EPIGRAPHICAL RECOGNITION AND RESTORATION USING DEEP NEURAL NETWORKS – AN APPROACH

PROJECT REPORT

Submitted by

KARTHIK S

SAI AAKAASH G R

VIDHARSHANA R

ANNA UNIVERSITY: CHENNAI

JUNE 2022

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"Epigraphical recognition and restoration using Deep Neural Networks – An Approach"** is the Bonafide work of **"Karthik S, Sai Aakaash G R, Vidharshana R"** who carried out the project work as a part of Creative and Innovative Project Laboratory.

DATE:	**SIGNATURE**
PLACE:	**Dr. K. THANGARAMYA**
	COURSE IN-CHARGE
	TEACHING FELLOW
	DEPATMENT OF CSE
	ANNA UNIVERSITY
	CHENNAI

ABSTRACT

The study of ancient inscriptions holds significant importance in unlocking the mysteries of civilizations' cultures, arts, medicines, and histories. However, the degradation and/or damage of a portion of these texts often results in their illegibility, hindering our knowledge of their contents. In this paper, we present a deep neural network-based approach to perform text restoration, along with the chronological and geographical attribution of the restored text. Our proposed model is trained on the unprocessed Packard Humanities Institute dataset, which consists of 178,551 transcribed inscriptions. The text is pre-processed to make it machine-readable and noise-reduced, and contextual information is captured through the joint representation of characters and words. Missing or damaged text is denoted by a special character [unk]. The model uses a transformer-based neural network to identify positional information, followed by shallow feed-forward networks for restoration, chronological, and geographical attribution. The collaboration between the model and skilled historians has the potential to maximize the accuracy of text restoration and attribution. The model produces a ranked set of decoded predictions, and multiple visualization techniques are used to increase the interpretability of the predictive hypothesis. This research serves as a breakthrough in expanding our current understanding of early civilizations' literary history and highlights the importance of ongoing research. The proposed method can be used to decode new encryptions more efficiently and augment our understanding of manually decoded ones, providing valuable information for future projects in this field and enhancing the interpretability of manuscripts of similar age and condition.

LIST OF CONTENTS

CHAPTER 1

INTRODUCTION

Ancient epigraphs, also known as inscriptions, are texts that have been engraved or written on hard surfaces such as stone, metal, or wood. These texts are important sources of historical, cultural, and linguistic information. Epigraphs are often found on ancient monuments, buildings, and artifacts, providing valuable insight into the beliefs, practices, and customs of past civilizations.

The study of epigraphs, also known as epigraphy, is a multidisciplinary field that involves archaeologists, historians, linguists, and computer scientists. Epigraphs are important for several reasons. First, they provide a tangible link to the past, giving us a glimpse into the daily lives of people who lived centuries or even millennia ago. Second, epigraphs can shed light on the social, political, and religious structures of ancient societies. Third, epigraphs provide valuable linguistic information, including dialects, scripts, and writing systems.

Epigraphs are particularly important for the study of ancient languages. Many ancient languages, such as Ancient Greek, Latin, and Sanskrit, are no longer spoken, and their written forms are the only way to learn about them. Epigraphs provide a wealth of linguistic data, including vocabulary, grammar, and syntax. This information can help linguists reconstruct ancient languages and understand their evolution over time. However, the process of deciphering and translating epigraphs is a difficult task due to various challenges faced during the restoration process.

One of the major challenges is the deterioration of the physical condition of the epigraphs over time. Epigraphs can get damaged or eroded, making it difficult to interpret the inscriptions. Additionally, parts of the epigraphs may be missing or broken, causing ambiguity in understanding the meaning of the inscriptions. This challenge is further compounded by the fact that epigraphs are often written in archaic or obscure languages that are no longer spoken or written.
In the past, traditional methods of translating epigraphs relied on the expertise of archaeologists and other professionals with a deep understanding of the language and culture of the civilization that created the inscription. However, these methods often resulted in inaccuracies and misinterpretations due to the aforementioned challenges. While the expertise of archaeologists and philologists has been crucial in deciphering these texts, their accuracy is often limited by the complexity of the texts and the level of fragmentation or damage. Moreover, traditional methods are time-consuming and resource-intensive, requiring the input of multiple experts and the manual correction of errors. To overcome these limitations, researchers are turning to modern technology, specifically deep neural

networks, to restore ancient texts. Various techniques have been developed in recent years to restore and translate ancient epigraphs using deep neural networks. These techniques range from image recognition algorithms to natural language processing models. One such approach involves taking an image of the epigraph as input, then creating a text file of the epigraph with unknown or missing text marked with a special character. The positional information of the words is then retrieved, and this information is fed into the torso of the proposed model which gives a range of predictions for the missing text, along with a corresponding probability score supporting the prediction.

This proposed approach has several benefits over traditional methods of translating epigraphs. First, it can restore damaged or missing parts of the epigraph with a high degree of accuracy. Second, it can translate inscriptions written in archaic or obscure languages without relying on the expertise of professionals. Finally, it can be applied to a wide range of epigraphs, making it a versatile tool for researchers and scholars. The proposed approach for ancient text restoration using deep neural networks is a promising technique that can help bridge the gap between the past and present. It takes advantage of advancements in deep learning and computer vision to restore epigraphs that are often fragmented, damaged, or missing parts. By creating a text file with marked unknown or missing text, the model can generate a range of predictions for the missing text with corresponding probability scores, thereby providing researchers and archaeologists with valuable insights into the past. This approach is an innovative solution to a long-standing problem in the field of epigraphy, which has been struggling to accurately interpret and translate ancient texts due to the limitations of traditional methods.

In contrast, deep neural networks offer a new level of accuracy and efficiency in the restoration of ancient texts. The proposed approach can process large amounts of data quickly, recognize patterns in the text, and generate predictions for missing or ambiguous sections. The use of positional information and probability scores further enhances the accuracy of the predictions, allowing researchers to confidently interpret the text and gain new insights into the past.
The benefits of this approach extend beyond the academic and archaeological communities. The restoration of ancient texts using deep neural networks can have significant implications for our understanding of history and culture, as well as for the preservation of cultural heritage. By providing a more accurate and accessible means of understanding ancient texts, this approach can help to bridge the gap between past and present and foster greater appreciation for the richness and diversity of human culture.

Thus restoration and translation of ancient epigraphs using deep neural networks is a promising field of research with significant potential for advancing our understanding of past civilizations. By leveraging the power of modern technology to overcome the limitations of traditional methods, this approach offers a new level of accuracy and efficiency in the interpretation and translation of ancient texts. With continued research and development, this approach could provide valuable insights into the language, culture, and history of ancient societies that were previously inaccessible due to the challenges of translating epigraphs. As the field continues to evolve, we can expect to see even more advanced techniques and applications emerge, further enhancing our understanding of the past and enriching our cultural heritage for generations to come.

CHAPTER 2

LITERATURE SURVEY

The concept of ancient text recognition and restoration using epigraphs has been a region of interest for Deep Learning enthusiasts recently. One of the first paper that was published regarding this issue is **paper by** Soumya et al., (2014) [8] **.** The paper discusses about the recognition of ancient Kannada Epigraphs.

Ancient Kannada script of two different periods Ashoka and Hoysala is used in this approach. s. Optical Character Recognition system to recognize ancient text of Kannada script is designed and implemented here. The input image of the epigraph is preprocessed to reduce noise, is segmented into individual characters and their centroid is computed. The statistical features are extracted by three methods, each of these methods provide a few more extraction methodologies. The one-dimensional statistical analyzer computes Mean, Variance, Standard deviation of an image. Histogram analyzer computes Skewness, kurtosis, Entropy values of an image and GLCM analyzer computes coarseness, smoothness, of an image. Overall, ten features per character are extracted. Considering them, the model value is computed using the Mamdani fuzzy function, which includes Gaussian computation. Each of the inputs is set with certain rules and a Gaussian function is used to get model value. The data stored as training samples for all the characters of Brahmi and Hoysala are retrieved. Model value is compared with training sample and relevant result is fetched. The result of the classifier is displayed in modern form as stored in the training set.

Performance Analysis of the system developed is carried out with respect to Segmentation rate and Recognition rate. The segmentation of characters has a direct impact on the output. Due to the cuts and bruises in the epigraph, on an average validate segmentation results up to 90%. The recognition results of Brahmi are observed to be comparatively better than Hoysala script

The paper discusses major disadvantages that follows. The proposed method works only on almost whole characters. Characters with cuts, bruises, smudges or missing parts are attributed as unrecognizable characters. The study is carried on properly denoised images of the epigraphs. The presence of noise greatly impacts the accuracy of the model. The accuracy of the results obtained in Segmentation and Recognition is determined by the versatile writing style practiced in handwritten documents. Less legible documents might not be recognized properly. [1]

In the paper by Manigandan et al., (2017) [7] , the proposed method for recognizing inscriptions in images from the 9th to 12th century involves collecting images from the Tamil Nadu Archaeology

Department in Chennai and processing them to remove any noise. After preprocessing, the images are transformed into binary format and segmented using Otsu's Thresholding Algorithm and Contourlet transformation. Feature extraction is performed on the segmented images using the GLCM feature extraction algorithm, and the resulting features are utilized for character classification using an SVM classifier. Additionally, a tri-grams approach is applied to accurately determine the specific letter being analyzed. Following character identification, a pattern matching algorithm is employed to map the character to the Unicode character set. The identified character is then stored in the Unicode corpus along with its corresponding Unicode value. In cases where the character is not present in the Unicode corpus, the user is prompted to recognize the character and assign a corresponding Unicode value, only in instances of new shorthand characters. Special shorthand characters are assigned a unique value and utilized as training data for the algorithm.

Accuracy, precision, F1 score can be used to evaluate the model. The method discussed in the study has been able to successfully recognize various characters and sentences in Tamil from the inscriptions from 9th to 12th century CE with satisfactory levels of accuracy. That being said, the accuracy of character depends on the quality of the input image, and more specifically, on the image resolution. As the resolution of the image decreases, the accuracy steadily declines. [7]

In the paper by Zhao et al.,(2020) [3] , the proposed model recognizes high-pixel and multi-dimensional images by conducting gradient level fragmentation processing, transforming the images through layers to gradually reach low-pixel and low-dimensional layers, and combining auxiliary information with the original input data to train a stable Laplacian adversarial neural network generative model. An unsupervised image cluster labeling algorithm is used to segregate similar characters based on the density peak and distance, and unsupervised labeling of text and image samples is completed by solving for the number of clusters and the threshold value of the clustering algorithm using the information entropy evaluation method. The classifier structure of the convolutional neural network is optimized by using the inter-class distance dc in the clustering algorithm, based on information entropy optimization. The Gaussian likelihood function and the Gaussian logarithmic likelihood function of the objective function are then derived, and the optimal value of dc is obtained by calculating the gradient and setting it equal to zero, allowing for the optimal classification result. The result can be used as a training data set for the convolutional neural network after a small amount of manual sorting and semantic annotation. The improved clustering performance optimizes the training of the convolutional neural network and enhances the recognition rate of Shui characters by the convolutional neural network.

‘‘Peak Signal to Noise Ratio’’ (PSNR) is an important indicator for measuring image quality in the field of super resolution, and the authors used it to evaluate the quality of generated images, along with the accuracy of the automatically labeled samples in the whole process. The latter grows rapidly in the beginning and stabilizes at 70%. The image samples of the Shui characters will have higher resolution through the proposed method. The Laplacian and classifier methods provide better outputs when compared to their individual results.

The main drawbacks included the following. The image quality is bad, classification will have more errors. The overall process encounters difficulties with noise expression. the residual and the parameter values are changed in the direction opposite the optimization, hence a large deviation of character recognition rate. Errors will add up, degrading the training accuracy. There are a certain number of errors from the character and automatic annotation, which let the authors choose some appropriate labeling strategies.

Along with these, we can also view many papers that include the algorithms that are being used for the recognition. In the paper by Phan et al. (2020) [4] are two among the most widely used techniques to construct deep learning models. Observing that the binarization of lightweight modules in neural networks can gain considerable outcomes, in this paper we propose Mobile Binary Network (or MoBiNet), a network that significantly compresses MobileNet to only a few megabytes while preserving good accuracy compared to other similar models. MoBiNet can be considered as a compact binary version of MobileNet.

The ILSVRC12 ImageNet dataset is used for both the training and assessment of image classification. The suggested method's separable convolution layers are made up of two compact modules: an 1x1 point-wise convolution and a 3x3 depthwise convolution with a single channel link. Skip/channel connections can be utilized to combat the information loss caused by depthwise convolution. There are three ablation studies done. The first compares mobile networks with and without skip connections in order to test our hypothesis that skip connections are beneficial for separable convolution binarization. The second is to compare the three additional block designs (Pre, Mid, and Post-block) added to MoBiNet to the binary original MobileNet and assess their efficacy. Also, it is shown how K dependency affects MoBiNet's improvement for a variety of hyper-parameter K values (K = 0, 1, 2, and 3). The five criteria Top-1, Top-5 classification accuracy, number of parameters, and number of FLOPs are used to compare MoBiNet to the most recent generation of cutting-edge binary neural networks.

The proposal results in an effectively small model while keeping the accuracy comparable to existing ones. Experiments on ImageNet dataset show the potential of the MoBiNet as it achieves 54.40% top-1 accuracy and dramatically reduces the computational cost with binary operators. MoBiNet, with Mid-block design and K = 4, achieves a speedup factor of 3.13× and a model saving factor of 1.34×. But in terms of accuracy MoBiNet outperforms all of the precedent methods except Bi-RealNet that we are 2% behind. This method was proposed 3 years ago, and efficient models have been developed henceforth. [4]

We can also take inputs from papers where the creation of our own datasets are discussed. In 'Creation Of Original Tamil Character Dataset Through Segregation Of Ancient Palm Leaf Manuscripts In Medicine' paper, a dataset is created in photoshop.

For the purpose of this study, palm leaf manuscripts written in Tamil (and in other languages) were gathered. The target characters in the investigation were identified using a variety of medical palm leaf text categories. The Gaussian filter is a tool used in image processing to blur pictures and eliminate background noise. On a blank piece of paper is one character from the Tamil palm leaf manuscript. Photoshop is used to transform the character into images that computers can understand. The character is also transformed into 224x224 pixel pictures in white and black or 8-bit greyscale. Following the conversion of the images, a pipeline object is produced, consisting of the output format and the source directory. Every object in the pipe is first rotated by 0.5% to the left, then by 0.5% to the right, creating around 50% mirror flips. No full mirror flips would be performed because the goal of this study is to locate a written article. For each image, a Gaussian distortion is added. Probability, grid width and height, magnitude, corner correction, infusion method, and a few x and y parameters are characteristics of Gaussian distortion. No specific target value is available; all values are utilized arbitrarily. This process is repeated until the desired form is obtained.

The proposed approach outputs a highly legible image of any illegible character from the palm leaf. These are manually mapped to the appropriate character, and thus the dataset is created. The dataset could be utilized by future researchers to develop expert systems that might, in turn, serve numerous purposes

This method requires the epigraph to be manually written on a separate sheet and then photoshopped and fed into the model. This could increase the amount of labor that goes into the process, especially if the quantity of palm leaves to be analyzed is large. Mapping of the enhanced image is also done manually, rather than being automated.

In the paper, ‘Restoring Ancient Text Using Deep Learning: A Case Study On Greek Epigraphy’ the proposed paper uses a LSTM-LM encoder and decoder to learn and predict missing characters and values from the Greek inscriptions. A Bi-directional LSTM Encoder was also used for better modeling. The Dataset was also generated via the use of a pipeline to convert Greek inscriptions into machine actionable text. The model proposed in the above-mentioned paper is called PYTHIA which works simultaneously at both a character- and a word level. PYTHIA takes an input of damaged text, and it is trained to predict character sequences comprising the hypothesized restorations.

The results obtained from the PYTHIA model consistently are better and of lower CER (Character Error Rate) than other proposed methods such as manual epigraphy, use of regular n-gram language models etc. In particular the bidirectional-LSTM model of PYTHIA was effective in predicting the top 20 of predictions 73.5% of the time with an CER of 30.1% as compared to Manual Epigraphy with a CER of 57.3%. The proposed method is curated specifically for Greek charecters, and cannot be scaled to any other language. [6]

After the paper about Mobinet, in the paper ‘Tamizhi: Historical Tamil-Brahmi Script Recognition Using Cnn And Mobilenet’**, t**he paper presents a method for recognizing Tamil-Brahmi script using Convolutional Neural Networks (CNN) and MobileNet architecture. The Tamil-Brahmi script is an ancient writing system used in Tamil Nadu, India, and recognizing it is important for historical and archaeological research. The proposed method involves pre-processing the images of the script, followed by training CNN and MobileNet models using the pre-processed images. The models achieved high accuracy in recognizing the Tamil-Brahmi script, which can be used in digitizing and preserving historical documents. The proposed method is efficient, accurate, and can be used for other historical scripts as well.

The algorithm used in "TAMIZHİ: Historical Tamil-Brahmi Script Recognition Using CNN and MobileNet" involves several steps. Preprocessing of the images of the script, including normalization, binarization, and deskewing, splitting the preprocessed images into training and testing sets, training a CNN and a MobileNet model using the training set. testing the models on the testing set and evaluating their accuracy. The results showed that the proposed method achieved high accuracy in recognizing the Tamil-Brahmi script, with the CNN model achieving an accuracy of 98.97% and the MobileNet model achieving an accuracy of 98.73%. The results demonstrate the effectiveness of using CNN and MobileNet for recognizing historical scripts.

The disadvantages include the following. It requires a large amount of training data to achieve high accuracy. In addition, the method is specific to Tamil-Brahmi script and may not be applicable to other historical scripts without modification. The recognition accuracy may be impacted by the quality of the input images, which can be difficult to control in historical documents. [2]

Finally in the paper 'Ancient Text Character Recognition Using Deep Learning' The paper presents a method for recognizing characters in ancient texts using deep learning techniques. The proposed method involves preprocessing the input image to remove noise and enhance contrast, segmenting the characters in the preprocessed image, and training a Convolutional Neural Network (CNN) model using the segmented characters as input.

The results showed that the proposed algorithm achieved high accuracy in recognizing ancient text characters, with an accuracy of 96.83%. The results demonstrate the effectiveness of using deep learning techniques for character recognition in ancient texts. This algorithm may not be applicable to texts with complex layouts or fonts, as the algorithm relies on character segmentation. Algorithm may require a large amount of training data to achieve high accuracy. [8]

Having viewed the papers, it is evident that there are ample research works available related to text recognition of languages other than English. Some of these also provide very high accuracy rates. Using MobileNet has always been a go-to option as it reduces the computational strain. Many models rely on neural networks to process the image, while there are other models that use feature extraction methods to extract certain features from the input text and then opt for methods like SVM and pattern matching. It is evident that as new models evolve, they tie with or outperform their peers in some way or the other. One reduces the computational complexity while the other trades the ease of execution with exceptional accuracy. Majority of the research work fails to be effective when the quality of the epigraph is not up to the mark. Moreover, those models which provide high accuracy often require large datasets which could not be feasible considering that these images are hard to collect.

Though there are so many models to recognize ancient text, it is of little use if the epigraph is broken, damaged or is missing a part of it. Though with the help of these text recognition models archeologists can faintly make out the damaged text, this is possible only when the extent of damage is within a certain range. Texts with huge chunks of letters or words missing cannot be processed further, and all the precious information that our forefathers preserved in them are lost. So for that reason, it is crucial to develop a model that can make an educated prediction on what the missing chunk could have been, essentially reconstructing the epigraph.

CHAPTER 03

PROPOSED SYSTEM

3.1. SYSTEM ARCHITECTURE

Initially we start by taking the input image, and preprocessing it. Then the text is recognized and the positional information is also captured. Then it is passed into the torse of the model where sparse multi head attention heads are used with residual connections and layer normalizations. Then the output is possed to a feed forward network, and the missing text is restored.

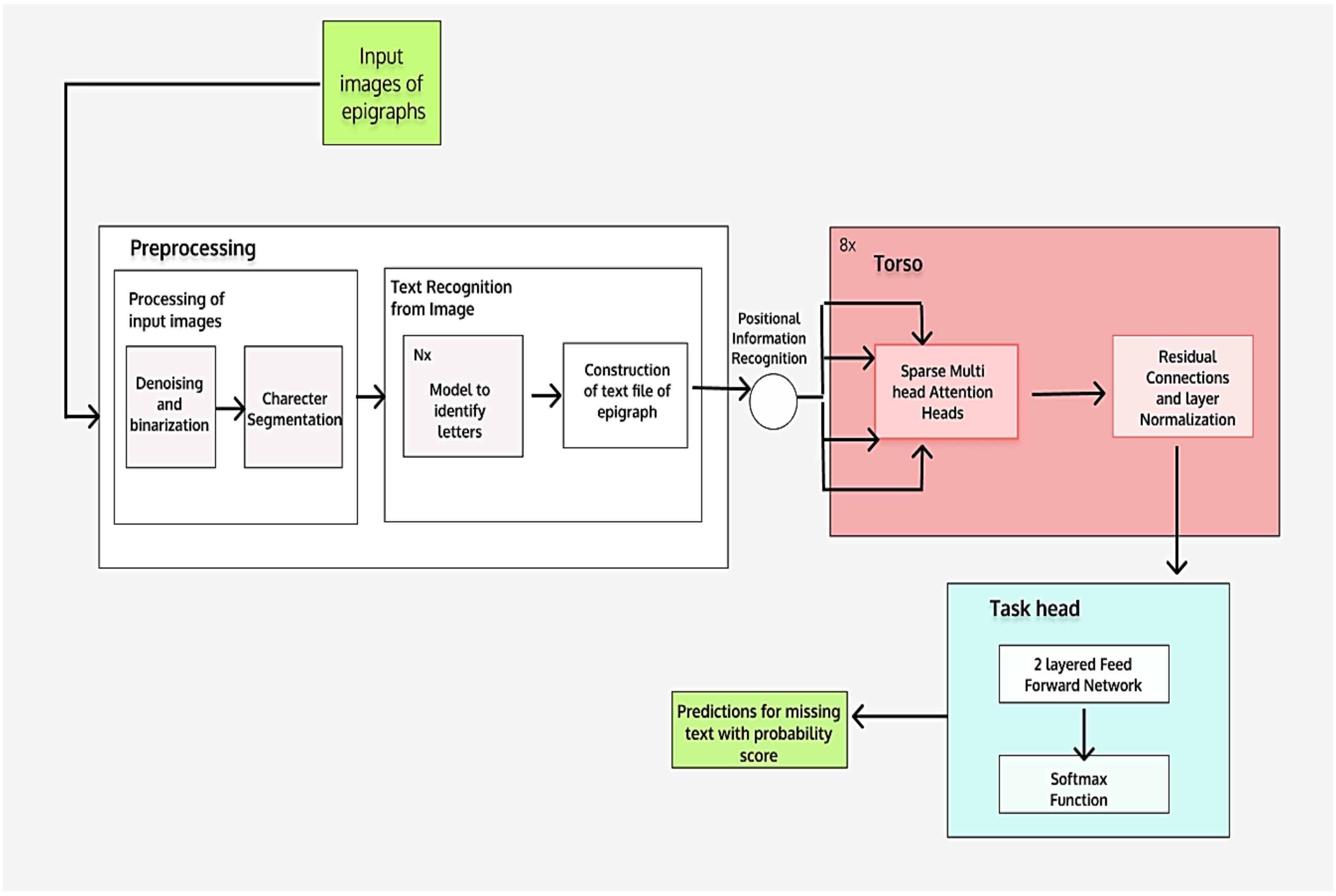

Input images of epigraphs
Preprocessing
Processing of input images
Denoising and binarization
Charecter Segmentation
Text Recognition from Image
Nx
Model to identify letters
Construction of text file of epigraph
Positional Information Recognition
8x
Torso
Sparse Multi head Attention Heads
Residual Connections and layer Normalization
Task head
2 layered Feed Forward Network
Softmax Function
Predictions for missing text with probability score

Figure 3.1.1 : OVERAL SYSTEM ARCHITECTURE

3.2 PROPOSED METHODOLOGY

The proposed methodology is to convert the given input epigraph into a text file and then marking the missing text using a special unknown character. This text is then passed to the 8 transformer layers. Each layer consists of sparse multi head attention heads. Then the output is also subjected to residual connection and layer normalization. Then a special 2 layer feed forward function is used to predict the missing text.

3.2.1 IMAGE PREPROCESSING

The aim of this module is to make the input image of the epigraph suitable for any machine learning model. The input image is first rotated to make it upright. After correctly positioning the image, it is converted to greyscale. Median filters are applied for Salt and pepper/impulse noise. Later, horizontal and vertical lines are removed from the image. The image is smoothened using Gaussian Blur Filter and final denoising of the image is done by using non-localized means. It is then converted to binarized form using adaptive thresholding method, where each pixel either takes 0 or 1.

3.2.2 CHARACTER SEGMENTATION

The output obtained from the previous module is now ready to be segmented. The dilate function is used to apply morphological filters on the image. The contours are identified from the image to get a rough idea about the position of each element. If the area of the contour is very small, it is ignored, and only those of considerable size are taken to be a letter. The ROI (Region of Interest) is identified in each of them. It is then cropped out and saved as a separate image. Thus, each image would correspond to one segment (letter/word) from the input. These images are labeled in the order in which they appear in the epigraph. These will collectively form a dataset and this can be fed to the ML model to identify what character it is.

3.2.3 TEXT IDENTIFICATION MODULE

In this module, the characters are identified and attributed to their corresponding form in modern literature using ML algorithms. For this purpose, Convolution Neural Network (CNN) algorithm is used. The module contains N such models, where N denotes the number of various eras the language can be classified into. We assign a numeric class to each letter that is present in the language, starting from 0. The mapping of each letter to a class is maintained in a file. With that, the predicted class is written in a text file as the matching modern-day version of that letter. Missing or unidentified characters or words are marked accordingly. Hence the given input image is converted into a text file which can now be processed further.

3.2.4. POSITIONAL EMBEDDING RECOGNITION

Sequence of character embeddings: real-valued vectors, each representing the character of the alphabet that occurs at the corresponding position of the inscription. Sequence of word embeddings which are equally long : real valued vector which represents the vocabulary word at the corresponding character position of the inscription. Positional embeddings: real valued vectors representing a position in the input sequence. Finally, the input sequence is padded with a start-of-sentence character ‘<’.

3.2.5. SPARSE MULTI HEAD CONNECTION MODULE

Concatenating the various embeddings per-character location allows the three input sequences to be combined, and the resultant sequence is then fed through the model's torso. The large-scale transformer model BigBird served as the inspiration for the eight stacked transformer decoder sections that make up model's torso. The model can manage longer sequences than classical transformers because each block employs four sparse attention heads (using global, local, and random attention mechanisms), which reduce the context-length dependency from quadratic to linear. The attention system is also "multi-headed" in the sense that it can be trained to take into account various kinds of information acquired from the input. Different attention heads, for instance, may be more receptive to certain character sequences or more perceptive to specific words and phrases with distinguishing morphosyntactic or semantic characteristics. Finally, each transformer block utilizes residual connections and layer normalization to address issues that prevent the stacking of such complex blocks.

3.2.5. TASK HEAD MODULE

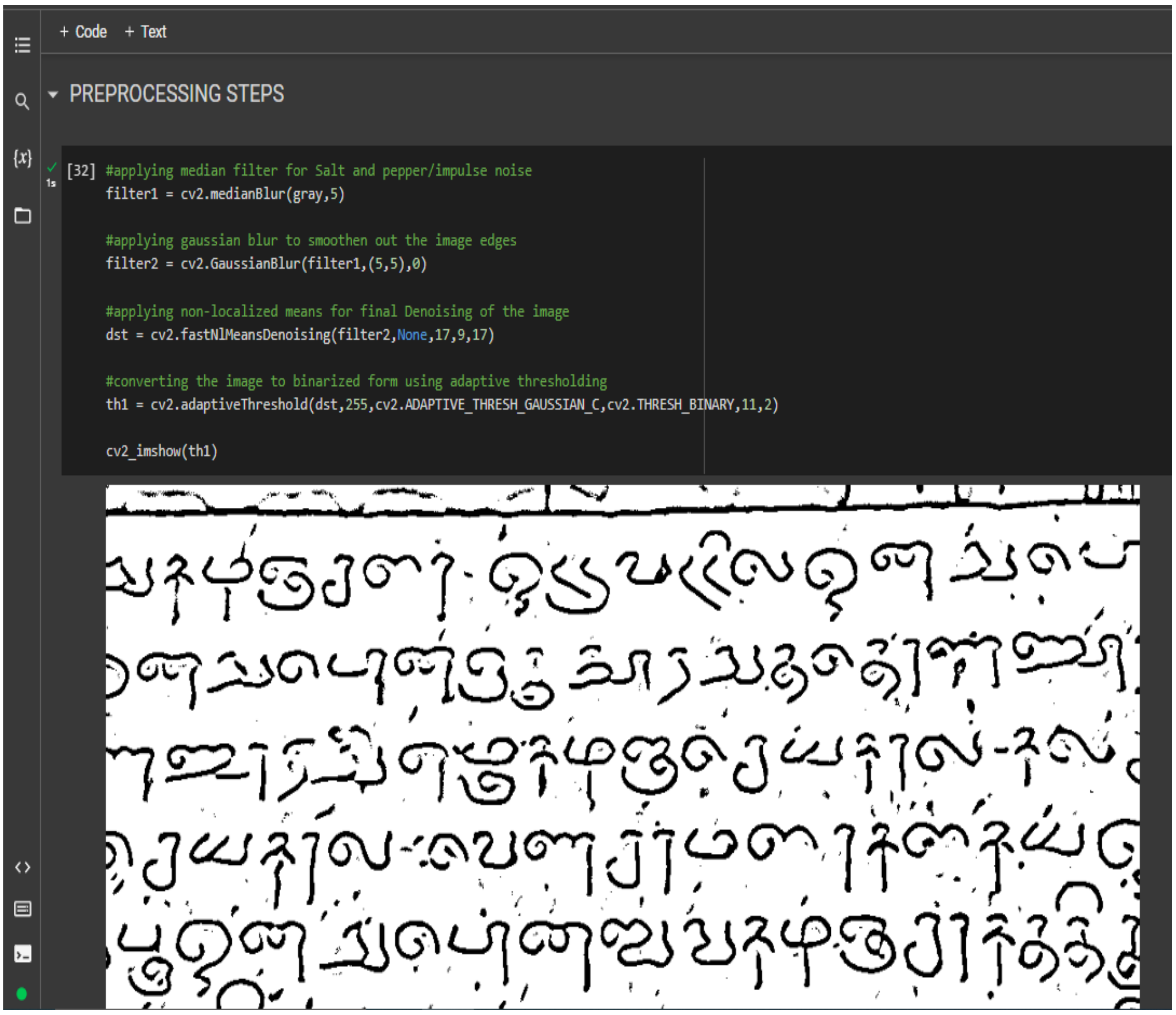
The output of the previous module is passed as the input. Task head consists of a two-layer feedforward network followed by a SoftMax function to covert the vector into a probability distribution of possible outcomes. There exists a direct correspondence between the input text and the output embeddings. For each missing character position, the corresponding output embedding of the torso is fed to the head of the restoration task, which predicts the missing character. Along with the prediction, appropriate probability score is also produced, which ranks the predictions from most suitable to the least.

CHAPTER 04

RESULT AND DISCUSSION

Image Preprocessing:

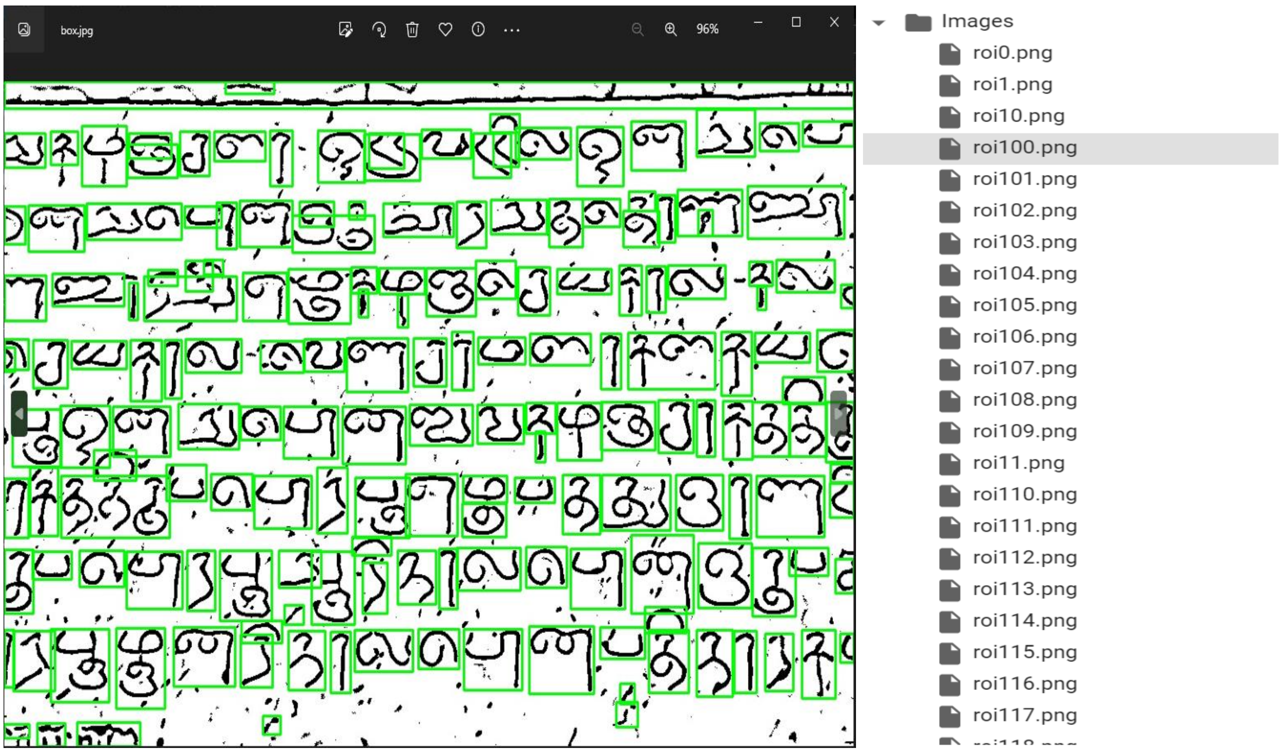
The final output obtained is a preprocessed binary image of the epigraph. No evaluation metrics are needed

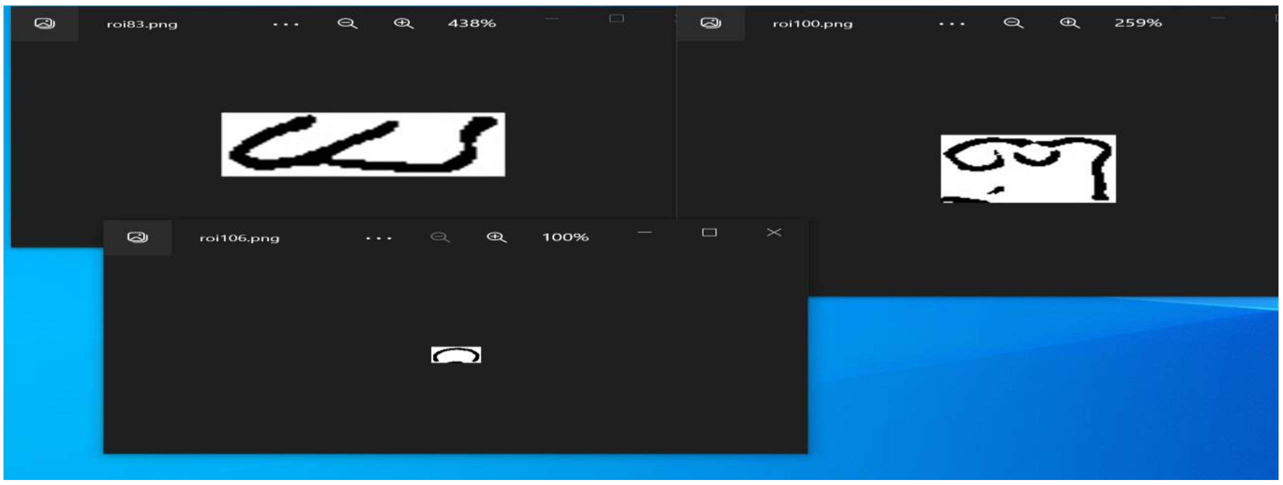


```
#applying median filter for Salt and pepper/impulse noise
filter1 = cv2.medianBlur(gray,5)

#applying gaussian blur to smoothen out the image edges
filter2 = cv2.GaussianBlur(filter1,(5,5),0)

#applying non-localized means for final Denoising of the image
dst = cv2.fastNlMeansDenoising(filter2,None,17,9,17)

#converting the image to binarized form using adaptive thresholding
th1 = cv2.adaptiveThreshold(dst,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,11,2)

cv2_imshow(th1)
```

Character Segmentation:

The final output obtained is a collection of images of each letter present in the epigraph. No evaluation metrics are needed for the module



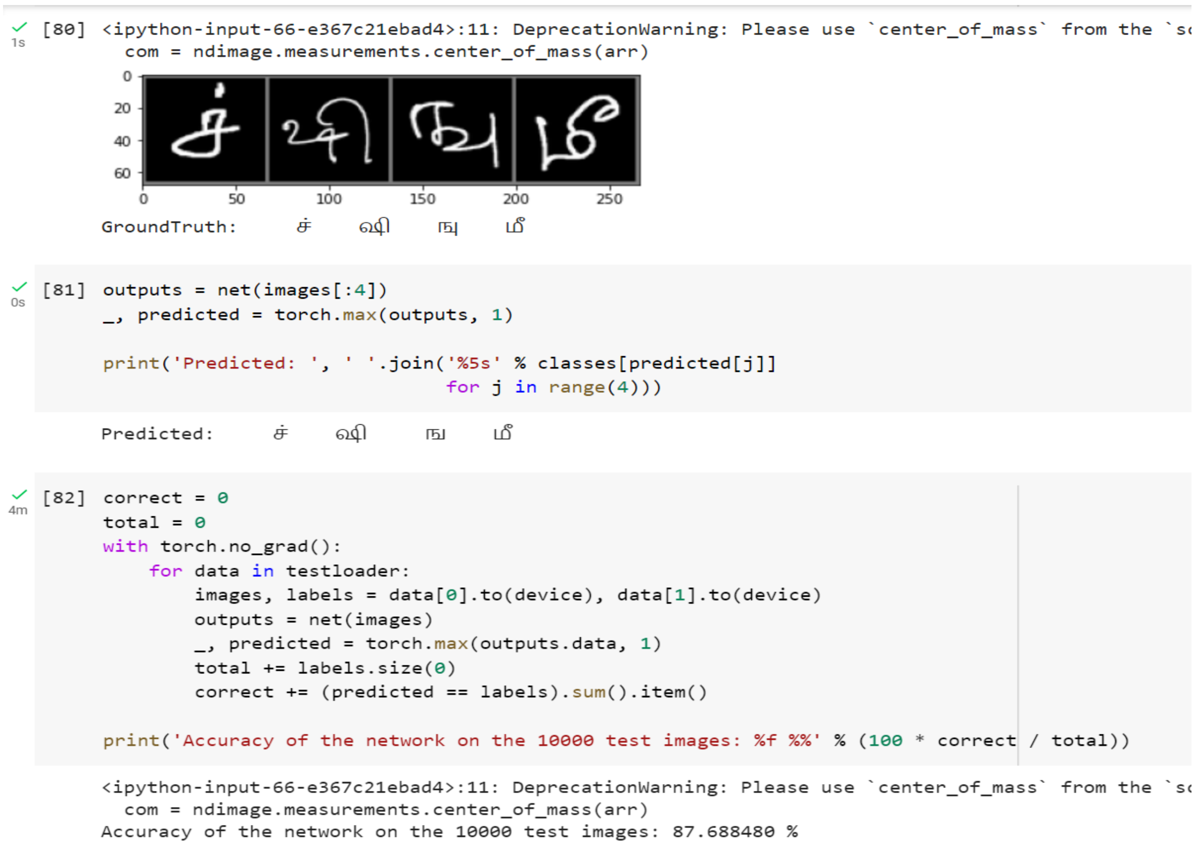


Text Recognition:

The output is the correctly predicted letter, in it modern form.

We evaluate the accuracy of the predicted letters in percentage accuracy.

The model has achieved 87.69% accuracy after being trained for 3 epochs.



```
[80] <ipython-input-66-e367c21ebad4>:11: DeprecationWarning: Please use `center_of_mass` from the `sc
       com = ndimage.measurements.center_of_mass(arr)
```

```
GroundTruth:    ச்    ஷி    ஙு    மீ
```

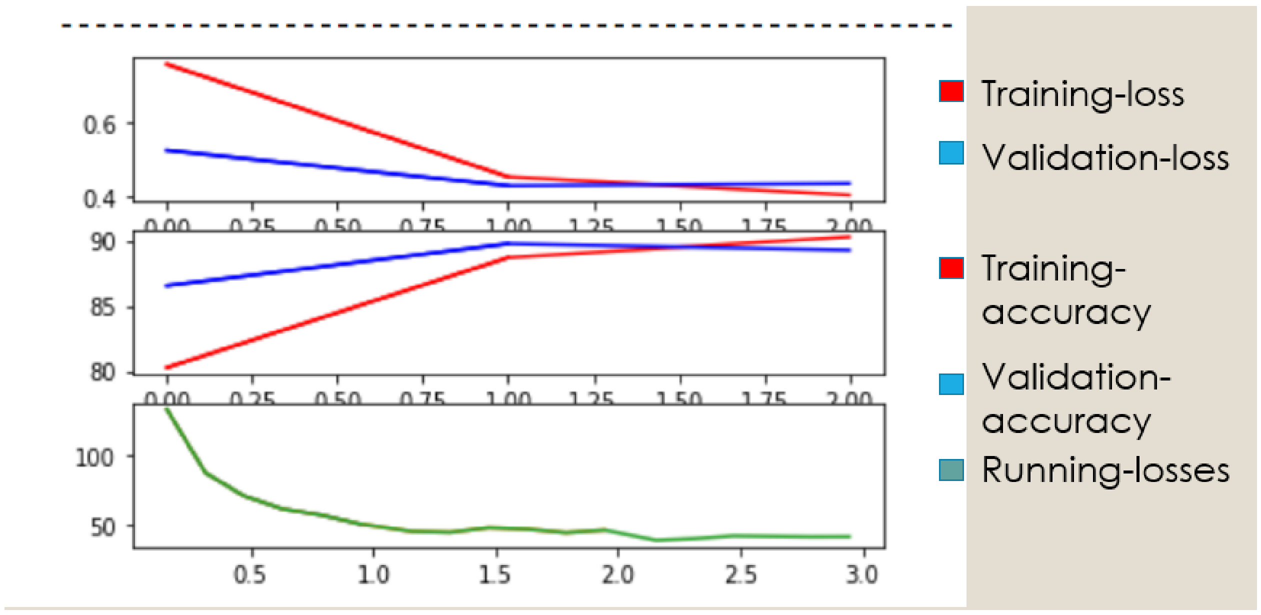
```
[81] outputs = net(images[:4])
     _, predicted = torch.max(outputs, 1)

     print('Predicted: ', ' '.join('%5s' % classes[predicted[j]]
                                   for j in range(4)))
```

```
Predicted:    ச்    ஷி    ஙு    மீ
```

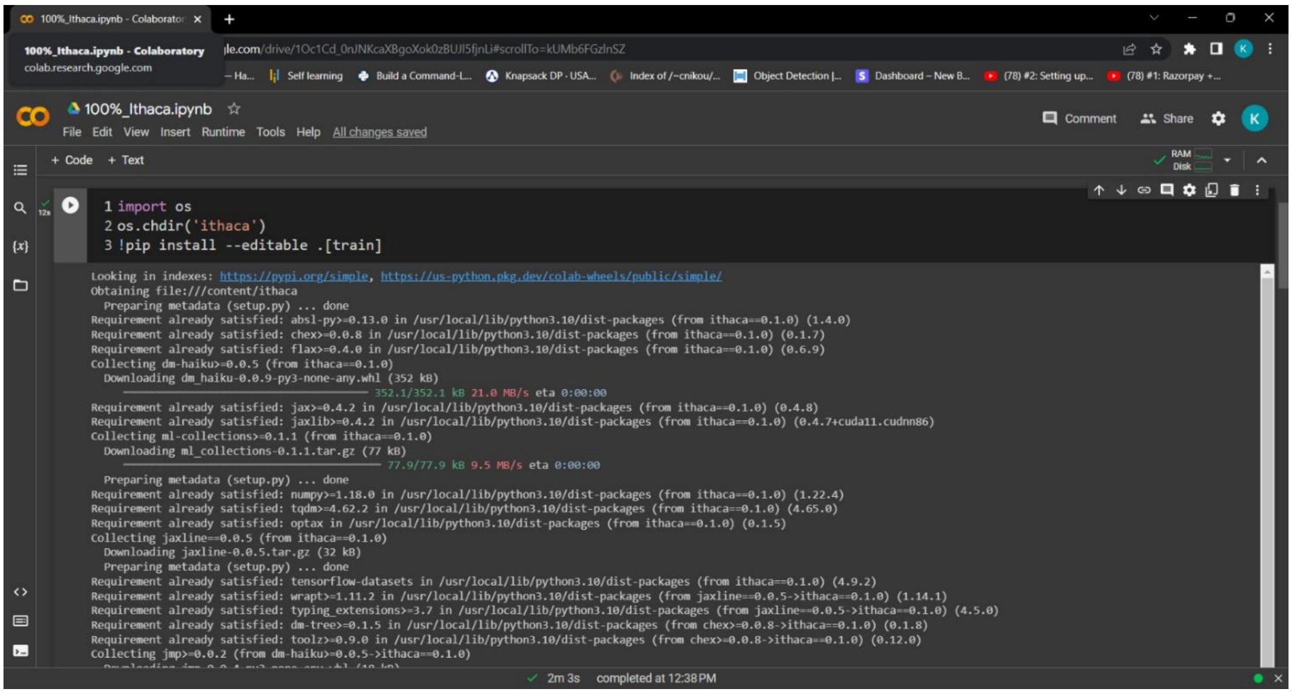
```
[82] correct = 0
     total = 0
     with torch.no_grad():
         for data in testloader:
             images, labels = data[0].to(device), data[1].to(device)
             outputs = net(images)
             _, predicted = torch.max(outputs.data, 1)
             total += labels.size(0)
             correct += (predicted == labels).sum().item()

     print('Accuracy of the network on the 10000 test images: %f %%' % (100 * correct / total))
```

```
<ipython-input-66-e367c21ebad4>:11: DeprecationWarning: Please use `center_of_mass` from the `sc
  com = ndimage.measurements.center_of_mass(arr)
Accuracy of the network on the 10000 test images: 87.688480 %
```

Training and Validation Statistics:



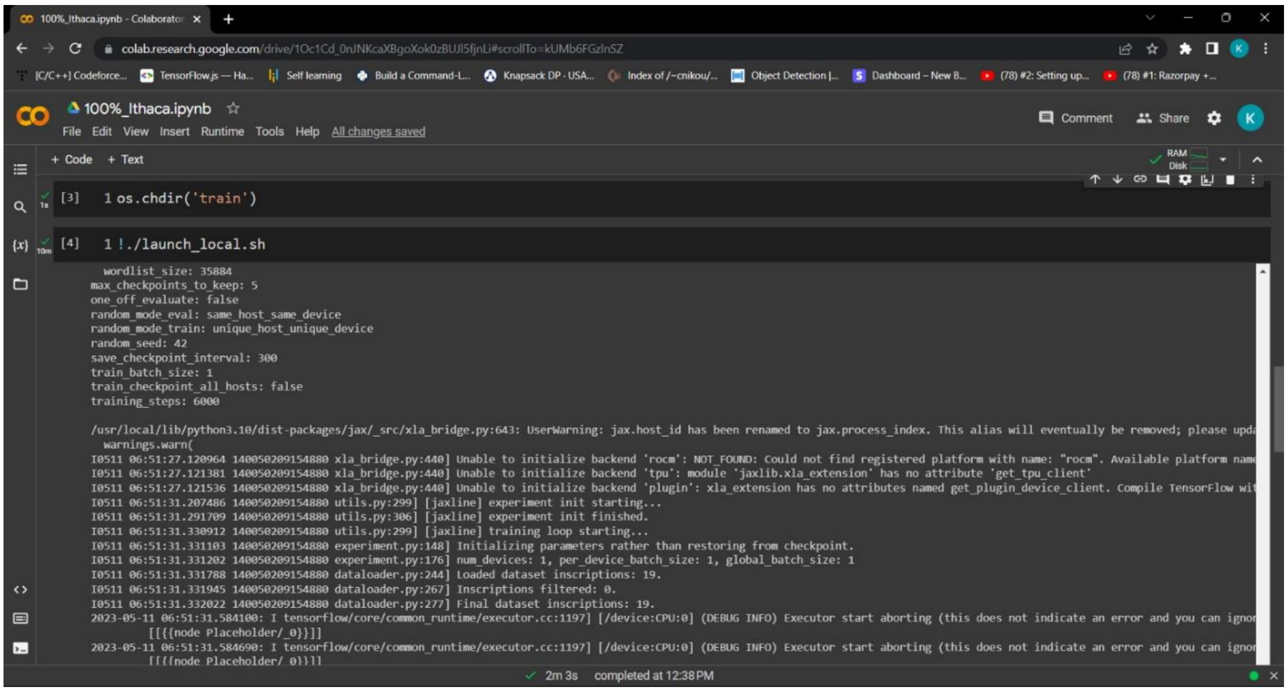
Training of Torso:



```
import os
os.chdir('ithaca')
!pip install --editable .[train]
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Obtaining file:///content/ithaca
  Preparing metadata (setup.py) ... done
Requirement already satisfied: absl-py>=0.13.0 in /usr/local/lib/python3.10/dist-packages (from ithaca==0.1.0) (1.4.0)
Requirement already satisfied: chex>=0.0.8 in /usr/local/lib/python3.10/dist-packages (from ithaca==0.1.0) (0.1.7)
Requirement already satisfied: flax>=0.4.0 in /usr/local/lib/python3.10/dist-packages (from ithaca==0.1.0) (0.6.9)
Collecting dm-haiku>=0.0.5 (from ithaca==0.1.0)
  Downloading dm_haiku-0.0.9-py3-none-any.whl (352 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 352.1/352.1 kB 21.0 MB/s eta 0:00:00
Requirement already satisfied: jax>=0.4.2 in /usr/local/lib/python3.10/dist-packages (from ithaca==0.1.0) (0.4.8)
Requirement already satisfied: jaxlib>=0.4.2 in /usr/local/lib/python3.10/dist-packages (from ithaca==0.1.0) (0.4.7+cuda11.cudnn86)
Collecting ml-collections>=0.1.1 (from ithaca==0.1.0)
  Downloading ml_collections-0.1.1.tar.gz (77 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 77.9/77.9 kB 9.5 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: numpy>=1.18.0 in /usr/local/lib/python3.10/dist-packages (from ithaca==0.1.0) (1.22.4)
Requirement already satisfied: tqdm>=4.62.2 in /usr/local/lib/python3.10/dist-packages (from ithaca==0.1.0) (4.65.0)
Requirement already satisfied: optax in /usr/local/lib/python3.10/dist-packages (from ithaca==0.1.0) (0.1.5)
Collecting jaxline==0.0.5 (from ithaca==0.1.0)
  Downloading jaxline-0.0.5.tar.gz (32 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: tensorflow-datasets in /usr/local/lib/python3.10/dist-packages (from ithaca==0.1.0) (4.9.2)
Requirement already satisfied: wrapt>=1.11.2 in /usr/local/lib/python3.10/dist-packages (from jaxline==0.0.5->ithaca==0.1.0) (1.14.1)
Requirement already satisfied: typing_extensions>=3.7 in /usr/local/lib/python3.10/dist-packages (from jaxline==0.0.5->ithaca==0.1.0) (4.5.0)
Requirement already satisfied: dm-tree>=0.1.5 in /usr/local/lib/python3.10/dist-packages (from chex>=0.0.8->ithaca==0.1.0) (0.1.8)
Requirement already satisfied: toolz>=0.9.0 in /usr/local/lib/python3.10/dist-packages (from chex>=0.0.8->ithaca==0.1.0) (0.12.0)
Collecting jmp>=0.0.2 (from dm-haiku>=0.0.5->ithaca==0.1.0)
```

100%_Ithaca.ipynb
File Edit View Insert Runtime Tools Help All changes saved
2m 3s completed at 12:38 PM

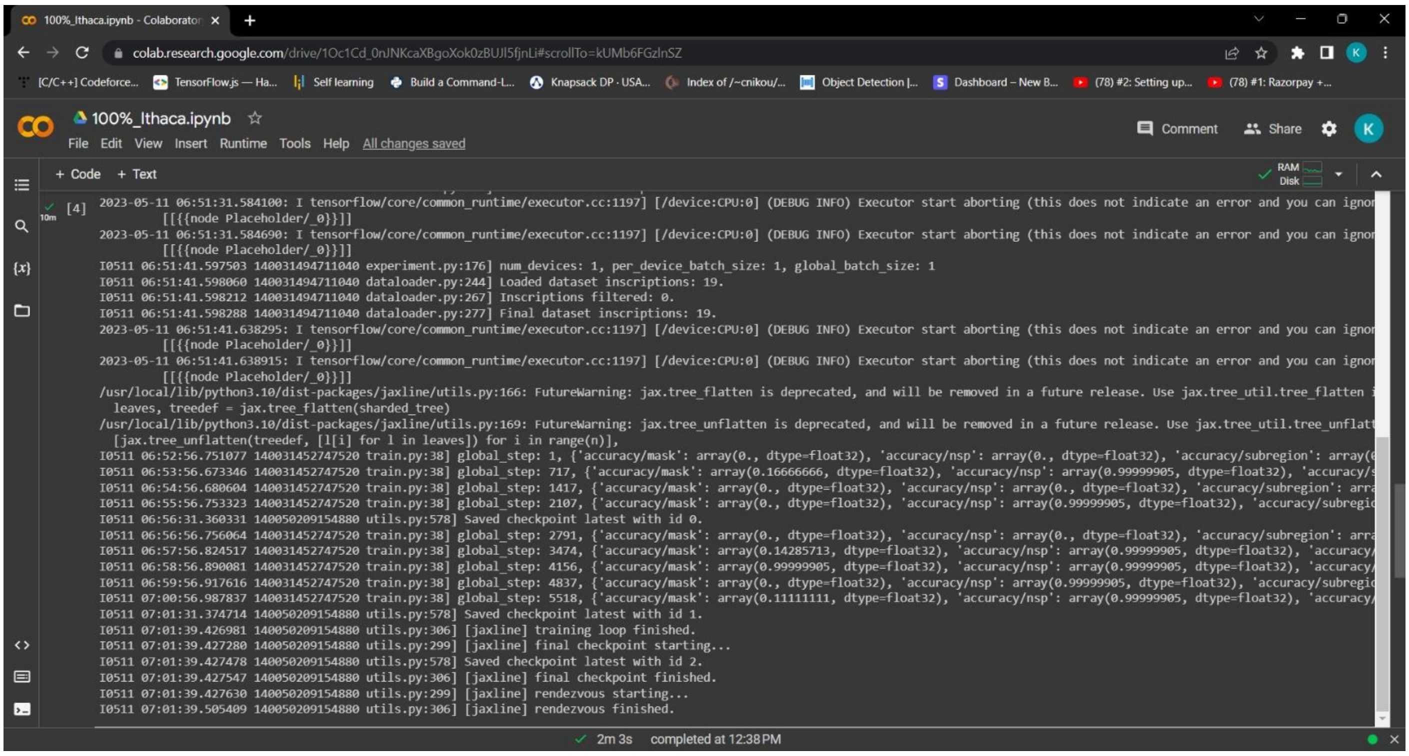


```
[3] os.chdir('train')
```

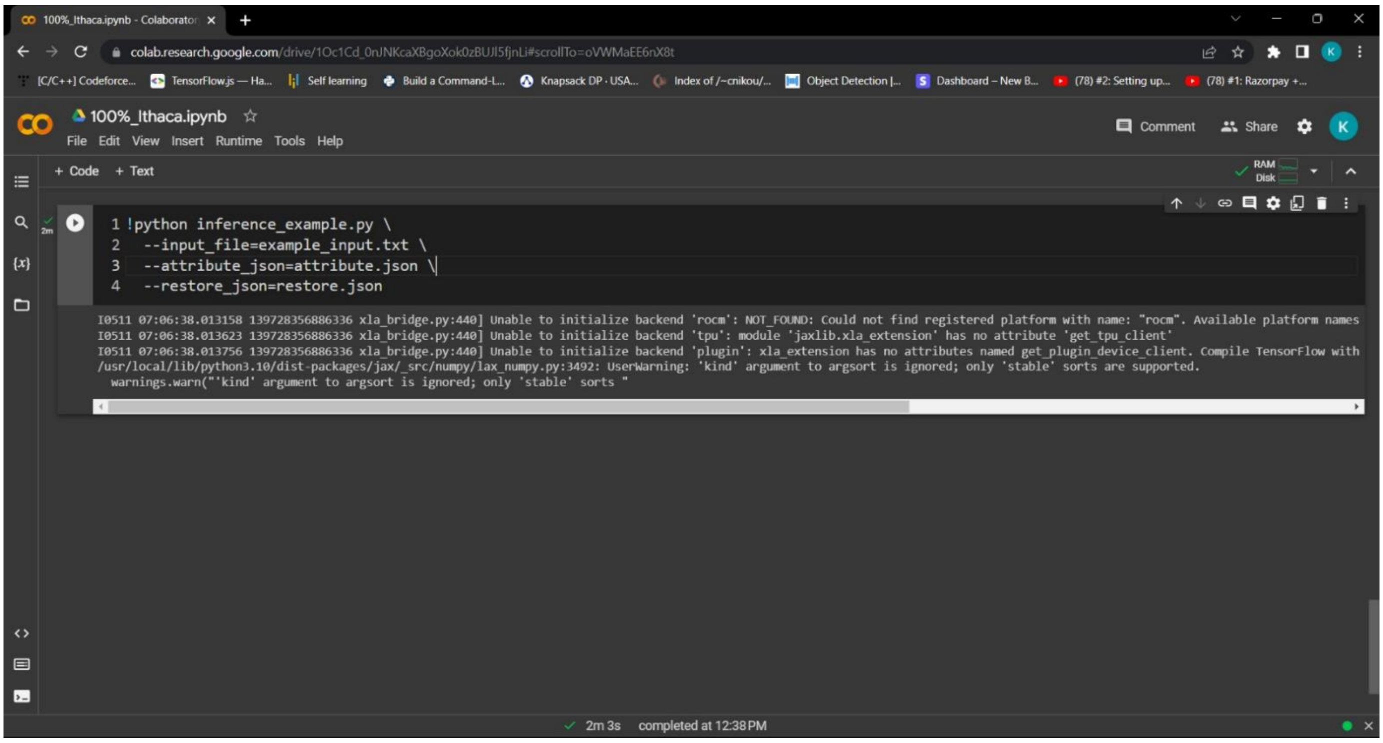
```
[4] !./launch_local.sh
```

```
  wordlist_size: 35884
max_checkpoints_to_keep: 5
one_off_evaluate: false
random_mode_eval: same_host_same_device
random_mode_train: unique_host_unique_device
random_seed: 42
save_checkpoint_interval: 300
train_batch_size: 1
train_checkpoint_all_hosts: false
training_steps: 6000

/usr/local/lib/python3.10/dist-packages/jax/_src/xla_bridge.py:643: UserWarning: jax.host_id has been renamed to jax.process_index. This alias will eventually be removed; please upda
  warnings.warn(
I0511 06:51:27.120964 140050209154880 xla_bridge.py:440] Unable to initialize backend 'rocm': NOT_FOUND: Could not find registered platform with name: "rocm". Available platform name
I0511 06:51:27.121381 140050209154880 xla_bridge.py:440] Unable to initialize backend 'tpu': module 'jaxlib.xla_extension' has no attribute 'get_tpu_client'
I0511 06:51:27.121536 140050209154880 xla_bridge.py:440] Unable to initialize backend 'plugin': xla_extension has no attributes named get_plugin_device_client. Compile TensorFlow wit
I0511 06:51:31.207486 140050209154880 utils.py:299] [jaxline] experiment init starting...
I0511 06:51:31.291709 140050209154880 utils.py:306] [jaxline] experiment init finished.
I0511 06:51:31.330912 140050209154880 utils.py:299] [jaxline] training loop starting...
I0511 06:51:31.331103 140050209154880 experiment.py:148] Initializing parameters rather than restoring from checkpoint.
I0511 06:51:31.331202 140050209154880 experiment.py:176] num_devices: 1, per_device_batch_size: 1, global_batch_size: 1
I0511 06:51:31.331788 140050209154880 dataloader.py:244] Loaded dataset inscriptions: 19.
I0511 06:51:31.331945 140050209154880 dataloader.py:267] Inscriptions filtered: 0.
I0511 06:51:31.332022 140050209154880 dataloader.py:277] Final dataset inscriptions: 19.
2023-05-11 06:51:31.584100: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignor
         [[{{node Placeholder/_0}}]]
2023-05-11 06:51:31.584690: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignor
         [[{{node Placeholder/_0}}]]
```

100%_Ithaca.ipynb
File Edit View Insert Runtime Tools Help All changes saved
2m 3s completed at 12:38 PM



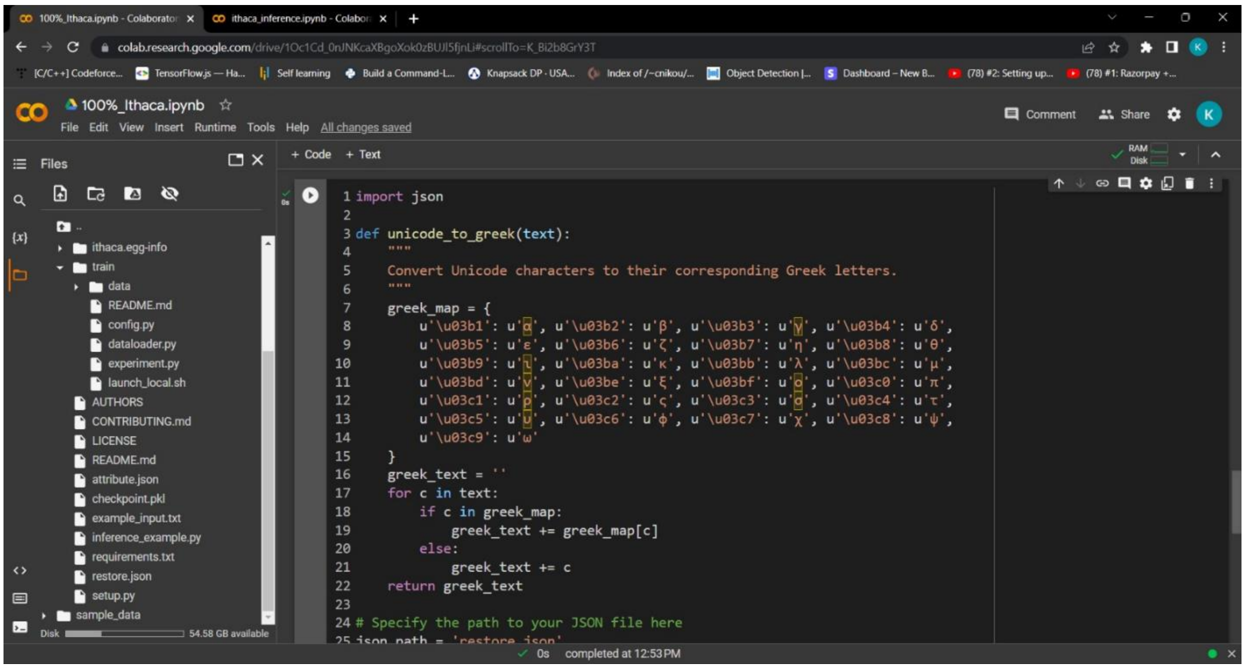
```
2023-05-11 06:51:31.584100: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignor
         [[{{node Placeholder/_0}}]]
2023-05-11 06:51:31.584690: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignor
         [[{{node Placeholder/_0}}]]
I0511 06:51:41.597503 140031494711040 experiment.py:176] num_devices: 1, per_device_batch_size: 1, global_batch_size: 1
I0511 06:51:41.598060 140031494711040 dataloader.py:244] Loaded dataset inscriptions: 19.
I0511 06:51:41.598212 140031494711040 dataloader.py:267] Inscriptions filtered: 0.
I0511 06:51:41.598288 140031494711040 dataloader.py:277] Final dataset inscriptions: 19.
2023-05-11 06:51:41.638295: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignor
         [[{{node Placeholder/_0}}]]
2023-05-11 06:51:41.638915: I tensorflow/core/common_runtime/executor.cc:1197] [/device:CPU:0] (DEBUG INFO) Executor start aborting (this does not indicate an error and you can ignor
         [[{{node Placeholder/_0}}]]
/usr/local/lib/python3.10/dist-packages/jaxline/utils.py:166: FutureWarning: jax.tree_flatten is deprecated, and will be removed in a future release. Use jax.tree_util.tree_flatten i
  leaves, treedef = jax.tree_flatten(sharded_tree)
/usr/local/lib/python3.10/dist-packages/jaxline/utils.py:169: FutureWarning: jax.tree_unflatten is deprecated, and will be removed in a future release. Use jax.tree_util.tree_unflatt
  [jax.tree_unflatten(treedef, [l[i] for l in leaves]) for i in range(n)],
I0511 06:52:56.751077 140031452747520 train.py:38] global_step: 1, {'accuracy/mask': array(0., dtype=float32), 'accuracy/nsp': array(0., dtype=float32), 'accuracy/subregion': array(
I0511 06:53:56.673346 140031452747520 train.py:38] global_step: 717, {'accuracy/mask': array(0.16666666, dtype=float32), 'accuracy/nsp': array(0.99999905, dtype=float32), 'accuracy/s
I0511 06:54:56.680604 140031452747520 train.py:38] global_step: 1417, {'accuracy/mask': array(0., dtype=float32), 'accuracy/nsp': array(0., dtype=float32), 'accuracy/subregion': arra
I0511 06:55:56.753323 140031452747520 train.py:38] global_step: 2107, {'accuracy/mask': array(0., dtype=float32), 'accuracy/nsp': array(0.99999905, dtype=float32), 'accuracy/subregio
I0511 06:56:31.360331 140050209154880 utils.py:578] Saved checkpoint latest with id 0.
I0511 06:56:56.756064 140031452747520 train.py:38] global_step: 2791, {'accuracy/mask': array(0., dtype=float32), 'accuracy/nsp': array(0., dtype=float32), 'accuracy/subregion': arra
I0511 06:57:56.824517 140031452747520 train.py:38] global_step: 3474, {'accuracy/mask': array(0.14285713, dtype=float32), 'accuracy/nsp': array(0.99999905, dtype=float32), 'accuracy/
I0511 06:58:56.890081 140031452747520 train.py:38] global_step: 4156, {'accuracy/mask': array(0.99999905, dtype=float32), 'accuracy/nsp': array(0.99999905, dtype=float32), 'accuracy/
I0511 06:59:56.917616 140031452747520 train.py:38] global_step: 4837, {'accuracy/mask': array(0., dtype=float32), 'accuracy/nsp': array(0.99999905, dtype=float32), 'accuracy/subregio
I0511 07:00:56.987837 140031452747520 train.py:38] global_step: 5518, {'accuracy/mask': array(0.11111111, dtype=float32), 'accuracy/nsp': array(0.99999905, dtype=float32), 'accuracy/
I0511 07:01:31.374714 140050209154880 utils.py:578] Saved checkpoint latest with id 1.
I0511 07:01:39.426981 140050209154880 utils.py:306] [jaxline] training loop finished.
I0511 07:01:39.427280 140050209154880 utils.py:299] [jaxline] final checkpoint starting...
I0511 07:01:39.427478 140050209154880 utils.py:578] Saved checkpoint latest with id 2.
I0511 07:01:39.427547 140050209154880 utils.py:306] [jaxline] final checkpoint finished.
I0511 07:01:39.427630 140050209154880 utils.py:299] [jaxline] rendezvous starting...
I0511 07:01:39.505409 140050209154880 utils.py:306] [jaxline] rendezvous finished.
```



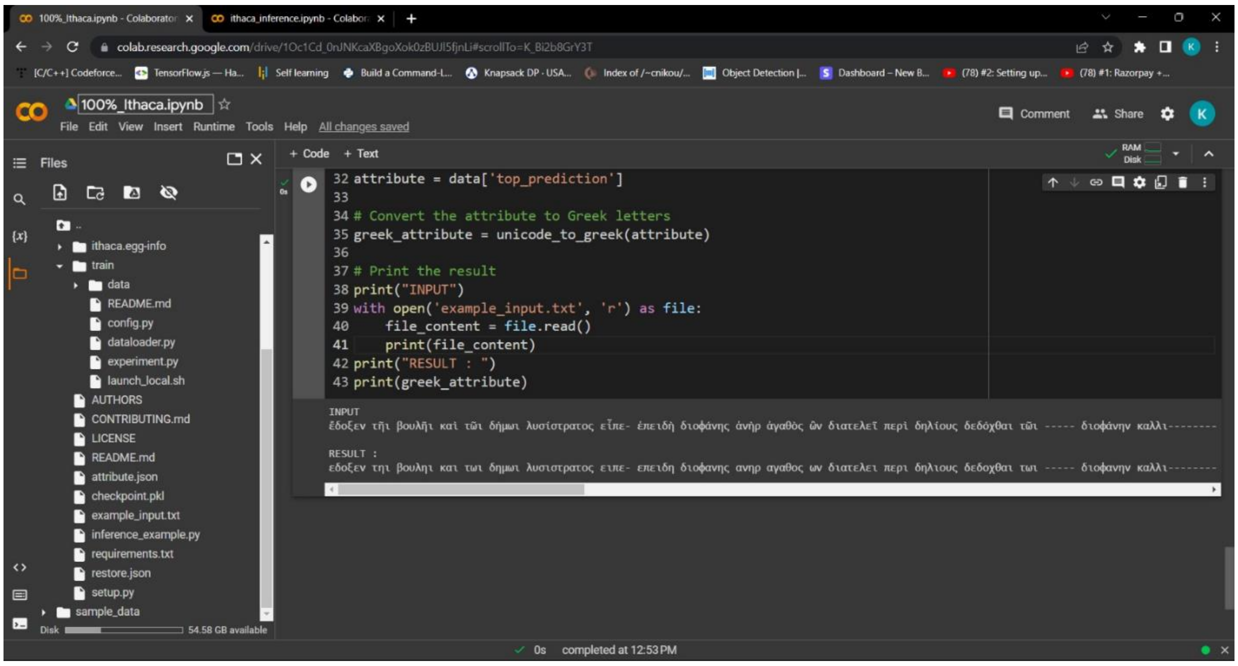
```
!python inference_example.py \
  --input_file=example_input.txt \
  --attribute_json=attribute.json \
  --restore_json=restore.json
```

```
I0511 07:06:38.013158 139728356886336 xla_bridge.py:440] Unable to initialize backend 'rocm': NOT_FOUND: Could not find registered platform with name: "rocm". Available platform names
I0511 07:06:38.013623 139728356886336 xla_bridge.py:440] Unable to initialize backend 'tpu': module 'jaxlib.xla_extension' has no attribute 'get_tpu_client'
I0511 07:06:38.013756 139728356886336 xla_bridge.py:440] Unable to initialize backend 'plugin': xla_extension has no attributes named get_plugin_device_client. Compile TensorFlow with
/usr/local/lib/python3.10/dist-packages/jax/_src/numpy/lax_numpy.py:3492: UserWarning: 'kind' argument to argsort is ignored; only 'stable' sorts are supported.
  warnings.warn("'kind' argument to argsort is ignored; only 'stable' sorts "
```

Results:



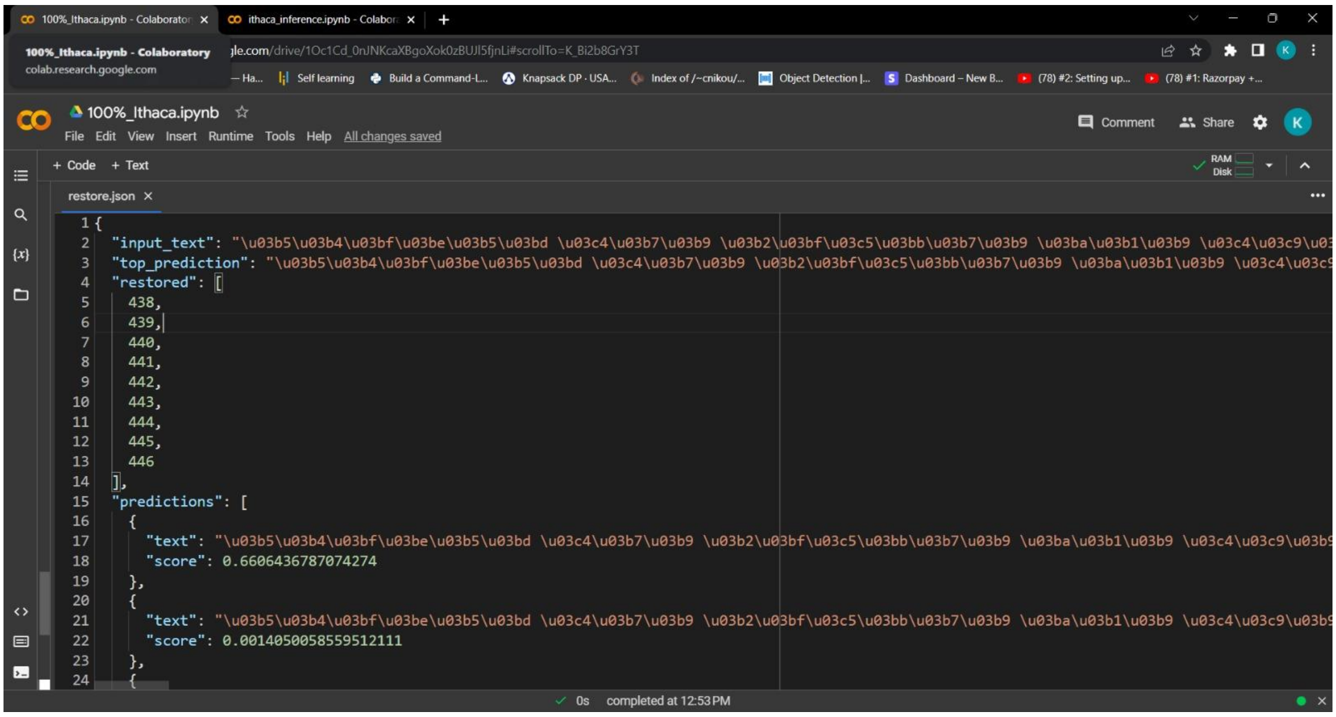
```python
import json

def unicode_to_greek(text):
    """
    Convert Unicode characters to their corresponding Greek letters.
    """
    greek_map = {
        u'\u03b1': u'α', u'\u03b2': u'β', u'\u03b3': u'γ', u'\u03b4': u'δ',
        u'\u03b5': u'ε', u'\u03b6': u'ζ', u'\u03b7': u'η', u'\u03b8': u'θ',
        u'\u03b9': u'ι', u'\u03ba': u'κ', u'\u03bb': u'λ', u'\u03bc': u'μ',
        u'\u03bd': u'ν', u'\u03be': u'ξ', u'\u03bf': u'ο', u'\u03c0': u'π',
        u'\u03c1': u'ρ', u'\u03c2': u'ς', u'\u03c3': u'σ', u'\u03c4': u'τ',
        u'\u03c5': u'υ', u'\u03c6': u'φ', u'\u03c7': u'χ', u'\u03c8': u'ψ',
        u'\u03c9': u'ω'
    }
    greek_text = ''
    for c in text:
        if c in greek_map:
            greek_text += greek_map[c]
        else:
            greek_text += c
    return greek_text

# Specify the path to your JSON file here
json_path = 'restore.json'
```



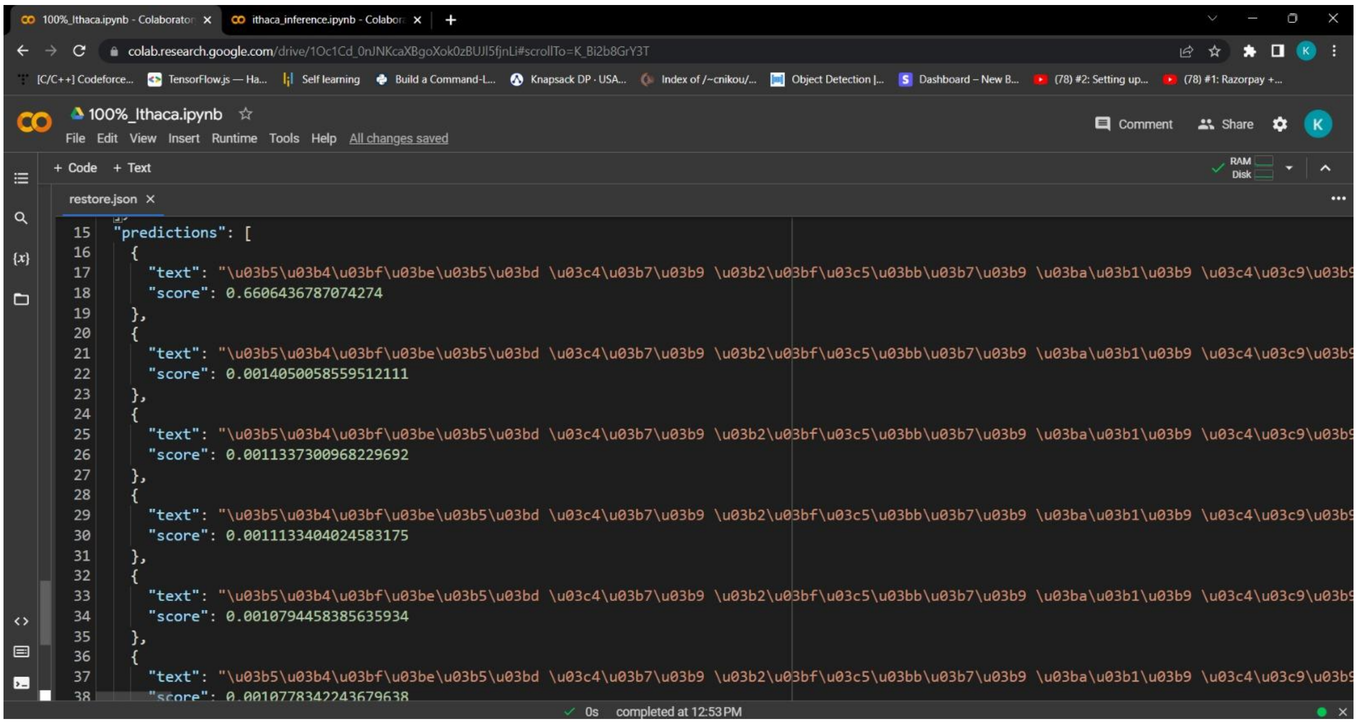
```python
attribute = data['top_prediction']

# Convert the attribute to Greek letters
greek_attribute = unicode_to_greek(attribute)

# Print the result
print("INPUT")
with open('example_input.txt', 'r') as file:
    file_content = file.read()
    print(file_content)
print("RESULT : ")
print(greek_attribute)
```

```
INPUT
ἔδοξεν τῆι βουλῆι καὶ τῶι δήμωι λυσίστρατος εἶπε- ἐπειδὴ διοφάνης ἀνὴρ ἀγαθὸς ὢν διατελεῖ περὶ δηλίους δεδόχθαι τῶι ----- διοφάνην καλλι-------

RESULT :
εδοξεν τηι βουληι και τωι δημωι λυσιστρατος ειπε- επειδη διοφανης ανηρ αγαθος ων διατελει περι δηλιους δεδοχθαι τωι ----- διοφανην καλλι-------
```

22



```
{
  "input_text": "\u03b5\u03b4\u03bf\u03be\u03b5\u03bd \u03c4\u03b7\u03b9 \u03b2\u03bf\u03c5\u03bb\u03b7\u03b9 \u03ba\u03b1\u03b9 \u03c4\u03c9\u03
  "top_prediction": "\u03b5\u03b4\u03bf\u03be\u03b5\u03bd \u03c4\u03b7\u03b9 \u03b2\u03bf\u03c5\u03bb\u03b7\u03b9 \u03ba\u03b1\u03b9 \u03c4\u03c9
  "restored": [
    438,
    439,
    440,
    441,
    442,
    443,
    444,
    445,
    446
  ],
  "predictions": [
    {
      "text": "\u03b5\u03b4\u03bf\u03be\u03b5\u03bd \u03c4\u03b7\u03b9 \u03b2\u03bf\u03c5\u03bb\u03b7\u03b9 \u03ba\u03b1\u03b9 \u03c4\u03c9\u03b9
      "score": 0.6606436787074274
    },
    {
      "text": "\u03b5\u03b4\u03bf\u03be\u03b5\u03bd \u03c4\u03b7\u03b9 \u03b2\u03bf\u03c5\u03bb\u03b7\u03b9 \u03ba\u03b1\u03b9 \u03c4\u03c9\u03b9
      "score": 0.001405005855951211
    },
    {
```



```
  "predictions": [
    {
      "text": "\u03b5\u03b4\u03bf\u03be\u03b5\u03bd \u03c4\u03b7\u03b9 \u03b2\u03bf\u03c5\u03bb\u03b7\u03b9 \u03ba\u03b1\u03b9 \u03c4\u03c9\u03b9
      "score": 0.6606436787074274
    },
    {
      "text": "\u03b5\u03b4\u03bf\u03be\u03b5\u03bd \u03c4\u03b7\u03b9 \u03b2\u03bf\u03c5\u03bb\u03b7\u03b9 \u03ba\u03b1\u03b9 \u03c4\u03c9\u03b9
      "score": 0.001405005855951211
    },
    {
      "text": "\u03b5\u03b4\u03bf\u03be\u03b5\u03bd \u03c4\u03b7\u03b9 \u03b2\u03bf\u03c5\u03bb\u03b7\u03b9 \u03ba\u03b1\u03b9 \u03c4\u03c9\u03b9
      "score": 0.0011337300968229692
    },
    {
      "text": "\u03b5\u03b4\u03bf\u03be\u03b5\u03bd \u03c4\u03b7\u03b9 \u03b2\u03bf\u03c5\u03bb\u03b7\u03b9 \u03ba\u03b1\u03b9 \u03c4\u03c9\u03b9
      "score": 0.0011133404024583175
    },
    {
      "text": "\u03b5\u03b4\u03bf\u03be\u03b5\u03bd \u03c4\u03b7\u03b9 \u03b2\u03bf\u03c5\u03bb\u03b7\u03b9 \u03ba\u03b1\u03b9 \u03c4\u03c9\u03b9
      "score": 0.0010794458385635934
    },
    {
      "text": "\u03b5\u03b4\u03bf\u03be\u03b5\u03bd \u03c4\u03b7\u03b9 \u03b2\u03bf\u03c5\u03bb\u03b7\u03b9 \u03ba\u03b1\u03b9 \u03c4\u03c9\u03b9
      "score": 0.0010778342243679638
```

INPUT TEXT

ἔδοξεν τῆι βουλῆι καὶ τῶι δήμωι λυσίστρατος εἶπε- ἐπειδὴ διοφάνης ἀνὴρ ἀγαθὸς ὢν διατελεῖ περὶ δηλίους δεδόχθαι τῶι ----- διοφάνην καλλι-------- --ηναῖον πρόξενον εἶναι δ--------- αὐτὸγ καὶ ἐκγόνους κ-- εἶναι αὐτοῖς ἀτέλειαν ἐν δήλωι πάντων καὶ γῆς καὶ οἰκίας ἔγκτησιν καὶ πρόσοδον πρὸς τὴμ βουλὴγ καὶ τὸν δῆμον

πρώτοις μετὰ τὰ ἱερὰ καὶ τὰ ἄλλα ὅσα καὶ τοῖς ἄλλοις προξένοις καὶ εὐεργέταις τοῦ ἱεροῦ δέδοται παρὰ ---ίων ἀναγράψαι δὲ τόδε ?????????α τὴν βουλὴν εἰς ----------ριον τοὺς -ὲ ----------------------.

RESULT TEXT

εδοξεν τηι βουληι και τωι δημωι λυσιστρατος ειπε- επειδη διοφανης ανηρ αγαθος ων διατελει περι δηλιους δεδοχθαι τωι ----- διοφανην καλλι-------- --ηναιον προξενον ειναι δ--------- αυτογ και εκγονους κ-- ειναι αυτοις ατελειαν εν δηλωι παντων και γης και οικιας εγκτησιν και προσοδον προς τημ βουληγ και τον δημον πρωτοις μετα τα ιερα και τα αλλα οσα και τοις αλλοις προξενοις και ευεργεταις του ιερου δεδοται παρα ---ιων αναγραψαι δε τοδε το ψηφισμα την βουλην εις ----------ριον τους -ε ----------------------.

CHAPTER 5
CONCLUSION

With 60% of the project complete, we have implemented the first 4 modules of our project. We have trained the character recognizer for 3 epochs which gives 87% accuracy. Higher accuracy is expected after a greater number of iterations. We've also constructed the basic skeleton of majorly important model. We have downloaded the letters from the input epigraph and the text recognition module feeds on these input images to create a full text file of the epigraph.

5.1 FUTURE WORKS

Future work related to the project includes the implementation of the next modules. The overall body of the module needs to be developed more, and it has to be integrated with the preprocessing part of the project. Once the project is completed it will be of great use to researchers and archeologists, along with whose supervision, the model could perform exceptionally good, and unveiling the treasured information captured in those epigraphs.

APPENDIX

ALGORITHMS CODE SNIPPETS

1. IMAGE PROCESSING

```
image = cv2.imread('/content/Original.jpg')
cv2_imshow(image)
def correct_skew(image, delta=1, limit=5):
    def determine_score(arr, angle):
        data = inter.rotate(arr, angle, reshape=False, order=0)
        histogram = np.sum(data, axis=1)
        score = np.sum((histogram[1:] - histogram[:-1]) ** 2)
        return histogram, score

    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1
]

    scores = []
    angles = np.arange(-limit, limit + delta, delta)
    for angle in angles:
        histogram, score = determine_score(thresh, angle)
        scores.append(score)

    best_angle = angles[scores.index(max(scores))]

    (h, w) = image.shape[:2]
    center = (w // 2, h // 2)
    M = cv2.getRotationMatrix2D(center, best_angle, 1.0)
    rotated = cv2.warpAffine(image, M, (w, h), flags=cv2.INTER_CUBIC, \
              borderMode=cv2.BORDER_REPLICATE)

    return best_angle, rotated
angle, rotated = correct_skew(image)
print(angle)
cv2.imwrite('rotated.jpg', rotated)
gray = cv2.cvtColor(rotated,cv2.COLOR_BGR2GRAY)
thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV + cv2.THRESH_OTSU)[1]
cv2_imshow(gray)

#applying median filter for Salt and pepper/impulse noise
filter1 = cv2.medianBlur(gray,5)

#applying gaussian blur to smoothen out the image edges
filter2 = cv2.GaussianBlur(filter1,(5,5),0)

#applying non-localized means for final Denoising of the image
dst = cv2.fastNlMeansDenoising(filter2,None,17,9,17)
```

```
#converting the image to binarized form using adaptive thresholding
th1 = cv2.adaptiveThreshold(dst,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINAR
Y,11,2)

cv2_imshow(th1)
```

2. CHARACTER SEGMENTATION

```
import cv2
import numpy as np
import imutils

image = cv2.imread("ImagePreProcessingFinal.jpg")
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
gray = cv2.GaussianBlur(gray, (7, 7), 0)

ret,thresh1 = cv2.threshold(gray ,127,255,cv2.THRESH_BINARY_INV)

dilate = cv2.dilate(thresh1, None, iterations=2)

cnts = cv2.findContours(dilate.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
cnts = cnts[0] # if imutils.is_cv2() else cnts[1]

sorted_ctrs = sorted(cnts, key=lambda ctr: cv2.boundingRect(ctr)[0] + cv2.boundingR
ect(ctr)[1] * image.shape[1] )
orig = image.copy()
i = 0
for cnt in sorted_ctrs:
    # Check the area of contour, if it is very small ignore it
    if(cv2.contourArea(cnt) < 200):
        continue

    # Filtered countours are detected
    x,y,w,h = cv2.boundingRect(cnt)

    # Taking ROI of the cotour
    roi = image[y:y+h, x:x+w]

    # Mark them on the image if you want
    cv2.rectangle(orig,(x,y),(x+w,y+h),(0,255,0),2)

    # Save your contours or characters
    cn = "Images/roi" + str(i) + ".png"
    cv2.imwrite(cn, roi)
    #print(cn)
    #cv2_imshow(roi)
    i = i + 1
```

3. TEXT RECOGNITION

```
class Process(object):
    def __call__(self, img):
        converted = img.convert("L")
        inverted = ImageOps.invert(converted)
        thick = inverted.filter(ImageFilter.MaxFilter(5))
        ratio = 48.0 / max(thick.size)
        new_size = tuple([int(round(x*ratio)) for x in thick.size])
        res = thick.resize(new_size, Image.LANCZOS)

        arr = np.asarray(res)
        com = ndimage.measurements.center_of_mass(arr)
        result = Image.new("L", (64, 64))
        box = (int(round(32.0 - com[1])), int(round(32.0 - com[0])))
        result.paste(res, box)
        return result
transform = transforms.Compose([Process(), transforms.ToTensor(), transforms.Normal
ize((0.5), (0.5))])
train_dir = "/content/LettersDataset/data/processed/train"
test_dir = "/content/LettersDataset/data/processed/test"

training_set = datasets.ImageFolder(train_dir, transform)
print(len(training_set))
trainsize = int(round(0.8 * len(training_set)))
trainset, valset = torch.utils.data.random_split(training_set, [trainsize, len(trai
ning_set) - trainsize], generator=torch.Generator().manual_seed(42))
print(len(trainset))
print(len(valset))
testset = datasets.ImageFolder(test_dir, transform)
print(len(testset))
trainloader = torch.utils.data.DataLoader(trainset, batch_size=64, shuffle=True)
valloader = torch.utils.data.DataLoader(valset, batch_size=64, shuffle=True)
testloader = torch.utils.data.DataLoader(testset, batch_size=64, shuffle=True)
df = pd.read_csv('/content/LettersDataset/data/TamilChar.csv', header=0)
unicode_list = df["Unicode"].tolist()
char_list = []

for element in unicode_list:
    code_list = element.split()
    chars_together = ""
    for code in code_list:
        hex_str = "0x" + code
        char_int = int(hex_str, 16)
        character = chr(char_int)
        chars_together += character
    char_list.append(chars_together)

classes = []
for i in range(156):
    index = int(testset.classes[i])
    char = char_list[index]
    classes.append(char)

print(classes)

def imshow(img):
    img = img / 2 + 0.5
    npimg = img.numpy()
    plt.imshow(np.transpose(npimg, (1, 2, 0)))
    plt.show()


dataiter = iter(trainloader)
images, labels = next(dataiter)

imshow(torchvision.utils.make_grid(images[:4]))
print(' '.join('%5s' % classes[labels[j]] for j in range(4)))
```

4. Positional Recognition

```
from . import bigbird
from . import common_layers

import flax.linen as nn
import jax
import jax.numpy as jnp


class Model(nn.Module):
  """Transformer Model for sequence tagging."""
  vocab_char_size: int = 164
  vocab_word_size: int = 100004
  output_subregions: int = 85
  output_date: int = 160
  output_date_dist: bool = True
  output_return_emb: bool = False
  use_output_mlp: bool = True
  num_heads: int = 8
  num_layers: int = 6
  word_char_emb_dim: int = 192
  emb_dim: int = 512
  qkv_dim: int = 512
  mlp_dim: int = 2048
  max_len: int = 1024
  causal_mask: bool = False
  feature_combine_type: str = 'concat'
  posemb_combine_type: str = 'add'
  region_date_pooling: str = 'first'
  learn_pos_emb: bool = True
```

```python
use_bfloat16: bool = False
dropout_rate: float = 0.1
attention_dropout_rate: float = 0.1
activation_fn: str = 'gelu'
model_type: str = 'bigbird'

def setup(self):
  self.text_char_emb = nn.Embed(
      num_embeddings=self.vocab_char_size,
      features=self.word_char_emb_dim,
      embedding_init=nn.initializers.normal(stddev=1.0),
      name='char_embeddings')
  self.text_word_emb = nn.Embed(
      num_embeddings=self.vocab_word_size,
      features=self.word_char_emb_dim,
      embedding_init=nn.initializers.normal(stddev=1.0),
      name='word_embeddings')

@nn.compact
def __call__(self,
             text_char=None,
             text_word=None,
             text_char_onehot=None,
             text_word_onehot=None,
             text_char_emb=None,
             text_word_emb=None,
             padding=None,
             is_training=True):
  """Applies Ithaca model on the inputs."""

  if text_char is not None and padding is None:
    padding = jnp.where(text_char > 0, 1, 0)
  elif text_char_onehot is not None and padding is None:
    padding = jnp.where(text_char_onehot.argmax(-1) > 0, 1, 0)
  padding_mask = padding[..., jnp.newaxis]
  text_len = jnp.sum(padding, 1)

  if self.posemb_combine_type == 'add':
    posemb_dim = None
  elif self.posemb_combine_type == 'concat':
    posemb_dim = self.word_char_emb_dim
  else:
    raise ValueError('Wrong feature_combine_type value.')

  # Character embeddings
  if text_char is not None:
    x = self.text_char_emb(text_char)
  elif text_char_onehot is not None:
    x = self.text_char_emb.attend(text_char_onehot)
  elif text_char_emb is not None:
```

```
  x = text_char_emb
else:
  raise ValueError('Wrong inputs.')

# Word embeddings
if text_word is not None:
  text_word_emb_x = self.text_word_emb(text_word)
elif text_word_onehot is not None:
  text_word_emb_x = self.text_word_emb.attend(text_word_onehot)
elif text_word_emb is not None:
  text_word_emb_x = text_word_emb
else:
  raise ValueError('Wrong inputs.')

if self.feature_combine_type == 'add':
  x = x + text_word_emb_x
elif self.feature_combine_type == 'concat':
  x = jax.lax.concatenate([x, text_word_emb_x], 2)
else:
  raise ValueError('Wrong feature_combine_type value.')

# Positional embeddings
pe_init = common_layers.sinusoidal_init(
    max_len=self.max_len) if self.learn_pos_emb else None
x = common_layers.AddPositionEmbs(
    posemb_dim=posemb_dim,
    posemb_init=pe_init,
    max_len=self.max_len,
    combine_type=self.posemb_combine_type,
    name='posembed_input',
)(
    x)
x = nn.Dropout(rate=self.dropout_rate)(x, deterministic=not is_training)
```

REFERENCES

[1] Subramani, K. and Subramaniam, M., 2021. Creation of original Tamil character dataset through segregation of ancient palm leaf manuscripts in medicine. *Expert Systems*, *38*(1), p.e12538.

[2] TAMIZHİ: Historical Tamil-Brahmi Script Recognition Using CNN and MobileNet ACM Transactions on Asian and Low-Resource Language Information Processing Volume 20 Issue 3 Article No.: 39pp 1–26

[3] Zhao, H., Chu, H., Zhang, Y. and Jia, Y., 2020. Improvement of Ancient Shui character recognition model based on convolutional neural network. *IEEE Access*, *8*, pp.33080-33087. -----------

[4] Phan, H., He, Y., Savvides, M. and Shen, Z., 2020. Mobinet: A mobile binary network for image classification. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision* (pp. 3453-3462).

[5] Ancient Text Character Recognition Using Deep Learning
International Journal of Engineering Research and Technology. ISSN 0974-3154, Volume 13, Number 9 (2020), pp. 2177-2184 © International Research Publication House

[6] Yannis Assael, Thea Sommerschield, and Jonathan Prag. 2019. Restoring ancient text using deep learning: a case study on Greek epigraphy. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), pages 6368–6375, Hong Kong, China. Association for Computational Linguistics.

[7] Manigandan, T.V.V.D.V.N.B., Vidhya, V., Dhanalakshmi, V. and Nirmala, B., 2017, August. Tamil character recognition from ancient epigraphical inscription using OCR and NLP. In *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)* (pp. 1008-1011). IEEE. -----

[8] Soumya, A. and Kumar, G.H., 2014, November. Recognition of ancient Kannada Epigraphs using fuzzy-based approach. In *2014 International Conference on Contemporary Computing and Informatics (IC3I)* (pp. 657-662). IEEE.----------