Enhanced Extraction of Textual Characters from Multimedia using Natural Language Processing

Dinesh K[1], Bharath T[2], Sumanth B[3] ***Mentor*** : Dr P Kavitha – Associate Professor

Department of Computer Science and Engineering, R.M.K Engineering College Thiruvallur, Tamil Nadu

dine19139.cs@rmkec.ac.in[1], bhar19120.cs@rmkec.ac.in[2], budh19126.cs@rmkec.ac.in[3]

Abstract - Enhanced extraction of textual characters is a complete framework for detecting and recognising textual content in video frames and images. Text detection from document photos allows Natural Language Processing algorithms to read the text and understand what the document is saying. Existing scene text detection approaches are incapable of accurately detecting text in videos with low contrast, complex backgrounds, or excessively small fonts. Furthermore, the text may be simply translated into multiple languages, and NLP text summarising uses transformers and pipelines to summarise the material into meaningful sentences. The summarizer text is then converted to audio using the Data-To-Speech module. NLP allows them to recognise text from grocery product labels, highway signs, and even billboards, allowing them to be an excellent translation and interpreter. The creation of this programme, which will be of tremendous assistance to persons with visual impairments and those who do not speak a different language. impairments and those who do not speak another language.

Keywords - Text Detection, Machine Learning, UnSupervised Learning, easyocr, matplotlib, pyttsx3, spacy, numpy, warnings.

I. INTRODUCTION

The amount of digital multimedia data, particularly video content, has increased dramatically in recent years, both in the form of video archives and live streaming. According to statistics, 300 hours of video are posted to YouTube every minute. The availability of low-cost smart phones with cameras is a major contributor to this massive surge. With such massive data collections, efficient and effective retrieval mechanisms are required to allow users to get the desired content. The term content can refer to visual material (for example, objects or people in a video), audio content (for example, spoken terms), or textual content (news tickers, score cards, and so on).

Textual content in the clip can be divided into two categories: scene text and caption text. Advertisement banners, sign boards, and lettering on a T-shirt are all examples of scene text. Scene text is frequently used in applications like as robot navigation and assistive technologies for the visually impaired. The primary components of a textual content-based indexing and retrieval system are text region detection, text extraction (segmentation from backdrop), script identification (for multi-script films), and ultimately text recognition (video OCR).

Text detection can be accomplished by unsupervised, supervised, or hybrid methods. To distinguish between text and non-text regions, unsupervised text detection employs image analysis techniques. Supervised approaches, on the other hand, include training a learning algorithm to distinguish between text and non-text regions. In other circumstances, a combination of the two techniques is used, with candidate text sections found by unsupervised methods confirmed by supervised methods.

This research proposes a comprehensive framework for multi-script video text detection and recognition. The important findings of this study are summarised below.

• Creation of a large collection of video frames containing ground truth information to aid in the evaluation of detection and recognition tasks.

• Research into cutting-edge deep learning CRAFT Algorithm-based object detectors, fine-tuning them to recognise textual content.

• Extensive experimentation is used to validate proposed procedures.

The following is how this document is structured. In the following section, we offer the database created during our research as well as the ground truth data. Section 3 provides a summary of the current state of the art in text detection and recognition, summarization, translation, and speech representation. Section 4 describes the suggested framework and experimental protocol in detail. Section 5 defines the actual results and the discussion that follows. Finally, Section 6 brings the paper and future work to a close.

II. LITERATURE SURVEY

We addressed word detection in video pictures in the last section, where the methods take individual frames of video as input. Because the suggested method uses temporal information for text detection in video, we will review publications that use temporal information for text detection in this part.

Li et al. [9] suggested a method for tracking video text using wavelet and moment characteristics. This technique identifies text candidates by combining wavelet decomposition, geographical information provided by moments, and a neural network classifier. Huang et al. [4] proposed exploiting temporal frames to recognise scrolling text in video. For detecting text, this approach employs motion vector estimation. This approach, however, is confined to scrolling text and does not support arbitrarily positioned text. To define text regions, Zhou et al. [38] used edge information and geometrical restrictions to create a coarse-to-fine technique. The approach, however, is incapable of dealing with dynamic text objects. Mi et al. [15] suggested a multi-frame text extraction method. For identifying text candidates, the edge features are investigated using a similarity metric. Wang et al. [31] extracted text items from video documents using a spatio-temporal wavelet transform. A three-dimensional in shape wavelet conversion with one scaling function and seven wavelet functions is applied to a sequence of video frames during the edge detection step. Tools for Multimedia Appl Huang [3] used temporal redundancy to detect video scene text. Because of camera or object movement, video scene texts in consecutive frames show arbitrary motion. As a result, this approach detects motion in 30 frames in a row to create a motion image. Zhao et al. [37] suggested a method for word identification in video utilising corners.

This method suggests using dense corners to identify text candidates. Finally, optical flow has been employed to detect moving text. Liu et al. [12] suggested a method for detecting video caption text utilising stroke-like edges and spatio-temporal data. The colour histogram is used to segment text data. Li et al. [10] suggested a multiple frame integration approach for video text detection. To extract text candidates, our approach makes use of edge information. To extract final text information from video, morphological techniques and heuristic principles are proposed. Khare et al. [8] have suggested a method for multi-oriented text detection in video that makes use of temporal information. The method's major goal is to create a new descriptor called the Histogram Oriented Moments Descriptor (HOM), which works similarly to the Histogram Oriented Gradients (HOG) for text detection in video but does not need temporal coherency to detect text candidates. This is because the method use optical flow to detect moving text using temporal information, but not text candidate detection. To put it another way, the approach analyses a frame as an individual image while looking for text candidates, and then it uses temporal information to track moving text candidates in temporal frames. As a result, the method's efficiency suffers for static foregrounds and dynamic backgrounds since moving text candidates may overlap with moving backdrops. Wu et al. [32] suggested a similar method for multi-oriented scene text line identification and tracking in video. For finding text candidates, the approach investigates gradient directional symmetry and spatial proximity between text components. Text lines are tracked by matching text component sub-graphs. The method's ability to handle arbitrarily oriented text and multi-script text lines in video is not tested.

According to the preceding literature survey, most approaches focus on horizontal or non-horizontal text identification, with relatively few on arbitrarily-oriented text detection in video. As a result, the recognition of multi-script text lines in video has received insufficient attention in the literature. Using a Gaussian mixture model and learning, Liu et al. [14] proposed a method for multi-lingual text extraction from scene photos. This approach detects text lines using binarization and linked component analysis. This concept works for photos with high contrast but not for images with low contrast, such as video frames. Huang et al. [5] also attempts to address text detection multi-scripts by investigating temporal information in consecutive frames based on a motion perception field. The methods are only available in a few languages, including Chinese, English, and Japanese. As a result, we can conclude that the following challenges demand researchers' immediate attention: (1) A method that works without regard to language or direction; (2) determining the exact number of temporal frames for performing operations when the method uses successive frames in video; and (3) determining automatic window sizes in video rather than using fixed windows for performing operations over an image or video. These concerns prompted us to propose a new strategy for addressing the aforementioned challenges. We propose to explore moments in a new way without wavelet support for classifying static and dynamic text clusters using temporal frames, inspired by the work presented in [27] for multi-oriented text detection in video using the combination of wavelet and moments, where it is shown that moments help in successfully detecting text candidates for text. Because the stroke width distance for characters in the video is nearly the same in [34], we use the same stroke width distance for them.

determining window size automatically to make the method Multimedia Tools Appl insensitive to fonts, font size, and so on, whereas methods [4, 8, 9] employ a fixed window size. The majority of known temporal frame approaches use a set number based on predetermined tests. As a result, the approaches' goal is to leverage temporal frames to improve low contrast text information. Unlike the previous methods, the one described here investigates moments for measuring deviations between temporal frames in order to autonomously pick temporal frames for arbitrarily-oriented-multi-script text identification in video. Furthermore, unlike previous systems, which use a set number of frames for text detection or an individual frame [8, 26], the suggested method includes an iterative procedure for choosing the number of temporal frames to be used out of 25-30 frames per second. The same iterative method aids with caption and scene text classification. We propose boundary-growing without angle projection for arbitrarily-oriented text extraction in video, inspired on the boundary-growing approach proposed in [27] for multi-oriented text detection in video. The proposed solution has the main advantage of working for static text with static and moving backdrops, as well as moving text with static and moving backgrounds. Furthermore, the proposed approach is script independent and rotation insensitive.

III. METHODOLOGY

Our talk is divided into four major areas. We start with the text identification problem, then go on to summary strategies, and finally translation techniques. Finally, we highlight the current state-of-the-art and open challenges in speech representation.

Text Detection and Recognition :

The detection of candidate text sections from retrieved video frames is the initial step in the proposed approach. We used cutting-edge CRAFT Algorithm-based object detectors to detect textual information in a particular frame. Although many object detectors are trained with thousands of class instances and achieve great accuracy in object identification and recognition, these object detectors cannot be used to recognise text regions in images. These models must be tailored to the unique problem of distinguishing text from non-text regions. These models' convolutional basis can be learned from scratch, or known pre-trained models (VGG, Inception, or ResNet) can be fine-tuned by training them on text and non-text regions. In our research, we looked into the following object detectors for text region localization. The goal is to determine which of these is best suited to the text detection challenge.

OpenCV:

OpenCV is a C/C++-based open-source computer vision library. It is mostly concerned with image processing. OpenCV has almost 2500 optimised algorithms. These algorithms can detect and recognise faces and text, identify items, track moving things, and so on.

Easy OCR:

Easy OCR is written in Python and uses the Pytorch deep learning framework; having a GPU could speed up the detecting process. The CRAFT method is used for detection, and the CRNN model is used for recognition.

To Extract the text from the scene image*:*

Over the last few years, many deep learning models in computer vision have been applied for scene text detection. However, when the text in the image is slanted or curved, the performance of these models suffers. In addition to regular text, the CRAFT model operates well on curved, lengthy, and deformed texts. To calculate region and affinity scores, the CRAFT text identification model use a convolutional neural network model. The region score is used to localise character regions, whilst the affinity score is used to aggregate characters together into text regions. CRAFT employs a VGG16-based fully convolutional neural network. The inference is accomplished by supplying word-level bounding boxes. The CRAFT text identification model is effective on unseen datasets and works well at many scales, from huge to small texts. When compared to the CRAFT text detection engine, the CTPN, EAST, and MSER text detection algorithms consume less time. When the text is long, curved, rotated, or deformed, CRAFT is more accurate, and the bounding boxes are more precise.

Optical Character Recognition (OCR) is a system that can recognise characters or text in a two-dimensional image. Text could be machine-printed or handwritten in the image. OCR can recognise a variety of languages, including English, Hindi, and German. Tesseract outperforms Easy OCR on CPU, but Easy OCR outperforms Tesseract on GPU.

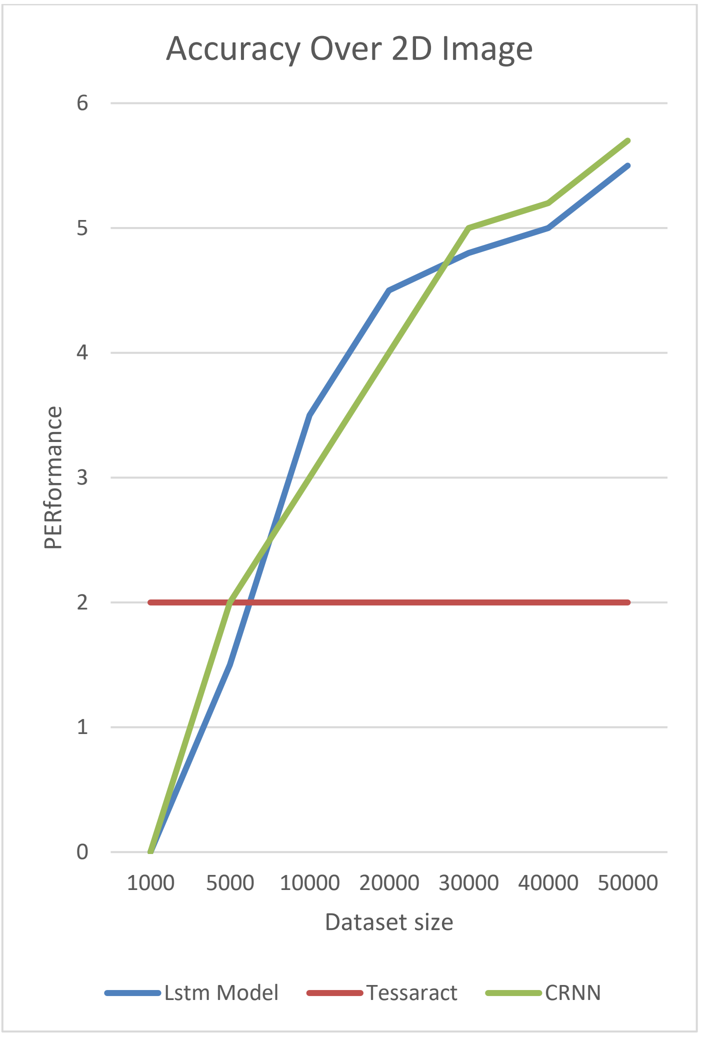


Figure (a)

To Recognize the text from bounded region:

Following the text detection process, regions with text are clipped and routed through convolutional layers to extract picture features. These features are then fed into a many-to-many LSTM architecture, which produces soft max probabilities throughout the lexicon. These outputs from various time steps are given into the CTC decoder, which produces the raw text from images. This is analogous to training any LSTM model with word embeddings such as word2vec, Glove, or rapid Text. We are creating a sequential model in the modelling. The first layer of the model is the embedding layer, which employs a 32-length vector, and the second layer is the LSTM layer, which employs 100 neurons as the model's memory unit. Following LSTM, the dense output layer with sigmoid function, assists in labelling.

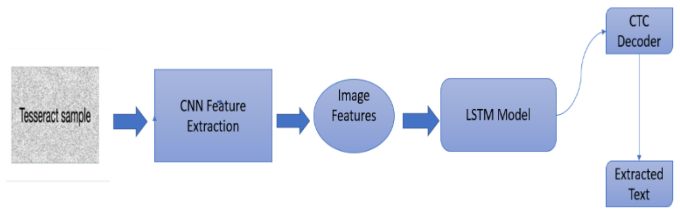


Figure (b)

The final fine-tuned model is fed a video frame and localises the text patches (Fig. 8). Once the text has been localised, the detected parts are given into the summary module, which summarises the detected text's script.

Summarization Techniques :

Text summarization is a natural language processing (NLP) operation that allows users to quickly summarise vast quantities of text without losing any vital information.

Spacy :

Spacy pipeline item for negating textual notions using the Neg Ex algorithm.

To Tokenize the recognized text :

The NegEx algorithm is based on four kinds of lexical cues: negation triggers (e.g., "denies"), pseudo-negation triggers (e.g., "no increase"), and termination terms (e.g., "but"). Any clinical state that falls under the purview of a negation trigger is negated. All NegEx lexical elements have an action—negation and pseudo-negation triggers can change information to the right (ahead in the sentence) or left (backward in the phrase). Termination terms include an action to end scope, which otherwise ends at the end of the sentence. Pseudo negation triggers try to compensate for NegEx's lack of syntax by listing exceptions to the occurrence of negation triggers like "no" in phrases like "no previous". Because the lexicon is the foundation of the algorithm, translating the lexical cues is the primary means of adapting Neg Ex to different languages.

NegEx was recently ported and tested on clinical texts in Swedish [6] and French [7], yielding decent results (recall 82%; precision 75%) for Swedish assessment sections of the Stockholm EPR corpus and higher results (recall 85%; precision 89%) for French cardiology notes. The customised NegEx systems achieved equivalent recall to the English NegEx in both studies, with visible changes in precision (differences of 9.3% and 4.4%, respectively).

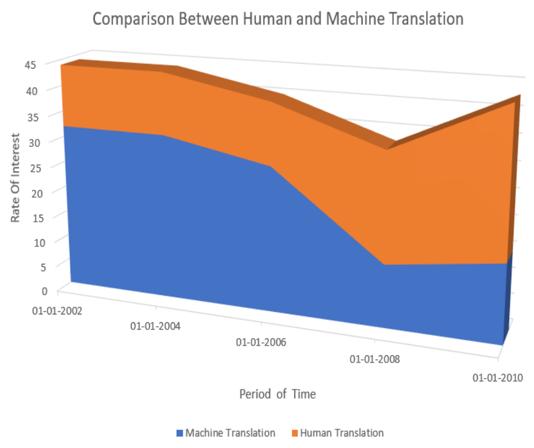


Figure (c)

A person might approach the task of summarising a document in the following way: 1. Read the entire book, 2. Understand the concepts being presented, 3. Highlight the most significant elements, and 4. Simplify them more concisely.

A semantic understanding of the text is required for a computer to execute the same work. While semantic analysis is possible with current NLP algorithms, it frequently necessitates a large amount of processing power and provides results that are comparable in quality to other extractive techniques.

Extractive summarization, rather than interpreting the text, relies on quantitative measurements created from the text itself, with no foreign meaning attached. Our method is straightforward:

1. Examine the frequency with which specific words are used.

2. Add up the frequencies in each sentence.

3. Sort the sentences depending on this total.

Of course, we assume that a more frequently used word conveys a more'significant' meaning. This may appear unduly basic, yet it frequently provides very effective results.

Translation Techniques :

We are frequently faced with a scenario in which we cannot understand a vital document or we must convey information to speakers of another language. Although online translation tools are available, they are not always accessible, and it may be preferable to provide a static translated page for our clients. To translate text from one language to another in Python, we will use the googletrans module, which has been called internally to the Google Translate API.

Googletrans:

Since then, Google Translate has been employing neural machine translation (NMT) instead of its previous statistical methods (SMT) that it had been using since October 2007 with its proprietary, in-house SMT technology. The NMT system used by Google Translate is a massive artificial neural network capable of deep learning.

To Translate the summarized part :

Neural Machine Translation (NMT) has emerged as the most powerful algorithm to do this task thanks to the strength of deep learning. While Google Translate is the most visible industrial example of NMT, tech companies all around the world are investing heavily in NMT. This cutting-edge technique is a deep learning programme that uses enormous datasets of translated sentences to create a model capable of translating between any two languages. The Encoder Decoder structure is one of the oldest and more established forms of NMT. We can repeat this procedure on every sentence in each language by building a vocabulary for both the input and output languages, allowing us to entirely change any corpus of translated sentences into a format fit for the task of machine translation.

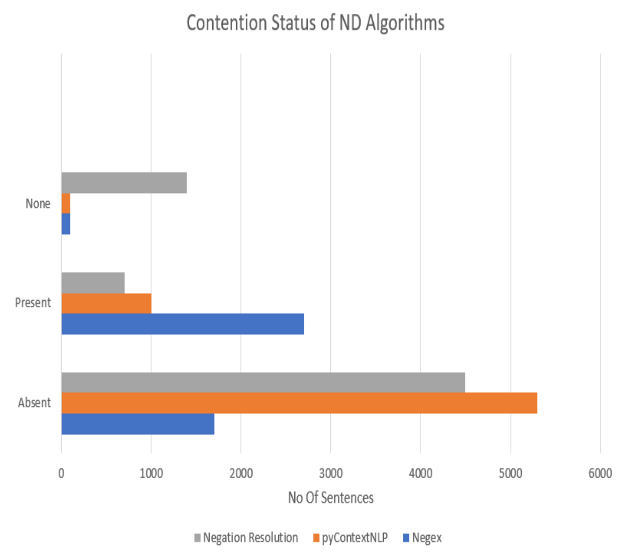


Figure (d)

Now that we know how to represent textual data numerically. At its most basic, the Encoder component of the model receives a sentence in the input language and generates a thought vector from it. This thought vector holds the sentence's meaning and is then handed to a Decoder, which provides the sentence's translation in the output language. The method is depicted in the diagram below.

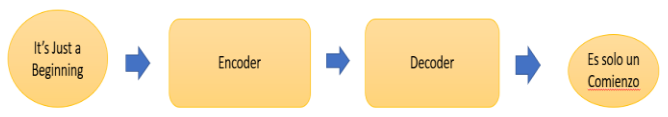


Figure (e)

Googletrans is a Python package that uses the Google Translate API and is completely free. This makes use of the Google Translate Ajax API to call methods like detect and translate. In reality, when it comes to translation accuracy, it's one of the top-rated translation tools, albeit the exact accuracy will rely on the language pairs you've picked.

Speech Representation :

The TTS system receives text as input, and then a computer algorithm known as the TTS engine examines the text, pre-processes the text, and synthesises the voice using mathematical models. As an output, the TTS engine typically generates sound data in an audio format.

How TTS Engine works :

The Acoustic feature (Mel-spectrogram) is converted to waveform output (audio). It is possible to use a mathematical model, such as Gryphon Lim's, or to train a neural network to learn the mapping from mel-spectrogram to waveforms. Learning-based strategies typically outperform the Gryphon Lim method. So, rather than predicting waveforms directly with the decoder, we divided this complicated and intricate work into two stages, first predicting mel-spectrogram from Latent processed data and then creating audio using mel-spectrogram.

You are now familiar with all of the essential components of a TTS system. We can deduce that you are ready to study and learn from advanced research papers such as Fastspeech, Tacotron, WaveNet, and others. Check out these audio clips to get a sense of how TTS works in practise.

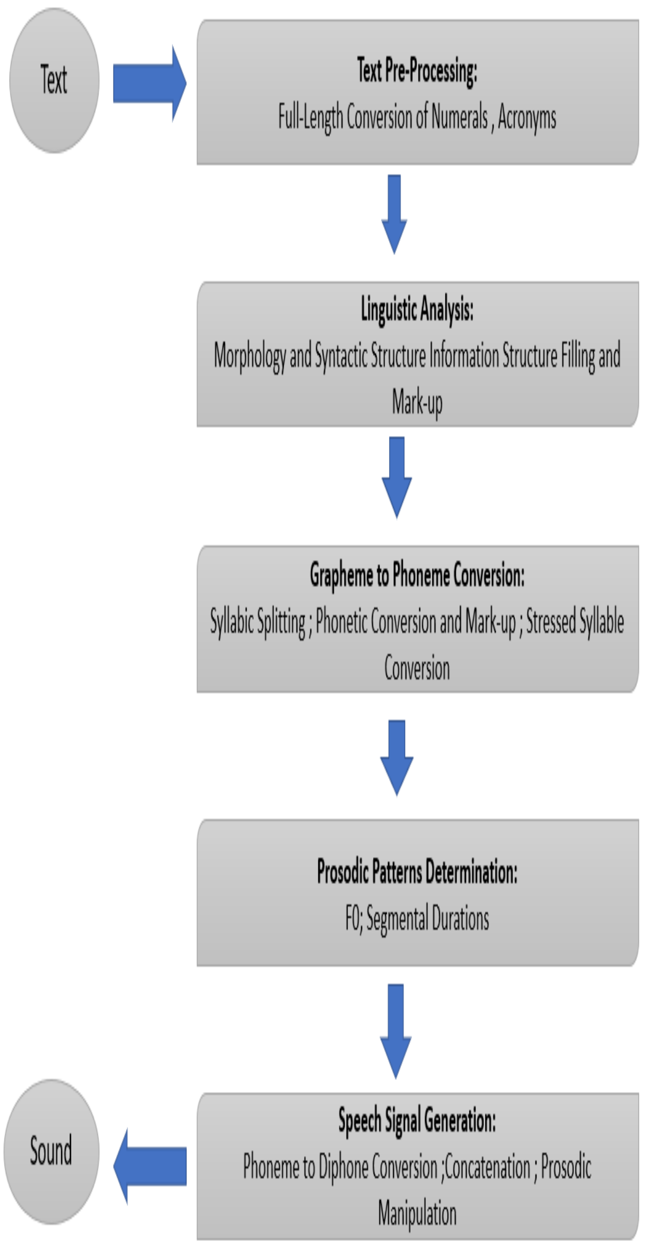

Text
Text Pre-Processing:
Full-Length Conversion of Numerals , Acronyms
Linguistic Analysis:
Morphology and Syntactic Structure Information Structure Filling and Mark-up
Grapheme to Phoneme Conversion:
Syllabic Splitting ; Phonetic Conversion and Mark-up ; Stressed Syllable Conversion
Prosodic Patterns Determination:
F0; Segmental Durations
Speech Signal Generation:
Phoneme to Diphone Conversion ;Concatenation ; Prosodic Manipulation
Sound

Figure (f)

Pyttsx3 is the Advanced module for speech conversion. It is a Python text-to-speech conversion package. It operates offline, unlike other libraries, and is compatible with Python 2 and 3. To obtain a reference to a pyttsx3. Engine instance, an application calls the pyttsx3.init() factory method. It is a simple tool that turns the entered text into speech.

IV. PROPOSED SYSTEM AND IMPLEMENTATION

This section describes the proposed framework in depth. The total system is made up of four key modules: text detector, text summarization, text translation, and speech representation. At the application layer, a variety of systems like as indexing and retrieval, key-word-based alert generation, and content summarising can be built on top of these modules. The text detector module is in charge of recognising and localising all textual material in a frame. Because text can appear in more than one script (within the same frame), the discovered textual sections are sent to the summary module, which summarises the text lines as the desired amount of content. The summarised text can also be translated into our preferred languages before being delivered to the various recognition engines of each script to turn the text into speech, which can then be used for a variety of applications. Each of these components is explored in depth below.

A) Text Detection and Recognition

Using OpenCV and OCR, we will create a text detector and extractor from picture or video frames in this Python project. Text detection techniques are often divided into unsupervised and supervised approaches. Unsupervised approaches rely on image analysis techniques to separate text from background, whereas supervised methods entail training a learning algorithm to distinguish between text and non-text regions. Unsupervised text identification algorithms include edge-based methods [2, 3, 16] that (assume and) exploit text-to-background contrast. Texture-based approaches [19, 20] view textual material in the image as a unique texture that separates itself from non-text regions, whereas linked component-based methods [17, 18] depend mostly on the color/intensity of text pixels.

OpenCV

It can first read the image from the given input and then modify and store the image's colour space in a variable. We use the function cv2.cvtColor(input_image, flag) to convert colours. The type of conversion is determined by the second parameter flag.We have the option of using cv2.COLOR_BGR2GRAY or cv2.COLOR R_BGR2HSV. In this case, we use cv2.COLOR_BGR2GRAY to convert an RGB image to a grayscale image. The cv2.threshold function is used to apply a threshold to the converted image. Thresholding is a popular segmentation technique for distinguishing a foreground object from its background. Binary Thresholding is the most fundamental Thresholding approach. The same threshold value is used for each pixel. If the pixel value is less than the threshold, it is set to zero, otherwise it is set to the maximum value.

We use the rectangular structural element (cv2.MORPH_RECT) to get a rectangular Structure. cv2.getStructuringElement requires an extra kernel parameter size.

Easy OCR

EasyOCR is implemented in Python and the PyTorch library. If you have a CUDA-capable GPU, the underlying PyTorch Deep Learning Library can greatly accelerate your text detection and OCR speed. EasyOCR Can Create OCR Texts In Over 70 Languages, Including English, Hindi, Russian, Chinese, and Others. EasyOCR Is Ideal For Cleaning

Document Scanning Would Improve Accuracy And Support For LSTM.

1. Install EasyOCR and the libraries required to open an image and recognise it.

2. Choose the language in which you want the text to be extracted.

3. Read and open the image from which you want to extract the text.

4. Determine the box bounds accuracy for the text in the image.

5. Draw the text box bounds in the image.

6. Output

Advantages :

• EasyOCR supports the GPU version, and the GPU performance is good.

• EasyOCR works better with noisy images.

• EasyOCR provides confidence in the extracted text, which can be used for further analysis.

B) Summarization Techniques

Text summarization is a Natural Language Processing (NLP) job that summarises information in huge texts so that it can be consumed more quickly without losing important information. Text summarising can be done in two ways: extractive text summarization and abstractive text summarization.

Extractive Text Summarization identifies noteworthy sentences and then adds them to the summary, which contains exact sentences from the original text.

Abstractive Text Summarization tries to identify relevant sections, evaluate the context, and provide an intelligent summary.

In this post, we'll use the abstractive approach to text extracted from a picture.

Spacy :

We'll utilise SpaCy to import a pre-trained NLP pipeline to assist in interpreting the text's grammatical structure. This will help us to determine the most common terms to filter out (i.e. STOP_WORDS) as well as the punctuation (i.e. punctuation). The n biggest function will also be used to extract a proportion of the most essential sentences. The following stages will be taken by our algorithm:

• Use the SpaCy pipeline to tokenize the text. Using grammatical principles specific to the English language, this segmented the text into words, punctuation, and so on.

• Count the number of times a word appears (without stopping words or punctuation), then normalise the count. A more commonly used term has a greater normalised count.

• Add together the normalised counts for each sentence.

• Take a percentage of the top-ranked sentences. These are our highlights.

C) Translation Techniques

To translate text from one language to another in Python, we will use the googletrans module, which has been called internally to the Google Translate API.

We imported the necessary package. We generated a Translator class instance and assigned it to the translator variable. We completed the text translation. (If the source language is not specified, Google Translate attempts to detect it. If no target language is specified, your material will be translated to English by default.) The translated text was printed. We translated the content from Korean to English by defining the destination, or target language, as en, and then printed it. Similarly, we can translate the English text to Tamil or any other language by providing the destination language.

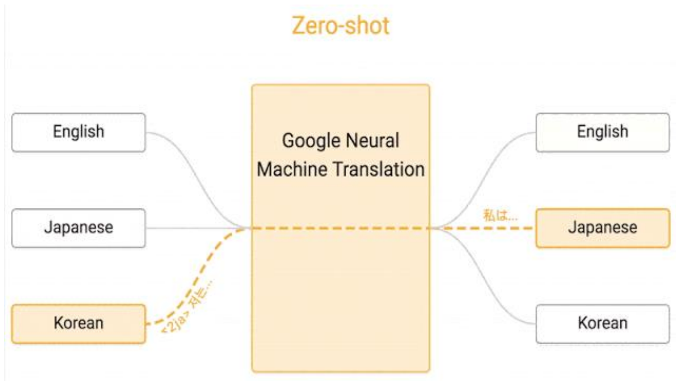
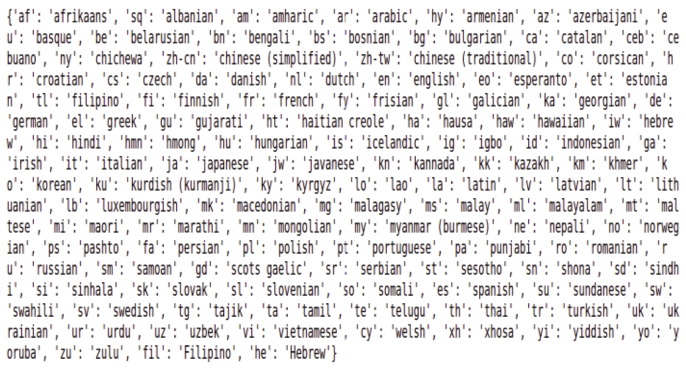


Figure (g)



```
{'af': 'afrikaans', 'sq': 'albanian', 'am': 'amharic', 'ar': 'arabic', 'hy': 'armenian', 'az': 'azerbaijani', 'e
u': 'basque', 'be': 'belarusian', 'bn': 'bengali', 'bs': 'bosnian', 'bg': 'bulgarian', 'ca': 'catalan', 'ceb': 'ce
buano', 'ny': 'chichewa', 'zh-cn': 'chinese (simplified)', 'zh-tw': 'chinese (traditional)', 'co': 'corsican', 'h
r': 'croatian', 'cs': 'czech', 'da': 'danish', 'nl': 'dutch', 'en': 'english', 'eo': 'esperanto', 'et': 'estonia
n', 'tl': 'filipino', 'fi': 'finnish', 'fr': 'french', 'fy': 'frisian', 'gl': 'galician', 'ka': 'georgian', 'de':
'german', 'el': 'greek', 'gu': 'gujarati', 'ht': 'haitian creole', 'ha': 'hausa', 'haw': 'hawaiian', 'iw': 'hebre
w', 'hi': 'hindi', 'hmn': 'hmong', 'hu': 'hungarian', 'is': 'icelandic', 'ig': 'igbo', 'id': 'indonesian', 'ga':
'irish', 'it': 'italian', 'ja': 'japanese', 'jw': 'javanese', 'kn': 'kannada', 'kk': 'kazakh', 'km': 'khmer', 'k
o': 'korean', 'ku': 'kurdish (kurmanji)', 'ky': 'kyrgyz', 'lo': 'lao', 'la': 'latin', 'lv': 'latvian', 'lt': 'lith
uanian', 'lb': 'luxembourgish', 'mk': 'macedonian', 'mg': 'malagasy', 'ms': 'malay', 'ml': 'malayalam', 'mt': 'mal
tese', 'mi': 'maori', 'mr': 'marathi', 'mn': 'mongolian', 'my': 'myanmar (burmese)', 'ne': 'nepali', 'no': 'norweg
ian', 'ps': 'pashto', 'fa': 'persian', 'pl': 'polish', 'pt': 'portuguese', 'pa': 'punjabi', 'ro': 'romanian', 'r
u': 'russian', 'sm': 'samoan', 'gd': 'scots gaelic', 'sr': 'serbian', 'st': 'sesotho', 'sn': 'shona', 'sd': 'sindh
i', 'si': 'sinhala', 'sk': 'slovak', 'sl': 'slovenian', 'so': 'somali', 'es': 'spanish', 'su': 'sundanese', 'sw':
'swahili', 'sv': 'swedish', 'tg': 'tajik', 'ta': 'tamil', 'te': 'telugu', 'th': 'thai', 'tr': 'turkish', 'uk': 'uk
rainian', 'ur': 'urdu', 'uz': 'uzbek', 'vi': 'vietnamese', 'cy': 'welsh', 'xh': 'xhosa', 'yi': 'yiddish', 'yo': 'y
oruba', 'zu': 'zulu', 'fil': 'Filipino', 'he': 'Hebrew'}
```

Figure (h)

D) Speech Representation

First and foremost, we'll import the pyttsx3 package that we loaded with pip.

Initialization of the Pyttsx3 Engine

engine = pyttsx3.init()

The preceding code loads the pyttsx3 package. The engine variable stores the instance of the initialised pyttsx3 package. We name it the variable engine because it acts as the engine and translates Text-To-Speech anytime the package's functions are executed.

Say Function in pyttsx3

engine.say("This is Text-To-Speech Engine Pyttsx3")

The pyttsx3 package includes a built-in say() method that takes a string value and speaks it out loud.

runAndWait Function

engine.runAndWait()

This function keeps track of when the engine begins translating text to voice and waits for that amount of time before closing the engine.

After all of the processes have completed, we terminate the engine by invoking the stop() function.

Change Text-to-Speech Voice In Pyttsx3 Python

We can also modify the engine's voice, which is currently the voice of a male named David.

To modify the voice of the pyttsx3 engine, we must first obtain a list of voice objects.

```
##Pyttsx3 getProperty
voices = engine.getProperty('voices')
```

The pyttsx3 package's getProperty() function accepts a string as a parameter and returns an object matching the string.

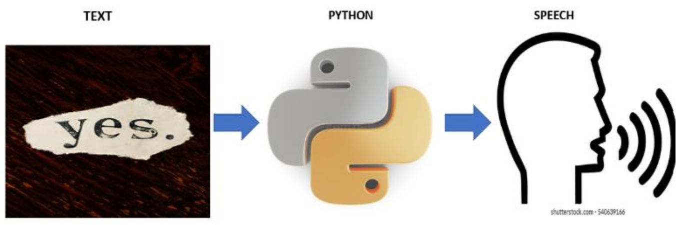


Figure (i)

V. RESULTS AND DISCUSSION

Performance of Modules :

Easy OCR:
The results reveal that Easy OCR predicted the number plate with greater than 95% accuracy, while Tesseract OCR only predicted it with 90% accuracy. As a result, Easy OCR surpasses Tesseract OCR since it employs a deep learning strategy for object detection and is effective in real-time prediction. It is ready-to-use OCR that supports 40+ languages, including Chinese, Japanese, Korean, and Thai.

Spacy :
Spacy, on the other hand, is the preferred method for app developers. While NLTK gives you access to a variety of algorithms, spacy is the best approach to get things done. It offers the most accurate and fastest syntactic analysis of any NLP library released to date. In terms of speed, NLTK is significantly slower than spacy.

Google trans :
For the most part, Google Translate is pretty accurate. In some circumstances, it's more than 94% accurate! In reality, when it comes to translation accuracy, it's one of the top-rated translation tools, albeit the exact accuracy will rely on the language pairs you've picked. The Spanish-English language pair has some of the most precise translations. Other languages, such as Chinese, German, and Portuguese, have a lower level of accuracy.

Pyttsx3 :
The Speech Engine's default speed rate is 200. If we set the Speed to less than 200, the voice will speak slowly, while setting it to more than 200 will increase the engine's speed rate. gTTS, which works flawlessly in Python3, requires an internet connection to function because it relies on Google to obtain audio data. However, Pyttsx3 is fully offline, works flawlessly, and supports many tts engines.

VI. FUTURE WORKS

This project may eventually link to the Cloud. This approach can also be integrated into an IoT-based system. Also, it can optimise the task to be implemented in web development.

VII. CONCLUSION

We examined approaches for developing an effective method of identifying and recognising text in video sequences through this study. Language translation is being implemented while viewing the image, which is faster than translating the extracted text. Implementing text summarization can help you understand the entire context of a text or paragraph. The creation of this programme, which will be of tremendous assistance to persons with visual impairments and those who do not speak another language.

REFERENCES

[1] Jamil et al., "Multilingual artificial text extraction and script identification from video images." International Journal of Advanced Computer Science and Applications, 1(7), 529-539 (2016)

[2] J. Hochberg, L. Kerns, P. Kelly, and T. Thomas, in Document Analysis and Recognition, vol. 1. "Automatic script identification from images using cluster-based templates" (IEEE, Montreal, 1995), pp. 378-381.

[3] Spitz, A. L., "Determination of the script and language content of document images." 19(3), 235-245 (1997) IEEE Trans. Patt. Anal. Mach. Intell.

[4] U. Pal and B. Chaudhuri, 2001, Proceedings of the International Conference on Document Analysis and Recognition. "Automatic identification of English, Chinese, Arabic, Devnagari, and Bangla script line" (IEEE, Washington, 2001), pp. 790-794.

[5] Advances in Multimedia Information Processing-PCM 2006, C. Zhu, W. Wang, and Q. Ning. "Text detection in images using texture features from strokes" (Springer, Hangzhou, 2006), pp. 295-301.

[6] "Effective and efficient video text extraction using key text points," Z. Li, G. Liu, X. Qian, D. Guo, and H. Jiang. Image Processing using IET." 5(8), 671–683 (2011)

[7] N. Sharma, S. Chanda, U. Pal, and M. Blumenstein, "Word-wise script identification from video frames" (IEEE, Washington, 2013), pp. 867-871 in Document Analysis and Recognition (ICDAR), 2013 12th International Conference On.

[8] G. Peake and T. Tan, 1997, Document Image Analysis (DIA'97). Workshop on "Script and language identification from document images" (IEEE, San Juan, 1997), pp. 10-17.

[9] P. Shivakumara, N. Sharma, U. Pal, M. Blumenstein, and C. L. Tan, "Gradient-angular features for word-wise video script identification" (IEEE, Stockholm, 2014), pp. 3098-3103.

[10] N. Sharma, U. Pal, and M. Blumenstein published "A study on word-level multi-script identification from video frames" in Neural Networks (IJCNN), 2014 International Joint Conference On (IEEE, Beijing, 2014), pp. 1827-1833.

[11] N. Sharma, R. Mandal, R. Sharma, U. Pal, and M. Blumenstein, in Document Analysis and Recognition (ICDAR), 15th International Conference On. Icdar2015 "competition on video script identification" (CVSI 2015) (IEEE, Nancy, 2015), pp. 1196-1200.

[12] J. Mei, L. Dai, B. Shi, and X. Bai, in Proceedings of the 23rd International Conference on Pattern Recognition (ICPR), 2016. "Scene text script identification with convolutional recurrent neural networks" (IEEE Cancn, 2016), pp. 4053-4058.

[13] Document Analysis Systems (DAS), 2016 12th IAPR Workshop On, A. K. Singh, A. Mishra, P. Dabral, and C. Jawahar. "A simple and effective solution for script identification in the wild" (IEEE, Santorini, 2016), pp. 428-433.

[14] Document Analysis Systems (DAS), 2016 12th IAPR Workshop On, L. Gomez, D. Karatzas. "A fine-grained approach to scene text script identification" (IEEE, Santorini, 2016), pages 192-197.

[15] M. Tounsi, I. Moalla, F. Lebourgeois, and A. M. Alimi, in Proceedings of the International Conference on Neural Information Processing. "CNN-based transfer learning for scene script identification" (Springer, California, 2017), pp. 702-711.

[16] L. Gomez, A. Nicolaou, and D. Karatzas, "Improving patch-based scene text script identification with ensembles of conjoined networks." 67, pp. 85-96 (2017).