning Repository, 2016. DOI: https://doi.org/10.24432/C5XS53.

[5] A. C. N. Matcha, "How to easily do handwriting recognition using machine learning." https://nanonets.com/blog/handwritten-character-recognition/. Last Accessed on: Aug. 21, 2023.

[6] U. Bhattacharya and B. Chaudhuri, "Handwritten numeral databases of indian scripts and multistage recognition of mixed numerals," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 3, pp. 444–457, 2009.

[7] M. A. Azad, H. S. Singha, and M. M. H. Nahid, "Bangla handwritten character recognition using deep convolutional autoencoder neural network," in *2020 2nd International Conference on Advanced Information and Communication Technology (ICAICT)*, pp. 295–300, 2020.

[8] V. K. .R and U. Babu, "Handwritten hindi character recognition using deep learning techniques," *International Journal of Computer Sciences and Engineering*, vol. 7, pp. 1–7, 02 2019.

[9] P. T. D. A. Krishn Verma, Ankit Sharma, "Identification of handwritten gujarati alphanumeric script by integrating transfer learning and convolutional neural networks identification of handwritten gujarati alphanumeric script by integrating

transfer learning and convolutional neural networks," *Behavior Research Methods*, vol. 47, (2022). DOI: https://doi.org/10.1007/s12046-022-01864-9.

[10] S. J. Pan and Q. Yang, "A survey on transfer learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010.

[11] N. Mohammed, S. Momen, A. Abedin, M. Biswas, R. Islam, G. Shom, and M. Shopon, "BanglaLekha-Isolated Dataset." Mendeley Data, V2, 2017. DOI: 10.17632/hf6sf8zrkc.2.

[12] J. Ferdous, S. Karmaker, A. K. M. S. A. Rabby, and S. A. Hossain, "MatriVasha: A multipurpose comprehensive database for bangla handwritten compound characters," in *Emerging Technologies in Data Mining and Information Security*, pp. 813–821, Springer Singapore, 2021.

[13] H. Y. R. J. P. K. Ark Verma, Vivek Sikarwar, "Shabd: A psycholinguistic database for hindi," *Behavior Research Methods*, vol. 54, pp. 830–844, (2022). DOI: https://doi.org/10.3758/s13428-021-01625-2.

[14] A. R. N. D. Mosin I Hasan, Prashant B Swadas, "Gujarati OCR Typed Gujarati Characters Dataset." https://www.kaggle.com/datasets/ananddd/gujarati-ocr-typed-gujarati-characters. Last Accessed on Aug. 21, 2023.

[15] A. A. Awan, "Introduction to t-sne." https://www.datacamp.com/tutorial/introduction-t-sne, Mar 2023. Last Accessed on Aug. 21, 2023.

[16] L. McInnes, J. Healy, N. Saul, and L. Großberger, "Umap: Uniform manifold approximation and projection," *Journal of Open Source Software*, vol. 3, no. 29, p. 861, 2018.

[17] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *J. Big Data*, vol. 3, no. 1, pp. 1–40, 2016.