**ABSTRACT**

Deep learning cab be said as subset of machine learning where artificial neural networks are used for knowledge acquisition. It is widely used in image recognition, speech processing, and other tasks. Deep learning algorithms have few networks that stack layers of artificial neurons to extract higher-level features from data almost similar to the human brain’s information processing.

HWR is still challenging, being associated with the variability inherent to writing styles and languages as well as the numerous representations that handwritten characters can have. From the four deep learning techniques discussed, CNNs and RNNs have shown unparalleled superiority in improving the HWR system’s accuracy together with efficiency.

This project aims to capitalize on deep learning in building a productive HWR system. Novel deep learning architectures and training methodologies will unlock state-of-the-art performances. The project will also investigate other pattern recognition challenges, such as picture and speech recognition, are solved using deep learning.

**Keywords***—*Deep Learning, Hand Writing Recognition, Convolutional Neural Networks, Recurrent Neural Networks, Alexnet, Bi-direction SLTM, SGDM.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Chapter No** | **Title** | **Page No.** |
| 1. | INTRODUCTION   * 1. Introduction   2. Statement of the problem   3. Objectives   4. Scope   5. Application   6. Limitation |  |
| 2. | LITERATURE SURVEY |  |
| 3. | ANALYSIS   1. Existing system 2. Design Constraints, Assumptions and Disadvantages |  |
| 4. | DESIGN   1. Realtime Model flow 2. Comparision with other system |  |
| 5. | IMPLEMENTATION   1. Dataset Used 2. CRNN Model Architecture |  |
| 6. | EXECUTION PROCEDURE & TESTING |  |
| 7. | RESULT & PERFORMANCE EVALUATION |  |
| 8. | CONCLUSION & FUTURE WORK |  |
|  | APPENDIX |  |
|  | REFERENCES |  |

**CHAPTER - 1**

**INTRODUCTION**

* 1. **Introduction:**

Handwriting recognition, or HTR (Handwritten information Recognition), is regarded as a current technology that marks a significant change in translating handwritten information into computer readable and editable formats. This research subject has received a lot of interest because to its many diverse applications, including document analysis, historical document digitalization, automation form processing, and others. However, deep learning, specifically the real applications of CNN (Convolutional Neural Networks) and RNN (Recurrent Neural Networks), stands out as a game changer in handwriting recognition, generating consistently higher performance ratings.

**Convolutional** **Neural Networks** are a natural choice in dealing with the problem of handwriting identification due to their success being quite significant when solving imaging-based problems. CNNs can detect tiny details and patterns in handwriting. The CNNs’ hierarchical representation is characteristically spatially precise in terms of fine details and spatial relationships between writing characters. For example, CNNs are utilized in handwriting recognition problems to assess character properties at the pixel level. The learning network is pretty effective in identifying the basic patterns—strokes, loops and curves- used for better character recognition process.

**Recurrent Neural Networks**, which govern the processing of sequential data, play an important role in extracting temporal innuendos from handwritten text. Handwriting is a highly dynamic process in which the order and timing of strokes reveal essential information about character contexts. RNNs can model sequential information, hence they are frequently employed to model handwriting dependencies. This enables the algorithm to determine how the characters are ordered in a word or sentence.

Modern handwriting recognition systems merge CNNs and RNNs into a single model to make use of their synergy. This composite strategy frequently consists of utilizing a CNN for the extraction of features from the input image first, and then an RNN is used to simulate the learned feature's sequential dependence. This synergistic method ensures that the system accurately identifies and translates both spatial and temporal components of handwritten text, hence improving overall performance.

However, the synergy between CNN and RNN is particularly effective when dealing with handwriting, because characters are not just expressive of their shape but also dependent on how they were written. The CNN component retrieves static visual signals, but RNN discloses the temporal sequencing of these features, providing a significantly more thorough understanding of handwritten information.

* 1. **Statement of the problem:**

Develop a deep learning-based handwriting recognition system that works with both pens and screens. Analyze different handwriting styles, from simple notes to complex forms, across languages and dialects. Use a large dataset to create an advanced and adaptable model. Our goal is to integrate with applications that can take advantage of handwritten data, such as document digitization, accessibility software, and automated form-filling techniques. Develop and implement a deep learning architecture capable of providing extremely accurate handwriting recognition across many domains. Examine each character, phrase, and sentence for style alterations, language shifts, and complexity. To achieve high precision and generalizability, train on a carefully chosen dataset. Seek a realistic and scalable model implementation that is simple to incorporate into applications such document image processing, assistive technologies, and intelligent data collection.

* 1. **Objectives:**

Handwriting recognition using deep learning typically aims to achieve several objectives:

* **Accuracy Improvement:** Enhance the accuracy of recognizing handwritten characters, words, or entire sentences. Deep learning models can learn complex patterns and variations in handwriting, enabling better recognition performance compared to traditional methods.
* **Robustness to Variability:** Make the recognition system robust to various handwriting styles, sizes, slants, and other forms of variability. Deep learning models can generalize well to different styles and variations in handwriting, making them adaptable to a wide range of inputs.
* **Real-time Recognition:** Enable real-time or near-real-time recognition of handwritten text. Deep learning models, particularly those optimized for efficiency, can provide fast inference times suitable for real-time applications such as digital note-taking, form processing, or gesture recognition.
* **Multi-language Support:** Support recognition of handwritten text in multiple languages. Deep learning models can be trained on diverse datasets containing handwriting samples from different languages and writing systems, enabling them to recognize text written in various scripts and languages.
* **Adaptability:** Develop models that can adapt to individual users' handwriting styles over time, improving recognition accuracy with continued use. Deep learning models can be fine-tuned on user-specific data to personalize the recognition system for individual users.
* **Integration with Other Systems:** Integrate handwriting recognition systems with other applications or systems, such as document digitization, text input for mobile devices, automatic form processing, or intelligent virtual assistants.
* **User Experience Improvement:** Enhance the user experience by providing intuitive and accurate handwriting input methods for digital devices. Deep learning-based handwriting recognition can enable natural and convenient text input methods, particularly on touch-enabled devices like tablets and smartphones.
* **Accessibility:** Improve accessibility for individuals with disabilities, such as those with motor impairments who may find typing difficult but can still write or draw. Handwriting recognition systems can provide alternative input methods for such users, enabling them to interact with digital devices more effectively.**Top of Form**
  1. **Scope:**

Handwriting recognition systems cater to a diverse range of target groups across different industries and sectors. Here is the scope of these systems:

* **Administrative Professionals:**

Administrative professionals in various organizations benefit from handwriting recognition systems when dealing with paperwork, form processing, and data entry tasks. It helps streamline administrative processes and reduce manual workload.

* **Healthcare Providers:**

Healthcare professionals use handwriting recognition systems to digitize and manage patient records, prescriptions, and other medical documents. This improves the accuracy and accessibility of health information.

* **Educational Institutions:**

Teachers, students, and administrators in educational institutions use handwriting recognition for grading exams, converting handwritten notes to digital format, and facilitating interactive learning through digital whiteboards.

* **Financial Institutions:**

Banks and financial organizations employ handwriting recognition for processing handwritten checks, forms, and documents. It enhances the efficiency of transactions and reduces errors in financial records.

* **Postal Services:**

Postal workers and logistics professionals benefit from handwriting recognition in sorting and routing mail. It helps automate the handling of handwritten addresses on envelopes.

* **Researchers and Historians:**

Researchers and historians dealing with archival documents and historical manuscripts use handwriting recognition to convert handwritten texts into digital format for preservation and analysis.

* **Business Professionals:**

Business professionals, including executives, managers, and office staff, use handwriting recognition systems for tasks such as converting meeting notes, filling out forms, and processing handwritten documents.

* **Legal Professionals:**

Lawyers, paralegals, and legal administrators utilize handwriting recognition for processing legal documents, verifying signatures, and managing case-related paperwork.

* **Market Researchers:**

Professionals in market research use handwriting recognition for processing handwritten survey responses and extracting valuable data for analysis.

* **Government Agencies:**

Various government agencies utilize handwriting recognition for tasks such as digitizing records, processing forms, and improving the efficiency of administrative processes.

* **Language Translation Services:**

Individuals and businesses seeking to translate handwritten text benefit from handwriting recognition integrated into language translation services.

* **Note-Taking and Productivity Apps Users:**

Individuals who use note-taking apps and productivity tools leverage handwriting recognition to convert handwritten notes into digital text for better organization and collaboration.

* **Users of Interactive Whiteboards:**

Teachers, presenters, and collaborative teams use interactive whiteboards equipped with handwriting recognition for digital presentations, note-taking, and collaborative work.

* **Mobile Users:**

Handwriting recognition is integrated into some mobile devices and tablets, targeting users who prefer to input text using a stylus or their fingers.

* 1. **Application:**

Handwriting recognition systems, also known as Optical Character Recognition (OCR) systems, have a variety of applications across different industries. Here are some of the key applications:

* **Document Digitization:**

OCR systems are commonly used to convert handwritten documents into digital formats. This is particularly useful for historical documents, archives, and paper-based records that need to be preserved and accessed electronically.

* **Data Entry Automation:**

Handwriting recognition can automate data entry processes, reducing the need for manual input. This is valuable in scenarios where large volumes of handwritten forms, surveys, or documents need to be processed quickly and accurately.

* **Banking and Finance:**

In the financial sector, OCR technology is employed to read handwritten checks, forms, and other financial documents. This enhances the efficiency of processing transactions and reduces the chances of errors associated with manual data entry.

* **Healthcare Records:**

Handwriting recognition is used in healthcare for converting handwritten patient records, prescriptions, and medical forms into electronic formats. This helps in maintaining accurate and easily accessible electronic health records.

* **Postal Services:**

OCR is utilized by postal services to automate the sorting and routing of mail. It can recognize handwritten addresses on envelopes, speeding up the delivery process.

* **Form Processing:**

Many organizations use handwritten forms for various purposes. Handwriting recognition systems help in automatically extracting information from these forms, improving the efficiency of form processing.

* **Education:**

Handwriting recognition can be applied in educational settings for grading handwritten exams and assignments. It enables quick and standardized assessment while reducing the workload on educators.

* **Digital Note-Taking:**

OCR technology is integrated into some note-taking apps, allowing users to convert handwritten notes into digital text. This makes it easier to organize, search, and share notes.

* **Authentication and Signature Verification:**

Handwriting recognition systems are employed for signature verification in legal and financial transactions. They help ensure the authenticity of handwritten signatures on documents.

* **Interactive Whiteboards:**

In educational and business environments, interactive whiteboards equipped with handwriting recognition can convert handwritten notes and diagrams into digital format, making it easy to save and share.

* **Smart Forms and Surveys:**

OCR technology is used to process handwritten responses on forms and surveys. This is valuable in market research, customer feedback, and other data collection activities.

* **Language Translation:**

Handwriting recognition can be integrated into translation tools, allowing users to input handwritten text for translation into different languages.

* 1. **Limitations:**

While deep learning has significantly advanced handwriting recognition, it still faces several limitations:

* **Data Requirements:**

Deep learning models require large amounts of annotated data for training. Building high-quality datasets for handwriting recognition can be time-consuming and expensive, especially for languages with complex scripts or rare handwriting styles.

* **Performance Variability:**

The performance of deep learning models for handwriting recognition can vary depending on the quality and quantity of training data, as well as the specific characteristics of the handwriting styles encountered during training and inference.

* **Robustness to Variability:**

Despite efforts to make recognition systems robust to variability in handwriting styles, deep learning models may still struggle with highly unconventional or distorted handwriting, especially in cases of low-quality input or unusual writing habits.

* **Resource Intensiveness:**

Training deep learning models for handwriting recognition typically requires substantial computational resources, including powerful hardware and significant time for training. This can pose challenges for deploying and scaling recognition systems, particularly in resource-constrained environments.

* **Domain Adaptation:**

Deep learning models trained on one dataset or handwriting style may not generalize well to other domains or handwriting styles without additional adaptation or fine-tuning. Adapting models to new domains or languages can require additional labeled data and fine-tuning procedures.

* **Interpretability:**

Deep learning models often lack interpretability, making it challenging to understand why a particular recognition error occurred or to diagnose and address performance issues effectively. This lack of interpretability can hinder debugging and refinement of handwriting recognition systems.

* **Edge Cases and Ambiguities:**

Handwriting recognition systems may struggle with ambiguous or illegible input, leading to recognition errors, particularly in cases of poor handwriting quality, unusual symbols, or ambiguous characters.

* **Language and Script Diversity:**

Deep learning models may exhibit biases or limitations in recognizing handwriting from languages with complex scripts, rare characters, or diverse writing styles that are not well-represented in the training data.

* **User Privacy and Security:**

Handwriting recognition systems may raise privacy concerns if they require access to sensitive handwritten documents or personal data. Ensuring the privacy and security of user data is essential but can pose challenges, particularly in cloud-based or distributed recognition systems.

**CHAPTER - 2**

**LITERATURE SURVEY**

Using a dataset including 79,684 images and 306 distinct Ethiopic characters, the study employed a deep learning technique to detect historical Ethiopic manuscripts [1]. WiRITE makes use of M5 model design, ReLU activation functions, and 10 layers for image classification and Chinese characters in VGG-10, a condensed form of VGG-16 [2]. The CNN-RNN hybrid architecture consists of a STN, residual convolutional blocks, stacked BLSTM, and a label transcription linear layer pre-trained on the IIITHWS dataset [3]. The CNN and MDLSTM layers make up the recommended architecture, and the RETURNN framework is used to create the SoftMax layer, which handles input and output sequence alignment [4].

The Recurrent neural networks (RNNs) and a lexicon are integrated in the new segmented handwritten text recognition approach [5], which produces the best recognition results by training two models with bidirectional LSTM and CTC techniques. CNN and LSTM are used by deep learning neural networks for object identification, along with connectionist temporal classification. Smaller Washington CRNNs receive knowledge by transfer learning from larger databases, and they retrain their parameters once they have been trained [6]. Using linked component analysis, the design offers an SVM classifier-based method for optical character recognition that can discriminate different items in data without considering color [7].

A different study presents a novel, lower-parameter, lower-layer HTR architecture based on Gated-CNN that processes huge text images more effectively than current state-of-the-art HTR architectures [8]. The Handwritten Text Recognition model extracts features from raw images with a probability of 0.4865 by combining a 5-layer CNN, a 2-layer RNN, and a BLSTM network. Using Artificial Neural Networks (ANNs), local extrema, and Multilayer Perceptrons (MLPs) for accurate estimations, this study offers novel

methods for normalizing text picture size and eliminating slant and slope from handwritten text lines [10].

The architectures utilized for text line image recognition include CNN, FCN, MD-LSTM, and a combination of CNN and LSTM. To solve sequence alignment concerns, CTC loss, attention-based encoder-decoder design, and an FCN+LSTM network are used [11]. The trained model consists of CNN and RNN layers, with an LSTM for RNN implementation. Handwritten text photos are handled using TensorFlow for word recognition and OpenCV for image processing [12].

The object detection techniques are R-CNN, Fast RCNN, Faster R-CNN, and Mask R-YOLOv3. The YOLOv3 model for handwriting recognition employs lexicon-free, sequential character detection without author style [13]. CNN architectures such as VGG-19, RESNET-18, and RESNET-34 are used for word classification models, which achieve high accuracy in OCR applications using Hidden Markov Models and the ADAM Optimizer [14]. Weights are optimized using Stochastic Gradient Descent on 28 x 28 pictures from the MNIST database. For feature estimation, a neural network employs ten neurons, thick layers, ReLU activation, and MaxPooling [15].

The competition included 450-page images with line detection and transcription challenges, as well as ground truth (GT) and a baseline system based on hidden Markov and 2-gram models [18]. The research aims to develop a fast, accurate BCHWTR system for Indian bank cheques using SVM-based classification techniques, reducing transaction workload and time [19]. The C5 Hattem Manuscript, a 15th-century Middle Dutch document, was used in experiments to study line localization and extraction, evaluating segmentation methods using ICDAR20 13 Handwritten Segmentation Contest metrics [20].

This study presents a network architecture using convolutional recurrent neural networks for High-Tensor Regression, avoiding 2D-LSTM layers and

utilizing large datasets for limited labeled inputs. This study describes a system for recognizing handwritten and typewritten text from document images using hidden Markov models, enhancing OCR accuracy, and employing a variety of classifiers and generative modeling methods [21][22]. The research describes a method for classifying words and lines in online handwritten documents into six primary scripts using CrossPad1, with an accuracy of 87.1% on 5-fold cross-validation [24].

The architecture uses five convolutional blocks, the LeakyReLU activation function, bidirectional 1D-LSTM layers, and the RmsProp algorithm to train and update parameters using CTC loss gradients [25]. The paper describes a method for detecting unconstrained English handwritten text utilizing a vast vocabulary, which includes preprocessing, feature extraction, and recognition via a hidden Markov model [23]. The database is designed for research in handwritten text recognition, storing data on city names, state names, and ZIP codes. A less expensive alternative, text localization, and recognition, reduces computational costs and optimizes CTC loss using a CNN-biLSTM network [16][17].

Hidden Markov Models estimate stroke time evolution of handwritten text, based on character representations. A digital tablet at Universitat Politècnica de València recorded 2964 Spanish words [27]. The study uses segmentation-based recognition for cursive handwritten character recognition, estimating parameters using non-parametric methods and the Baum-Welch procedure for rapid prototyping [26]. The recognizer uses a sliding window approach to recognize handwritten texts, extracting feature vectors and modeling them using continuous density Hidden Markov Models [28].

**CHAPTER - 3**

**ANALYSIS**

* 1. **Existing System:**

In some baseline projects, handwriting recognition is accomplished through the implementation of a conventional neural network, yielding an impressive accuracy exceeding 90.3%. The employed algorithm demonstrates notable efficiency, delivering effective results for character recognition. Notably, the project attains optimal accuracy when applied to text with minimal noise interference. The pivotal factor influencing accuracy is the quality and size of the dataset; expanding the dataset has shown to positively correlate with improved accuracy.

Furthermore, the algorithm exhibits enhanced performance when cursive writing is avoided. This observation underscores the importance of preprocessing steps or specialized handling to ensure optimal results, particularly when dealing with diverse writing styles.

Continued refinement of the dataset, incorporating additional diverse and representative samples, remains a key consideration for ongoing improvement. Implementing data augmentation techniques, such as rotations and flips, can further enhance the variability of the dataset. Additionally, exploring alternative neural network architectures, optimization strategies, and regularization techniques may unveil opportunities for fine-tuning and improving the overall model performance.

As the project progresses, attention to validation strategies becomes crucial to accurately assess model performance and identify potential overfitting concerns. Future directions may involve the deployment of

the model in real-world scenarios or expanding its applicability to recognize characters in handwritten documents. Documenting findings and embracing a dynamic, iterative approach to model enhancement will contribute to sustained success in character classification.

# Design Constraints, Assumptions, and Disadvantages:

**Design Constraints:**

* + An uninterrupted internet connection is required at all times, else the system wouldn’t take the input as image and give the required output.
  + Doesn’t predict properly if the image is blurred.

**Design Assumptions:**

* + The predictions are limited to English language and doesn't support any other languages.
  + The input is assumed to have no background across the text.
  + The input given by the user is an image

**Disadvantages:**

1. Variability in Handwriting Styles:

* Different individuals have unique handwriting styles, making it challenging to create a system that can generalize across all these variations.
* Historical manuscripts, in particular, exhibit significant variability due to the diversity of languages, scripts, and regional writing conventions.

1. Limited Scope and Language Support:

* Handwriting recognition systems face limitations due to the vast array of manuscripts written in various languages and scripts.
* These systems require thorough review to ensure the accurate preservation of the original manuscript in electronic formats.

1. Poor Generalization:

* Traditional optical character recognition (OCR) tools, which have been around since the 1970s, struggle with handling handwritten text.
* Each person's handwriting is distinct, and traditional OCR tools cannot effectively recognize everyone's unique style.

1. Complexity of Handwriting Composition:

* The composition of handwritten text poses challenges. Writers and scribes have their own idiosyncrasies, making it difficult to create a onesizefitsall system.
* Variations in letter shapes, spacing, and stroke patterns further complicate the recognition process.

1. Loss of Spatial Information:

* During the recognition process, spatial information about rotation, location, scale, and other positional attributes can be lost.
* This occurs, for example, when pooling techniques are applied, leading to a reduction in spatial context.

1. The project gives best accuracy only when the text has less noise.
2. The project deliberately doesn’t yield good results for the text containing the cursive handwriting.

**CHAPTER - 4**

**DESIGN**

A hybrid architecture is designed using CNNs and RNNs which applies Alex Net and Bi-directional LSTM respectively.

The IAM dataset is imported from the official website. The data from it is unzipped to create a folder of words. Then a list of words is made from the folder. This list is split into train, test and validation sets.

**Unlocking the Secrets of Handwriting: The Power of the IAM Database -**

Regarding Handwritten Text Recognition (HWR), the IAM Database is groundbreaking. This vast repository, with samples from more than 657 writers touches on various subjects in different languages and writing styles making about 1,539 pages and 1,15,320 words in handwritten form. Deep Learning models for HWR, when provided with very precise ground truth marked on each page reliably yields a training environment that is perfect.

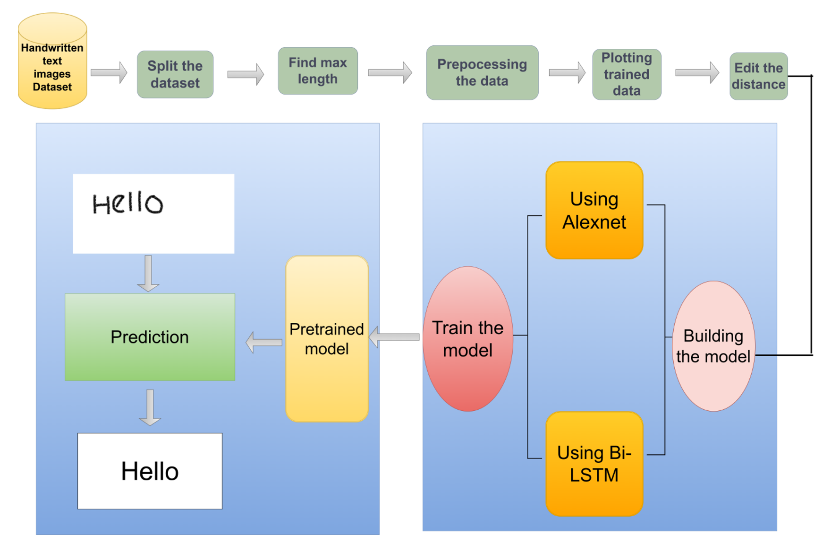
******

Fig: Flow of control of the designed model

Handwriting recognition using the AlexNet architecture coupled with Bidirectional Long Short-Term Memory (Bi-LSTM) and the Stochastic Gradient Descent with Momentum (SGDM) optimizer presents a powerful approach for accurate and efficient character identification. The fusion of these advanced techniques synergistically enhances the model's capability to decipher diverse and intricate handwritten scripts. The AlexNet architecture, renowned for its deep convolutional layers, enables the extraction of intricate features from input images, providing a strong foundation for recognizing intricate patterns in handwriting.

The incorporation of Bi-LSTM layers introduces a dynamic element to the model, allowing it to capture contextual dependencies and sequential information inherent in handwritten characters. The bidirectional nature of the LSTM layers enables the network to analyze the input sequence from both forward and backward directions, enhancing its ability to understand the nuanced structures present in handwriting.

The SGDM optimizer optimally fine-tunes the model parameters during the training process. Its combination of stochastic gradient descent with momentum facilitates efficient convergence and accelerates the learning process, ultimately contributing to the model's overall accuracy.

This hybrid architecture, encompassing AlexNet, Bi-LSTM, and SGDM, showcases a sophisticated and effective approach for handwriting recognition. The deep learning capabilities of AlexNet, coupled with the sequential understanding of Bi-LSTM and the optimized training process facilitated by SGDM, collectively result in a robust system capable of accurately deciphering diverse handwriting styles with notable efficiency.

**Advantages of Proposed System:**

1. The incorporation of AlexNet, Bi-LSTM, and SGDM in handwriting recognition collectively results in a sophisticated and powerful

system that excels in extracting features, understanding context, and optimizing training.

1. SGDM optimizer facilitates efficient and accelerated training of the model by combining stochastic gradient descent with momentum. This optimization technique helps prevent the model from getting stuck in local minima, enabling faster convergence and improved overall training efficiency.
   1. **Realtime model flow:**

The image is uploaded to Google Colab which is then processed by reading it. In the preprocessing step, the uploaded image undergoes the aforementioned preprocessing operations. Predictions are generated using the constructed CRNN model. During the testing phase, only the handwritten text is provided as input to the CTC function. CTC function decodes the digital text based on its learned patterns. As an output a graph is plotted for the uploaded image with the predicted text.

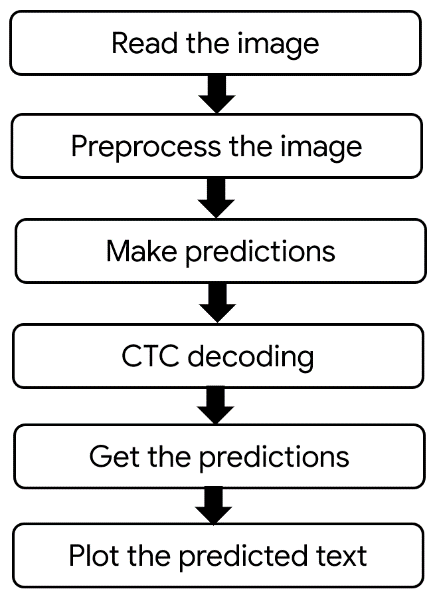
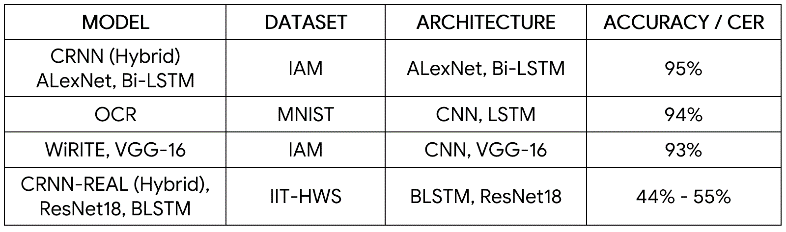


Fig: Flow of model in real time

* 1. **Comparision with other system:**



**CHAPTER - 5**

**IMPLEMENTATION**

The proposed architecture is a hybrid between CNNs and RNNs which uses Alex Net and Bi-directional LSTM respectively.

* **Alexnet:**

The proposed architecture follows key principles of AlexNet, utilizing convolutional, reshaping, dense, and recurrent layers.A (128, 32, 1) shaped input layer processes the input image in a Convolutional Recurrent Neural Network (CRNN). The ReLU activation function is employed throughout. The CRNN architecture is inspired by AlexNet and encompasses five Convolutional Layers. The first layer yields an output space dimensionality of 32, while the subsequent layers (second to fifth) have an output space dimensionality of 64. Following each convolutional layer, BatchNormalization is applied to normalize inputs. MaxPooling layers are additionally incorporated. A Reshape layer adjusts the input size. The model further includes two dense layers with ReLU activation, a third dense layer with a softmax function, and dropout layers to facilitate seamless processing from input to output. The Connectionist Temporal Classification (CTC) layer serves as the loss function. After convolutional processing, a reshaping operation prepares the features for further analysis.

* **Bi-LSTM:**

A Bidirectional LSTM (Bi-LSTM) is a recurrent neural network used primarily on natural language processing. Unlike standard LSTM, the input flows in both directions, and it’s capable of utilizing information from both sides, which makes it a powerful tool for modeling the sequential dependencies between words and phrases in both directions of the sequence. Bi-LSTM adds one more LSTM layer, which reverses the direction of information flow. In the proposed architecture two Bidirectional LSTM layers are used to capture sequential information and context from the features extracted by the CNN layers. Bidirectional LSTM processes the input sequence in both forward and backward directions, providing a more comprehensive understanding of the context. The output of the Bi-LSTM layers is passed through a Dense layer with softmax activation, producing the final output probabilities for each class.

* **Model Compilation:**

The model is compiled using Stochastic Gradient Descent (SGD) as the optimizer with a learning rate of 0.01 and momentum of 0.9. The model is configured to measure accuracy as a metric during training. Both models are compiled using the same optimizer SGDM. The summary of each model is printed to provide an overview of the model architecture.

* **Edit Distance:**

The similarity measure used is edit distance. At the end of epoch, the predictions are made for validation images. For every batch, labels are converted into sparse tensors. This computes Levenshtein distance.

**5.1 Dataset Used:**

The IAM dataset is imported from the official website. The data from it is unzipped to create a folder of words. Then a list of words is made from the folder. This list is split into train, test and validation sets.

The IAM Database with more than 1,150 pages of segmented and labeled text serves as the reference. Its detailed annotations which analyze words,

lines, and characters enable in-depth analysis and model refinement. The wide and varied source has helped the researchers in addressing hard HWR problems.

******

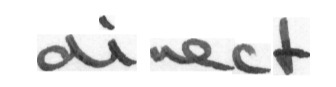
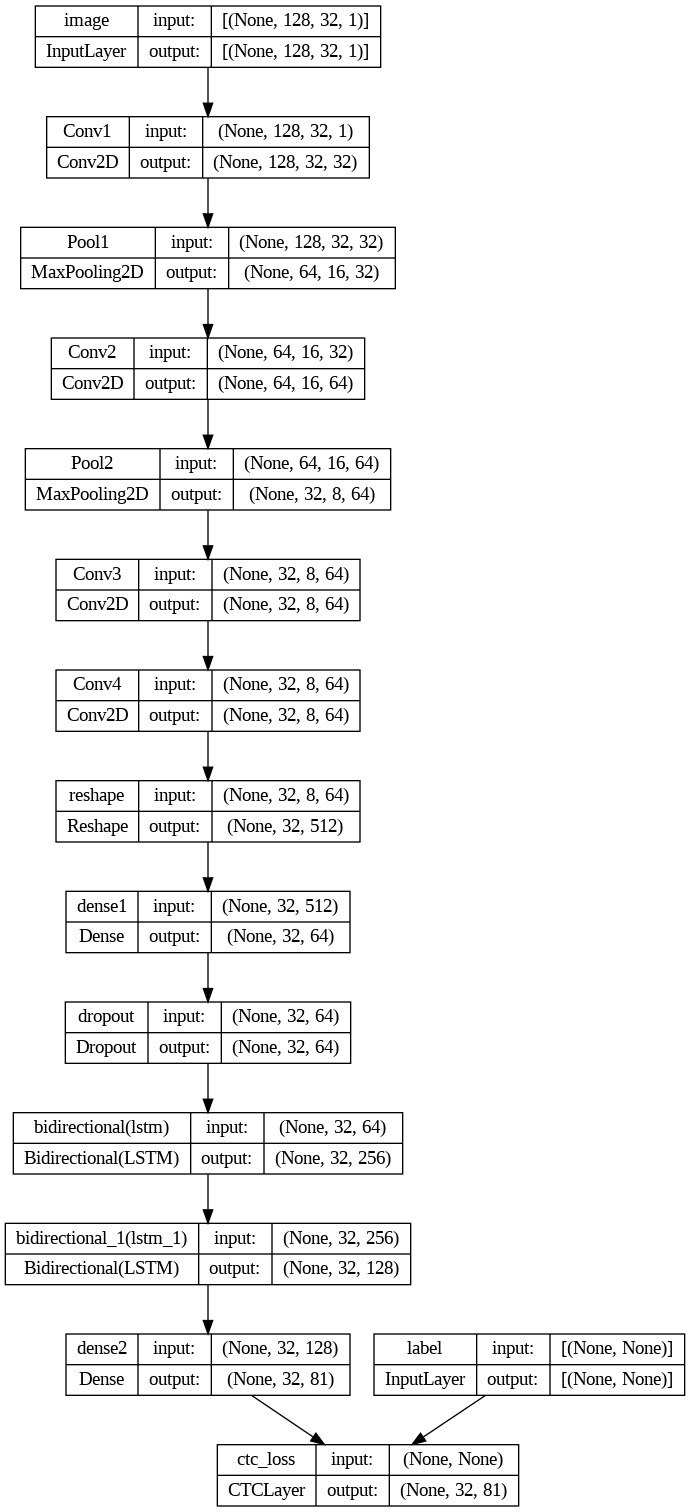
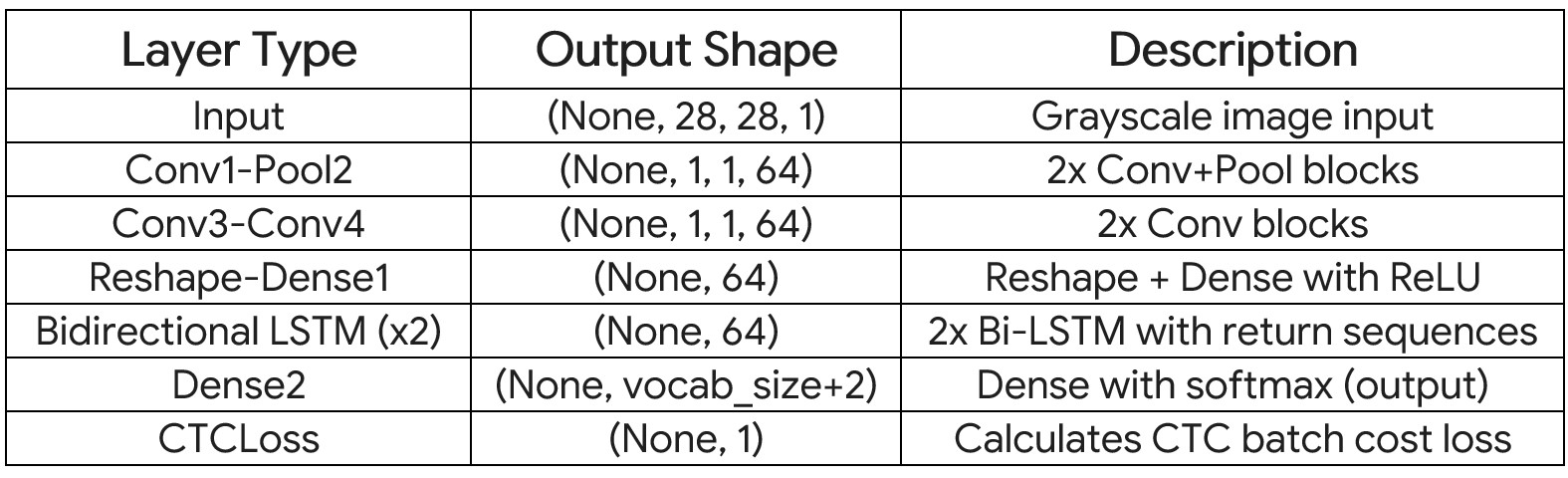
******

Fig: Images of words in the IAM dataset

**5.2 CRNN Model Architecture:**

****

Tabulation of layers along with specifications



**PERFORMANCE EVALATION METRICS**

**Character Error Rate (CER):**

Character Error Rate (CER) is a valuable metric used to evaluate the performance of a handwriting recognition system, especially when dealing with sequence-to-sequence tasks such as recognizing handwritten text.

The Character Error Rate measures the difference between the predicted sequence of characters and the ground truth sequence. CER shines when dealing with the inherent variability in how sequences align, making it a valuable tool for tasks like interpreting handwritten words.

CER**=**Edit Distance/Total No of Characters in Ground Truth Sequence

where

Edit Distance represents the minimum number of single-character edits (insertions, deletions, substitutions) required to transform the predicted sequence into the ground truth sequence.

**Word Error Rate (WER):**

Word Error Rate (WER) is another relevant metric to evaluate the performance of a handwriting recognition system, especially to recognize entire words or sequences of characters.

The Word Error Rate assesses the disparity between the predicted word sequence and the actual ground truth sequence. It takes into account insertions, deletions, and substitutions of entire words.

WER**=**Edit Distance/Total No of Words in Ground Truth Sequence

where

Edit Distance signifies the minimum number of word-level modifications (insertions, deletions, substitutions) necessary to transform the predicted sequence into the ground truth sequence.

**Sentence Error Rate (SER):**

Although the Character Error Rate (CER) and Word Error Rate (WER) are frequently employed metrics to assess the performance of handwriting recognition systems, yet a direct "Sentence Error Rate" may not be standardized. It assess the performance of your model at the sentence level.

SER can be defined as the ratio of the total number of errors (insertions, deletions, substitutions) in the predicted sentences to the total number of sentences in the ground truth.

SER**=**Total No of Sentences in Ground Truth/Total Sentence Edit Distance​

where

Total Sentence Edit Distance is the sum of the edit distances between each predicted sentence and its corresponding ground truth sentence.

**EXECUTION PROCEDURE**

The project is executed in Google Colab. The following guide gives a detailed description of the procedure for executing python code in Google Colab.

**Table of Contents**

****

1. GOOGLE COLAB – INTRODUCTION

2. GOOGLE COLAB – WHAT IS GOOGLE COLAB?

3. GOOGLE COLAB – YOUR FIRST COLAB NOTEBOOK

4. GOOGLE COLAB – SAVING YOUR WORK

5. GOOGLE COLAB – SHARING NOTEBOOK

6. GOOGLE COLAB – EXECUTING EXTERNAL PYTHON FILES

7. GOOGLE COLAB – GRAPHICAL OUTPUTS

8. GOOGLE COLAB – INSTALLING ML LIBRARIES

9. GOOGLE COLAB – USING FREE GPU

10. GOOGLE COLAB – CONCLUSION

**1. Google Colab – Introduction**

Google is quite aggressive in AI research. Over many years, Google developed AI framework called **TensorFlow** and a development tool called **Colaboratory**. Today TensorFlow is open-sourced and since 2017, Google made Colaboratory free for public use. Colaboratory is now known as Google Colab or simply **Colab**.

Another attractive feature that Google offers to the developers is the use of GPU. Colab supports GPU and it is totally free. The reasons for making it free for public could be to make its software a standard in the academics for teaching machine learning and data science. It may also have a long term perspective of building a customer base for Google Cloud APIs which are sold per-use basis.

**Why Use Google Colab?**

You might wonder, why use Google Colab for machine learning and deep learning? Well, there are several compelling reasons:

* **Accessibility**

Google Colab provides free access to powerful computational resources like GPUs and TPUs (tensor processing units). These resources, typically reserved for high-end, expensive hardware, are key to training deep learning models efficiently.

* **Ease of Use**

With Google Colab, there's no need for complex setup procedures. Everything runs in your browser, meaning you can focus on writing and executing your code rather than dealing with installation issues. This cloud-based nature also ensures that you can work from anywhere, on any device that has internet access.

* **Collaboration and Sharing**

Google Colab inherits Google Docs' collaborative features. You can share your notebooks, have others comment on them, and even edit them in real-time - an excellent feature for team projects or teaching.

* **Integration with Google Drive**

Google Colab automatically saves your work in Google Drive. This auto-save feature is a lifesaver, making sure you never lose your work, even if you forget to save manually.

* **Pre-installed Libraries**

Google Colab comes pre-installed with popular Python libraries like TensorFlow, PyTorch, and Keras. This convenience allows you to jump right into coding without worrying about installing and updating libraries.

In a nutshell, Google Colab is an accessible, user-friendly platform that alleviates much of the typical setup pains associated with machine learning and deep learning, leaving you free to focus on what matters most - building and refining your models.



**2. Google Colab – What is Google Colab**

If you have used **Jupyter** notebook previously, you would quickly learn to use Google Colab. To be precise, Colab is a free Jupyter notebook environment that runs entirely in the cloud. Most importantly, it does not require a setup and the notebooks that you create can be simultaneously edited by your team members - just the way you edit documents in Google Docs. Colab supports many popular machine learning libraries which can be easily loaded in your notebook.

**What Colab Offers You?**

****

As a programmer, you can perform the following using Google Colab.

* Write and execute code in Python
* Document your code that supports mathematical equations
* Create/Upload/Share notebooks
* Import/Save notebooks from/to Google Drive
* Import/Publish notebooks from GitHub
* Import external datasets e.g. from Kaggle
* Integrate PyTorch, TensorFlow, Keras, OpenCV
* Free Cloud service with free GPU

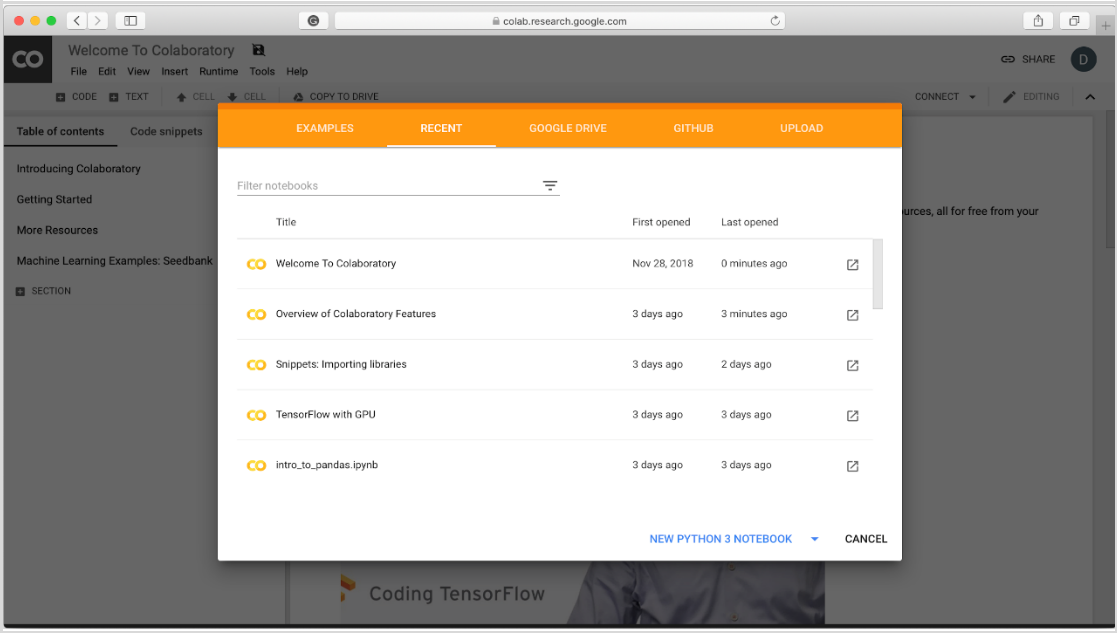
**3. Google Colab – Your First Colab Notebook**

In this chapter, you will create and execute your first trivial notebook. Follow the steps that have been given wherever needed.

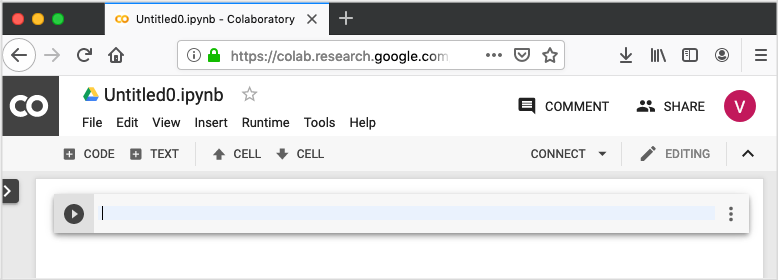
**Note**: As Colab implicitly uses Google Drive for storing your notebooks, ensure that you are logged in to your Google Drive account before proceeding further.

**Step 1:** Open the following URL in your browser: [https://colab.research.google.com](https://colab.research.google.com/)

Your browser would display the following screen (assuming that you are logged into your Google Drive):



**Step 2:** Click on the **NEW PYTHON 3 NOTEBOOK** link at the bottom of the screen. A new notebook would open up as shown in the screen below.

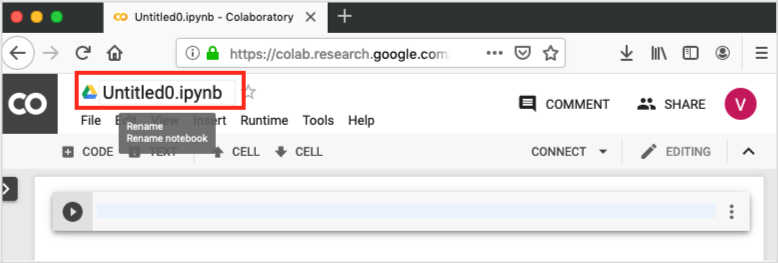


Google Colab

**Setting Notebook Name**

****

By default, the notebook uses the naming convention **UntitledXX.ipynb**. To rename the notebook, click on this name and type in the desired name in the edit box as shown here:



We will call this notebook as **MyFirstColabNotebook**. So type in this name in the edit box and hit ENTER. The notebook will acquire the name that you have given now.

**Entering Code**

****

You will now enter a trivial Python code in the code window & execute it.

Enter the following two Python statements in the code window:



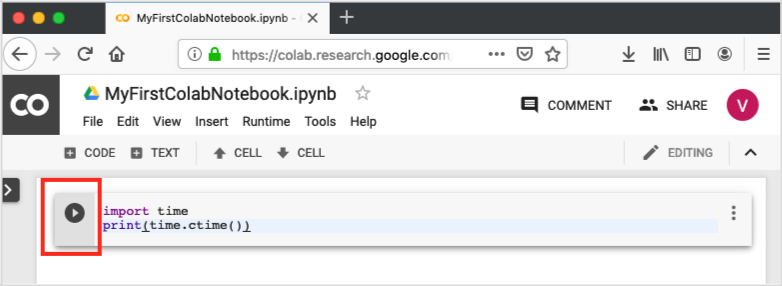
import time

print(time.ctime())

**Executing Code**

****

To execute the code, click on the arrow on the left side of the code window.



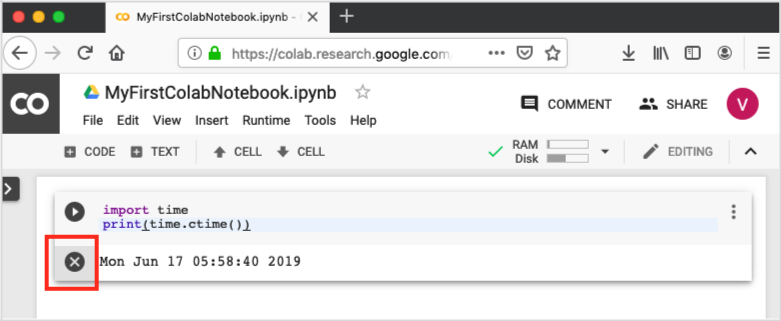
After a while, you will see the output underneath the code window, as shown here:



Mon Jun 17 05:58:40 2022

Google Colab

You can clear the output anytime by clicking the icon on the left side of the output display.



**Adding Code Cells**

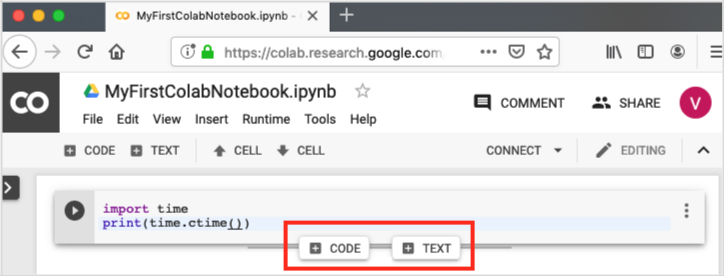
****

To add more code to your notebook, select the following **menu** options:



Insert / Code Cell

Alternatively, just hover the mouse at the bottom center of the Code cell. When the **CODE** and **TEXT** buttons appear, click on the CODE to add a new cell. This is shown in the screenshot below:



A new code cell will be added underneath the current cell. Add the following two statements in the newly created code window:



time.sleep(5)

print (time.ctime())

Now, if you run this cell, you will see the following output:



Mon Jun 17 04:50:27 2023

Google Colab

**Run All**

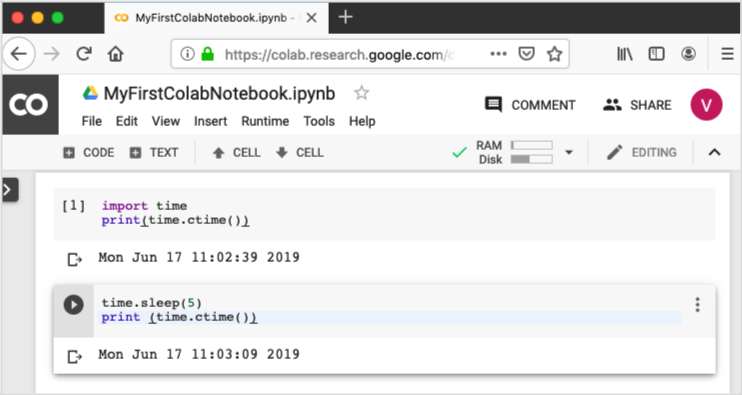
****

To run the entire code in your notebook without an interruption, execute the following menu options:



Runtime / Reset and run all…

It will give you the output as shown below:



Note that the time difference between the two outputs is now exactly 5 seconds.

The above action can also be initiated by executing the following two menu options:



Runtime / Restart runtime…

or



Runtime / Restart all runtimes…

Followed by



Runtime / Run all

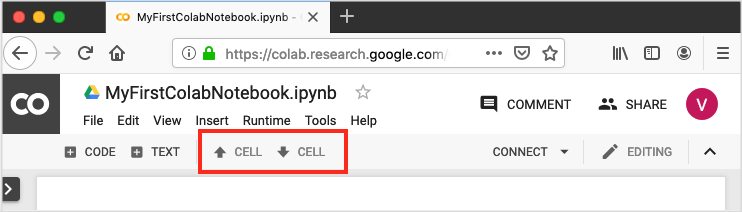
Study the different menu options under the **Runtime** menu to get yourself acquainted with the various options available to you for executing the notebook.

Google Colab

**Changing Cell Order**

****

When your notebook contains a large number of code cells, you may come across situations where you would like to change the order of execution of these cells. You can do so by selecting the cell that you want to move and clicking the **UP CELL** or **DOWN CELL** buttons shown in the following screenshot:

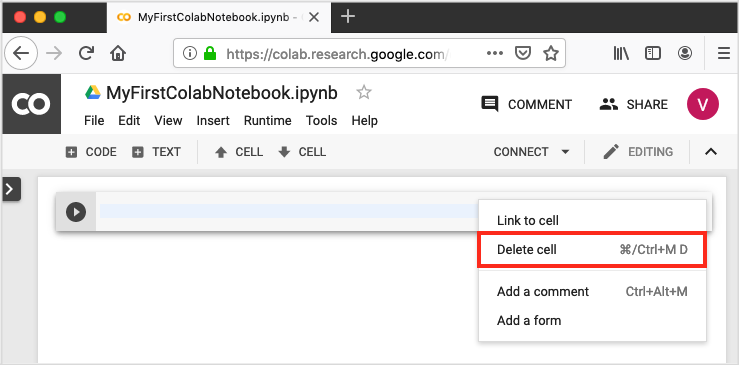


You may click the buttons multiple times to move the cell for more than a single position.

**Deleting Cell**

****

During the development of your project, you may have introduced a few now-unwanted cells in your notebook. You can remove such cells from your project easily with a single click. Click on the vertical-dotted icon at the top right corner of your code cell.



Click on the **Delete cell** option and the current cell will be deleted.

Now, as you have learned how to run a trivial notebook, let us explore the other capabilities of Colab.

**4. Google Colab – Saving Your Work**

Colab allows you to save your work to Google Drive or even directly to your GitHub repository.

**Saving to Google Drive**

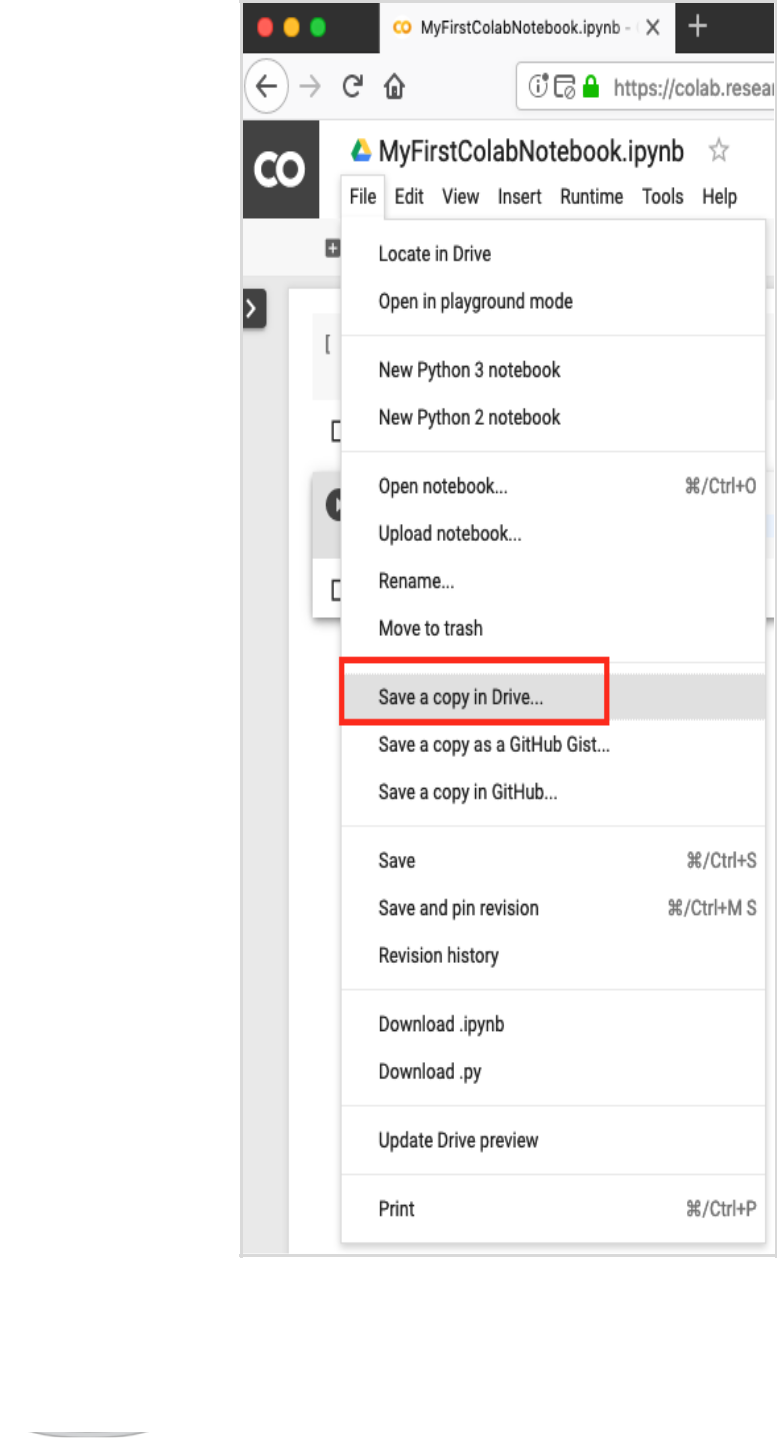
****

Colab allows you to save your work to your Google Drive. To save your notebook, select the following menu options:



File / Save a copy in Drive…

You will see the following screen:



Google Colab

The action will create a copy of your notebook and save it to your drive. Later on you may rename the copy to your choice of name.

**Saving to GitHub**

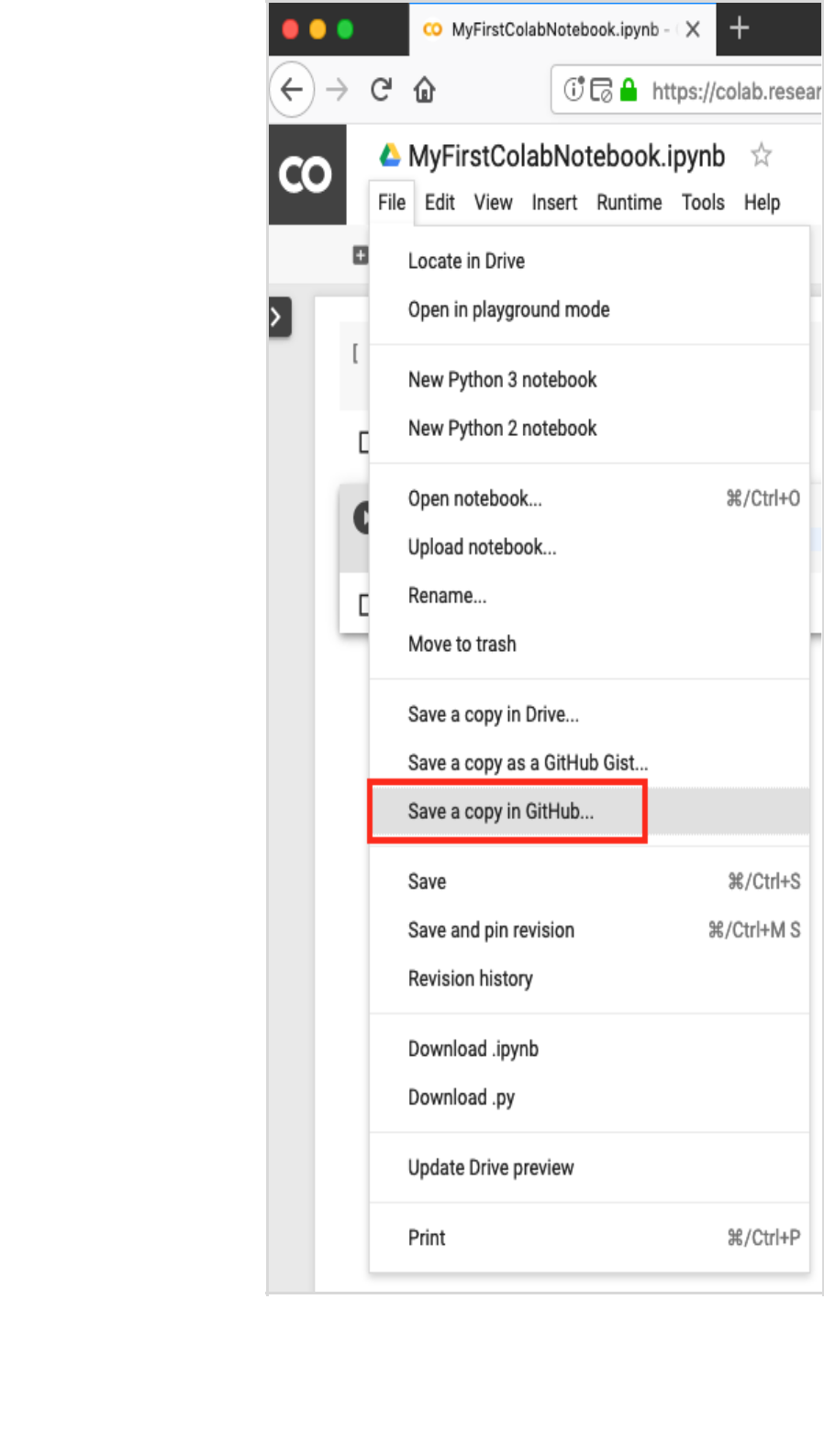
****

You may also save your work to your GitHub repository by selecting the following menu options:



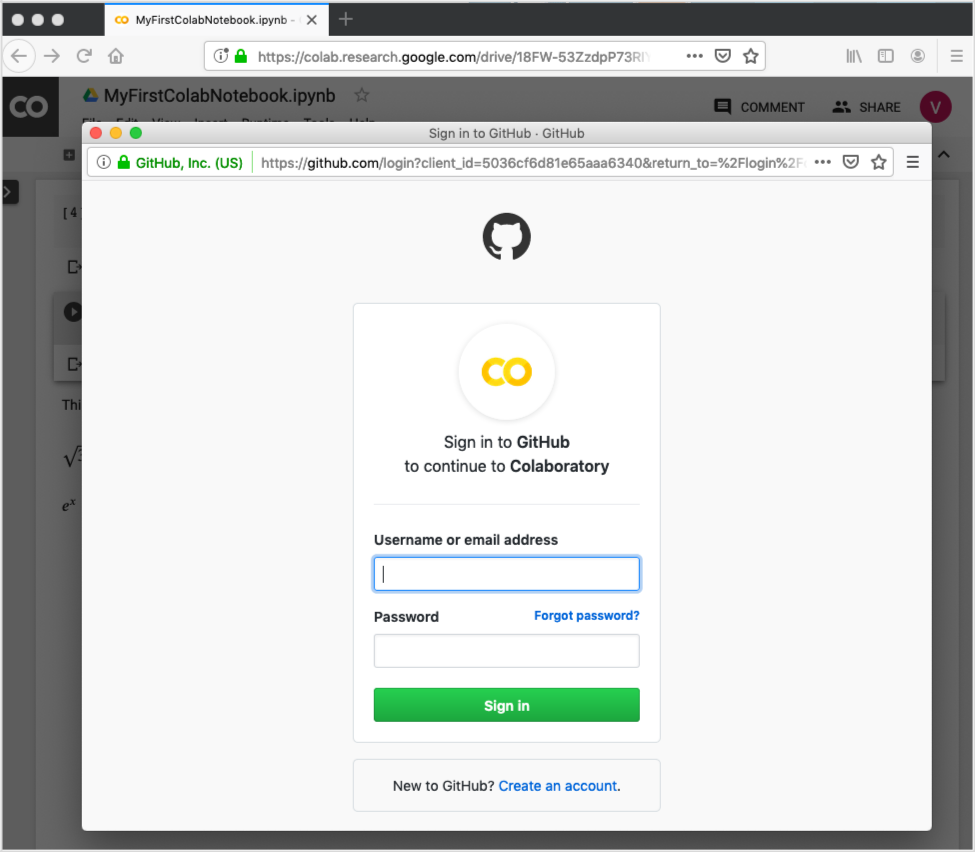
File / Save a copy in GitHub...

The menu selection is shown in the following screenshot for your quick reference:

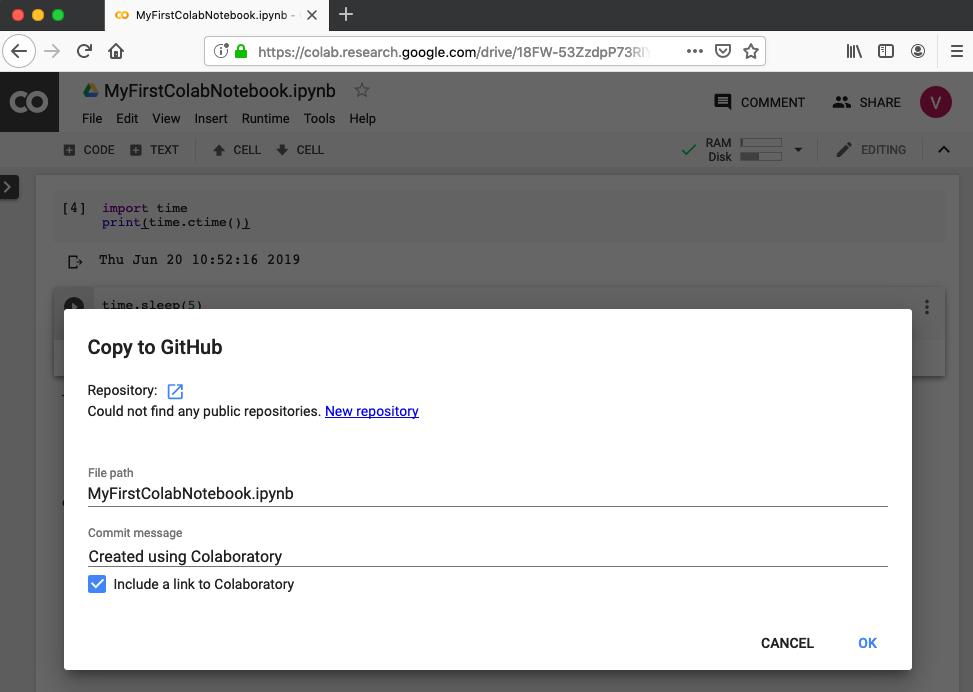


Google Colab

You will have to wait until you see the login screen to GitHub.



Now, enter your credentials. If you do not have a repository, create a new one and save your project as shown in the screenshot below:



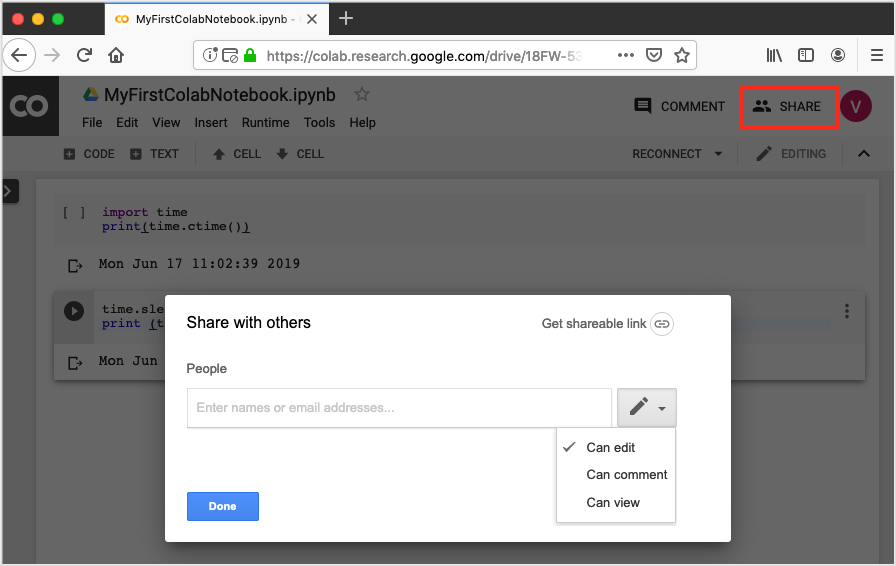
In the next chapter, we will learn how to share your work with others.

**5. Google Colab – Sharing Notebook**

To share the notebook that you have created with other co-developers, you may share the copy that you have made in your Google Drive.

To publish the notebook to general audience, you may share it from your GitHub repository.

There is one more way to share your work and that is by clicking on the **SHARE** link at the top right hand corner of your Colab notebook. This will open the share box as shown here:



You may enter the email IDs of people with whom you would like to share the current document. You can set the kind of access by selecting from the three options shown in the above screen.

Click on the **Get shareable link** option to get the URL of your notebook. You will find options for whom to share as follows:

* Specified group of people
* Colleagues in your organization
* Anyone with the link
* All public on the web

Now. you know how to create/execute/save/share a notebook. In the Code cell, we used Python so far. The code cell can also be used for invoking system commands. This is explained next.

**6. Google Colab – Executing External Python Files**

Suppose, you already have some Python code developed that is stored in your Google Drive. Now, you will like to load this code in Colab for further modifications. In this chapter, we will see how to load and run the code stored in your Google Drive.

**Mounting Drive**

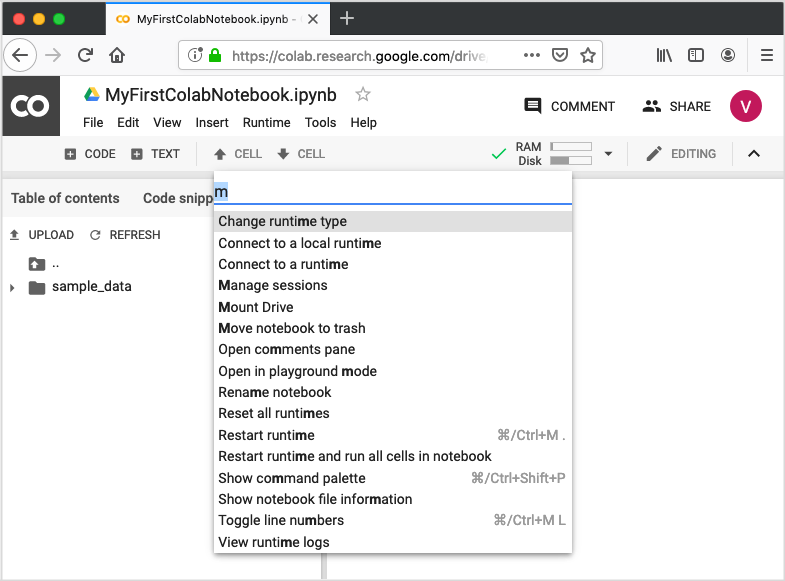
****

First, you need to mount your Google Drive in Colab. Select the following menu options:

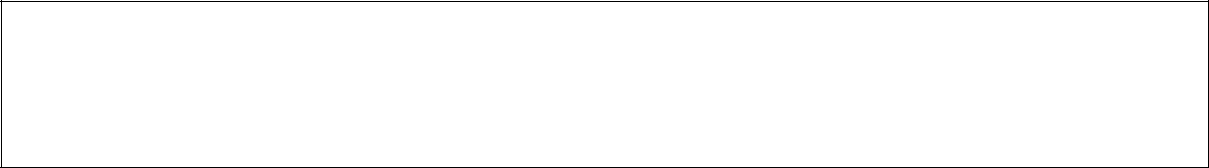


Tools / Command palette

You will see the list of commands as shown in this screenshot:



Type a few letters like “m” in the search box to locate the mount command. Select **Mount Drive** command from the list. The following code would be inserted in your Code cell.



* Run this cell to mount your Google Drive. from google.colab import drive drive.mount('/content/drive')



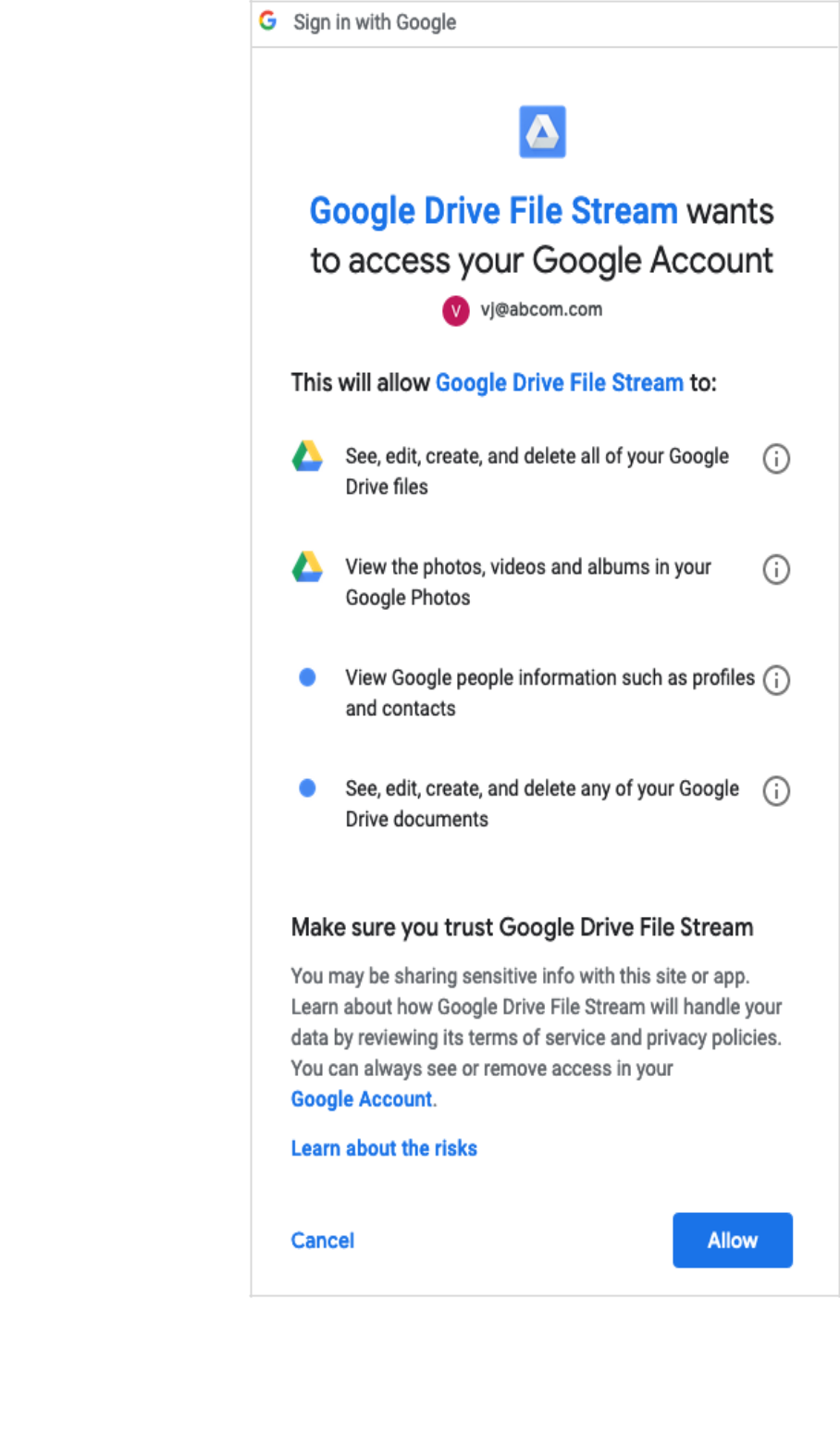
Google Colab

If you run this code, you will be asked to enter the authentication code. The corresponding screen looks as shown below:



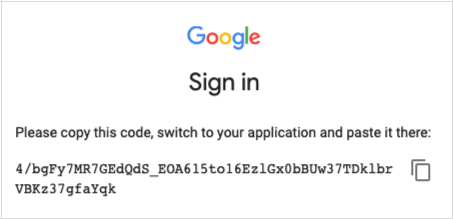
Open the above URL in your browser. You will be asked to login to your Google account.

Now, you will see the following screen:



Google Colab

If you grant the permissions, you will receive your code as follows:



Cut-n-paste this code in the Code cell and hit ENTER. After a while, the drive will be mounted as seen in the screenshot below:



Now, you are ready to use the contents of your drive in Colab.

**Listing Drive Contents**

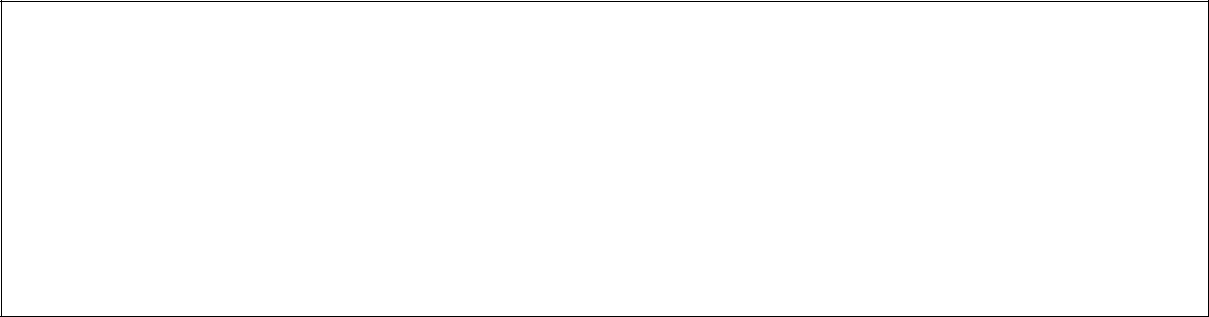
****

You can list the contents of the drive using the **ls** command as follows:



!ls "/content/drive/My Drive/Colab Notebooks"

This command will list the contents of your Colab Notebooks folder. The sample output of my drive contents are shown here:



Greeting.ipynb

hello.py

LogisticRegressionCensusData.ipynb

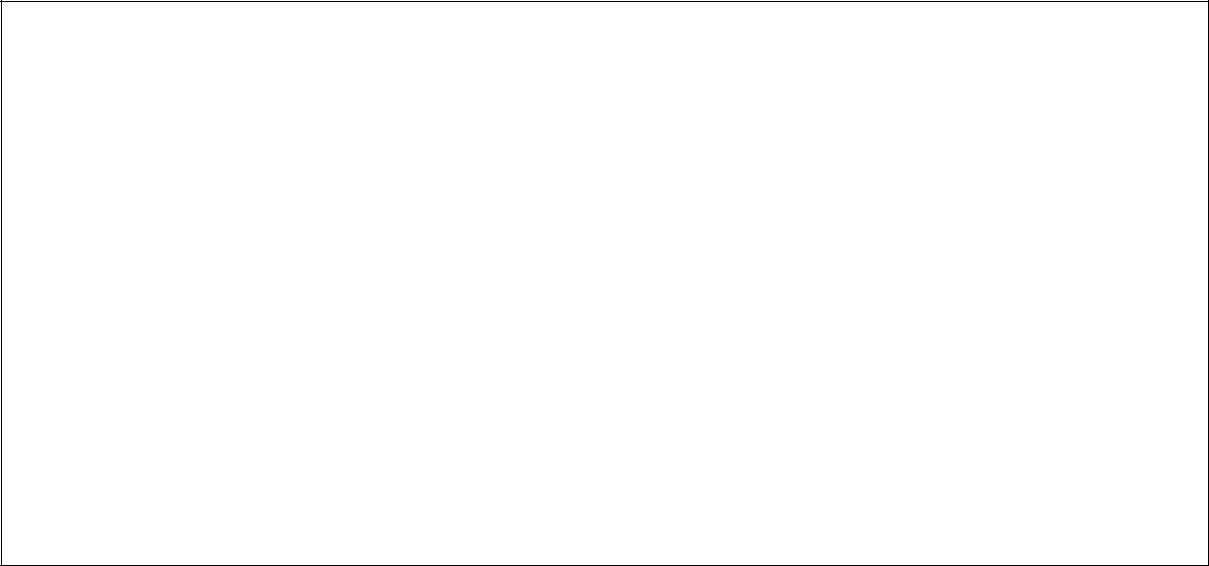
LogisticRegressionDigitalOcean.ipynb

MyFirstColabNotebook.ipynb

SamplePlot.ipynb

**7. Google Colab – Graphical Outputs**

Colab also supports rich outputs such as charts. Type in the following code in the Code cell.



import numpy as np

from matplotlib import pyplot as plt

y = np.random.randn(100)

x = [x for x in range(len(y))]

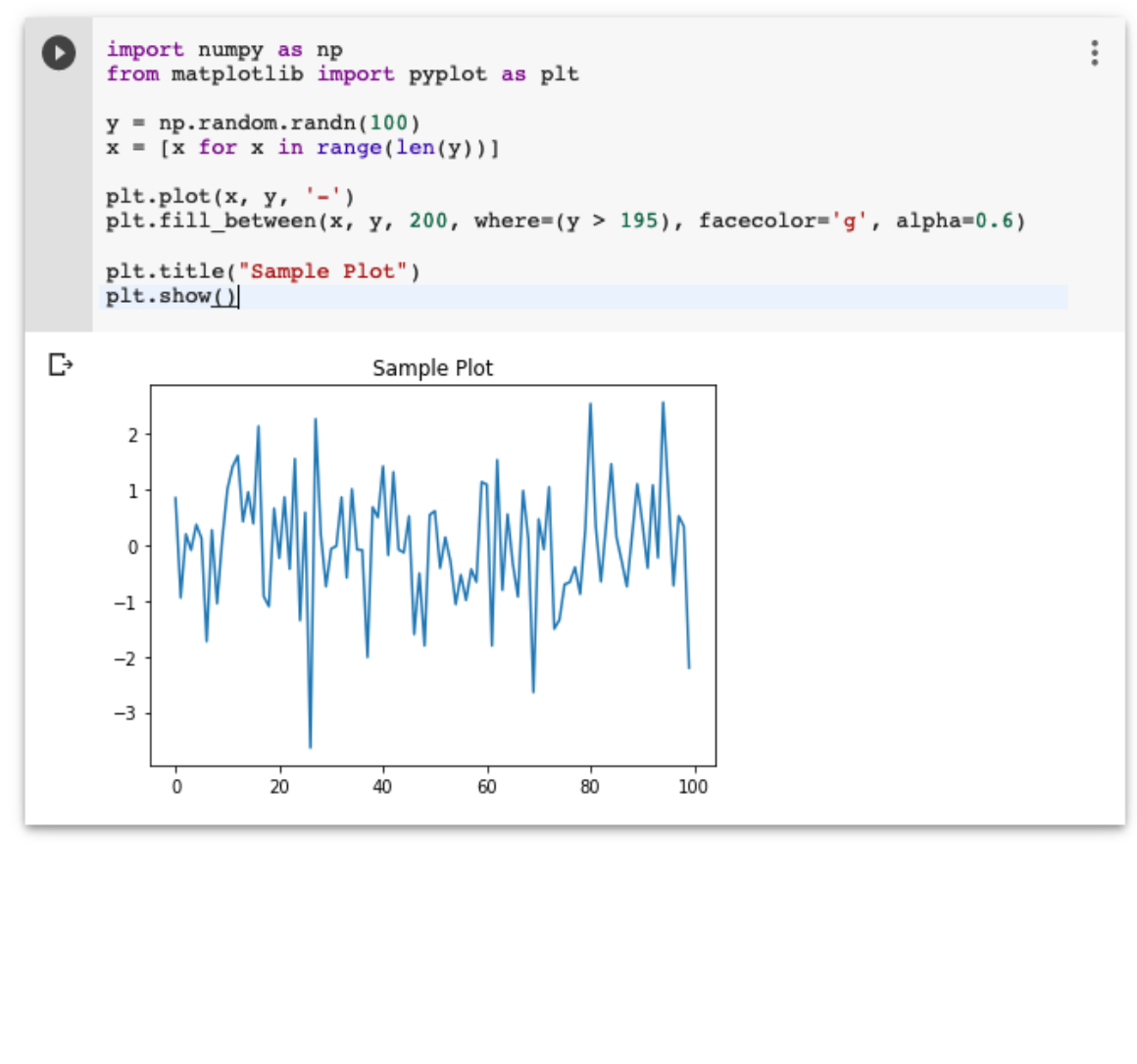
plt.plot(x, y, '-')

plt.fill\_between(x, y, 200, where=(y > 195), facecolor='g', alpha=0.6)

plt.title("Sample Plot")

plt.show()

Now, if you run the code, you will see the following output:



Note that the graphical output is shown in the output section of the Code cell. Likewise, you will be able to create and display several types of charts throughout your program code.

Now, as you have got familiar with the basics of Colab, let us move on to the features in Colab that makes your Python code development easier.

**8. Google Colab – Installing ML Libraries**

Colab supports most of machine learning libraries available in the market. In this chapter, let us take a quick overview of how to install these libraries in your Colab notebook.

To install a library, you can use either of these options:



!pip install

**Keras**

****

Keras, written in Python, runs on top of TensorFlow, CNTK, or Theano. It enables easy and fast prototyping of neural network applications. It supports both convolutional networks (CNN) and recurrent networks, and also their combinations. It seamlessly supports GPU.

To install Keras, use the following command:



!pip install -q keras

**PyTorch**

****

PyTorch is ideal for developing deep learning applications. It is an optimized tensor library and is GPU enabled. To install PyTorch, use the following command:



!pip3 install torch torchvision

**MxNet**

****

Apache MxNet is another flexible and efficient library for deep learning. To install MxNet execute the following commands:



!apt install libnvrtc8.0

!pip install mxnet-cu80

**OpenCV**

****

OpenCV is an open source computer vision library for developing machine learning applications. It has more than 2500 optimized algorithms which support several applications such as recognizing faces, identifying objects, tracking moving objects, stitching images, and so on.

To install OpenCV use the following command:



!apt-get -qq install -y libsm6 libxext6 && pip install -q -U opencv-python

**XGBoost**

****

XGBoost is a distributed gradient boosting library that runs on major distributed environments such as Hadoop. It is highly efficient, flexible and portable. It implements ML algorithms under the Gradient Boosting framework.

To install XGBoost, use the following command:



!pip install -q xgboost==0.4a30

**GraphViz**

****

Graphviz is an open source software for graph visualizations. It is used for visualization in networking, bioinformatics, database design, and for that matter in many domains where a visual interface of the data is desired.

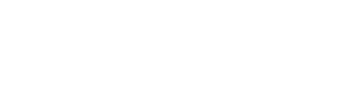
To install GraphViz, use the following command:



!apt-get -qq install -y graphviz && pip install -q pydot

By this time, you have learned to create Jupyter notebooks containing popular machine learning libraries. You are now ready to develop your machine learning models. This requires high processing power. Colab provides free GPU for your notebooks.

In the next chapter, we will learn how to enable GPU for your notebook.



**9. Google Colab – Using Free GPU**

Google provides the use of free GPU for your Colab notebooks.

**Enabling GPU**

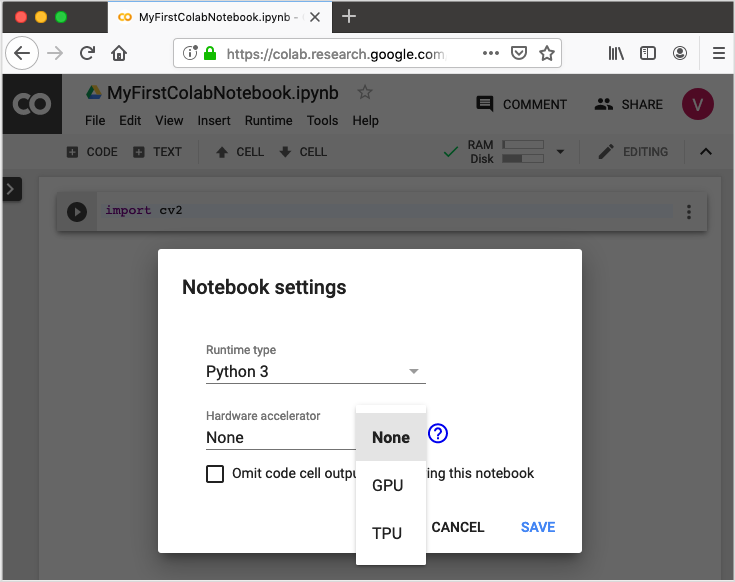
****

To enable GPU in your notebook, select the following menu options:



Runtime / Change runtime type

You will see the following screen as the output:



Select **GPU** and your notebook would use the free GPU provided in the cloud during processing. To get the feel of GPU processing, try running the sample application from **IAM** tutorial that you cloned earlier.



!python3 "/content/drive/My Drive/app/mnist\_cnn.py"

Try running the same Python file without the GPU enabled. Did you notice the difference in speed of execution?

Google Colab

**Testing for GPU**

****

You can easily check if the GPU is enabled by executing the following code:



import tensorflow as tf

tf.test.gpu\_device\_name()

If the GPU is enabled, it will give the following output:



'/device:GPU:0'

**Listing Devices**

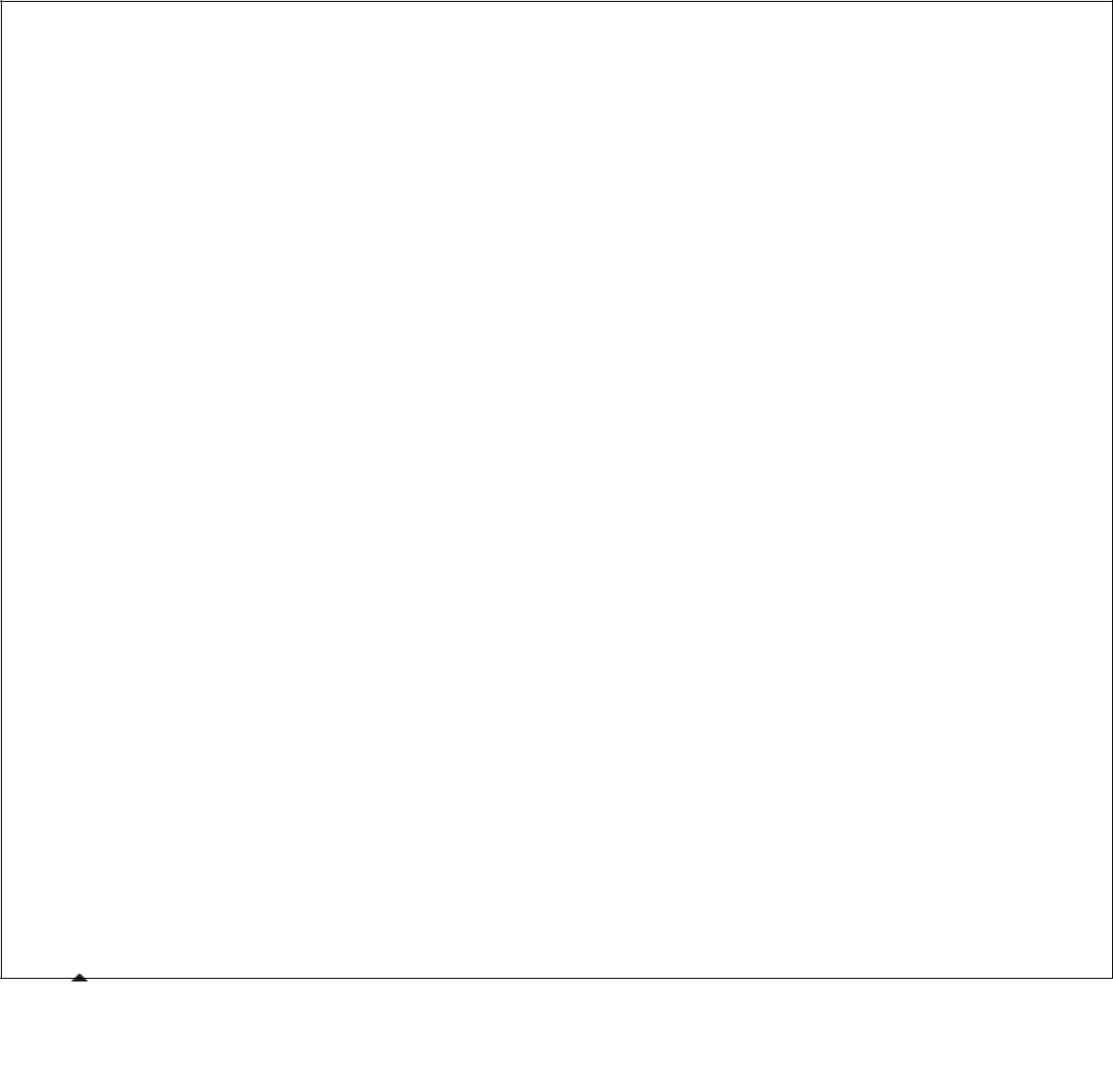
****

If you are curious to know the devices used during the execution of your notebook in the cloud, try the following code:



from tensorflow.python.client import device\_lib device\_lib.list\_local\_devices()

You will see the output as follows:



[name: "/device:CPU:0"

device\_type: "CPU"

memory\_limit: 268435456

locality {

}

incarnation: 1734904979049303143, name: "/device:XLA\_CPU:0"

device\_type: "XLA\_CPU"

memory\_limit: 17179869184

locality {

}

incarnation: 16069148927281628039

physical\_device\_desc: "device: XLA\_CPU device", name: "/device:XLA\_GPU:0"

device\_type: "XLA\_GPU"

memory\_limit: 17179869184

locality {

}

incarnation: 16623465188569787091

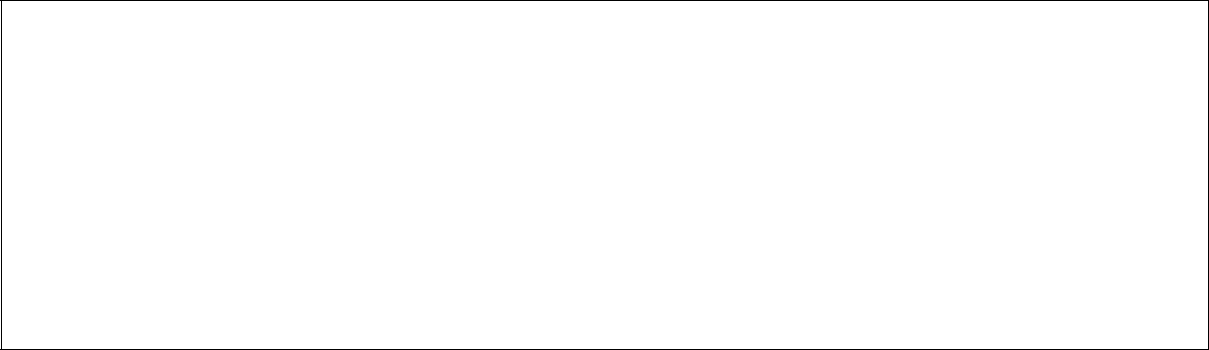
physical\_device\_desc: "device: XLA\_GPU device", name: "/device:GPU:0"

device\_type: "GPU"

memory\_limit: 14062547764

locality {

Google Colab



bus\_id: 1

links {

}

}

incarnation: 6674128802944374158

physical\_device\_desc: "device: 0, name: Tesla T4, pci bus id: 0000:00:04.0,

compute capability: 7.5"]

**Checking RAM**

****

To see the memory resources available for your process, type the following command:



!cat /proc/meminfo

You will see the following output:

|  |  |  |
| --- | --- | --- |
| MemTotal: | 13335276 | kB |
| MemFree: | 7322964 | kB |
| MemAvailable: | 10519168 | kB |
| Buffers: | 95732 | kB |
| Cached: | 2787632 | kB |
| SwapCached: | 0 | kB |
| Active: | 2433984 | kB |
| Inactive: | 3060124 | kB |
| Active(anon): | 2101704 | kB |
| Inactive(anon): | 22880 | kB |
| Active(file): | 332280 | kB |
| Inactive(file): | 3037244 | kB |
| Unevictable: | 0 | kB |
| Mlocked: | 0 | kB |
| SwapTotal: | 0 | kB |
| SwapFree: | 0 | kB |
| Dirty: | 412 | kB |
| Writeback: | 0 | kB |
| AnonPages: | 2610780 | kB |
| Mapped: | 838200 | kB |
| Shmem: | 23436 | kB |
| Slab: | 183240 | kB |
| SReclaimable: | 135324 | kB |
| SUnreclaim: | 47916 | kB |
|  |  |  |

Google Colab



|  |  |  |  |
| --- | --- | --- | --- |
| KernelStack: | 4992 | kB |  |
| PageTables: | 13600 | kB |  |
| NFS\_Unstable: | 0 | kB |  |
| Bounce: | 0 | kB |  |
| WritebackTmp: | 0 | kB |  |
| CommitLimit: | 6667636 | kB |  |
| Committed\_AS: | 4801380 | kB |  |
| VmallocTotal: | 34359738367 | | kB |
| VmallocUsed: | 0 | kB |  |
| VmallocChunk: | 0 | kB |  |
| AnonHugePages: | 0 | kB |  |
| ShmemHugePages: | 0 | kB |  |
| ShmemPmdMapped: | 0 | kB |  |
| HugePages\_Total: | 0 |  |  |
| HugePages\_Free: | 0 |  |  |
| HugePages\_Rsvd: | 0 |  |  |
| HugePages\_Surp: | 0 |  |  |
| Hugepagesize: | 2048 | kB |  |
| DirectMap4k: | 303092 | kB |  |
| DirectMap2M: | 5988352 | kB |  |
| DirectMap1G: | 9437184 | kB |  |

You are now all set for the development of machine learning models in Python using Google Colab.



**10. Google Colab – Conclusion**

Google Colab is a powerful platform for learning and quickly developing machine learning models in Python. It is based on Jupyter notebook and supports collaborative development. The team members can share and concurrently edit the notebooks, even remotely. The notebooks can also be published on GitHub and shared with the general public. Colab supports many popular ML libraries such as PyTorch, TensorFlow, Keras and OpenCV. The restriction as of today is that it does not support R or Scala yet. There is also a limitation to sessions and size. Considering the benefits, these are small sacrifices one needs to make.

**TESTING & TRAINING**

The model undergoes a training on the subset of words from IAM dataset comprising of 86810 total training samples.

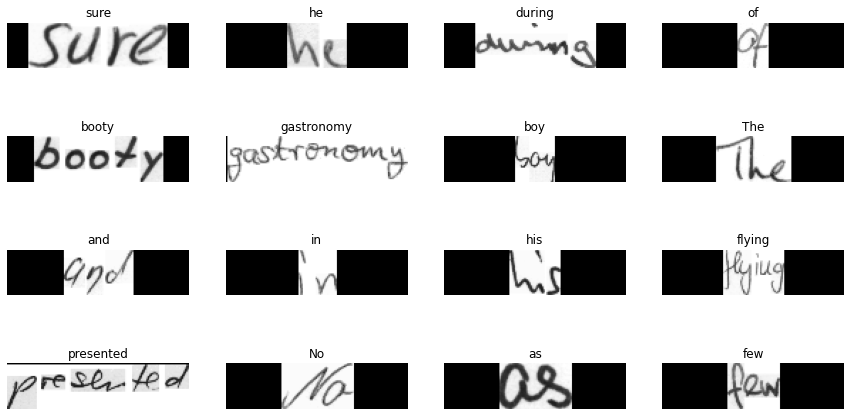
******

Fig: Training set predictions for CRNN

Total validation samples: 9646

The model is trained on various subsets of words extracted from the IAM dataset, which encompasses a total of around 4823 test samples.

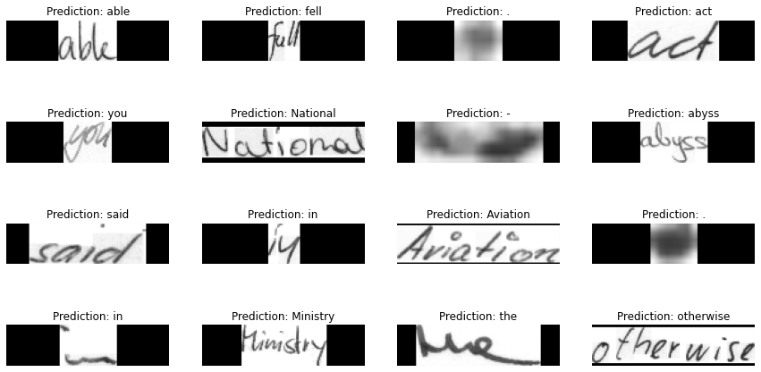


Fig: Test set predictions for CRNN

**CHAPTER - 7**

**RESULTS & PERFORMANCE EVALUATION**

The CRNN model undergoes training for 240 epochs, whereas the AlexNet model undergoes training for 25 epochs. The decision to limit AlexNet to 25 epochs is driven by the observation that exceeding this threshold may result in overfitting, as evident from the model's loss graph.

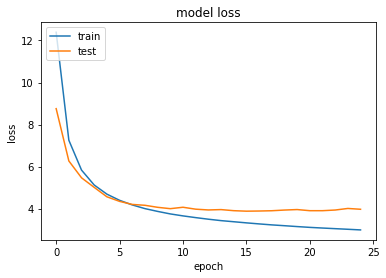
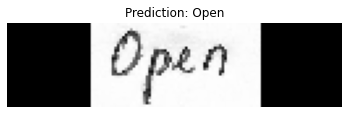
******

Fig: Model Loss Graph of CRNN

With increasing epochs, the model is giving above 95% accuracy.

******

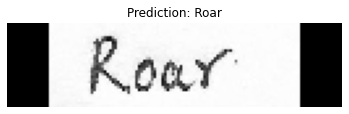
******

Fig: outputs predicted

**List of Abbreviations/Nomenclature:**

ML-Machine Learning

DL-Deep Learning

HIR-Handwriting Information Recognition

CNN-Convolutional Neural Networks

RNN-Recurrent Neural Networks

HTR-Handwritten Text Recognition

LSTM-Long Short Term Memory

BiLSTM-Bidirectional Long Short Term Memory

FCN-Fully Convolutional Network

CTC-Connectionist Temporal Classification

VGG-Visual Geometry Group

RESNET-Residual Networks

OCR-Optical Character Recognition

SVM-Support Vector Machine

ZIP-Zone Improvement Plan

IAM-Identity and Access Management

Wi-Fi -Wireless Fidelity

DAN-Document Attention Network

CRNN-Convolutional Recurrent Neural Network

**CHAPTER - 8**

**Conclusion and Future work**

This research introduced a hybrid architecture to recognize handwritten text that effectively applies preprocessing techniques like grayscale conversion, noise reduction, slant correction, and segmentation and effectively uses a hybrid architecture that combines AlexNet for feature extraction with a bidirectional LSTM units (Bi-LSTM) for sequence modeling and SGDM is chosen as the optimizer because it effectively guides the model towards the ideal combination of weights and biases, minimizing the overall error. In proposed model promising results are achieved by the use of IAM dataset, demonstrating its ability to accurately recognize handwritten text.

**Key Findings:**

•**Thorough Preprocessing:** The very systematized preprocessing pipeline consisting of segmentation, thresholding, noise reduction, and slant correction as well as grayscale conversion greatly animated the quality of input images enhancing the model’s performance.

•**Robust Feature Extraction:** Through its deep convolutional layers, AlexNet showed adequacy in discriminative feature extraction from images of handwritten text. This capability empowered the network to identify both local and global patterns, essential for achieving accurate recognition.

•**Efficient Sequence Modeling:** By identifying long-range relationships between words and sentences, the Bi-LSGM component effectively captured the sequential character of text. Its capacity to concentrate on pertinent context to make correct predictions was further strengthened by the inclusion of GRU units and attention processes.

•**Hybrid architecture:** The benefits of these two types were united in AlexNet and Bi-LSGM, which allowed them to read handwritten text.

•**Attention mechanism:** The attention mechanism of Bi-LSGM enabled the model to focus on critical elements and improve its performance.

•**Total Outcome:** The proposed design was established as a successful approach in dealing with the complexity of handwritten text since it has attained a high level of accuracy from word recognition trials

**Future Work:**

* Explore several CNN architectures for feature extraction.
* Explore alternate sequence modeling methods, like transformer-based models.
* Assess the model's performance against other handwritten text datasets.
* Address difficulties like handwriting variances and low image quality.
* Use language models for contextual word prediction.
* Exploring alternate feature extraction and sequence modelling methods.
* Evaluating model performance on handwritten text datasets across languages and styles.
* Investigating model optimization and compression solutions for real-time applications.

**APPENDIX**

Create a Deep Learning-based handwriting recognition system that successfully deciphers various handwritten texts. They examine characters, words, and sentences in a variety of styles, languages, and levels of complexity. Using a huge dataset, create an enduring method for transcribing handwriting into digital text. Smoothly integrate it into applications such as document scanning, accessibility aids, and data entry automation.

Make the most of handwritten data with a sophisticated deep-learning handwriting recognition system. Overcome the limitations of traditional ways to effectively read diverse writing styles in languages and difficulties. Train with a vast volume of data to attain high accuracy and robustness. Equip applications that might benefit from the convenience of handwriting-to-text conversion, such as document digitization, accessibility aids, and data automation.

**REFERENCES**

[1] Hassan, Hagar Hany, and Ayla Gülcü. "Handwritten Text Recognition using Deep Learning Methods." In 2023 7th International Electromagnetic Compatibility Conference (EMC Turkiye), pp. 1-7. IEEE, 2023.

[2] Zhang, Yanbo, Weiping Sun, and Mo Li. "WiRITE: General and Practical Wi-Fi Based Hand-Writing Recognition." IEEE Transactions on Mobile Computing (2023).

[3] Coquenet, Denis, Clément Chatelain, and Thierry Paquet. "DAN: a segmentation-free document attention network for handwritten document recognition." IEEE Transactions on Pattern Analysis and Machine Intelligence (2023).

[4] Malhotra, Ruchika, and Maru Tesfaye Addis. "End-to-End Historical Handwritten Ethiopic Text Recognition using Deep Learning." IEEE Access (2023).

[5] de Sousa Neto, Arthur Flor, Byron Leite Dantas Bezerra, Alejandro Héctor Toselli, and Estanislau Baptista Lima. "HTR-Flor: A deep learning system for offline handwritten text recognition." In 2020 33rd SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI), pp. 54-61. IEEE, 2020.

[6] Nikitha, A., J. Geetha, and D. S. JayaLakshmi. "Handwritten text recognition using deep learning." In 2020 International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT), pp. 388-392. IEEE, 2020.

[7] Mishra, Piyush, Pratik Pai, Mihir Patel, and Reena Sonkusare. "Extraction of information from handwriting using optical character recognition and neural networks." In 2020 4th International Conference on Electronics, Communication and Aerospace Technology (ICECA), pp. 1328-1333. IEEE, 2020.

[8] Dutta, Kartik, Praveen Krishnan, Minesh Mathew, and C. V. Jawahar. "Improving CNN-RNN hybrid networks for handwriting recognition." In 2018 16th international conference on frontiers in handwriting recognition (ICFHR), pp. 80-85. IEEE, 2018.

[9] Arora, Shefali, and MP S. Bhatia. "Handwriting recognition using deep learning in keras." In 2018 International Conference on Advances in Computing, Communication Control and Networking (ICACCCN), pp. 142-145. IEEE, 2018.

[10] Nair, Rathin Radhakrishnan, Nishant Sankaran, Bharagava Urala Kota, Sergey Tulyakov, Srirangaraj Setlur, and Venu Govindaraju. "Knowledge transfer using Neural network-based approach for handwritten text recognition." In 2018 13th IAPR International Workshop on Document Analysis Systems (DAS), pp. 441-446. IEEE, 2018.

[11] Jaramillo, José Carlos Aradillas, Juan José Murillo-Fuentes, and Pablo M. Olmos. "Boosting handwriting text recognition in small databases with transfer learning." In 2018 16th International Conference on Frontiers in Handwriting Recognition (ICFHR), pp. 429-434. IEEE, 2018.

[12] Rajnoha, Martin, Radim Burget, and Malay Kishore Dutta. "Offline handwritten text recognition using support vector machines." In 2017 4th International Conference on Signal Processing and Integrated Networks (SPIN), pp. 132-136. IEEE, 2017.

[13] Su, Bolan, Xi Zhang, Shijian Lu, and Chew Lim Tan. "Segmented handwritten text recognition with recurrent neural network classifiers." In 2015 13th International Conference on Document Analysis and Recognition (ICDAR), pp. 386-390. IEEE, 2015.

[14] Espana-Boquera, Salvador, Maria Jose Castro-Bleda, Jorge Gorbe-Moya, and Francisco Zamora-Martinez. "Improving offline handwritten text recognition with hybrid HMM/ANN models." IEEE transactions on pattern analysis and machine intelligence 33, no. 4 (2010): 767-779.

[15] Mondal, Riktim, and Samir Malakar. "Handwritten English word recognition using a deep learning-based object detection." Proceedings of the IEEE 770: 778.

[16] Hull, Jonathan J. "A database for handwritten text recognition research." IEEE Transactions on Pattern Analysis and Machine Intelligence 16, no. 5 (1994): 550-554.

[17] Chung, Jonathan, and Thomas Delteil. "A computationally efficient pipeline approach to full page offline handwritten text recognition." In 2019 International Conference on document analysis and recognition workshops (ICDARW), vol. 5, pp. 35-40. IEEE, 2019.

[18] Sanchez, Joan Andreu, Veronica Romero, Alejandro H. Toselli, and Enrique Vidal. "ICFHR2016 competition on handwritten text recognition on the READ dataset." In 2016 15th International Conference on Frontiers in Handwriting Recognition (ICFHR), pp. 630-635. IEEE, 2016.

[19] Ghosh, Rajib, Chinmaya Panda, and Prabhat Kumar. "Handwritten text recognition in bank cheques." In 2018 Conference on Information and Communication Technology (CICT), pp. 1-6. IEEE, 2018.

[20] Romero, Veronica, Joan Andreu Sanchez, Vicente Bosch, Katrien Depuydt, and Jesse de Does. "Influence of text line segmentation in handwritten text recognition." In 2015 13th International Conference on Document Analysis and Recognition (ICDAR), pp. 536-540. IEEE, 2015.

[21] Aradillas, José Carlos, Juan José Murillo-Fuentes, and Pablo M. Olmos. "Boosting offline handwritten text recognition in historical documents with few labeled lines." IEEE Access 9 (2021): 76674-76688.

[22] Cao, Huaigu, Rohit Prasad, and Prem Natarajan. "Handwritten and typewritten text identification and recognition using hidden Markov models." In 2011 International Conference on Document Analysis and Recognition, pp. 744-748. IEEE, 2011.

[23] Marti, U-V., and Horst Bunke. "Text line segmentation and word recognition in a system for general writer independent handwriting recognition." In Proceedings of Sixth International Conference on Document Analysis and Recognition, pp. 159-163. IEEE, 2001.

[24] Namboodiri, Anoop M., and Anil K. Jain. "Online handwritten script recognition." IEEE Transactions on Pattern Analysis and Machine Intelligence 26, no. 1 (2004): 124-130.

[25] Puigcerver, Joan. "Are multidimensional recurrent layers really necessary for handwritten text recognition?" In 2017 14th IAPR international conference on document analysis and recognition (ICDAR), vol. 1, pp. 67-72. IEEE, 2017.

[26] Nosary, Ali, Thierry Paquet, Laurent Heutte, and Ameur Bensefia. "Handwritten text recognition through writer adaptation." In Proceedings Eighth International Workshop on Frontiers in Handwriting Recognition, pp. 363-368. IEEE, 2002.

[27] Pastor, Moisés, Alejandro Toselli, and Enrique Vidal. "Writing speed normalization for on-line handwritten text recognition." In Eighth International Conference on Document Analysis and Recognition (ICDAR'05), pp. 1131-1135. IEEE, 2005.

[28] Bunke, Horst, Samy Bengio, and Alessandro Vinciarelli. "Offline recognition of unconstrained handwritten texts using HMMs and statistical language models." IEEE transactions on Pattern analysis and Machine intelligence 26, no. 6 (2004): 709-720.