

Handwritten Text to Speech Converter



MakerNova 2.0

TEAM

<u>Members:</u>	<u>Mentors:</u>
Harshvardhan Gupta	Yagna Rao Neelgiri
Kushal Joshi	Vatsal Shah
Mohit Agrawal	
Diya Pansheriya	
Sweta Rana	
Anmol Kumar Srivastav	
Brinda Soneji	

Table of Contents

1.	Abstract.....	3
2.	Introduction.....	3
3.	Study of Modules	4
4.	Machine Learning	5
5.	Task	7
6.	Research	9
7.	Architecture of Model.....	10
8.	Errors and Solution.....	14
9.	Time-Line.....	15
10.	Conclusion.....	15
11.	References & Learning Resources.....	16

Abstract

This research presents a novel machine learning approach for the accurate recognition and classification of handwritten characters and words. By employing convolutional neural networks (CNNs), a state-of-the-art deep learning architecture, the proposed system demonstrates exceptional performance in extracting meaningful features from handwritten input. Through extensive training on a diverse dataset, the model is capable of handling various handwriting styles and producing highly accurate transcriptions. The potential applications of this technology are far-reaching, encompassing fields such as document automation, medical record digitization, and accessibility solutions for individuals with visual impairments.

Introduction

In today's world, converting handwritten text into digital speech can be challenging. The Handwritten Text-to-Speech Converter simplifies this by automatically recognizing and vocalising handwritten content.

This system employs machine learning and convolutional neural networks to analyse and convert handwritten words from images into digital text. The model is trained on a dataset containing handwritten words. The dataset used is the `iam_handwriting_word_database` which is available on kaggle. The model is trained to accurately interpret entire handwritten words, which are then processed through text-to-speech technology to transform them into speech.

This approach enables seamless conversion of handwritten words into audible content, making handwritten text more accessible to users.

Resources Required:

- Python and its libraries
- VS Code / Google Collab

Study of Modules

1. NumPy:

Purpose: Fundamental library for numerical computations in Python.

Key Features:

Supports multidimensional arrays.

Fast operations on arrays (element-wise and matrix operations).

Provides mathematical functions, linear algebra, and random number generation.

2. Pandas:

Purpose: Data manipulation and analysis library built on top of NumPy.

Key Features:

Provides DataFrame and Series for handling tabular and 1D data.

Tools for data cleaning, merging, reshaping, and aggregation.

Supports time series analysis and handling of missing data.

3. TensorFlow:

Purpose: Machine learning and deep learning.

Key Features:

It is a powerful tool for machine learning and deep learning. It can handle complex numerical computations, automatically calculate gradients for training neural networks, and provides pre-built components for common deep learning architectures. Additionally, Keras offers a user-friendly interface for building and training models.

Usage: Ideal for building and training machine learning models, especially deep neural networks, for tasks like image classification, natural language processing, and time series analysis.

4. Keras:

Purpose: High-level neural networks API.

Key Features:

Simplifies building and training neural networks with a user-friendly API.

Built on top of TensorFlow (or other backends like Theano or CNTK).

Offers pre-built layers, optimizers, and loss functions for quick model development.

Usage: Ideal for rapid prototyping and experimentation with neural network models.

Machine Learning

1. Supervised Learning :

Supervised learning involves training models on labelled data to make predictions.

Classification predicts discrete categories, such as identifying spam emails or classifying images into types. **Regression** predicts continuous values, like estimating house prices or forecasting stock trends. In classification, the output is categorical, while in regression, it's numerical. Both methods require labelled data for training and are used across various applications to solve different types of prediction problems.

Cost Function :

A cost function is a measure of how well a machine learning model performs. It quantifies the error between predicted and expected values and presents that error in the form of a single real number. The cost function is also called the loss function.

Linear Regression:

Linear regression models the relationship between a dependent variable and one or more independent variables using a linear equation. **Simple linear regression** fits a line to data, while **multiple linear regression** fits a plane or hyperplane. It predicts outcomes and analyzes relationships by minimising prediction errors.

Logistic Regression:

Logistic regression is used for binary classification, predicting the probability of an outcome using a logistic function. It transforms linear combinations of input features into probabilities between 0 and 1. It's effective for tasks like spam detection and disease diagnosis, providing both probabilities and class predictions.

Logistic Function :

The **logistic function** maps any real-valued input to a probability between 0 and 1. It produces an S-shaped curve, making it ideal for binary classification tasks by transforming inputs into probabilities, which can then be used to classify data into two categories.

Overfitting:

It occurs when a model is trained too well on the training data. Consequently, while it performs excellent on the training data, it struggles to generalise to new, unseen data, resulting in poor performance on the test data or in real-world applications.

Underfitting:

It fails even to fit the training data well and as a result, performs poorly on the training data and the test data. The model might be underfitting because it's too simplistic, or because it hasn't been trained for long enough to learn the underlying patterns in the data.

Gradient-Descent:

Gradient Descent is an optimization algorithm used to minimise a cost function by iteratively adjusting model parameters in the direction that reduces the error. It calculates gradients to determine the direction and size of updates, aiming to find the optimal model parameters.

2. Unsupervised Learning:

Unsupervised learning explores data without predefined labels to find patterns. Clustering groups similar data (e.g., customer segmentation), dimensionality reduction simplifies data for easier analysis (e.g., visualisation), and anomaly detection identifies outliers (e.g., fraud detection). It uncovers hidden structures and relationships in data.

3. Reinforcement Learning:

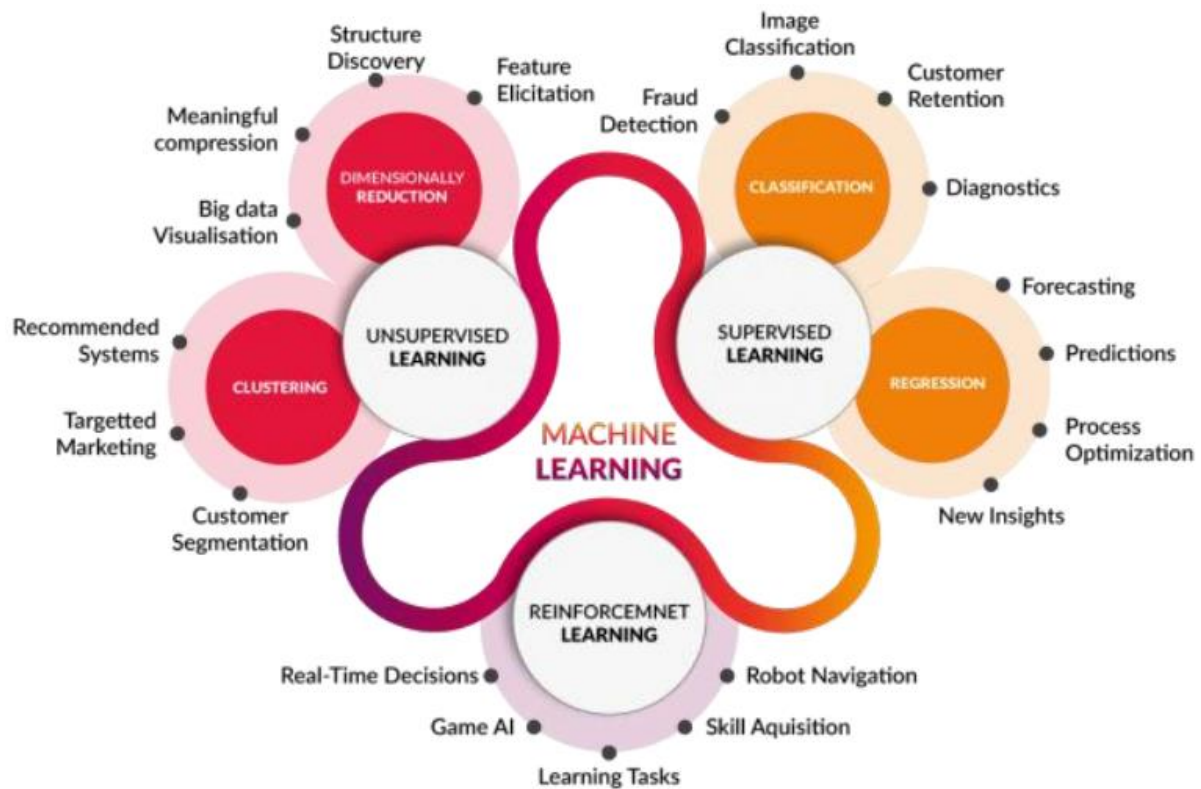
Reinforcement Learning (RL) is the science of decision making. It is about learning the optimal behaviour in an environment to obtain maximum reward. In RL, the data is accumulated from machine learning systems that use a trial-and-error method. Data is not part of the input that we would find in supervised or unsupervised machine learning. Reinforcement learning uses algorithms that learn from outcomes and decide which action to take next. After each action, the algorithm receives feedback that helps it determine whether the choice it made was correct, neutral or incorrect. It is a good technique to use for automated systems that have to make a lot of small decisions without human guidance.

4. Convolutional Neural Networks (CNNs):

Designed for processing grid-like data, such as images.

Key Components: Use convolutional layers to automatically learn spatial hierarchies of features, pooling layers to reduce dimensionality, and fully connected layers for classification or regression.

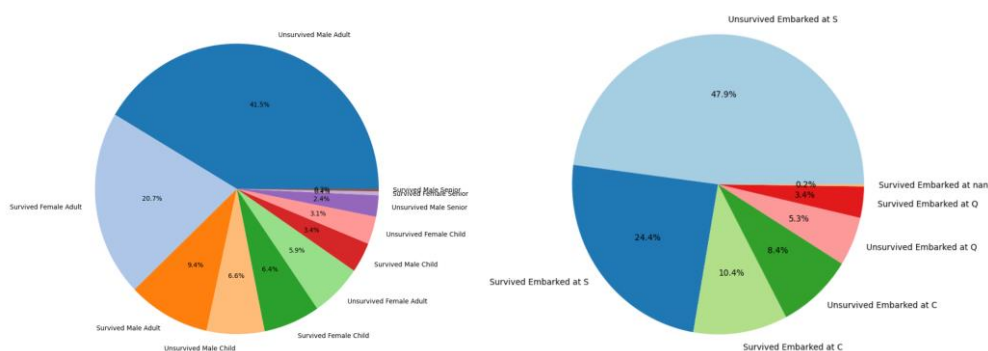
Strengths: Excellent for image recognition and visual tasks due to their ability to capture spatial relationships and patterns.



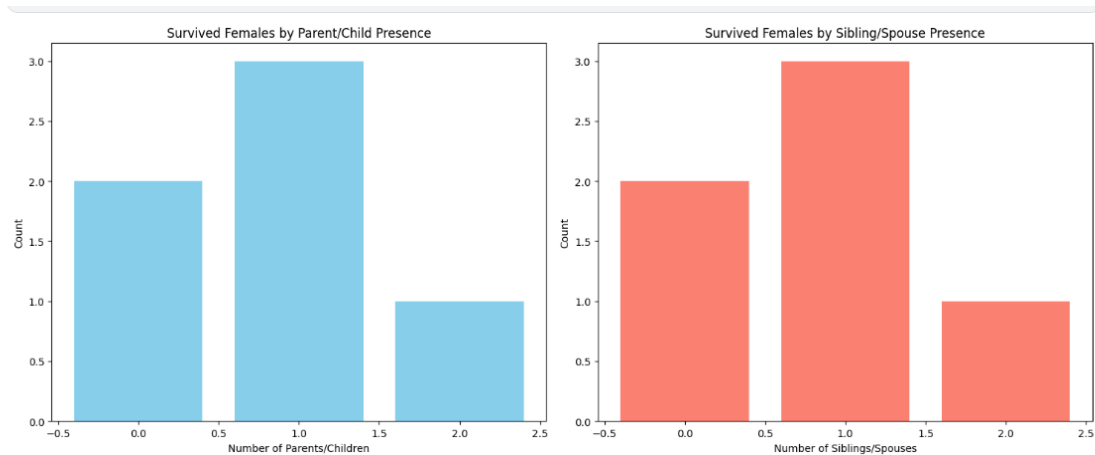
Tasks

During our learning period, we were given the task of implementing the libraries of python such as Numpy, Pandas, Matplotlib on the Titanic dataset to explore various survival patterns. We plotted graphs such as pie charts and bar graphs to analyse survival rates across different categories like gender, age, and passenger class. These visualisations provided valuable insights that helped us understand the dataset's structures.

Graphs:



- Pie chart showing gender and age category wise survival status
- Pie chart showing embarking based survival



c. Bar graph showing Survival of females based on parent/child presence and sibling/spouse presence

Research

This research undertook a comprehensive investigation into various approaches for handwritten character recognition. We had two plans, first was to go letter by letter, and second was to train the model directly on the word. After careful evaluation we finalised to go with the word approach.

To facilitate the development process, we explored existing libraries and frameworks, ultimately selecting Keras, TensorFlow as the core components of our project. These tools provided a robust foundation for constructing and training our machine learning model.

For the training and testing of our model, we utilised the iam_handwriting_word_database dataset from kaggle, a widely recognized benchmark for handwritten character recognition. The dataset's extensive collection of labelled examples proved invaluable in ensuring the model's ability to generalise to diverse handwriting styles.

Following a rigorous research and development phase, we successfully constructed a machine learning model capable of accurately recognizing individual alphabets and numerals.

Architecture of Model

1.Import Necessary Libraries

NumPy is used for efficient array operations and numerical computations. Matplotlib is used for image visualisation. OpenCV handles image processing tasks such as reading, resizing, and normalising images. TensorFlow serves as the core framework for building and training the neural network model. Keras, a high-level API, simplifies model definition, training, and inference.

2. Data Preprocessing and Loading

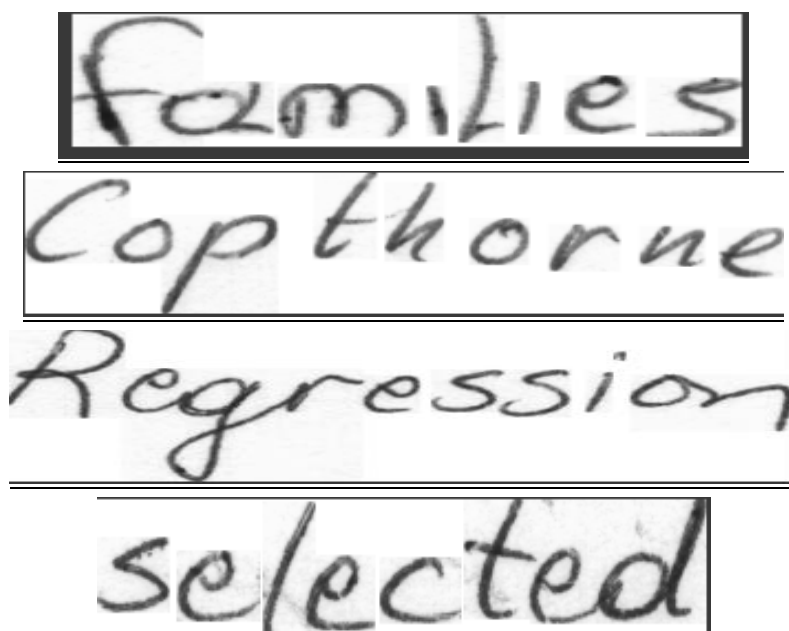
Reshaping: Reshaped the images in the dataset to have the expected shape for the model to size of (32, 128, 1) [height, width, channels].

Normalisation: Normalised the pixel values to a range of 0 to 1 to improve model convergence.

One-hot encoding: Converted the categorical labels into one-hot encoded vectors for easier processing by the model.

Splitting of Data: Divided the dataset in a 9:1 ratio for training and validation.

Padding: We have added padding post the labels in the dataset for consistent size of the encoded labels



Dataset Example

Model Architecture

Convolutional layers: Extracted features from the image data using convolutional layers which apply filters to the input and learn to detect patterns. We have used 5 Conv2D layers in the model.

Pooling layers: The images in the data are very noisy, to counter that we max pooled (MAXPooling2D) the feature maps to reduce computational cost and improve invariance to small translations.

Bidirectional LSTM: The 3 Bidirectional LSTM layers in this OCR model work together to improve text recognition. They read the text both forwards and backwards, helping the model understand the context of each character. This approach allows the model to learn complex patterns in text, including how letters relate to each other over longer distances. The layers also use a technique called dropout to prevent the model from relying too heavily on any single feature, which helps it perform better on new, unseen text. By processing text in both directions and learning at different levels, these layers significantly boost the model's ability to accurately recognize a wide variety of written words and phrases.

Dropout layers: Introduced randomness to prevent overfitting by randomly dropping neurons during training of these layers.

Dense layers: The final Dense layer in an OCR model translates the learned features into character probabilities. It uses softmax to ensure these probabilities sum to 1. By processing the entire image, this layer helps the model predict the complete text, effectively converting the image into readable words.

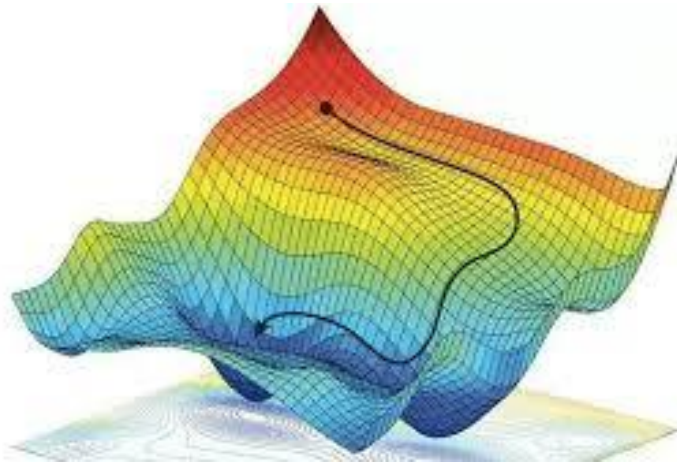
CTC Loss Function: For model compilation and training, a loss function is essential. In our application, we employed the widely used cross-entropy loss function. This function effectively aligns the model's variable output predictions with the true labels provided in the dataset. To address the challenges of variable-length sequences, we incorporated the Connectionist Temporal Classification (CTC) loss function. CTC utilises conditional probability to identify and mitigate inconsistencies between the model's predictions and the ground truth labels.

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	[(None, 32, 128, 1)]	0
conv2d (Conv2D)	(None, 32, 128, 64)	640
max_pooling2d (MaxPooling2D)	(None, 16, 64, 64)	0
conv2d_1 (Conv2D)	(None, 16, 64, 128)	73856
max_pooling2d_1 (MaxPooling2D)	(None, 8, 32, 128)	0
conv2d_2 (Conv2D)	(None, 8, 32, 256)	295168
max_pooling2d_2 (MaxPooling2D)	(None, 4, 32, 256)	0
conv2d_3 (Conv2D)	(None, 4, 32, 512)	1180160
batch_normalization (BatchNormalization)	(None, 4, 32, 512)	2048
conv2d_4 (Conv2D)	(None, 4, 32, 512)	2359808
batch_normalization_1 (BatchNormalization)	(None, 4, 32, 512)	2048
max_pooling2d_3 (MaxPooling2D)	(None, 2, 32, 512)	0
reshape (Reshape)	(None, 64, 512)	0
bidirectional (Bidirectional)	(None, 64, 512)	1574912
bidirectional_1 (Bidirectional)	(None, 64, 512)	1574912
bidirectional_2 (Bidirectional)	(None, 64, 512)	1574912
dense (Dense)	(None, 64, 79)	40527
Total params: 8,678,991		
Trainable params: 8,676,943		
Non-trainable params: 2,048		

Model Summary

Model Compilation

Optimizer: Choose the Adam optimizer, a popular optimization algorithm that combines the best aspects of gradient descent and RMSprop.



Model Training:

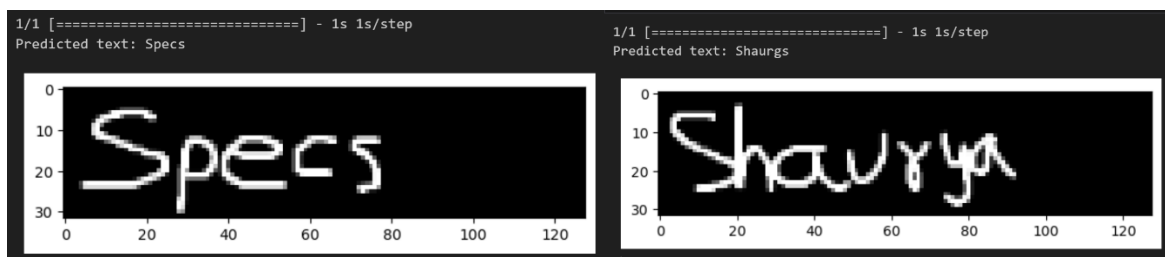
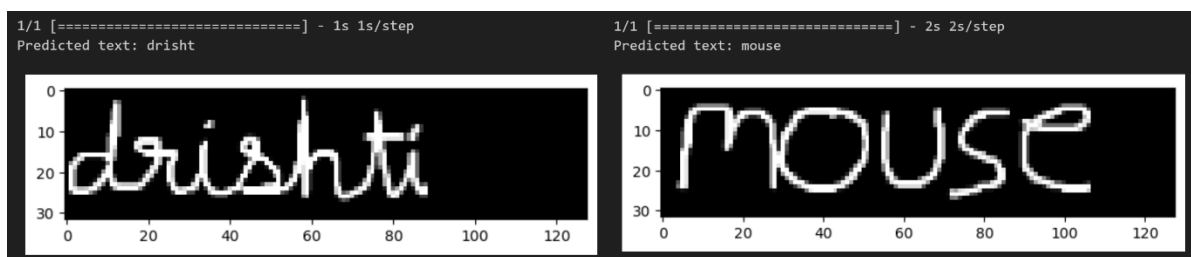
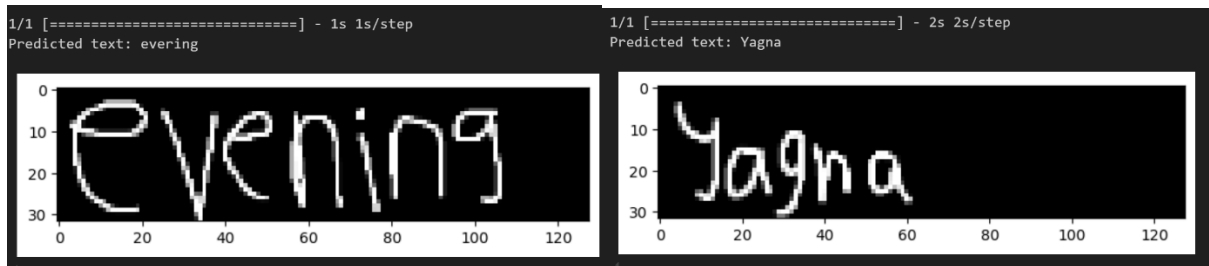
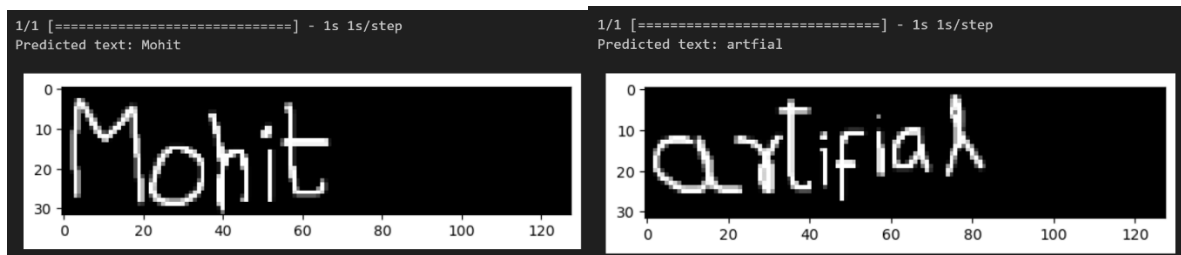
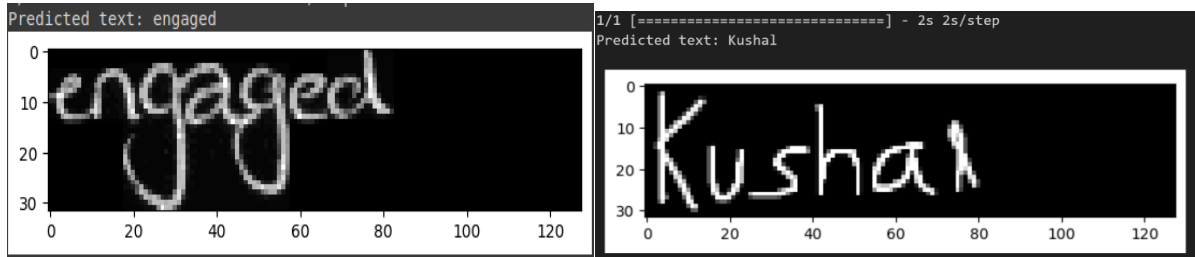
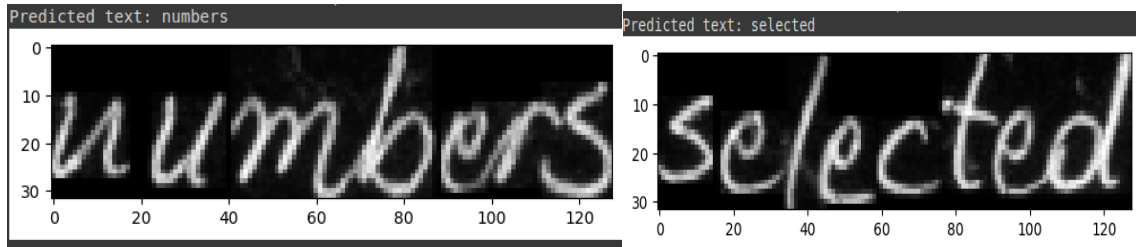
Used the fit() method to train the model on the training set. We have set the number of epochs (iterations over the entire batchsize) to 25 and batch size (number of samples processed at once) to 32, to minimise loss and increase accuracy.

Model Accuracy and Testing:

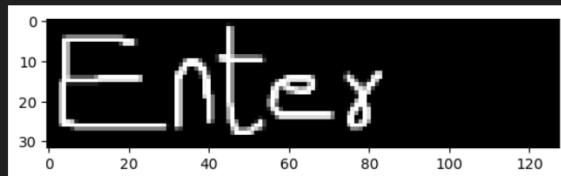
The model has a Character Accuracy of 89.41% and a Word Accuracy of 70.80% on the Validation Set.

```
83/83 [=====] - 2s 21ms/step  
-Character Acc: 89.4124% -Word Acc: 70.8018%  
743/743 [=====] - 95s 128ms/step - loss: 0.6072 - val_loss: 2.1779
```

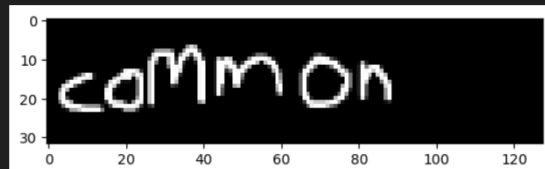
We have tested the model on different custom handwritten images and have got an accurate prediction from the model. We had written our words on the Paint Application and used them for testing. Below are some predictions made by the model.



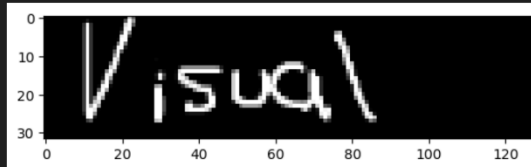
1/1 [=====] - 1s 1s/step
Predicted text: Entes



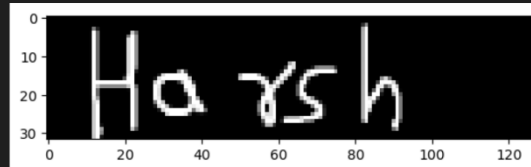
1/1 [=====] - 1s 1s/step
Predicted text: common



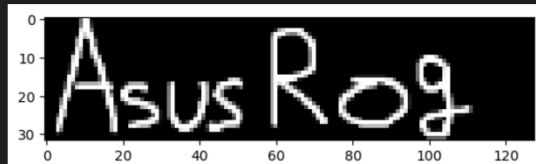
1/1 [=====] - 1s 1s/step
Predicted text: Visual



1/1 [=====] - 1s 1s/step
Predicted text: Harsh



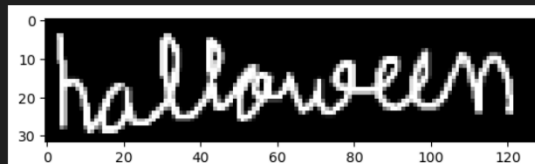
1/1 [=====] - 1s 1s/step
Predicted text: AsusRos



1/1 [=====] - 2s 2s/step
Predicted text: halloween



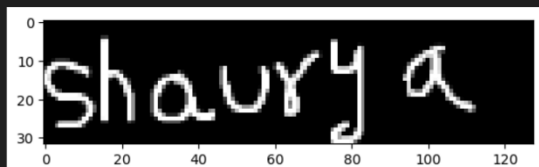
1/1 [=====] - 1s 1s/step
Predicted text: halloween



1/1 [=====] - 1s 1s/step
Predicted text: abedefghi



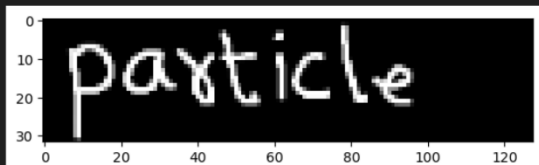
1/1 [=====] - 1s 1s/step
Predicted text: shaurya



1/1 [=====] - 1s 804ms/step
Predicted text: battery



1/1 [=====] - 1s 1s/step
Predicted text: particle



Errors and Solution

1. Model was overfitted:

During initial training and testing, our model achieved a validation accuracy of 71%. However, when presented with novel input data, the model produced inaccurate predictions. To address this overfitting issue, we implemented a dropout layer within the Bidirectional LSTM architecture. The dropout rate was set to 0.3, which effectively mitigated overfitting and significantly improved the model's generalisation performance, resulting in more accurate predictions on unseen data.

2. Padding of labels:

During the model development process, we encountered an error related to inconsistent batch sizes. Upon investigation, it was determined that the labels associated with the images exhibited varying lengths. To address this incompatibility with the model's requirement for uniform input lengths, we implemented padding post-encoding of the label '78.' This padding strategy ensured that all input sequences were aligned, enabling the model to process batches of varying lengths effectively.

3. Model Saving:

Initially, we encountered difficulties in saving the trained model using the .h5 format. Upon further investigation, it was determined that saving the model in the .keras format was the appropriate solution. By adopting this approach, we successfully preserved the trained model and were able to deploy it for image prediction tasks.

Time-line:

- **4 August:** Learning Modules such as NumPy, Pandas and Matplotlib.
- **8 August:** Matplotlib Data Visualisation on Titanic dataset.
- **12 August:** Learning the Machine Learning Basics (supervised, unsupervised and reinforcement learning, cnn)
- **18 August:** Research on similar projects and understanding the model by analysing research papers.
- **22 August:** Dataset Finalization (handwritten text) and Data Cleaning and Preprocessing.
- **26 August:** Implementing the code of the training and testing model and finalisation of architecture.
- **30 August:** Group Presentation
- **8 September:** Changed the dataset so model can predict cursives also, started working on improving accuracy
- **12 September:**Finalisation of model
- **13 September:**Started working on web interface

Conclusion

This project successfully developed an OCR model capable of converting handwritten text into speech. The model utilized a combination of Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) to extract relevant features from the handwritten text images. The Connectionist Temporal Classification (CTC) loss function was employed to address the challenges of variable-length sequences and improve the model's accuracy.

By leveraging the Keras and TensorFlow frameworks, the model was efficiently trained and evaluated on a dataset obtained from Kaggle. The final model demonstrated promising results in converting handwritten text into accurate speech output, highlighting the potential of this technology for various applications, including document transcription, text-to-speech systems, and accessibility tools.

References & Learning Resources

Basic Libraries (Numpy, Pandas, Matplotlib):

<https://youtube.com/playlist?list=PLjVLYmrlmjGfgBKkIFBkMNGG7qyRfo00W&si=DDKBDyroy0H49-pS>

<https://youtube.com/playlist?list=PLjVLYmrlmjGdEE2jFpL71LsVH5QjDP5s4&si=lgX-jGYMt8E2Sh5>

https://youtube.com/playlist?list=PLjVLYmrlmjGcC0B_FP3bkJ-JIPkV5GuZR&si=H1LMF1YeRCMPcBM6

Basics of machine Learning:

https://youtube.com/playlist?list=PLkDaE6sCZn6FNC6YRfRQc_FbeQrF8BwGI&si=zPpugSuaeLt-dCSJ

Data-Set:

<https://www.kaggle.com/datasets/nibinv23/iam-handwriting-word-database>

