

LANGUAGE RECOGNITION FROM HANDWRITING BASED ON MACHINE LEARNING & DEEP LEARNING

Submitted in partial fulfillment of the requirements for
the award of
Bachelor of Engineering degree in Computer Science and Engineering

by
Jayesh Pranav (38110214)
Joel Thomas Zachariah (38110217)



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
SCHOOL OF COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Accredited with Grade "A" by NAAC

**JEPPIAAR NAGAR, RAJIV GANDHI SALAI,
CHENNAI - 600 119**

MARCH – 2022



SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)**

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of Jayesh Pranav (38110214), Joel Thomas Zachariah (38110217) who carried out the project entitled "LANGUAGE RECOGNITION FROM HANDWRITING BASED ON MACHINE LEARNING & DEEP LEARNING" under my supervision from June 2021 to December 2021.

Internal Guide

(Name in Capital letters with signature)

Head of the Department

Submitted for Viva voce Examination held on _____

Internal Examiner

External Examiner

DECLARATION

We, Jayesh Pranav and Joel Thomas Zachariah, hereby declare that the Project Report entitled "LANGUAGE RECOGNITION FROM HANDWRITING BASED ON MACHINE LEARNING & DEEP LEARNING" done by us under the guidance of Dr. R. Gomathi is submitted in partial fulfillment of the requirements for the award of Bachelor of Engineering / Technology degree in Computer Science and Engineering.

DATE: 29/03/2021

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATES:

A handwritten signature in blue ink, appearing to read 'Jayesh', enclosed within a light blue rectangular border.A handwritten signature in blue ink, appearing to read 'Joel', enclosed within a light blue rectangular border.

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to **Board of Management of SATHYABAMA** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. Sasikala M.E., Ph.D, Dean**, School of Computing **Dr. L. Lakshmanan M.E., Ph.D. ,** and **Dr.S.Vigneshwari M.E., Ph.D. Heads** of the Department of Computer Science and Engineering for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide Dr. R. Gomathi ma'am for her valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Computer Science and Engineering** who were helpful in many ways for the completion of the project.

ABSTRACT

In order to develop the understandings of the machinery mind, there has been a lot of upfold in the development of machine learning. As humans, we learn how to do a task by learning it, and optimize the tasks by learning from the mistakes in the process. Just like the brain's neurons automatically trigger and quickly perform learnt tasks, machines can comprehend to the situations of strengthening developed neurons. Deep learning is just as interesting as the human brain's concept. Usage of different types of architectures for such neural networks collide for different types of problems, like image and sound classification, object recognition, image segmentation, object detection, etc. Following these layers of different commemorations and accuracy that Artificial Intelligence provides, one can come up with answers to many unresolved issues, problems, and tasks, since machine is now with the capability of approaching the task as if it is of a human's approach, but with the ability to drive through the task with the machine's work function. The fact that Machine Learning and Deep Learning had taken the industry up by storm is the very reason for its unending usages in this field. The very fabrics of precision and balance that a task completed under the machine's supervision is where we can entrust tasks that can bring a change for many lives with challenges, ailments and difficulties. The challenges that humans label as an impairment or a dysfunctional occurrence is where a machine could potentially step as a helping hand, or in our case, a hearing aid.

CONTENTS

	Title	Page No.
	LIST OF FIGURES	(i)
	LIST OF TABLES	(ii)
	LIST OF ABBREVIATIONS	(iii)
1	INTRODUCTION 1.1. IDEA BEHIND THE LANGUAGE RECOGNITION PROCESS 1.2. TIMELINE OF THE ML-DL COURSES AND THE RESOURCE COLLECTION 1.3. THE BASIC STRUCTURE AND RENDITION OF THE UI TO BE MADE FOR USERS	1 1 2 4
2	AIM, SCOPE AND LITERATURE REVIEW OF THE PROJECT 2.1. AIM 2.2. SCOPE 2.3. LITERATURE REVIEW	6 6 6 7
3	METHODOLOGY 3.1. SPECIFYING THE NEEDS OF THE PROJECT ACCORDING TO AIM 3.2. AVAILING THE INITIAL PROJECT REQUIREMENTS 3.3. UNDERSTANDING THE MODEL CORE REQUIREMENTS 3.4. IMPLEMENTATION OF	10 10 11 14

	THE PYTHON MODULES	16
	3.5. IMPLEMENTING THE LEARNING MODEL WITH NNs	17
	3.6. POINTERS BEFORE DEVELOPING THE PROGRAM	19
	3.7. DEVELOPING THE LANGUAGE RECOGNITION PROJECT	20
	3.8. RUNNING THE PROJECT WITH MODEL-WEBSITE INTEGRATION AND MODULATING THE RESULT WITH FEATURE ENABLING	26
4	RESULTS AND DISCUSSION	30
	4.1. RESULTS OBTAINED FROM THE WEBSITE	30
	4.2. BETA REVIEW AND DISCUSSION FROM USERS	32
5	CONCLUSION AND FUTURE WORKS	34
6	REFERENCES	38
7	APPENDICES	40
	7.1. SOURCE CODE	
	7.2. SAMPLE SCREENSHOTS	
	7.3. PUBLICATION PROOF (WITH PLAGIARISM REPORT)	

LIST OF FIGURES

- Fig 2.1. Underlying process of the Language Recognition
- Fig. 3.1. The different Neural Networks present in the project
- Fig. 3.2. – The flowchart of Language Recognition Project
- Fig 3.3. Home Page of project website
- Fig 3.4. – About Us page of the project website
- Fig 3.5. – The Front-End Flowchart of the project
- Fig 3.6. – The Model flowchart of the project
- Fig 3.7. – The Back-End flowchart of the project
- Fig 3.8. – The Visual Studio Code IDE (VSC) containing the project code
- Fig 3.9. – The Visual Studio Code IDE (VSC) terminal stated to the main directory
- Fig 3.10. – Running the flask application to execute the project
- Fig 3.11. – Running flask application with the link to open the project website
- Fig 3.12. – The website with the file input mode to give the manuscript input to the model
- Fig. 4.1. Input for the Model's Prediction (CMD Execution)
- Fig. 4.2. Output of the Model's Prediction (CMD Execution)
- Fig. 4.3. Input for the Model's Prediction (Website Input)
- Fig. 4.4. Output of the Model's Prediction (Website Output)
- Fig 5.1. Cursive Writing that is removed from classification
- Fig 5.2. Graph depicting the Accuracy percentage of the Model's training-testing sequence

- **Table 3.1 – Different Python modules in the project**

- **API – Application Process Interface**
- **PIP – Python Installation of Packages management**
- **CNN – Convolutional Neural Network**
- **RNN – Recurrent Neural Network**
- **CTC – Connectionist Temporal Classification**
- **HTML – Hypertext Markup Language**
- **CSS – Cascading Style Sheets**
- **JS – JavaScript**
- **GTTS – Google Text-To-Speech**

1. INTRODUCTION

1.1. Idea behind the Language Recognition Process:

In recent times, many new inventions have been made in order to help people with a certain set of disabilities/disorders. We see the development of speech-to-neural transmitter (earpiece) for the deaf, VR-enabled course looker (yet to be developed) for the physically disabled, and so on. One of the main problems that the blind people have to tackle in their daily life is to convert any piece of information that is non-Braille in nature. Since everyone isn't very keen in the aspect of sharing their piece of information in the usable method for the disabled, we seek to find a method that disassembles the correct way of bringing about a valid piece of technology to help this part of community.

This search led us to learning and use the knowledge of Deep Learning, ML, and UI/UX, to develop an interface that allows you to bring forth a piece of information (currently in English), either written or typed, to the program as an input, and in return, the machine gives out a voice note of the very information that has been entered, along with numerous help for the set of information provided. Since Machine Learning has its niche in a specter of domains and researches, many approaches have been partially or totally established. Bayesian Network, Clustering, Decision Tree Learning and many more have collectively made Deep Learning only a part of the approaches.

The following review mainly focuses on the collated usage of deep learning, the basics of forming this proposed application in said and different fields. Additionally, it presents several figures portraying the viable usage of this project across many nuances. The paper that helped the most to bring this project to its fullest potential was the “Object Recognition Development for Android Mobile Devices with Text-to-Speech Function Created for Visually Impaired People” developed by Andrei Burta, Roland Szabo, and Aurel Gontean. The paper was issued in accordance of developing a speech application that could scan the given picture’s input and speak out the name/description of that product.

This project was implemented in order to help the old-aged people with identifying some of the essential products while purchasing them in markets. The project basically ran under the guidance of Convolutional NNs and Recurrent NNs. So, we learnt the utmost capabilities of CNN and RNN following the understandings of integrating multi-substantial layers to the model, which was later integrated to a rich and intuitive website UI to be used by people for test.

1.2. Timeline of the ML-DL Courses and the Resources Collection

With the courtesy of the courses – “Machine Learning A-Z™: Hands-On Python & R In Data Science” and “Deep Learning A-Z™: Hands-On Artificial Neural Networks” from Udemy, we had started our journey in learning the core details and workings of Machine Learning

and Deep Learning. It took us about 6 months of screentime to finish both the courses that summed upto 60+ hours of course content. After the completion of the courses, we came with the basic notions of progress.

We must understand of all the various parameters that has been used for building a perfectly working Machine Learning model. The main parameter to analyze and learn is the types of models and NNs to implement. We know that in order to study data that are in the form of images, audios, or videos, we take the primary help of a Sequential Model and Convolutional Neural Network. We learnt in the courses that for every type of ML model, there could be a new module developed, that in turn developed new skills to improve our understanding of this challenging field. To get on with the idea, we did use the help of a few research papers from the IEEE website, with a few of them present in the Literature Review.

The whole concept of bringing forth and developing a model that can decipher the given text into an audio clip can be split into three works – the scanning of the input for words from the manuscript, the identification of the words by the model as a sub-output, and the conversion of the sub-output as the necessary voice output. In the case of scanning the input, we need a neural network layer that can split the data into parts of necessary schema that needs deciphering. For identification, the help of the Sequential Model along with the guidance of a Recurrent NN is required.

For the conversion of the deciphered manuscript as a voice clip, a text-to-speech module is to be implemented that can create an automated voice note of the sub-output clearly. In total, from collecting various variants of dataset, to configuring nominal number of neural networks, it took us about 4-5 months to build the initial sketch of the working model. After that, we parallelly started improving the model's accuracy while we started developing the website for the user interface purpose. By the end on about 7 months since, we had a prototype ready to be executed.

1.3. The Basic Structure and Rendition of The UI Made for Users

The learning model needed an apt User Interface for the users to feel at ease while using the model. When creating a perfect UI for your website, there are many factors to consider. The first is to ensure that the layout is easy to navigate and that all important information is easily accessible with the proper information structure. The design should also be visually appealing to enable the visitors to use the model correctly.

It is also essential to ensure that the website is responsive and works well on all devices in a proper layout. As we knew that a nominal number of users might face visual impairments or meager understanding of the language barrier, we had to make a clear website that used enriched colors and better guidance system for the users to be well-guided in getting their predicted manuscripts right. The website development needed core understanding of HTML, CSS and Flask. Hence, we took the course of "Learn Flask for Beginners" by freeCodeCamp YouTube channel, and "Mastering HTML and CSS" course developed by Howard Hamilton and Sun Developers via

Udemy. The courses with a combined screentime of 25 hours took us about 15-20 days of our learning and concentration. Post-learning the courses, we had started developing the apt intuitive website for the recognition model.

We realized that in our website, we needed the interface of our website to be easy to navigate. If they are not easy to use or navigate, the visitors will be clueless as to where they must access the model from the website. To create an easy and accessible website, focusing on the UI's ease of access was crucial. The UI were to be designed to quickly understand what they need to do to achieve their goal. The UX should be designed so that users can easily navigate through the model of the website.

The website had the following elements that we wished to add in – Home page explaining the websites' purpose, an About Us page introducing ourselves to the users, a Support page for the users to add their valuable feedbacks, and an "IMPLEMENT SCRIPT" button which leads the user to the file upload page. The websites' color scheme was designed to be orange and black since besides the color of red, orange and black are perhaps the more cautious and eye-catching color, and at the same time, soothing and appealing for the user to work with the website. The buttons are kept as a top bar at a rather large size to the right side of the website for the user to access the pages easily. The file upload and camera modes are also kept easily accessible and baselined with the design for the people of older generation to not be startled by the basic usage of a website.

2. AIM, SCOPE AND LITERATURE REVIEW OF THE PROJECT

2.1. Aim

The aim of the project is to develop an end-to-end user interface enabled application for users to avail the process of converting the given handwritten manuscript (input) into an automated voice note.

2.2. Scope

The scope of the project is to provide the people with partial or permanent visionary deficits to be able to understand/learn the information that is given to them in a written format, or even in typed format. This will help the said set of people able to understand the information from the manuscript in an instant without any external resource/support needed.

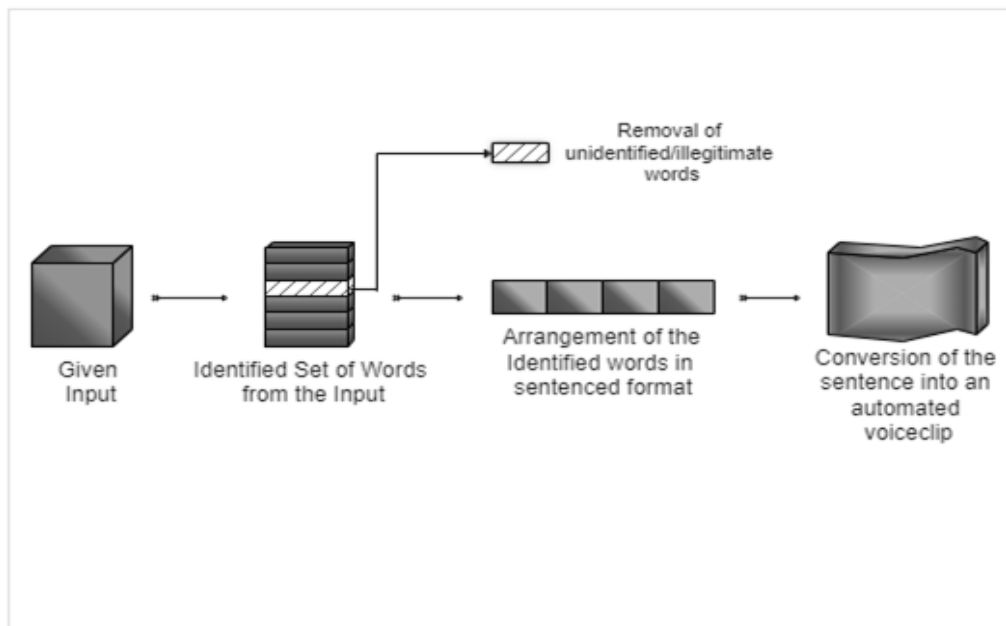


Fig 2.1. Underlying process of the Language Recognition

2.3. Literature Review

- Qinge Xiao; Congbo Li; Ying Tang; Xingzheng Chen (2020) contributed to the IEEE Transactions on Automation Science and Engineering on developing an Energy Efficiency modelling that could configure a variably dependent cohesive model development using Machine Learning.
- Tahir, Ghalib Ahmed; Loo, Chu Kiong (2020) contributed for an open-ended Continual Learning (CL) model that is deemed on Food Recognition by the means of Extreme Learning Machines using the incrementations of class via IEEE Access.
- L. Liu, P. Wang., and J. Lin (2021) contributed for describing the detection of intrusion caused by imbalanced Network Traffic purely dependent on ML and DL via IEEE Access.
- A thesis on enabling an Ensemble Hierarchical EL (Extreme machine learning) for dereverberation of different speech patterns was conducted by Tassadaq Hussain, Sabato Marco Siniscalchi, Hsiao-Lan Sharon Wang, Yu Tsao, Salerno Valerio Mario, and Wen-Hung Liao via IEEE 2020
- Dali Zhu; Long Yang; Zhanxun Li; and Hualin Zeng contributed to IEEE Symposium on Computers and Communications (ISCC Rennes, 2020) with a marvellous project on Remote Speech Extraction from a given Speckle Image using the functions of the modulated CNN (convolutional neural network).

- Lavanya Bhaskar and Ranjith R attended the 2020 11th International Conference on Computing, Communication and Networking Technologies to demonstrate the Robust Text Extraction in Images for Personal Event Planner.
- Fuwei Cui; Qian Cui; and Yongduan Song (2020) made a convoluted survey on Learning-Based approaches for computation and execution of Human-Machine Dialog Systems via IEEE.
- Hao Tang; Hong Liu; Wei Xiao; and Nicu Sebe (2020) developed a thesis to demonstrate the intuitive connections of Machine Learning and Deep Learning's coding NNs for Image Recognition with Limited Data via IEEE.
- Guoqiang Zhong; Kang Zhang; and Hongxu; Yuchen Zheng; and Junyu Dong (2019) discussed on the vast sectors of Marginal Deep Architecture, which further followed up on the learning modules and stacking features to develop Deep Learning Models via IEEE
- Lazhar Khelifi and Max Mignotte (2020) came up with the analysis of different variants of Remote Sensing Images congregated with its Change Detection in reasoning to Deep Learning via IEEE

- Asghar Ali Chandio, M. D. Asikuzzaman and Mark R. Pickering (2020) had developed a Cursive-writing recognition from the inputs of Natural Scenary by the usage multilevel CNNs and Neural Network Fusion via IEEE
- Yi-Chao Wu, Fei Yin, Zhuo Chen, and Cheng-Lin Liu had developed a model on Handwritten Text Recognition that could recognize the handwritten Chinese manuscripts using Separable Multi-Dimensional RNNs via IEEE
- Yucheng Zhou and Zhixian Gao (2019) propelled the idea of executing the Medical Motion Image Recognition with the combined help of Convolutional Neural Network and IoTs (internet of things) via IEEE
- Yulia S. Chernyshova, Alexander V. Sheshkus and Vladimir V. Arlazarov came up with the development of a Two-Step CNN Framework that serves the purpose of Text Line Recognition from the images captured via cameras in different places via IEEE.

3. METHODOLOGY

3.1. Specifying The Needs of The Project According to Aim

So far, we have seen the needs of the project and the final product the algorithm aims for. Typically, any project's life cycle includes four stages:

Initiation - Where initial research is done and scope is identified. In our case, we have scouted our initial scope to be developing an accurate model and an intuitive UI for the model in the form of a website.

Planning - Where you create a plan, determine end results, and finalize the project timeline. The timeline of learning the given modules and languages, and starting to gather the given datasets, followed by developing the model and website takes an achievement time of 4-6 months depending on external and internal factors.

Execution - Where you break down the project into smaller milestones, start the work, collaborate, and reach the end goals of the project. The team of two here had assigned the tasks of developing the initial model and training the model as the first set of tasks for each member, and developing the UI (website) and designing the logo and website thoroughly for the user's ease as the second and final set of individual tasks. The final step of integrating the model to the website is as mentioned, an integrated teamwork.

Closure – The result is evaluated and checked for accessibility and quality. By the end of execution, minor debugging and updates with the model accuracy and website's responsiveness are necessary before deploying the product for the users.

Not just is this project for a sustainable end-product, but also a susceptible end-product for the users as mentioned before. Thus, the project needs the utmost care with every aspect of elements chosen – starting from the layers of neural networks chosen, accuracy of the learning model, accessible UI for the model website with the right elements and a proper integration of the model with the website so that the website doesn't lag or execute a deadlock.

3.2. Availing the Initial Project Requirements

The project mainly required the support of some key programming languages, the necessary package files, trainable datasets, and other expertise in employing the right and efficient APIs and Databases. With that in mind, we integrated the availability of the following tools:

3.2.1. Usage of Python, Flask, and Web Applications

In case of the back-end side of things, the correct flowing of the project requires the following set of installations and implementations – Ver 3.8 of Python, Ver 1.3 of TensorFlow, and any or latest versions of NumPy and OpenCV. These tools are used accordingly in a secure open-source IDE (such as Visual Studio Code in our case) to maintain an ideal and superlative process in the project. Some of the common Python packages are also taken into consideration since the project requires a few sets of APIs for conversions. The Python packages that we used in this project are – Flask, RoughLine, and Transfer. They have the purposes of integrating to the developed website, maintaining a small data sheet for the derived predictions, and congregation of the predictions to the

database that will be integrated in future. (Since some of the high-level databases require financial installments)

In order to develop an intuitive user interface i.e., a website, that collects the data from user and gives the output in form of a voice note, we've employed the services of implementing framework using Bootstrap, and making the interactive UI using HTML & CSS, with the functionality procedures of Js. So far, we've thought of either making the model with just a one-time use and destroy interface that uses the input only once to replicate the output rather than saving the whole process. If the latter is in the need of implementation, we employ either AWS or Firebase for the creation of a nominal database. The website has deemed to contain all the logical and technical information regarding the project, such as About Us, Review Link, Project Explanation, and the actual application. This was fabricated from the beginning and was implemented with the usage of the above-mentioned web applications and Flask.

3.2.2. Usage of Certain APIs in the Project

Now, apart from all the Python and Web App implementations, certain APIs played a key role in developing the nuances of the project in a formidable way, such as GoogleTTS, which was the interface that delivered the automated voicenote i.e., conversion of derived text from the model to speech to the user. This was one of the easier options to use for the text-to-speech module, since developing another model or program to perform that task might cause a clash of memory input and complexity with the ongoing text-prediction model. Hence, GTTS was the right option to choose. Apart from this, we

implemented an easy core socket for the future developer to add a database to the project with ease and develop a directory without hassles. We have left it as an option of access between the modern cloud-based servers such as AWS or Firebase as we mentioned earlier.

3.2.3. Gathering of the Dataset from IAM and Classification for Line and Word Format

The dataset that was used to train the model with is from the IAM Dataset Collection. The IAM Handwriting Database is a publicly accessible and freely available handwriting dataset that is used for non-commercial research purposes. Upon using data from the IAM Handwriting Database, a registration is required in order to be aware of who is using their data. Since we are publishing scientific work based on the IAM Handwriting Database, we were requested to include IAMs reference to the paper.

The IAM Handwriting Database is hierarchically structured into forms that are of different speech sets: “words” database for the training of model with singular words, “lines” database to train model with a couple of words (1-3 words in a line), and “sentences” database that could train the model for a set of sentences (1-2 lines). The datasets that we used in our project were lines and sentences databases. The model had showed steady epochs with the given database. IAM also provided certain broken images (that we implemented to filter separately in our model) to find the necessary utterances that could help the model identify outliers, or in our case, disoriented or meaningless words.

3.3. Understanding of the Model's Core Requirements

To manifest the application that we dreamt of seeing in action, we decided to implement the base of the application on the cores of the following model – Convolutional Neural Network, that has always been known as the apt Neural Network model to handle and process audio, image, and video clips, Long Short-Term Memory (LSTM) implementation of Recurrent Neural Network (RNN) has proven applicable to compile and transfer information in a larger surface of data layer with better data training, and the Connectionist Temporal Classification (CTC) links with the RNN to compute the loss value. CTC's interference comes up as the task of mining the given array of dataset and finding the apt data node for a certain tested data and give the final text output.

To see every neural networks' purpose in detail, we have to understand the neural network in detail. In the case of a Convolutional Neural Network, they are considered the most volumetric Neural Network for the amount of data to be streamed from it. CNN is a type of neural network that would only require a specific convolutional layer, but can have other types of nonlinear, pooling, and fully connected layers as well, which would create a deeper convolution to the pre-existing convolutional neural network. Depending on our application, CNN can be beneficial, since to handle multiple words in a sentence, we require a lot of accuracy to be built up. However, it brings additional parameters for training. As we know, in the CNN, convolutional filters are trained via backpropagation method. The shapes of the filter structure depend on the given sentence, or sometimes the input. For instance, let us review the application, one filter can perform letter extraction, whereas another can carry out

curve extraction. However, we do not fully control these filters in CNN, and their values are determined through learning.

Numerous works have been published on the LSTM type of an RNN, many of them reporting on the results achieved across a wide variety of application domains where data is sequential. The impact of the LSTM network has been notable in language modeling, speech-to-text transcription, machine translation, and other applications. All major open-source machine learning frameworks offer efficient, production-ready implementations of a number of RNN and LSTM network architectures. The advantage of this lengthier LSTM path is that it affords an opportunity to build a correct choice of selection that can prove beneficial during all phases of the process of incorporating an open-source module to suit the needs of their research effort or a long data collecting application, preparing the dataset, data troubleshooting, and tuning.

Labelling unsegmented sequence data is rather a tedious problem in a sequence learning model. It is particularly common in tasks such as handwriting recognition, speech recognition, gesture recognition and any other learning task where noisy, real-valued input streams are annotated with strings of discrete labels, such as letters or words, such as our project. This is where Connectionist Temporal Classification comes to play. A CTC network has a SoftMax output layer with one more unit than there are labels in L , the labels being every predicted word's accuracy. The activations of the first L units are interpreted as the probabilities of observing the corresponding labels at particular times. The activation of the extra unit is the probability of observing a 'blank', or no label, defining the letter/word has no value.

Together, these outputs of the collective neural networks define the probabilities of all possible ways of aligning all possible sequences of the words with the input sentence. The total probability of any one labelled prediction can then be found by adding the probabilities of the sentence's different alignments in the neural network summation.

3.4. Implementation of the Python Modules

As the first block, a Neural Network is built that is completely trained on images of words and sentences from a designated dataset (IAM). Keeping the input layer moderately small for the word/line images, every other layer is compelled with the initial layers, making it feasible for the NN training on the CPU. These are the initial steps to build a base (i.e., a Neural Network) for the conversion that we're aiming in this project. The main layers that are needed in this case are as follows: Convolutional NN layer for depicting the word-line image, Recurrent NN for the feature scaling, and a final layer of CTC (Connectionist Temporal Classification) for length and relevancy domain. The depiction of the Neural Network is also feasible in a more formal way as a functional equation, mapping a precise image M (considering a matrix) with a length between 0 and L . The necessity of classifying the words on a character level is plausible for the improvement of the model's overall purpose of increasing its accuracy on the "words and sentences" scheme of things. Thus,

$$\text{NN: } \underset{W \times H}{M} \rightarrow \underset{0 \leq n \leq L}{(C_1, C_2, \dots, C_n)}$$

With the implementation of the Python modules in this project, we use about 5 modules that are assigned to the various tasks of the prediction model and text-to-speech conversion. Firstly, we have the "sample pre-processor" module that pre-processes the dataset of

images for the model. Which is followed by “data loader” module that reads the obtained samples and loads them in batches to create a sort of iterator-interface to read the whole set of samples, and the actual “model” that predicts the word/sentence from the given input using the samples’ pattern. The “T2SConvert” module converts the predicted sentence from the input to an automated voice output for the user as the final product. As a main method to make a collective working of all the modules mentioned above, we use a HTRWork.py module that integrates the model’s functions, and links with the flask python module to be integrated with the website/UI.

HTRWork.py	Executes all the relevant .py programs together and acts as a link to the Flask UI module
SamplePreprocessor.py	Accesses the dataset of images from the Neural Network and prepares the data
DataLoader.py	Reads all the pre-processed samples given and segregates the common data
Model.py	Creates, stores and loads the developing model for the program
def T2SConvert ():	Converts the text module given as an output in the model.py to an automated speech

**T
a
b
l
e
3
.**

. Different Python modules in the project

3.5. Implementing the Learning Model with NNs

As discussed earlier, the usage of CNN and RNN (in form of LSTM) is the primary key role of this project. In order to build the accuracy of the model and to prevent overfitting, we have used the functions of Feature Scaling, Ensembling, k-fold Cross Validation and Regeneralization.

The input is a grey-value image. It is usually either a maximum grey-scale image or a minimalistic B/W, with a size of 128x32. Finding a few images from the dataset out of proportion and size could be common, hence the images are resized which occurs without distortion. The image is later copied to a target image of the same specified size until the image is scaled either between the choices of “128 width” or “32 height”. Finally, the normalization of the grey-values of the image is performed, to simplify the Neural Network’s performance.

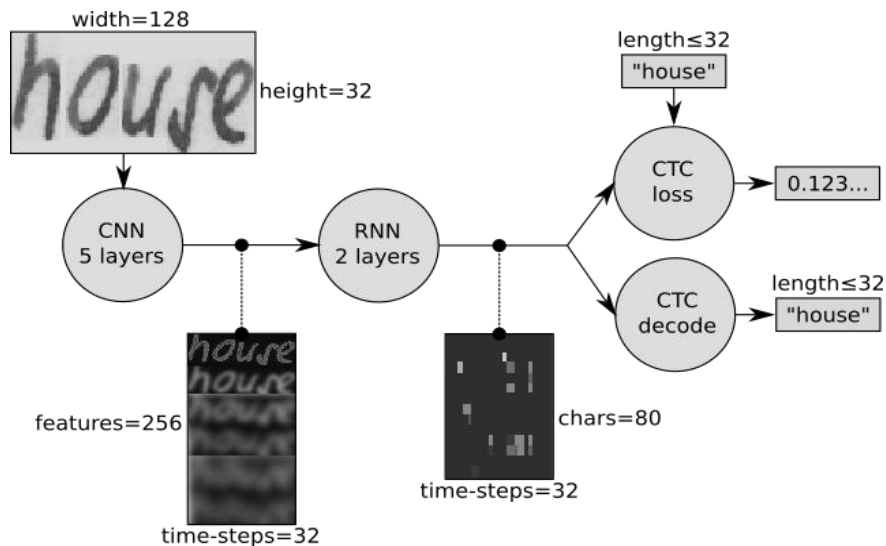


Fig. 3.1. The different Neural Networks present in the project

In the case of CNN, the CNN layers fix the output rate length at a sequence of 32. There are 256 features for every entry, which are processed furthermore by the secondary layers of RNN. Even amidst

all this, some cases are definitive where high correlation is seen amongst a few sets of handwriting from the input, such as the characters “e” and “l” have a lot in common when it comes in the cursive handwriting, or in the case of duplicate or misspelt characters (classic case of comparing “ll” to “tt”). In the case of the writing of the word “little”, it can be seen that most of the time, the predictions of the images are done in contrast to the placement of that letter in the word, like the letter “i” in comparison to the letter “l” or “t”. Since the CTC operation is a position-free entity and seldom goes out of proportion for a handwritten word, the characters “l”, “i”, “t”, “e” and the void label of the text’s classification is a minimalistic task to decode, hence we concatenate the most similar pattern of the words in the sentence to that output line. This creates a best-case scenario for that sentence, thus eliminating all repeated words, letters, and blanks, and with that, we conclude “l---ii--t-t--l---e” → “l---i--t-t--l---e” → “little”.

3.6. Pointers before developing the program

With the implementation of the said neural networks and the python modules, it is time to get forward with developing the learning model, and finally, the Language Recognition program. Before we move forward with the development part, please make sure to keep the following pointers in mind:

- Nowhere must the files that are about to be used and executed in this program leave or be accessed from a different directory altogether. Please try to maintain all the files regarding the project to be maintained in one directory for the ease of access.
- Please do not run the model in cmd (Command Prompt) for its

smooth running. You may find the cmd to execute to model just as fine as any other IDEs, however the main drive will hold unnecessary cache that will be associated with the main directory in future use and cause unwanted disputes of running the file in any other platform.

- Please do not refrain from using different dataset (either from third party websites, or your own created ones) on this model, since you might want to develop the same model with any different language, such as your mother tongue. This will be of great use to the world regardless, so please access the “regain” notepad file from the data file for more information on that.
- Do not shift any file from the given folder to any other folders and subfolders in the main directory as it could cause confusions while executing the end product.

Now, let us start with developing the Language Recognition Project.

3.7. Developing the Language Recognition Project

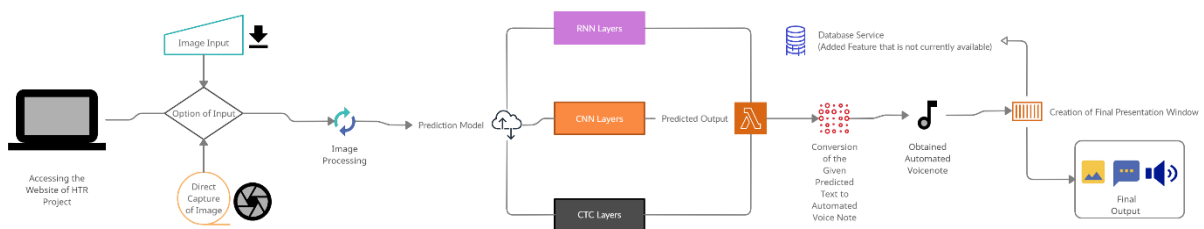


Fig. 3.2. – The flowchart of Language Recognition Project

This is the flowchart of the final end-to-end project's working. The preview of this image might be small, but this is indeed split in a closer view as we proceed with the explanation. For a clear view and explanation of the purposes that happen in front-end, model and back-end of this project, we go in detail with the mentioned sub-parts of the project one by one.

3.7.1. Front-End Development

The first work that would be visible to the user is the website. We shall discuss this in two parts of development – one being the initial pages of Home, About Us, etc. along with gathering the input from the user, and the one being the Model-Website Integration. Let us begin with the initial works of the website.

The website was developed from scratch using HTML, CSS, and Js. On opening the website, the website looks something like this:

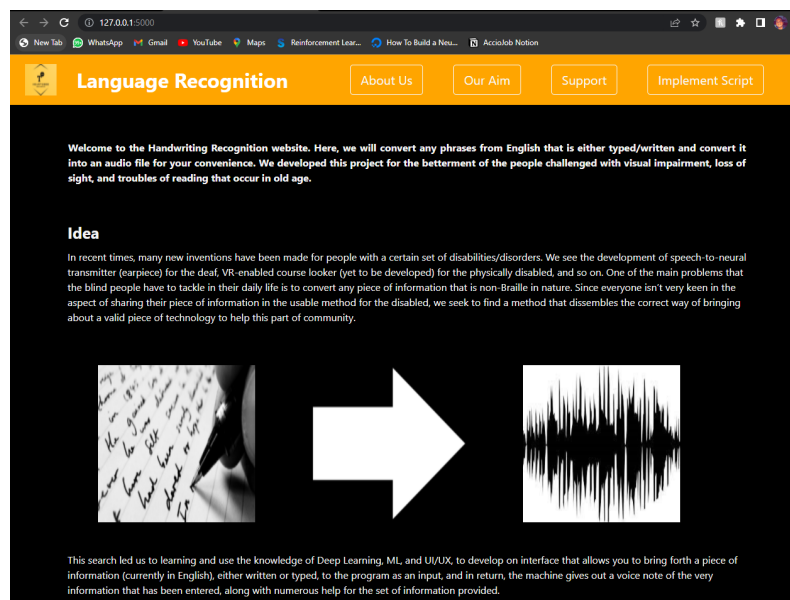


Fig 3.3. Home Page of project website

The website will show the basic aim and scope of the project in the “Home” title, along with the initial builds and runs.

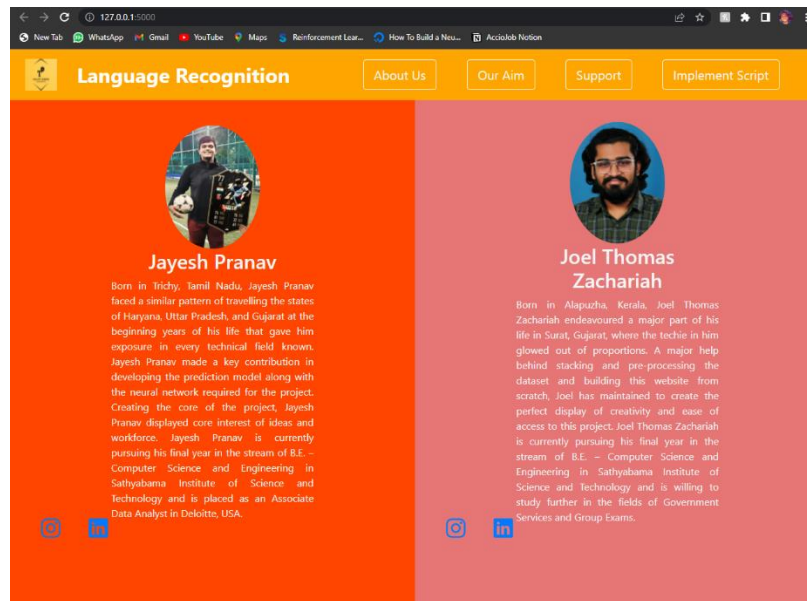


Fig 3.4. – About Us page of the project website

This is the “About us” page, where we introduce ourselves and describe our college life and career interests. This will be followed by our opinions and updates regarding the project in the “Support” page which is also attached with survey links and

supporting documents that could let the user share their feedbacks and also transparently see our motives.

The final page to visit is the “Implement Script” page. This page was meant to be named “HearToSee”, after the name of our project idea “HearToSee”. This is where the model is implemented with the website and the user can give their inputs.



In case of the first set of giving input to the model, the flowchart here explains how that is done. Here, the “Implement Script” page is accessed where the website asks for user’s input using either a “File Upload” button or opens a camera (using OpenCV) that will directly capture an image via face camera, converts the input into grayscale and sends the image to the model for the prediction.

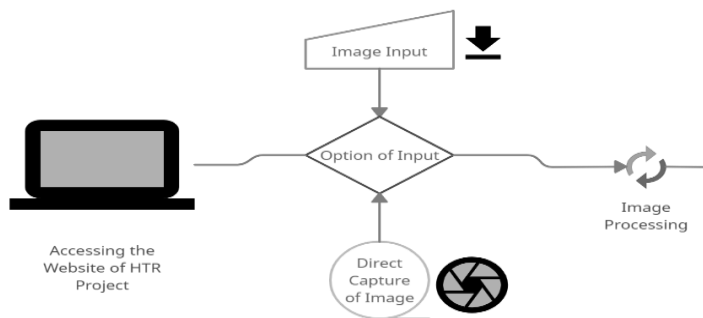


Fig 3.5. – The Front-End Flowchart of the project

3.7.2. Model Development

The algorithms used in the project are solely based on the models of CNN, RNN, CTC and LSTM. The flow goes in the manner of deriving the data from the dataset that we use in the program, the IAM Dataset. Following that, we use the pre-processing methods to batch, loss-take and load the data into the model. The model also has a character error check module which will continually improve the model's accuracy as the error is received via the model running, and keeps the model in its sanity if the model has predicted the input just about right.

Word validations occur predominantly as the pre-processor will get the image from OpenCV and infer the model's run for the specified. This will also check the error ratio and rate for the further simplification and improvement of the model in the future. An inferring function is run alongside the cv2 module to convert the given image in a recognizable manner for the machine (grey-scale) and give the recognized output with the probability percentage of the given phrase's success in the model.

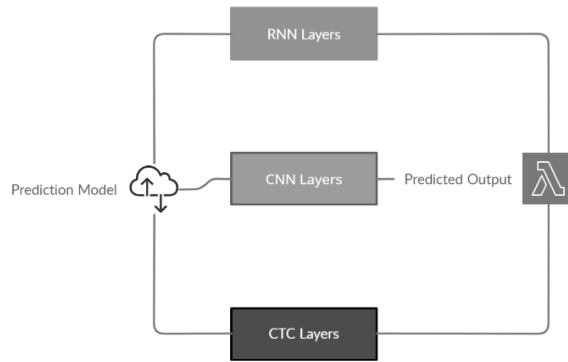


Fig 3.6. – The Model flowchart of the project

The processed output line is then sent into the next function that converts the given output text phrase into an automated voice note for the user to listen to and gather the information. This concludes the flowchart of the project and defines its full functionality.

3.7.3. Back-End (Output) Development

With the prediction output from the model delivered back to the python module, the final works of converting the given predicted text to speech occurs now. The GoogleTTS API that we mentioned earlier comes into work as it develops an automated voice note (in any accent that is wished to be kept, editable in the code) of the given text in a playable format (either .mp3 or .wav). This is then stored in the database (if one such is availed, otherwise the predicted data is volatile) for the user's purpose. A final webpage is created with the user's given input image, the predicted text and the prepared automated voice note.

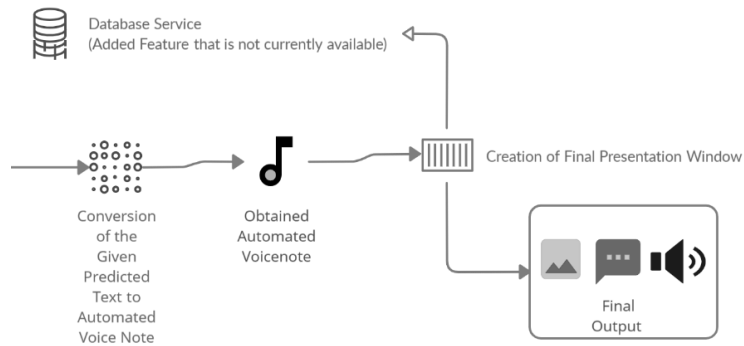


Fig 3.7. – The Back-End flowchart of the project

With that, the project’s development is complete. The project is now an end-to-end volatile application that can predict the given image text and convert it to an automated voice note.

3.8. Running The Project with Model-Website Integration and Modulating the Result with Feature Enabling

With all the modules mentioned above executed, the Language Recognition project must be developed perfectly. The Flask should be integrated correctly with the model and website developed as well. However, there are the final set of integrations, file inclusions and OS allowances that must be kept in mind and executed before running the project in order to connect the project’s loose ends:

- The *model.py* and *htrwork.py* must definitely be imported into the flask’s python application (*app.py*) in order to integrate the model and website and encapsulate these two individual entities as a whole and working project.
- All the python applications are made sure to be kept in the

main file of the directory and not in any subdirectories to avoid any functionality errors in the website or models' working.

- The files of inputs (image) and outputs (prediction voice note) are made to be saved at the “files” sub-directory of the project. But as mentioned before, anybody willing to develop a database for the project in the future can be enabled to integrate the same using the code of input and database management found in *preprocessor.py* and *model.py* properly.
- The camera input is currently integrated using the OpenCV module and the model is efficient to predict the text from the image in a webcam, however with less accuracy due to the picture with pixelization and less resolution obtained from a usual webcam. Consider using the webcam in a well-lit environment with the manuscript shown well. As an Android/Apple application, the project may thrive well with volatile and immediate camera-based inputs.
- If the prediction is found to have any sort of demeaned or inaccurate prediction, please make sure to enable the default word decoder to be “WordBeamSearch” for more accuracy, or try developing the model's accuracy by training the model with new dataset either user-made or third party accessed.

The project should be ready to be accessed. In order to run the project, perform the following steps:

- Open a terminal in the IDE that is used (VSC in this case).

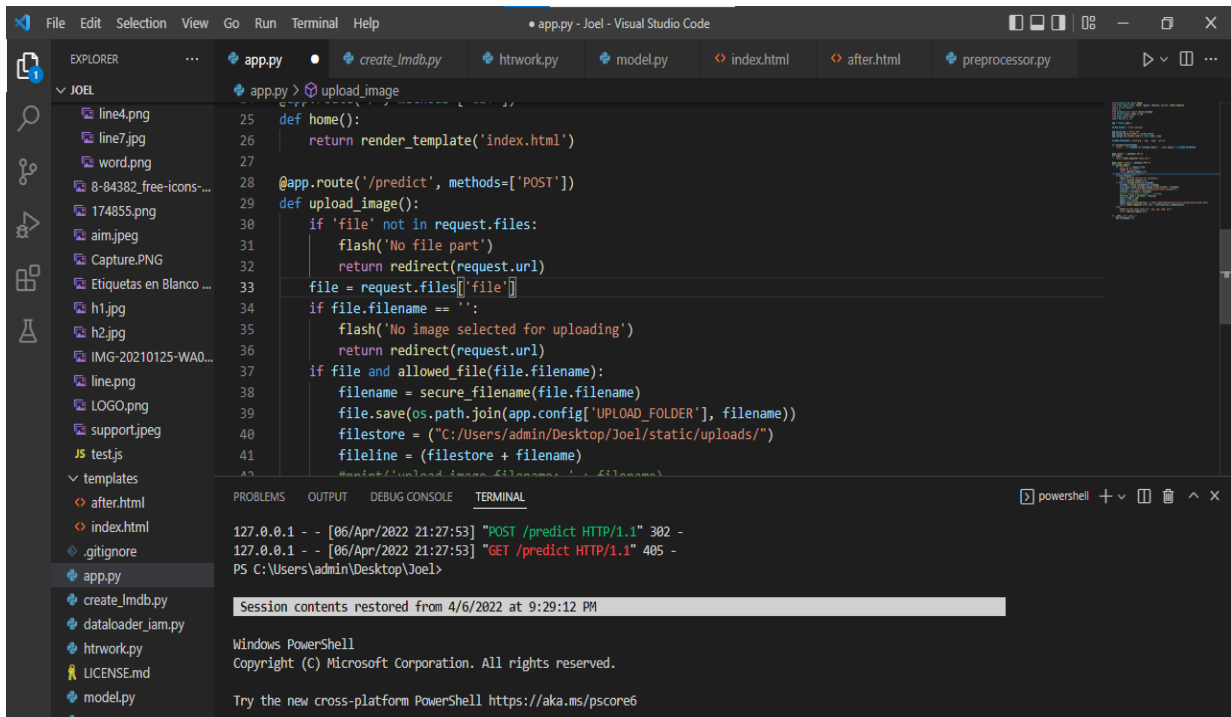


Fig 3.8. The Visual Studio Code IDE (VSC) containing the project code

- Open the file path of the terminal to the main directory of the project.

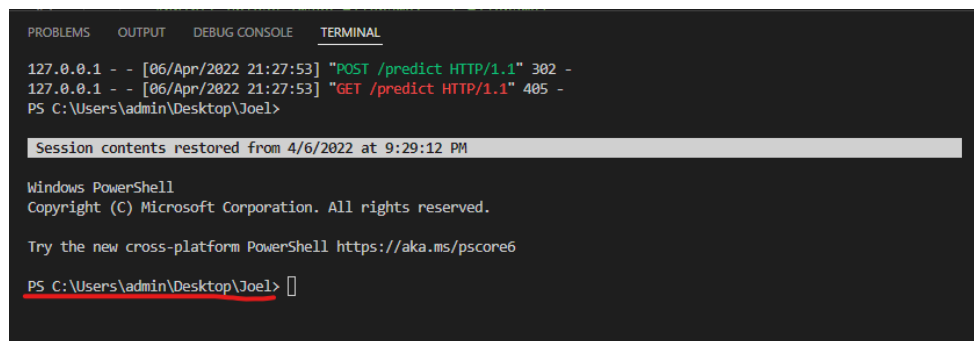
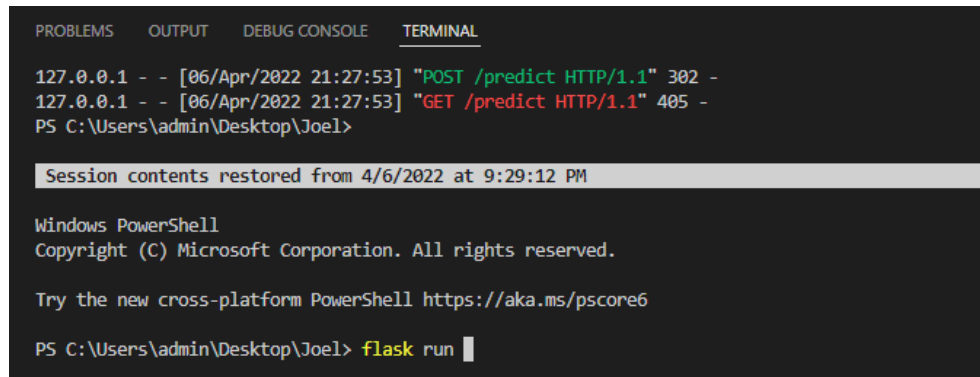


Fig 3.9. The Visual Studio Code IDE (VSC) terminal stated to the main directory

- Execute the command "flask run" in order to run the flask applications present across all the python modules in the directory in revised order of the website.



```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL

127.0.0.1 - - [06/Apr/2022 21:27:53] "POST /predict HTTP/1.1" 302 -
127.0.0.1 - - [06/Apr/2022 21:27:53] "GET /predict HTTP/1.1" 405 -
PS C:\Users\admin\Desktop\Joel>

Session contents restored from 4/6/2022 at 9:29:12 PM

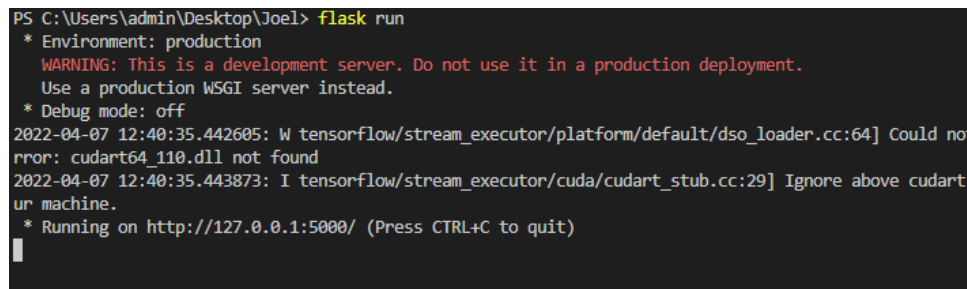
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\admin\Desktop\Joel> flask run
```

Fig 3.10. Running the flask application to execute the project

- Upon executing, the flask will run the supporting modules (TF, RoughLine, etc.) and a portal link will be visible. Click the link to open the website in the default browser of your device.



```
PS C:\Users\admin\Desktop\Joel> flask run
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
2022-04-07 12:40:35.442605: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudart64_110.dll' not found
2022-04-07 12:40:35.443873: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart error message.
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Fig 3.11. Running flask application with the link to open the project website

- Click the “Implement Script” link on the top-right side of the website to open the model. Give the desired input in the form of the following extensions - .JPEG, .PNG, and .JPG, and press “Predict” button to see the results.

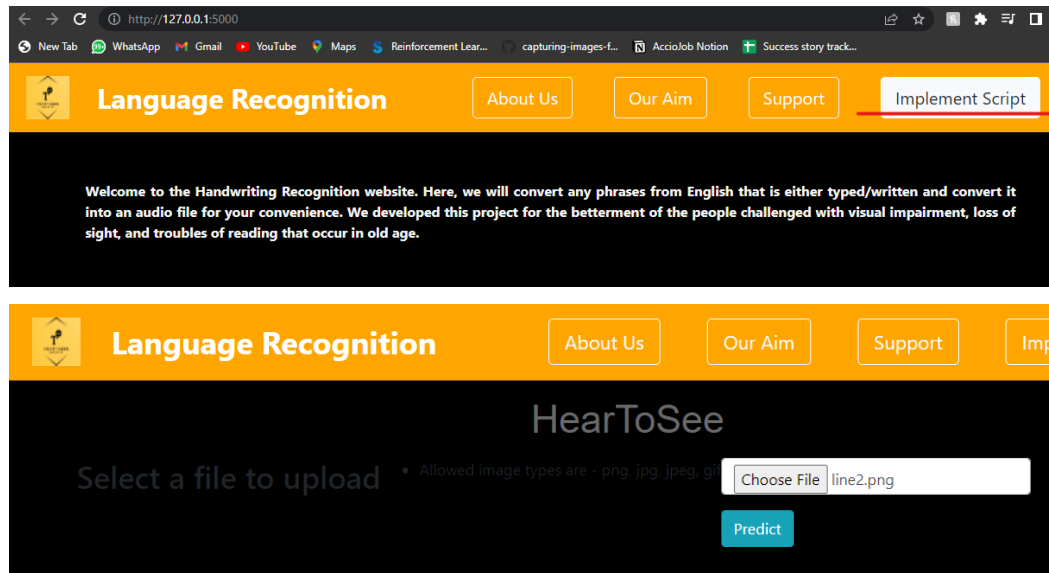


Fig 3.12. The website with the file input mode to give the manuscript input to the model

4. RESULTS AND DISCUSSIONS

4.1. RESULTS OBTAINED FROM THE WEBSITE

The project had been implemented and developed perfectly, which resulted in the execution of the application successfully. Following the implementation of the python modules and the learning model, the website is then integrated with flask and the project is run, as we have seen.

The “Home” screen of the website as we have seen earlier, gets navigated to the “Implement Script” page, we find the options for

giving the input via the file upload scheme. Upon selecting the input and uploading, the website then uploads that to the model.

The model predicts the text from the image and later loads to a new page where the following output comes up (cmd output given for now). Here we show the model's input and output format followed by the website's input and output format:

or work on line level F

ig. 4.1. Input for the Model's Prediction (CMD Execution)

```
Call initializer instance with the dtype argument instead of passing it to the constructor
Python: 3.8.10 (tags/v3.8.10:3d8993a, May  3 2021, 11:48:03) [MSC v.1928 64 bit (AMD64)]
TensorFlow: 2.5.0
2022-04-07 12:59:57.513619: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary
(oneDNN) to use the following CPU instructions in performance-critical operations: AVX
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
Init with stored values from ../model/snapshot-13
*
*
*
*
*
Welcome to the Language Recognition Project. Your image's audio result will arrive shortly!

L
D
A
D
I
N
G
.
.
.

Recognized: "or work on line level"
Probability: 66.7436752319336
Do you want me to repeat the phrase? Press Y/N
no
Thank you for using our project! Come back soon!
```

Fig. 4.2. Output of the Model's Prediction (CMD Execution)

trigger-finger and the ritual slaughter of beasts

Fig. 4.3 Input for the Model's Prediction (Website Input)

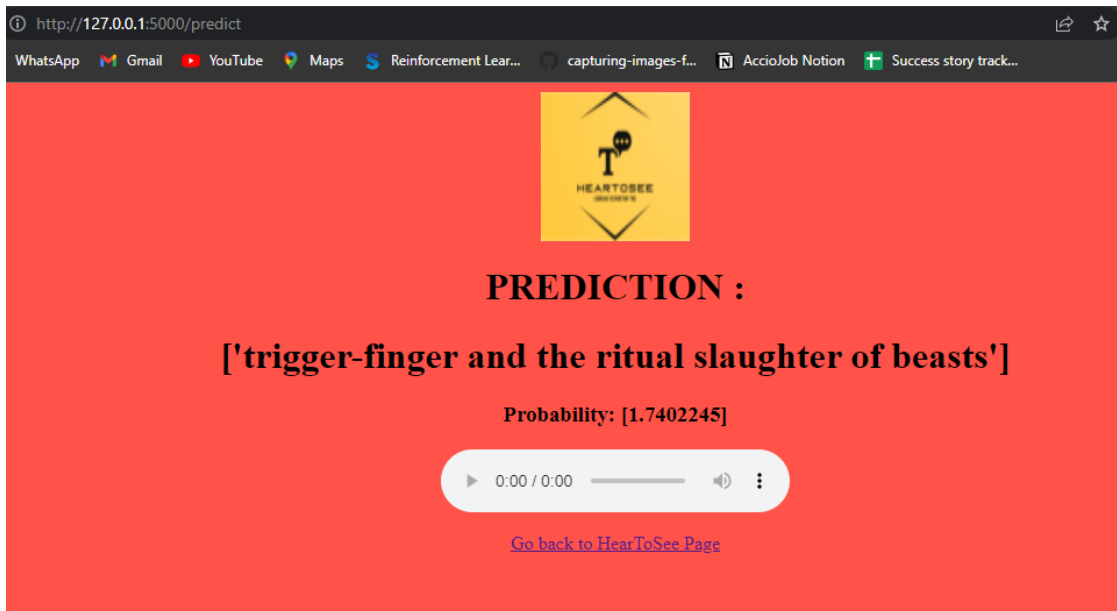


Fig. 4.4. Output of the Model's Prediction (Website Output)

With the output that comes up right with the probability, the user will be provided with the accurate answer for the input that they give. Thus, it would serve the purpose that was mentioned in the scope quite correctly. Over the period of developing the project, we found that the perfect intuitive website was needed for the user to feel at ease with the execution of the Language Recognition and get the desired voice output, and the website stands for the purpose.

4.2. BETA Review and the Discussion with Users

We displayed the Language Recognition project with a few users as a beta test. Preliminarily taking people with low to negligible visual faults, we reviewed the performance of the website with their inputs in

a piece of paper. With 7 users on our side, they were instructed to write a line in the piece of paper, click the image in the camera and upload it into the model. Upon doing the following, the results came back right for about four people, and partially right for the remaining three people. The partial outputs had either misinterpreted a few words or misaligned a few words upon completing prediction.

The users had found the website very simple and deliverable for anyone with layman knowledge over websites to use for easily. However, they found that the convergence of the picture data to be a bit better, along with the recognition accuracy of the model. Apart from that, the main look of the interface and the small details found in the website towards the aim of the project sat well with the user-friendly aspect. If accessed by the people with partial or permanent visual impairment, would have a convincing result with the usage of the project.

Some of the key points of improvement agreed to be implemented as a future update were the following:

- The development of an Android and iOS application for the following model could bring a lot of ease for anyone wishing to use this model from a convenient place anytime. This would need the deployment of the model to a user-friendly application via ADS (Android Development Studio) or Swift for Android and iOS application respectively.
- The development of the model in different languages used in our country commonly such as Hindi, Tamil, Telugu, Marathi, Urdu, etc. for the users belonging to certain regions of India be able to convert manuscripts of certain language to the automated voice notes. So far, the model

can assess data and labels of different languages perfectly, in accordance to a completely prepared and well-defined dataset and labels of that language to be used for training the model.

- Assessing the neural network with adding a few more CNNs and RNNs for more accuracy. According to our experimentation, the minimal number of layers needed for the model to have a threshold accuracy of identifying a few to couple of lines to be predicted is about 5 for Convoluted NNs and 2 for Recurrent NNs. Any more RNNs added to the model results in a good accuracy of predicting the correct phrase from the given inputs of a CNN, which in turn may also require a good improved input from CNN's end that needs more significant number of CNN layers. A predictive and accurate scoring from CTC can also be prepared for an accurate loss error rate of the predicted words.

5. CONCLUSION, IMPROVEMENTS, AND FUTURE WORKS

We discussed about a Neural Network that was able to recognize text in images, whether handwritten or typed, and converted the text format to a voice note. A Python (.py) implementation using TensorFlow (TF) is provided, along with their roles in the project. Some important parts of the model were introduced separately and inspected for their functions. As different entities, they worked out to be a consistent application with the help of different modules.

The final NN model had consisted of 5 Convolutional NN and 2 Recurrent NN layers, which outputs a character-probability matrix. This matrix is either used as a throughput to the model improvement techniques, that were displayed to improve the recognition accuracy of the shown developed model.

Overall, the project was executed successfully with the algorithm of development performed rightly, and the end model of the project was entertained well with the beta users, guides, and the internal examiners.

For improving the recognition accuracy of the model, we followed some of the common regimens – we enabled Data augmentation to be used as to expand the size of the dataset by enabling non-sequential input transformation of the images via Regeneration and Duplication.

By increasing the size of the input of the NN layer, if the layer is large enough, we could enable the text-lines completely. We confined the output words to be a proper word from the dictionary is permitted, and if the dictionary doesn't seem to find the predicted word from the input, then a text correction is done by searching for the most similar one, and the cursive writing style was removed from the input images, or classified as broken images to conclude the input invalid for conversion.

A sample of handwritten text in cursive script, showing the words "He is the friend of the family, like the". The text is written in black ink on a white background.

Fig 5.1. Cursive Writing that is removed from classification

Looking at the initial stages of the model's training aspects, the training data – testing data was split in the ratio of 4:1 (i.e., 80% - 20%). The model was trained by the mean of the loss values of the batch elements.

With regards to analyzing the preloaded dataset (IAM Sentences version and some self -made dataset), the images of various pre-written sentences is linked and trained along with their actual phrase in the ASCII (or word) format.

In the middle of the training process, some broken images are also added to identify the potential outliers in the input of such were present. During the Training Phase, the model is set to find a stable accuracy rate with every count batch it analyses, and the completion of a standard 25 epochs (batch analysis) of the whole training dataset, the accuracy increases up to about 78%.

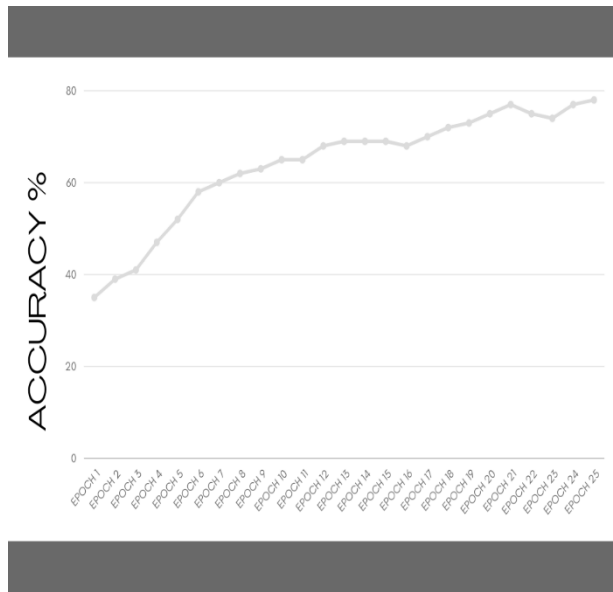


Fig 5.2. Graph depicting the Accuracy percentage of the Model's training-testing sequence

The missed-out features of user account formation and database facilities could possibly be the major improvements to be implemented in the future works. As said earlier, this project is capable of being trained with different language's datasets that could enable the model to predict different languages' texts and convert them to the voice of that language, hence there is not much feature editing needed.

This paper has been possible by learning the works of the output of the project developed by Andrei Burta, Roland Szabo, and Aurel Gontean. In their paper discussed before, their sole purpose was to enact the task of captioning the shown product to the camera to its nomenclature (name) for the user to know.

We had taken the initiative to have changed the approach of reading a product, to reading a script. In the end, the aim from both the ends were to serve the people with either visual impairment or lack of literal knowledge would be benefited by this project.

REFERENCES

Books:

[1] Aurélien Géron (2019). Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow: Second Edition. *Pages 324-481*

[2] Hisham, A., Harin, S. (2017). Deep Learning – the new kid in Artificial Intelligence. *Pages 97-210*

[3] Jeffrey Zeldman (2009). Designing with Web Standards (3rd Edition). *Pages 129-305*

[4] Robin Nixon (2014). Learning PHP, MySQL, JavaScript, CSS & HTML5: A Step-by-Step Guide to Creating Dynamic Websites. *Full Book*

Conference Papers:

[1] Dali Zhu, Long Yang, Zhanxun Liyz, Hualin Zeng, Chinese Academy of Sciences, Beijing, China. Remote Speech Extraction from Speckle Image by Convolutional Neural Network

[2] K. Sasindu Anuradha. and Samantha Thelijagoda. (2020). Sri Lanka Institute of Information Technology, Sri Lanka. Machine translation system to convert Sinhala and English Braille documents into voice.

[3] K. Sakthidasan @ Sankaran, K.Kavitha, S. Haritha Priya. Content Based Image Retrieval Process for Speech Annotated Digital Images.

[4] Albawi, Saad; Mohammed, Tareq Abed; Al-Zawi, Saad (2017). [IEEE 2017 International Conference on Engineering and Technology (ICET) - Antalya, Turkey (2017.8.21-2017.8.23)] 2017 International Conference on Engineering and Technology (ICET) – “Understanding of a convolutional neural network.”, ppt.1–6.

[5] Bourbakis, Nikolaos (2008). [IEEE 2008 20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI) - Dayton, OH, USA (2008.11.3-2008.11.5)] 2008 20th IEEE International Conference on Tools with Artificial Intelligence - Automatic Image-to-Text-to-Voice Conversion for Interactively Locating Objects in Home Environments.

Videos and Courses:

- [1] FreeCodeCamp – Learn Flask as a beginner with FreeCodeCamp
https://www.youtube.com/watch?v=Z1RJmh_OqA
- [2] Udemy - Deep Learning A-Z™: Hands-On Artificial Neural Networks -
<https://www.udemy.com/course/deeplearning/>
- [3] Udemy - Machine Learning A-Z™: Hands-On Python & R In Data Science -
<https://www.udemy.com/course/machinelearning/>
- [4] Udemy - Learn Bootstrap 4: Create Modern Responsive Websites -
<https://www.udemy.com/course/learn-bootstrap-4-create-modern-responsive-websites-in-2019/>

APPENDICES

Source Code

HTRWork.py

```
import argparse
```

```
import json
```

```

from typing import Tuple, List

import cv2

import editdistance

import time

import pyttsx3

from path import Path


from dataloader_iam import DataLoaderIAM, Batch

from model import Model, DecoderType

from preprocessor import Preprocessor


class FilePaths:

    """Filenames and paths to data."""

    fn_char_list = '../model/charList.txt'

    fn_summary = '../model/summary.json'

    fn_corpus = '../data/corpus.txt'


def get_img_height() -> int:

    """Fixed height for NN."""

```

```
return 32
```

```
def get_img_size(line_mode: bool = False) -> Tuple[int, int]:
```

```
    """Height is fixed for NN, width is set according to training mode
    (single words or text lines)."""
```

```
    if line_mode:
```

```
        return 256, get_img_height()
```

```
    return 128, get_img_height()
```

```
def write_summary(char_error_rates: List[float], word_accuracies:
List[float]) -> None:
```

```
    """Writes training summary file for NN."""
```

```
    with open(FilePaths.fn_summary, 'w') as f:
```

```
        json.dump({'charErrorRates': char_error_rates, 'wordAccuracies':
word_accuracies}, f)
```

```
def train(model: Model,
```

```
        loader: DataLoaderIAM,
```

```
        line_mode: bool,
```

```

        early_stopping: int = 25) -> None:

    """Trains NN."""

    epoch = 0 # number of training epochs since start

    summary_char_error_rates = []

    summary_word_accuracies = []

    preprocessor = Preprocessor(get_img_size(line_mode),
data_augmentation=True, line_mode=line_mode)

    best_char_error_rate = float('inf') # best validation character error
rate

    no_improvement_since = 0 # number of epochs no improvement
of character error rate occurred

    # stop training after this number of epochs without improvement

    while True:

        epoch += 1

        print('Epoch:', epoch)

        # train

        print('Train NN')

        loader.train_set()

        while loader.has_next():

            iter_info = loader.get_iterator_info()

            batch = loader.get_next()

```

```

    batch = preprocessor.process_batch(batch)

    loss = model.train_batch(batch)

    print(f'Epoch: {epoch} Batch: {iter_info[0]}/{iter_info[1]} Loss:
{loss}')

# validate

char_error_rate, word_accuracy = validate(model, loader,
line_mode)

# write summary

summary_char_error_rates.append(char_error_rate)

summary_word_accuracies.append(word_accuracy)

write_summary(summary_char_error_rates,
summary_word_accuracies)

# if best validation accuracy so far, save model parameters

if char_error_rate < best_char_error_rate:

    print('Character error rate improved, save model')

    best_char_error_rate = char_error_rate

    no_improvement_since = 0

    model.save()

else:

```

```

        print(f'Character error rate not improved, best so far:
{char_error_rate * 100.0}%')

        no_improvement_since += 1

    # stop training if no more improvement in the last x epochs

    if no_improvement_since >= early_stopping:

        print(f'No more improvement since {early_stopping} epochs.
Training stopped.')

        break

```

```

def validate(model: Model, loader: DataLoaderIAM, line_mode: bool) -
> Tuple[float, float]:

    """Validates NN."""

    print('Validate NN')

    loader.validation_set()

    preprocessor = Preprocessor(get_img_size(line_mode),
line_mode=line_mode)

    num_char_err = 0

    num_char_total = 0

    num_word_ok = 0

    num_word_total = 0

    while loader.has_next():

```

```

iter_info = loader.get_iterator_info()

print(f'Batch: {iter_info[0]} / {iter_info[1]}')

batch = loader.get_next()

batch = preprocessor.process_batch(batch)

recognized, _ = model.infer_batch(batch)


print('Ground truth -> Recognized')

for i in range(len(recognized)):

    num_word_ok += 1 if batch.gt_texts[i] == recognized[i] else 0

    num_word_total += 1

    dist = editdistance.eval(recognized[i], batch.gt_texts[i])

    num_char_err += dist

    num_char_total += len(batch.gt_texts[i])

    print('[OK]' if dist == 0 else '[ERR:%d]' % dist, "" +
batch.gt_texts[i] + "", '->',

        "" + recognized[i] + "")


# print validation result

char_error_rate = num_char_err / num_char_total

word_accuracy = num_word_ok / num_word_total

print(f'Character error rate: {char_error_rate * 100.0}%. Word
accuracy: {word_accuracy * 100.0}%.')

```

```
return char_error_rate, word_accuracy
```

```
def infer(model: Model, fn_img: Path) -> None:
```

```
    """Recognizes text in image provided by file path."""
```

```
    img = cv2.imread(fn_img, cv2.IMREAD_GRAYSCALE)
```

```
    assert img is not None
```

```
    preprocessor = Preprocessor(get_img_size(),  
dynamic_width=True, padding=16)
```

```
    img = preprocessor.process_img(img)
```

```
    batch = Batch([img], None, 1)
```

```
    recognized, probability = model.infer_batch(batch, True)
```

```
    return(recognized)
```

dataloader_iam.py

```
import pickle
```

```
import random
```

```
from collections import namedtuple
```

```
from typing import Tuple
```

```

import cv2

import lmdb

import numpy as np

from path import Path


Sample = namedtuple('Sample', 'gt_text, file_path')

Batch = namedtuple('Batch', 'imgs, gt_texts, batch_size')


class DataLoaderIAM:
    """
    Loads data which corresponds to IAM format,
    see: http://www.fki.inf.unibe.ch/databases/iam-handwriting-database
    """

    def __init__(self,
                  data_dir: Path,
                  batch_size: int,
                  data_split: float = 0.95,
                  fast: bool = True) -> None:
        """Loader for dataset."""

```

```

assert data_dir.exists()

self.fast = fast

if fast:

    self.env = lmdb.open(str(data_dir / 'lmdb'), readonly=True)

self.data_augmentation = False

self.curr_idx = 0

self.batch_size = batch_size

self.samples = []

f = open(data_dir / 'gt/words.txt')

chars = set()

bad_samples_reference = ['a01-117-05-02', 'r06-022-03-05'] #
known broken images in IAM dataset

for line in f:

    # ignore comment line

    if not line or line[0] == '#':

        continue

    line_split = line.strip().split(' ')

```

```

assert len(line_split) >= 9

# filename: part1-part2-part3 --> part1/part1-part2/part1-part2-
part3.png

file_name_split = line_split[0].split('-')

file_name_subdir1 = file_name_split[0]

file_name_subdir2 = f'{file_name_split[0]}-{file_name_split[1]}'

file_base_name = line_split[0] + '.png'

file_name = data_dir / 'img' / file_name_subdir1 /
file_name_subdir2 / file_base_name

if line_split[0] in bad_samples_reference:

    print('Ignoring known broken image:', file_name)

    continue

# GT text are columns starting at 9

gt_text = ''.join(line_split[8:])

chars = chars.union(set(list(gt_text)))

# put sample into list

self.samples.append(Sample(gt_text, file_name))

```

```

# split into training and validation set: 95% - 5%

split_idx = int(data_split * len(self.samples))

self.train_samples = self.samples[:split_idx]

self.validation_samples = self.samples[split_idx:]


# put words into lists

self.train_words = [x.gt_text for x in self.train_samples]

self.validation_words = [x.gt_text for x in self.validation_samples]


# start with train set

self.train_set()


# list of all chars in dataset

self.char_list = sorted(list(chars))


def train_set(self) -> None:

    """Switch to randomly chosen subset of training set."""

    self.data_augmentation = True

    self.curr_idx = 0

    random.shuffle(self.train_samples)

    self.samples = self.train_samples

```

```

self.curr_set = 'train'

def validation_set(self) -> None:

    """Switch to validation set."""

    self.data_augmentation = False

    self.curr_idx = 0

    self.samples = self.validation_samples

    self.curr_set = 'val'

def get_iterator_info(self) -> Tuple[int, int]:

    """Current batch index and overall number of batches."""

    if self.curr_set == 'train':

        num_batches = int(np.floor(len(self.samples) /
self.batch_size)) # train set: only full-sized batches

    else:

        num_batches = int(np.ceil(len(self.samples) / self.batch_size))
# val set: allow last batch to be smaller

    curr_batch = self.curr_idx // self.batch_size + 1

    return curr_batch, num_batches

def has_next(self) -> bool:

    """Is there a next element?"""

```

```

    if self.curr_set == 'train':

        return self.curr_idx + self.batch_size <= len(self.samples) #
train set: only full-sized batches

    else:

        return self.curr_idx < len(self.samples) # val set: allow last
batch to be smaller


def _get_img(self, i: int) -> np.ndarray:

    if self.fast:

        with self.env.begin() as txn:

            basename = Path(self.samples[i].file_path).basename()

            data = txn.get(basename.encode("ascii"))

            img = pickle.loads(data)

        else:

            img = cv2.imread(self.samples[i].file_path,
cv2.IMREAD_GRAYSCALE)

    return img


def get_next(self) -> Batch:

    """Get next element."""

    batch_range = range(self.curr_idx, min(self.curr_idx +
self.batch_size, len(self.samples)))

```

```
imgs = [self._get_img(i) for i in batch_range]

gt_texts = [self.samples[i].gt_text for i in batch_range]


self.curr_idx += self.batch_size

return Batch(imgs, gt_texts, len(imgs))
```

model.py

```
import os

import sys

from typing import List, Tuple


import numpy as np

import tensorflow as tf


from dataloader_iam import Batch


# Disable eager mode

tf.compat.v1.disable_eager_execution()
```

```
class DecoderType:
```

```
    """CTC decoder types."""
```

```
    BestPath = 0
```

```
    BeamSearch = 1
```

```
    WordBeamSearch = 2
```

```
class Model:
```

```
    """Minimalistic TF model for HTR."""
```

```
    def __init__(self,
```

```
        char_list: List[str],
```

```
        decoder_type: str = DecoderType.BestPath,
```

```
        must_restore: bool = False,
```

```
        dump: bool = False) -> None:
```

```
    """Init model: add CNN, RNN and CTC and initialize TF."""
```

```
    self.dump = dump
```

```
    self.char_list = char_list
```

```
    self.decoder_type = decoder_type
```

```
    self.must_restore = must_restore
```

```
    self.snap_ID = 0
```

```

# Whether to use normalization over a batch or a population
self.is_train = tf.compat.v1.placeholder(tf.bool, name='is_train')

# input image batch

self.input_imgs = tf.compat.v1.placeholder(tf.float32,
shape=(None, None, None))

# setup CNN, RNN and CTC

self.setup_cnn()

self.setup_rnn()

self.setup_ctc()

# setup optimizer to train NN

self.batches_trained = 0

self.update_ops =
tf.compat.v1.get_collection(tf.compat.v1.GraphKeys.UPDATE_OPS)

with tf.control_dependencies(self.update_ops):

    self.optimizer =
tf.compat.v1.train.AdamOptimizer().minimize(self.loss)

# initialize TF

```

```

self.sess, self.saver = self.setup_tf()

def setup_cnn(self) -> None:

    """Create CNN layers."""

    cnn_in4d = tf.expand_dims(input=self.input_imgs, axis=3)

    # list of parameters for the layers

    kernel_vals = [5, 5, 3, 3, 3]

    feature_vals = [1, 32, 64, 128, 128, 256]

    stride_vals = pool_vals = [(2, 2), (2, 2), (1, 2), (1, 2), (1, 2)]

    num_layers = len(stride_vals)

    # create layers

    pool = cnn_in4d # input to first CNN layer

    for i in range(num_layers):

        kernel = tf.Variable(

            tf.random.truncated_normal([kernel_vals[i], kernel_vals[i],
feature_vals[i], feature_vals[i + 1]],

                                stddev=0.1))

        conv = tf.nn.conv2d(input=pool, filters=kernel,
padding='SAME', strides=(1, 1, 1, 1))

```

```

        conv_norm = tf.compat.v1.layers.batch_normalization(conv,
training=self.is_train)

        relu = tf.nn.relu(conv_norm)

        pool = tf.nn.max_pool2d(input=relu, ksize=(1, pool_vals[i][0],
pool_vals[i][1], 1),

                                strides=(1, stride_vals[i][0], stride_vals[i][1], 1),
padding='VALID')

        self.cnn_out_4d = pool

def setup_rnn(self) -> None:

    """Create RNN layers."""

    rnn_in3d = tf.squeeze(self.cnn_out_4d, axis=[2])

    # basic cells which is used to build RNN

    num_hidden = 256

    cells =
[tf.compat.v1.nn.rnn_cell.LSTMCell(num_units=num_hidden,
state_is_tuple=True) for _ in

        range(2)] # 2 layers

    # stack basic cells

```

```
stacked = tf.compat.v1.nn.rnn_cell.MultiRNNCell(cells,  
state_is_tuple=True)
```

```
# bidirectional RNN
```

```
# BxTxF -> BxTx2H
```

```
(fw, bw), _ =  
tf.compat.v1.nn.bidirectional_dynamic_rnn(cell_fw=stacked,  
cell_bw=stacked, inputs=rnn_in3d,  
  
dtype=rnn_in3d.dtype)
```

```
# BxTxH + BxTxH -> BxTx2H -> BxTx1x2H
```

```
concat = tf.expand_dims(tf.concat([fw, bw], 2), 2)
```

```
# project output to chars (including blank): BxTx1x2H ->  
BxTx1xC -> BxTxC
```

```
kernel = tf.Variable(tf.random.truncated_normal([1, 1,  
num_hidden * 2, len(self.char_list) + 1], stddev=0.1))
```

```
self.rnn_out_3d = tf.squeeze(tf.nn.atrous_conv2d(value=concat,  
filters=kernel, rate=1, padding='SAME'),  
  
axis=[2])
```

```
def setup_ctc(self) -> None:
```

```
"""Create CTC loss and decoder."""
```

```

# BxTxC -> TxBxC

self.ctc_in_3d_tbc = tf.transpose(a=self.rnn_out_3d, perm=[1, 0,
2])

# ground truth text as sparse tensor

self.gt_texts = tf.SparseTensor(tf.compat.v1.placeholder(tf.int64,
shape=[None, 2]),

                                tf.compat.v1.placeholder(tf.int32, [None]),

                                tf.compat.v1.placeholder(tf.int64, [2]))

# calc loss for batch

self.seq_len = tf.compat.v1.placeholder(tf.int32, [None])

self.loss = tf.reduce_mean(

    input_tensor=tf.compat.v1.nn.ctc_loss(labels=self.gt_texts,
inputs=self.ctc_in_3d_tbc,

                                sequence_length=self.seq_len,

                                ctc_merge_repeated=True))

# calc loss for each element to compute label probability

self.saved_ctc_input = tf.compat.v1.placeholder(tf.float32,

                                shape=[None, None,

len(self.char_list) + 1])

```

```

        self.loss_per_element =
tf.compat.v1.nn.ctc_loss(labels=self.gt_texts,
inputs=self.saved_ctc_input,

                                sequence_length=self.seq_len,
ctc_merge_repeated=True)

# best path decoding or beam search decoding

if self.decoder_type == DecoderType.BestPath:

    self.decoder =
tf.nn.ctc_greedy_decoder(inputs=self.ctc_in_3d_tbc,
sequence_length=self.seq_len)

elif self.decoder_type == DecoderType.BeamSearch:

    self.decoder =
tf.nn.ctc_beam_search_decoder(inputs=self.ctc_in_3d_tbc,
sequence_length=self.seq_len,

                                beam_width=50)

# word beam search decoding (see
https://github.com/githubharald/CTCWordBeamSearch)

elif self.decoder_type == DecoderType.WordBeamSearch:

    # prepare information about language (dictionary, characters
in dataset, characters forming words)

    chars = "".join(self.char_list)

    word_chars =
open('../model/wordCharList.txt').read().splitlines()[0]

```

```

corpus = open('../data/corpus.txt').read()

# decode using the "Words" mode of word beam search

from word_beam_search import WordBeamSearch

self.decoder = WordBeamSearch(50, 'Words', 0.0,
corpus.encode('utf8'), chars.encode('utf8'),

                                word_chars.encode('utf8'))

# the input to the decoder must have softmax already applied

self.wbs_input = tf.nn.softmax(self.ctc_in_3d_tbc, axis=2)

def setup_tf(self) -> Tuple[tf.compat.v1.Session,
tf.compat.v1.train.Saver]:

    """Initialize TF."""

    print('Python: ' + sys.version)

    print('Tensorflow: ' + tf.__version__)

    sess = tf.compat.v1.Session() # TF session

    saver = tf.compat.v1.train.Saver(max_to_keep=1) # saver saves
model to file

    model_dir = '../model/'

```

```
latest_snapshot = tf.train.latest_checkpoint(model_dir) # is there
a saved model?
```

```
# if model must be restored (for inference), there must be a
snapshot
```

```
if self.must_restore and not latest_snapshot:
```

```
    raise Exception('No saved model found in: ' + model_dir)
```

```
# load saved model if available
```

```
if latest_snapshot:
```

```
    print('Init with stored values from ' + latest_snapshot)
```

```
    saver.restore(sess, latest_snapshot)
```

```
else:
```

```
    print('Init with new values')
```

```
    sess.run(tf.compat.v1.global_variables_initializer())
```

```
return sess, saver
```

```
def to_sparse(self, texts: List[str]) -> Tuple[List[List[int]], List[int],
List[int]]:
```

```
    """Put ground truth texts into sparse tensor for ctc_loss."""
```

```
    indices = []
```

```

values = []

shape = [len(texts), 0] # last entry must be max(labelList[i])

# go over all texts
for batchElement, text in enumerate(texts):

    # convert to string of label (i.e. class-ids)
    label_str = [self.char_list.index(c) for c in text]

    # sparse tensor must have size of max. label-string
    if len(label_str) > shape[1]:
        shape[1] = len(label_str)

    # put each label into sparse tensor
    for i, label in enumerate(label_str):
        indices.append([batchElement, i])
        values.append(label)

return indices, values, shape

def decoder_output_to_text(self, ctc_output: tuple, batch_size: int) -
> List[str]:
    """Extract texts from output of CTC decoder."""

    # word beam search: already contains label strings

```

```

if self.decoder_type == DecoderType.WordBeamSearch:

    label_strs = ctc_output

# TF decoders: label strings are contained in sparse tensor
else:

    # ctc returns tuple, first element is SparseTensor

    decoded = ctc_output[0][0]

    # contains string of labels for each batch element

    label_strs = [[] for _ in range(batch_size)]

    # go over all indices and save mapping: batch -> values
    for (idx, idx2d) in enumerate(decoded.indices):

        label = decoded.values[idx]

        batch_element = idx2d[0] # index according to [b,t]

        label_strs[batch_element].append(label)

    # map labels to chars for all batch elements

    return ["".join([self.char_list[c] for c in labelStr]) for labelStr in
label_strs]

def train_batch(self, batch: Batch) -> float:

```

```

        """Feed a batch into the NN to train it."""

        num_batch_elements = len(batch.imgs)

        max_text_len = batch.imgs[0].shape[0] // 4

        sparse = self.to_sparse(batch.gt_texts)

        eval_list = [self.optimizer, self.loss]

        feed_dict = {self.input_imgs: batch.imgs, self.gt_texts: sparse,
                      self.seq_len: [max_text_len] * num_batch_elements,
self.is_train: True}

        _, loss_val = self.sess.run(eval_list, feed_dict)

        self.batches_trained += 1

        return loss_val

```

```

@staticmethod

```

```

def dump_nn_output(rnn_output: np.ndarray) -> None:

```

```

    """Dump the output of the NN to CSV file(s)."""

```

```

    dump_dir = './dump/'

```

```

    if not os.path.isdir(dump_dir):

```

```

        os.mkdir(dump_dir)

```

```

    # iterate over all batch elements and create a CSV file for each
one

```

```

    max_t, max_b, max_c = rnn_output.shape

```

```

for b in range(max_b):

    csv = ""

    for t in range(max_t):

        for c in range(max_c):

            csv += str(rnn_output[t, b, c]) + ';'

        csv += '\n'

    fn = dump_dir + 'rnnOutput_' + str(b) + '.csv'

    print('Write dump of NN to file: ' + fn)

    with open(fn, 'w') as f:

        f.write(csv)

```

```

def infer_batch(self, batch: Batch, calc_probability: bool = False,
probability_of_gt: bool = False):

```

```

    """Feed a batch into the NN to recognize the texts."""

```

```

    # decode, optionally save RNN output

```

```

    num_batch_elements = len(batch.imgs)

```

```

    # put tensors to be evaluated into list

```

```

    eval_list = []

```

```

    if self.decoder_type == DecoderType.WordBeamSearch:

```

```

        eval_list.append(self.wbs_input)

    else:

        eval_list.append(self.decoder)

    if self.dump or calc_probability:

        eval_list.append(self.ctc_in_3d_tbc)

    # sequence length depends on input image size (model
    downsizes width by 4)

    max_text_len = batch.imgs[0].shape[0] // 4

    # dict containing all tensor fed into the model

    feed_dict = {self.input_imgs: batch.imgs, self.seq_len:
    [max_text_len] * num_batch_elements,

        self.is_train: False}

    # evaluate model

    eval_res = self.sess.run(eval_list, feed_dict)

    # TF decoders: decoding already done in TF graph

    if self.decoder_type != DecoderType.WordBeamSearch:

        decoded = eval_res[0]

```

```

        # word beam search decoder: decoding is done in C++ function
compute()

    else:

        decoded = self.decoder.compute(eval_res[0])

        # map labels (numbers) to character string

        texts = self.decoder_output_to_text(decoded,
num_batch_elements)

        # feed RNN output and recognized text into CTC loss to
compute labeling probability

        probs = None

        if calc_probability:

            sparse = self.to_sparse(batch.gt_texts) if probability_of_gt
else self.to_sparse(texts)

            ctc_input = eval_res[1]

            eval_list = self.loss_per_element

            feed_dict = {self.saved_ctc_input: ctc_input, self.gt_texts:
sparse,

                        self.seq_len: [max_text_len] * num_batch_elements,
self.is_train: False}

            loss_vals = self.sess.run(eval_list, feed_dict)

            probs = np.exp(-loss_vals)

```

```

# dump the output of the NN to CSV file(s)

if self.dump:

    self.dump_nn_output(eval_res[1])


return texts, probs


def save(self) -> None:

    """Save model to file."""

    self.snap_ID += 1

    self.saver.save(self.sess, '../model/snapshot',
global_step=self.snap_ID)

```

sample_preprocessor.py

```

import random

from typing import Tuple


import cv2

import numpy as np


from dataloader_iam import Batch

```

```
class Preprocessor:
```

```
    def __init__(self,  
                  img_size: Tuple[int, int],  
                  padding: int = 0,  
                  dynamic_width: bool = False,  
                  data_augmentation: bool = False,  
                  line_mode: bool = False) -> None:
```

```
        # dynamic width only supported when no data augmentation  
        happens
```

```
        assert not (dynamic_width and data_augmentation)
```

```
        # when padding is on, we need dynamic width enabled
```

```
        assert not (padding > 0 and not dynamic_width)
```

```
        self.img_size = img_size
```

```
        self.padding = padding
```

```
        self.dynamic_width = dynamic_width
```

```
        self.data_augmentation = data_augmentation
```

```
        self.line_mode = line_mode
```

```
    @staticmethod
```

```
    def _truncate_label(text: str, max_text_len: int) -> str:
```

```
"""
```

Function `ctc_loss` can't compute loss if it cannot find a mapping between text label and input

labels. Repeat letters cost double because of the blank symbol needing to be inserted.

If a too-long label is provided, `ctc_loss` returns an infinite gradient.

```
"""
```

```
cost = 0
```

```
for i in range(len(text)):
```

```
    if i != 0 and text[i] == text[i - 1]:
```

```
        cost += 2
```

```
    else:
```

```
        cost += 1
```

```
    if cost > max_text_len:
```

```
        return text[:i]
```

```
return text
```

```
def _simulate_text_line(self, batch: Batch) -> Batch:
```

```
    """Create image of a text line by pasting multiple word images
    into an image."""
```

```
    default_word_sep = 30
```

```

default_num_words = 5

# go over all batch elements

res_imgs = []

res_gt_texts = []

for i in range(batch.batch_size):

    # number of words to put into current line

    num_words = random.randint(1, 8) if self.data_augmentation
else default_num_words

    # concat ground truth texts

    curr_gt = ' '.join([batch.gt_texts[(i + j) % batch.batch_size] for j
in range(num_words)])

    res_gt_texts.append(curr_gt)

# put selected word images into list, compute target image
size

sel_imgs = []

word_seps = [0]

h = 0

w = 0

for j in range(num_words):

```

```

        curr_sel_img = batch.imgs[(i + j) % batch.batch_size]

        curr_word_sep = random.randint(20, 50) if
self.data_augmentation else default_word_sep

        h = max(h, curr_sel_img.shape[0])

        w += curr_sel_img.shape[1]

        sel_imgs.append(curr_sel_img)

        if j + 1 < num_words:

            w += curr_word_sep

            word_seps.append(curr_word_sep)

# put all selected word images into target image

target = np.ones([h, w], np.uint8) * 255

x = 0

for curr_sel_img, curr_word_sep in zip(sel_imgs, word_seps):

    x += curr_word_sep

    y = (h - curr_sel_img.shape[0]) // 2

    target[y:y + curr_sel_img.shape[0]:, x:x +
curr_sel_img.shape[1]] = curr_sel_img

    x += curr_sel_img.shape[1]

# put image of line into result

res_imgs.append(target)

```

```

return Batch(res_imgs, res_gt_texts, batch.batch_size)

def process_img(self, img: np.ndarray) -> np.ndarray:
    """Resize to target size, apply data augmentation."""

    # there are damaged files in IAM dataset - just use black image
    instead

    if img is None:

        img = np.zeros(self.img_size[:-1])

    # data augmentation

    img = img.astype(np.float)

    if self.data_augmentation:

        # photometric data augmentation

        if random.random() < 0.25:

            def rand_odd():

                return random.randint(1, 3) * 2 + 1

            img = cv2.GaussianBlur(img, (rand_odd(), rand_odd()), 0)

        if random.random() < 0.25:

            img = cv2.dilate(img, np.ones((3, 3)))

        if random.random() < 0.25:

```

```

img = cv2.erode(img, np.ones((3, 3)))

# geometric data augmentation

wt, ht = self.img_size

h, w = img.shape

f = min(wt / w, ht / h)

fx = f * np.random.uniform(0.75, 1.05)

fy = f * np.random.uniform(0.75, 1.05)

# random position around center

txc = (wt - w * fx) / 2

tyc = (ht - h * fy) / 2

freedom_x = max((wt - fx * w) / 2, 0)

freedom_y = max((ht - fy * h) / 2, 0)

tx = txc + np.random.uniform(-freedom_x, freedom_x)

ty = tyc + np.random.uniform(-freedom_y, freedom_y)

# map image into target image

M = np.float32([[fx, 0, tx], [0, fy, ty]])

target = np.ones(self.img_size[:-1]) * 255

img = cv2.warpAffine(img, M, dsize=self.img_size, dst=target,
borderMode=cv2.BORDER_TRANSPARENT)

```

```

# photometric data augmentation

if random.random() < 0.5:

    img = img * (0.25 + random.random() * 0.75)

    if random.random() < 0.25:

        img = np.clip(img + (np.random.random(img.shape) - 0.5) *
random.randint(1, 25), 0, 255)

        if random.random() < 0.1:

            img = 255 - img

# no data augmentation

else:

    if self.dynamic_width:

        ht = self.img_size[1]

        h, w = img.shape

        f = ht / h

        wt = int(f * w + self.padding)

        wt = wt + (4 - wt) % 4

        tx = (wt - w * f) / 2

        ty = 0

    else:

        wt, ht = self.img_size

```

```

h, w = img.shape

f = min(wt / w, ht / h)

tx = (wt - w * f) / 2

ty = (ht - h * f) / 2

# map image into target image

M = np.float32([[f, 0, tx], [0, f, ty]])

target = np.ones([ht, wt]) * 255

img = cv2.warpAffine(img, M, dsize=(wt, ht), dst=target,
borderMode=cv2.BORDER_TRANSPARENT)

# transpose for TF

img = cv2.transpose(img)

# convert to range [-1, 1]

img = img / 255 - 0.5

return img

def process_batch(self, batch: Batch) -> Batch:

    if self.line_mode:

        batch = self._simulate_text_line(batch)

```

```

        res_imgs = [self.process_img(img) for img in batch.imgs]

        max_text_len = res_imgs[0].shape[0] // 4

        res_gt_texts = [self._truncate_label(gt_text, max_text_len) for
gt_text in batch.gt_texts]

        return Batch(res_imgs, res_gt_texts, batch.batch_size)

```

```
def main():
```

```

    import matplotlib.pyplot as plt

    img = cv2.imread('../data/test.png', cv2.IMREAD_GRAYSCALE)

    img_aug = Preprocessor((256, 32),
data_augmentation=True).process_img(img)

    plt.subplot(121)

    plt.imshow(img, cmap='gray')

    plt.subplot(122)

    plt.imshow(cv2.transpose(img_aug) + 0.5, cmap='gray', vmin=0,
vmax=1)

    plt.show()

```

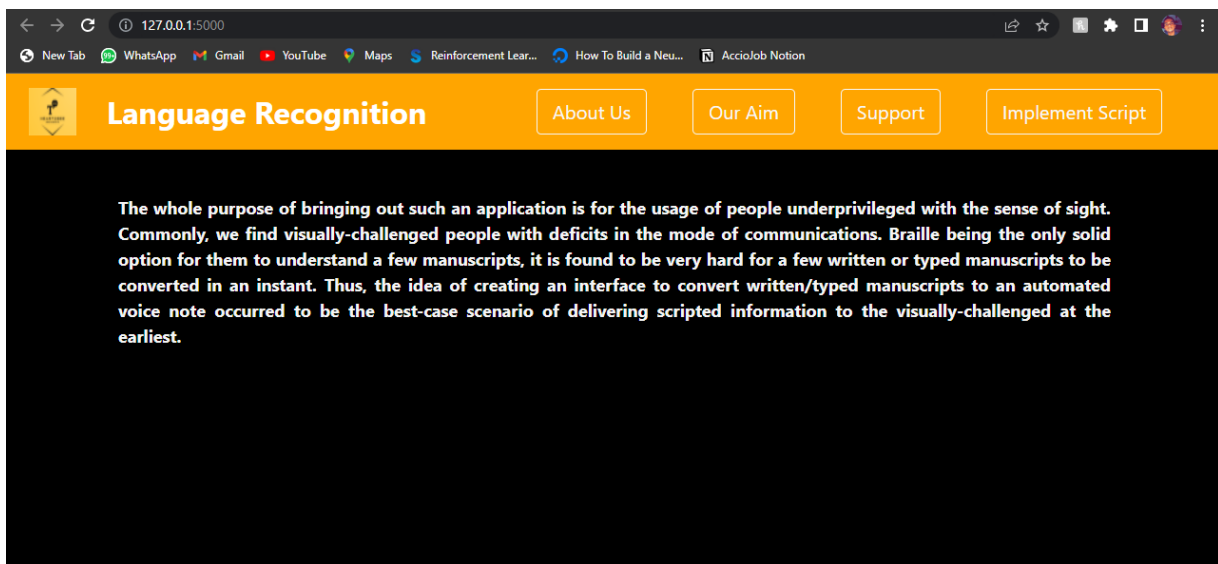
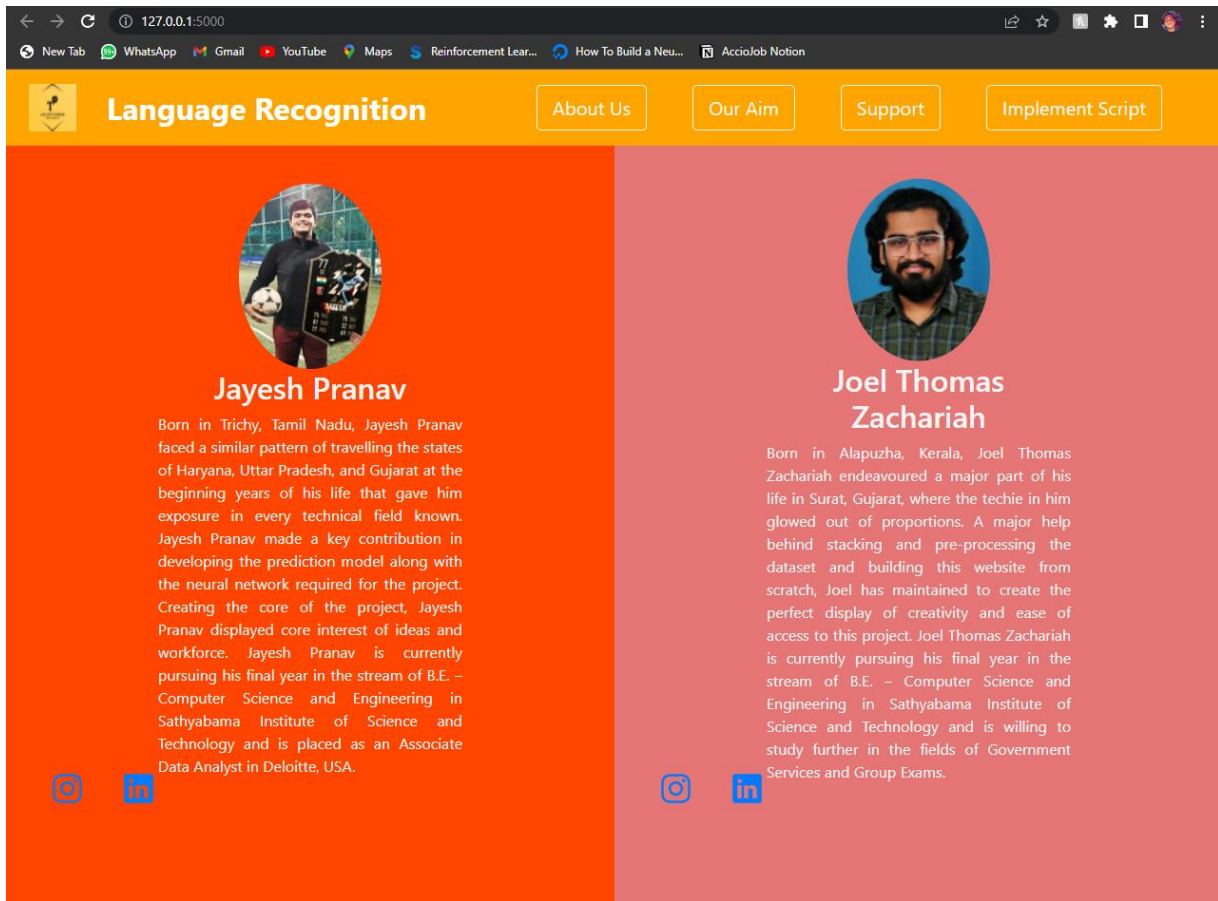
```

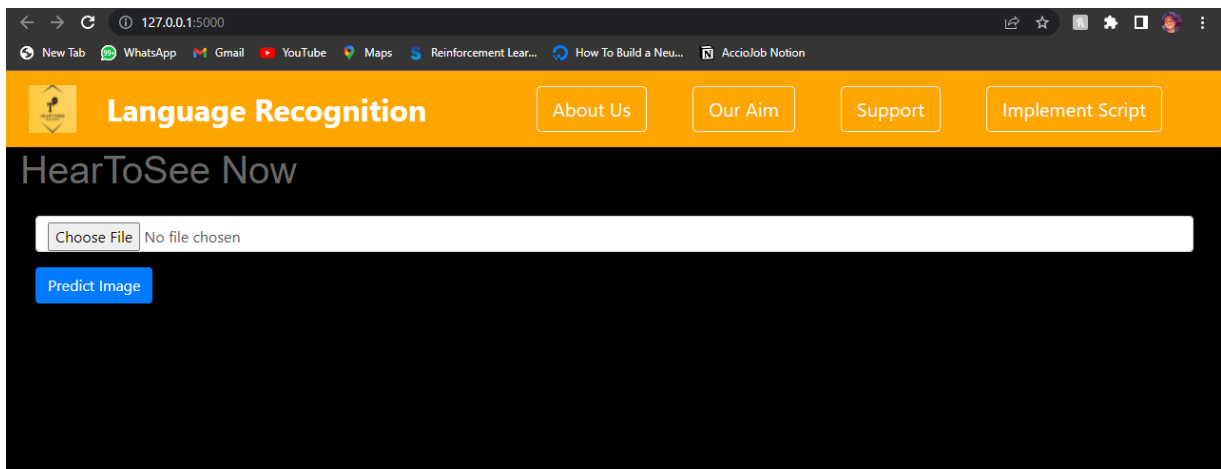
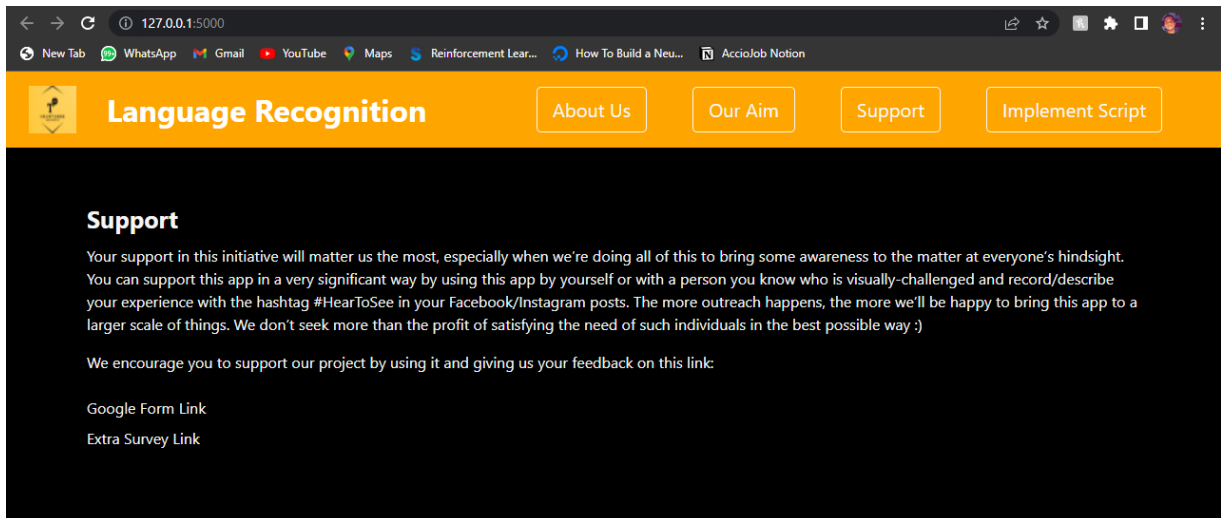
if __name__ == '__main__':

    main()

```

Sample Screenshots





```
Instructions for updating:
Call initializer instance with the dtype argument instead of passing it to the constructor
Python: 3.8.10 (tags/v3.8.10:3d8993a, May 3 2021, 11:48:03) [MSC v.1928 64 bit (AMD64)]
TensorFlow: 2.5.0
2022-03-30 20:45:53.430289: I tensorflow/core/platform/cpu_feature_guard.cc:142] This TensorFlow binary is optimized with oneAPI Deep Neural Network Library
(oneDNN) to use the following CPU instructions in performance-critical operations: AVX
To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.
Init with stored values from ../model/snapshot-13
*
*
*
*
*
Welcome to the Language Recognition Project. Your image's audio result will arrive shortly!

L
O
A
D
I
N
G
.
.
.

Recognized: "or work on line level"
Probability: 66.7436752319336
Do you want me to repeat the phrase? Press Y/N
y
Recognized: "or work on line level"
Do you want me to repeat the phrase? Press Y/N
n
Thank you for using our project! Come back soon!
```

http://127.0.0.1:5000/predict

WhatsApp Gmail YouTube Maps Reinforcement Lear... capturing-images-f... AccioJob Notion Success story track...



PREDICTION :

['trigger-finger and the ritual slaughter of beasts']

Probability: [1.7402245]

0:00 / 0:00

[Go back to HearToSee Page](#)

IEEE Paper Publication (with Plagiarism Proof)





SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)

Accredited with Grade 'A' By NAAC | 12B Status by UGC | Approved By AICTE
www.sathyabama.ac.in



School of Computing

Department of Information Technology

Certificate of Presentation

This is to certify that

Joel Thomas Zachariah

of

Sathyabama Institute of Science and Technology

has presented a paper entitled

Language Recognition from Handwriting Based on Machine Learning and Deep Learning

in the AICTE sponsored

International Conference on Artificial Intelligence and Machine Learning (IAIM-2022),

held from 27th January 2022 - 29th January 2022.

Dr. R. SUBHASHINI
HOD [Information Technology]

Dr. T. SASIKALA
Dean [School of Computing]

Dr. T. SASIPRABA
Vice Chancellor

FYPResearchPaper - IAIM_096.doc

ORIGINALITY REPORT

5 %	3 %	4 %	1 %
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	"[Front matter]", 2020 Fourth World Conference on Smart Trends in Systems, Security and Sustainability (WorldS4), 2020 Publication	1 %
2	Submitted to Universitaet Hamburg Student Paper	1 %
3	dblp.dagstuhl.de Internet Source	1 %
4	www.ijert.org Internet Source	<1 %
5	www.semanticscholar.org Internet Source	<1 %
6	dokumen.pub Internet Source	<1 %
7	Submitted to British University In Dubai Student Paper	<1 %
8	cps-vo.org Internet Source	<1 %

easychair.org

9	Internet Source	<1 %
10	toc.proceedings.com Internet Source	<1 %
11	www.researchgate.net Internet Source	<1 %
12	www.udemy.com Internet Source	<1 %
13	Tassadaq Hussain, Sabato Marco Siniscalchi, Hsiao-Lan Sharon Wang, Yu Tsao, Salerno Valerio Mario, Wen-Hung Liao. "Ensemble Hierarchical Extreme Learning Machine for Speech Dereverberation", IEEE Transactions on Cognitive and Developmental Systems, 2020 Publication	<1 %

Exclude quotes On
Exclude bibliography On

Exclude matches Off