University of Khartoum

Faculty of Engineering

Department of Electrical and Electronics Engineering

Final-Year Project Thesis

Software Engineering

Scene captured Arabic Recognition (SCAR)

Prepared By:

Yazeed Mohi-Eldeen Mohammed
Mohammed Eltigani Elshareef

Supervised By:

Prof. Sharif Fadul Babikir
Ust. Mohanned Ahmed

8th of September 2020

DEDICATION

To our families, teachers and colleagues.

ABSTRACT

(English)

The field of Arabic text recognition is lagging when it comes to the application of more advanced recognition techniques. Moreover, the datasets available for public usage does contain irregularities and unclarities making them unable to achieve good results. The same can be said for the localization of Arabic text. In this research, we introduce a cleaned up version of one of the popular Arabic recognition datasets (the KHATT dataset), we provide an implementation for the advanced state of the art recognition technique called GCNNs, and we adapt the popular localization technique called EAST for the Arabic language. And finally, we have created a fully robust system that does all the work from the localization of text boxes, then binarizing their colors, then acquiring the recognition results for each, and finally replacing the original text boxes in the image with the recognized ones.

ABSTRACT
(Arabic)

يتأخر مجال التعرف على النص العربي عندما يتعلق الأمر بتطبيق تقنيات التعرف الأكثر تقدمًا عن اللغات الأخرى. علاوة على ذلك ، فإن مجموعات البيانات المتاحة للاستخدام العام تحتوي على الكثير من عيوب مما يجعلها غير قادرة على تحقيق نتائج جيدة. يمكن قول الشيء نفسه عن مجال إيجاد خطوط النص العربي في الصور بصورة تلقائية. في هذا البحث ، نقدم نسخة مُنظَّفة و محسنة من إحدى مجموعات بيانات التعرف على اللغة العربية الشائعة (مجموعة بيانات KHATT)، ونوفر تطبيقًا لتقنية التعرف على خطوط الكتابة اليدوية في الصور المتقدمة المسماة بتقنية GCNNs ، ونكيف تقنية إيجاد خطوط النص في الصورة المسماة بتقنية EAST من أجل اللغة العربية. وأخيرًا ، أنشأنا نظامًا قويًا بالكامل يقوم بجميع الأعمال بدءًا من إيجاد مربعات خطوط النص ، ثم تحويلها إلى صيغة الألوان الثنائية ، ثم الحصول على نتائج التعرف على كل منها ، وأخيراً استبدال مربعات النص الأصلية في الصورة بتلك التي تم التعرف عليها.

INTRODUCTION

Background:

In the excruciating advancement in document image analysis technology, the need for reliable and efficient Optical Character Recognition (OCR) systems has become inevitable. Using the old methods of representing written documents digitally as images makes the job of transferring, maintaining, and accessing such documents a tedious task. Such legacy technology produces image documents that are unsearchable, non-editable, and generally occupy more storage. The newer technologies emerging from later research has focused on improving such technologies to meet the demands of network bandwidth and storage for both businesses and communities. The primary task of OCR is the conversion of document images into digitized form like ASCII/UNICODE. One of the essential objectives of OCR is to provide text to speech recognition for visually impaired and illiterate people in any language. Furthermore, it adds to the availability of documents and ensures their readability. Deep learning (a Machine learning approach) have a sound reputation in solving a huge number of classification and regression problems including: character recognition, speech recognition, machine translation, etc. Some of the well-known deep learning model are AlexNet [1], GoogleNet, VGG Bi-directional Long Short-Term Memory (BLTM) [2], Multi-Dimensional Long Short-Term Memory (MDLSTM) [3].

In the case of Arabic text recognition, some difficulties emerge from the fact that Arabic text has a wide diversity in its forms, additionally, the lack of datasets readily available for researchers also poses an issue.

The other important component in designing OCR systems is the text localization part, where the system is given a scene picture with some text lines within it, and its job is to automatically detect those text lines and output their coordinates for later usage. While the field of Arabic text recognition is still immature, the same cannot be said about text localization, as there has been a number of well-designed systems that has been successfully used for a variety of applications, though such systems are still imperfect and enhancements can indeed be achieved.

Key areas:

Here we will include some of the datasets used by previous researchers, also going through the description of some of the models and methods for data collection, localization, and recognition.

Datasets:

For the datasets used, they can be divided into two sectors:

Text recognition datasets:

English text:

IAM: A collection of handwritten passages by several writers. Generally, they use that data to classify writers according to their writing styles.

Bentham: Used in tranScriptorium is a subset of the transcribed documents.

Rimes: (Reconnaissance et Indexation de données Manuscrites et de fac similÉS / Recognition and Indexing of handwritten documents and faxes) was created to evaluate automatic systems of recognition

and indexing of handwritten letters. Of particular interest are cases such as those sent by postal mail or fax by individuals to companies or administrations.

Saint Gall: Contains a handwritten historical manuscript with following characteristics: 9th century, Latin language, single writer, Carolingian script, ink on parchment

Arabic text:

KHATT: A database of unconstrained handwritten Arabic Text written by 1000 different writers. This research database's development was undertaken by a research group from KFUPM, Dhahran, S audi Arabia headed by Professor Sabri Mahmoud in collaboration with Professor Fink from TU-Dortmund, Germany and Dr. Märgner from TU-Braunschweig, Germany. The database includes 2000 similar-text paragraph images and 2000 unique-text paragraph images and their extracted text line images. The images are accompanied with manually verified ground-truth and Latin representation of the ground-truth. The database can be used in various handwriting recognition related researches like, but not limited to, text recognition, and writer identification. Interested readers can refer to the paper [1], and [2] for more details on the database. The version 1.0 of the KHATT database is available free of charge (for academic and research purposes) to the researchers.

KHATTX: A modified version of the KHATT dataset created by our team. With removed outliers, normalized image size, and background color correction.

Localization datasets:

ICDAR: (International Conference on Document Analysis and Recognition). A dataset created by International Association of Pattern Recognition (IAPR), used in real-time text extraction.

ICPR: (International Conference on Patter Recognition). A dataset created by International Association of Pattern Recognition (IAPR), used several competitions for text localization.

Total Text: A comprehensive dataset for scene text detection and recognition. This dataset fills the gap that is present in older datasets by taking into account the curvature an orientation of text, it facilitates a new research direction for the scene text community. On top of the conventional horizontal and multi-oriented texts, it features curved-oriented text. Total-Text is highly diversified in orientations, more than half of its images have a combination of more than two orientations

SVT: The Street View Text (SVT) dataset was harvested from Google Street View. Image text in this data exhibits high variability and often has low resolution. In dealing with outdoor street level imagery. And since that image text often comes from business signage and that business names are easily available through geographic business searches. These factors make the SVT set uniquely suited for word spotting in the wild: given a street view image, the goal is to identify words from nearby businesses.

Methods and models:

Here, we will take a look at the previous models and methods used in text recognition and localization.

Text recognition:

Previous methods for text recognition includes the following:

Segmentation and character recognition: Where each line of text is segmented into its composing characters, each character is then recognized individually using a convolutional neural network, the results are then merged to compose the final recognized text line.

BLSTM: Uses Bidirectional Long Short-Term Memory neural networks, the text line is recognized in a sequential matter where the network scans the text in a horizontal direction, whilst memorizing the context from the previous steps, the recognized text is then passed through a Connectionist Temporal Classification algorithm to construct the final result.

MDLST: Same as the previous method but instead of scanning the text only in the horizontal direction, this method scans it in both horizontal and vertical directions for improved accuracy, it does that with the assistance of a Multi-Dimensional LSTM network.

GCNN: (Gated Convolutional Neural Networks), Uses gated convolutional layers to imitate the context memorization of LSTM networks, yields the best performance and accuracy.

Text localization:

Canny edge detection: Where a canny edge detection algorithm is used to extract the borders of the characters composing the text, a bounding box is created for each of the characters and a larger box for the text line is created through merging the smaller boxes. This method is not sufficient for images where there is high frequency changes of colors in the background.

EAST: (Efficient and Accurate Scene Text detector), a deep learning model, based on a novel architecture and training pattern, it provides high accuracy and low latency.

Data processing:

Here, we will describe data processing techniques used in the research, for both text recognition and localization.

Image normalization: Whereby the average and the standard deviation of the image are calculated and then the value for each pixel is replaced by its value minus the average divided by the standard deviation.

Augmentation: Whereby from each image a set of new images are created which are scaled, dilated, rotated, eroded, or shifted versions of the original image. This helps increasing the size of the training set for the dataset.

Size preprocessing: Whereby each image is resized to fit the required input size for the neural network model.

Illumination compensation: Whereby the uneven illumination of images is compensated and images in normal lighting conditions are reconstructed. The reconstructed images are then used for classification. An illumination compensation scheme includes the following modules: lighting category evaluation, shape normalization, and lighting compensation.

Cursive hand written text normalization: Whereby the cursive style is removed from the input image text which improves the recognition accuracy.

Image binarization: Whereby the image is converted to a 1 bit black and white image, which yields better performance for recognition.

NMS: (Non-Maximum Suppression), A technique used in text localization for the EAST model, calculates the coordinates of the bounding box for the text using the activation values from the network.

Adjust-to-see: A function used to adjust the image for better visual interpretation by the user.

Methodology:

An agile software development methodology scheme was followed, as in, the project was carried out in the shape of sprints, each of which complements and builds upon the one preceding it. A component based architecture was followed for the design of the system, where each module is implemented independently and then the complete system is built from these components, this provides higher upgradability and easier detection for bugs and performance bottlenecks.

The final system is constructed from three major parts: text localization, text recognition and replacement of the original text with the recognized one.

The system for the text recognition was built first, where a highly optimized and robust system was created that allows for reading from various datasets of text recognition for both the English and Arabic languages, it also provides an implementation for the image preprocessing methods stated earlier.

Next, the localization system was created. A similar implementation to the recognition system was followed, where the created system allowed for reading from various datasets and using different preprocessing schemes, in addition, the system provides the ability to visualize the steps followed in detecting the text lines.

Finally, a text replacement system was implemented, whereby the previous systems are used in tandem to extract and recognize text lines, the locations of the text lines in the image are stored and are replaced later by the recognized text lines, where both the background color of the text area and the font size are estimated to allow for more natural looking output images.

Future impact:

The techniques and ideas implemented in this research will greatly impact both the fields of text localization and recognition, here we include some of the major ways in which this research contributes to improving knowledge.

Text recognition:

In the text recognition department, previous methods such as CNNs, BLSTMs, and MDLSTMs were not good enough in terms of performance, they had long training times, high inference latencies, and were in general complex to understand. Whereas, the method introduced in this research remedies those issues with the use of Gated Convolutional Neural Networks, due to the simplicity of their infrastructures, GCNNs generally yield better performance when compared to older methods.

As for Arabic text recognition in specific, models for recognizing handwritten text were rather inaccurate and inadequate to use in most applications. This actually comes from the fact that Arabic machine language datasets are usually not available publicly, and for the ones which are available, they usually contain many irregularities and needs heavy preprocessing before they can be used. In this research, we took one of the most popular Arabic text recognition datasets (the KHATT dataset) and processed it by removing outliers and normalizing the image sizes, this lead to better training performance.

Text localization:

While the text localization field is indeed mature enough, most of the techniques used in this field were only applicable to certain languages on which their respective models were trained. In this research we take a look at a rather new technique that allows for training the network on two different stages, where in the first stage the model is trained on a general model for recognizing the contents of the introduced images. And on the second stage, the network is further trained for detecting text in the required language. This allows for high reusability of trained models and lower training times (since the user can download a model that is already pre-trained on the first stage and only needs to be trained on the second one).

Research questions:

The major questions that are to be answered by this research are focused upon the ability to apply the latest text localization and recognition techniques for Arabic text. Here we include some of the more important ones.

The most important question that this research answers is whether or not is there an ability to improve Arabic text recognition systems and algorithms, and there appears to be such ability, how is it going to implemented and how beneficial and more accurate is it compared to previous methods, what modifications are required for the current systems, what preprocessing methods yield better performance, and is there a need for creating new datasets or modifying existing ones for recognizing Arabic text.

And for the localization part, what are the existing methods for Arabic text localization, is there a way to use other state-of-the-art methods tested on other languages to the localization of Arabic text, and how feasible is it.

And generally for the system design, what are the inevitable compromises, what are the acceptable levels of performance for the overall system, where does bottlenecks occur, and how feasible is it in terms of implementation and inference.

Organization

Here, we describe the general organization of the research documentation.

Literature review:

Where we take note of previous techniques used in text recognition and localization, we will discuss each of the previous methods and models in detail and provide a general description for the base of knowledge, individual studies will be included, contribution of previous studies on the different areas will be considered, and a description of the state of the art will conclude the chapter.

Methodology:

In this chapter, we will describe the methods followed in the research, discussing the design of methodology, explaining why the considered methods is a good match to answer the research question. The steps for conduction the study will be noted, and the stages of the desired system will be discussed in detail.

Results and data analysis:

For this part, we will first go through the findings that are within the context of the setting, we will provide visual representations for easier interpretation, we will carry out an analysis for the result, and we will describe how we made sense out of the findings.

Conclusion:

We start this part by providing the research results and discussing the findings, we will provide information for future research, we will describe future recommendations and the lessons learned from the research, and we will finally include some ideas that are to be implemented in the future.

LITERATURE REIVEW

Key areas:

Here, we will describe the key areas that contributes to the base of knowledge, including: models, preprocessing techniques, post-processing techniques, and general theories related to this research.

CNNs:

Convolutional neural networks [1] are heavily used for image recognition tasks due to their good performance with images, a CNN is composed of a number of convolutional layers, each of which builds upon its predecessor, and finally a fully connected layer is imposed to do the classification based on the output of the final convolutional layer.

A convolutional layer is composed of convolutional units, which are basically image filtering scanning blocks, each unit has a known width and height, and each unit also has its own activations and bias. The convolutional layer is then simply a number of these units stacked in a certain order, each of the units scans the content of the previous layer in a linear manner, and then, an activation function is applied to provide the final output of the layer.

CTC:

Connectionist temporal classification [2] is a method used with recurrent neural networks in natural language processing to extract results from the recognized output time series.

When an RNN is used, the network generates an output for each time step defined in the input series, though in some cases, the time step is rather too short compared to the period for which a certain output is presented, for example, when an RNN is used for voice recognition, the length of each letter might be longer than the time step defined for the network, in this case, the CTC technique is applied to remove all the duplicate results in the classification, it can also detect actual duplicates in the input series with the use of an additional classification label that is called a “blank”.

LSTMs:

Long short-term memory neural networks [3] are recurrent neural networks that make use of controlled gates for keeping or discarding context stored within them.

Each LSTM unit contains a number of gates which define how it operates, the forget gate is used to discard the content of the unit, the input gate is used to store information into the unit, the keep gate is used to let the unit know that it should keep the information it has stored, and the output gate is used to allow the unit to use the value stored within and output it to the outside world. Each of these gates can be trained to only be activated when necessary.

MDLSTMs:

Multi-Dimensional LSTMs are neural networks based on LSTM units, their difference to vanilla LSTM networks is that they scan the input in multiple directions instead of only scanning it in the conventional direction, an example for that is when a text image is scanned I all four directions (up, down, right, and left). This usually helps with keeping more context of the input for better classification results, though it generally also results in more complex models and rather long training times.

GCNNs:

Gated convolutional neural networks [4] are different from the normal in that the convolutional layers contains controlled gates that allows them to keep context from previous classifications, this allows for better context keeping in the overall network when compared to normal convolutional layers connected to recurrent layers. Though, this method is currently rather new, more testing is indeed required.

Canny edge detection:

A method used for text localization (detection), where a canny edge detection algorithm [5] is used to extract the borders of the characters composing the text, a bounding box is created for each of the characters and a larger box for the text line is created through merging the smaller boxes. This method is not sufficient for images where there is high frequency changes of colors in the background.

Image normalization:

A preprocessing method used for yielding better training results, whereby the average and the standard deviation of the image are calculated and then the value for each pixel is replaced by its value minus the average divided by the standard deviation, this technique is useful for eliminating outliers and adjusting the training set of the machine learning model to become centralized.

Augmentation:

A preprocessing method used usually with smaller datasets [6] (but is also used for larger ones), whereby from each image a set of new images are created which are scaled (resized), dilated (stretched), rotated, eroded, or shifted versions of the original image. This helps increasing the size of the training set for the dataset.

Illumination compensation:

A recently invented preprocessing technique used majorly for text recognition [7], whereby the uneven illumination of images is compensated and images in normal lighting conditions are reconstructed. The reconstructed images are then used for classification. An illumination compensation scheme includes the following modules: lighting category evaluation, shape normalization, and lighting compensation.

Cursive hand written text normalization:

A preprocessing method used for the recognition of cursive handwritten text [8], whereby the cursive handwriting style of the writer is estimated using different methods, and then the text image is reconstructed without the cursive style which improves the recognition accuracy. In this technique, the input image is rotated and text lines are eroded or dilated as required, so that all the characters are vertically separated.

Image binarization:

A post-processing technique used for the recognition of text lines extracted from a larger image [9], whereby the text line image is converted to a 1 bit black and white image, which yields better performance for recognition since the text can be represented with black and the background with white (as the case is for most of the text recognition datasets).

NMS:

Non-Maximum Suppression [10] is a post-processing technique used in text localization for the EAST model, where the activation layer together with the geometry layers of the model output is used to calculate the coordinates of the bounding boxes.

Datasets:

Here, we will provide detailed descriptions for the datasets used in this study. This part will be divided into two major sections: Text recognition datasets and text localization datasets.

Text recognition datasets:

English text:

IAM:

The IAM Handwriting Database contains forms of handwritten English text which can be used to train and test handwritten text recognizers and to perform writer identification and verification experiments.

The database was first published in [11] at the ICDAR 1999. Using this database an HMM based recognition system for handwritten sentences was developed and published in [12] at the ICPR 2000. The segmentation scheme used in the second version of the database is documented in [13] and has been published in the ICPR 2000.

The database contains forms of unconstrained handwritten text, which were scanned at a resolution of 300dpi and saved as PNG images with 256 gray levels.

Partitions	Characters	Words	Lines
Train	274,964	53,728	6,161
Validation	40,770	7,899	900
Test	81,676	17,616	1,861
Total	397,410	79,246	8,922

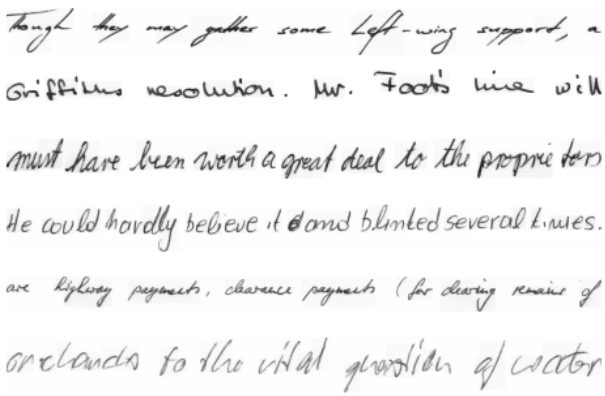


Figure (2.1): IAM dataset samples

Table (2.1): IAM dataset properties

Bentham:

The Bentham collection consists of a set of images of a collection of works on law and moral philosophy written by the philosopher Jeremy Bentham.

The selected subset has been written by several hands (Bentham himself and his secretaries) and entails significant variabilities and difficulties regarding the quality of text images and writing styles. Training and test data were provided in the form of carefully segmented line images, along with the corresponding transcripts.

This dataset is free available for research purposes and it is provided into two parts: the images and the GT. The GT includes information about the layout and the transcription at line level of each image in PAGE format.

Partitions	Characters	Words	Lines

Train	424,199	86,015	9,195
Validation	68,679	12,958	1,415
Test	38,429	6,960	860
Total	527,307	105,933	11,470

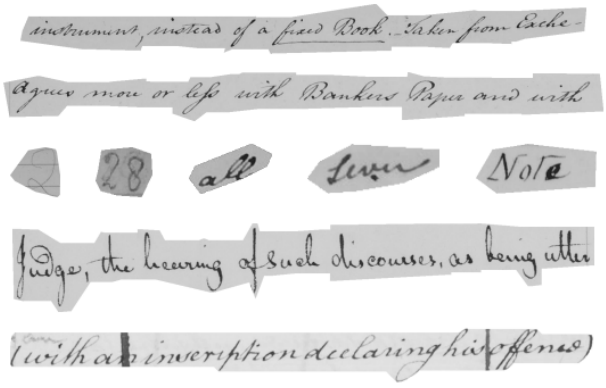


Figure (2.2): Bentham dataset samples

Table (2.2): Bentham dataset properties

Rimes:

(Reconnaissance et Indexation de données Manuscrites et de fac similÉS / Recognition and Indexing of handwritten documents and faxes) was created to evaluate automatic systems of recognition and indexing of handwritten letters. Of particular interest are cases such as those sent by postal mail or fax by individuals to companies or administrations.

Partitions	**Characters**	**Words**	**Lines**
Train	458,737	73,798	10,193
Validation	51,968	8,404	1,133
Test	35,194	5,639	778
Total	545,899	87,841	12,104

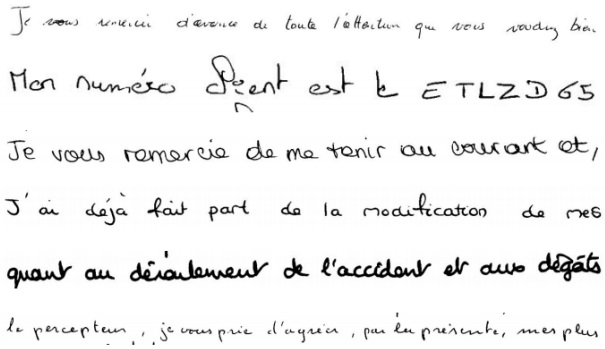


Figure (2.3): Rimes dataset samples

Table (2.3): Rimes dataset properties

Saint Gall:

Contains a handwritten historical manuscript with the following characteristics:

- 9th century
- Latin language
- Single writer
- Carolingian script
- ink on parchment

Partitions	Characters	Words	Lines
Train	458,737	73,798	10,193
Validation	51,968	8,404	1,133
Test	35,194	5,639	778
Total	545,899	87,841	12,104

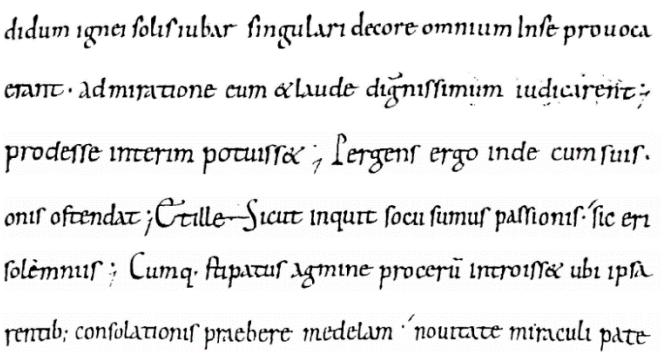


Figure (2.4): Saint Gall dataset samples

Partitions	Lines
Train	9,523
Validation	1,906
Test	2,009
Total	13,438

Table (2.4): Saint Gall dataset properties

Arabic text:

KHATT (KFUPM Handwritten Arabic TexT):

A database of unconstrained handwritten Arabic Text written by 1000 different writers. This research database's development was undertaken by a research group from KFUPM, Dhahran, Saudi Arabia headed by Professor Sabri Mahmoud in collaboration with Professor Fink from TU-Dortmund, Germany and Dr. Märgner from TU-Braunschweig, Germany. The database includes 2000 similar-text paragraph images and 2000 unique-text paragraph images and their extracted text line images. The images are accompanied with manually verified ground-truth and Latin representation of the ground-truth. The database can be used in various handwriting recognition related researches like, but not limited to, text recognition, and writer identification. Interested readers can refer to the paper [1], and [2] for more details on the database. The version 1.0 of the KHATT database is available free of charge (for academic and research purposes) to the researchers.

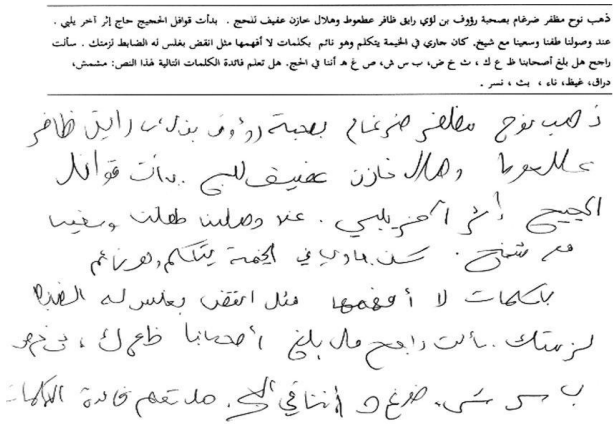


Figure (2.5): KHATT dataset samples

Table (2.5): KHATT dataset properties

KHATTX:

A modified version of the KHATT dataset created by our team. With removed outliers, normalized image size, and background color correction.

Partitions	Lines
Train	4,993
Validation	872
Test	1082
Total	6,744

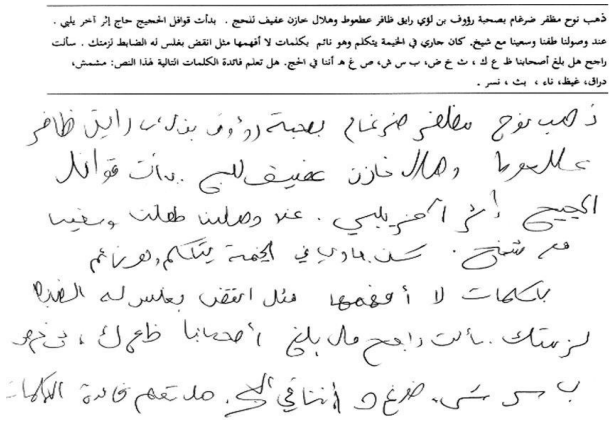


Figure (2.6): KHATTX dataset samples

Table (2.6): KHATTX dataset properties

Localization datasets:

ICDAR:

(International Conference on Document Analysis and Recognition). A dataset created by International Association of Pattern Recognition (IAPR), used in real-time text extraction.

Partitions	Images
Train	5,603
Validation	2,321
Test	2,242
Total	10,166

Figure (2.7): ICDAR19 dataset samples

Table (2.7): ICDAR19 dataset properties

ICPR:

(International Conference on Patter Recognition). A dataset created by International Association of Pattern Recognition (IAPR), used several competitions for text localization.

Partitions	Images
Train	10,000

Validation	N/A
Test	10,000
Total	20,000

Figure (2.8): ICPR dataset samples

Table (2.8): ICPR dataset properties

Total Text:

A comprehensive dataset for scene text detection and recognition. This dataset fills the gap that is present in older datasets by taking into account the curvature an orientation of text, it facilitates a new research direction for the scene text community. On top of the conventional horizontal and multi-oriented texts, it features curved-oriented text. Total-Text is highly diversified in orientations, more than half of its images have a combination of more than two orientations.

Partitions	Images
Train	1,255
Validation	N/A
Test	300
Total	1,555

Figure (2.9): TotalText dataset samples

Table (2.9): TotalText dataset properties

SVT:

The Street View Text (SVT) dataset was harvested from Google Street View. Image text in this data exhibits high variability and often has low resolution. In dealing with outdoor street level imagery. And since that image text often comes from business signage and that business names are easily available through geographic business searches. These factors make the SVT set uniquely suited for word spotting in the wild: given a street view image, the goal is to identify words from nearby businesses.

Partitions	Images
Train	300
Validation	N/A
Test	50
Total	350

Figure (2.10): SVT dataset samples

Table (2.10): SVT dataset properties

Individual studies:

Here, we will consider previous studies done on the field, while grouping them according to common denominators, and describing how each study contributes to the body of knowledge.

Image Recognition (Common to both text recognition and localization):

Object Recognition with gradient based learning (Yann LeCun):

In this research, the problem of finding an appropriate set of features is considered due to how essential it is in the design of shape recognition systems. They show that for recognizing simple objects with high shape variability such as handwritten characters, it is possible, and even advantageous, to feed the system directly with minimally processed images and to rely on learning to extract the right set of features. Where convolutional neural networks were shown to be particularly well suited to this task. They also show that these networks can be used to recognize multiple objects without requiring explicit segmentation of the objects from their surroundings. The second part of their research presents the graph transformer network model which extends the applicability of gradient based learning systems that use graphs to represent features, objects, and their combinations.

This research in particular was a breakthrough in the image recognition field of research, where the advantage of using convolutional neural networks was explained in detail, going also through what makes these networks suitable for image recognition or object detection tasks.

ImageNet Classification with Deep Convolutional Neural Networks (Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton):

In this research, a large, deep convolutional neural network was trained to classify the 1.2 million high-resolution images in the ImageNet LSVRC-2010 contest into the 1000 different classes. On the test data, this approach achieved top-1 and top-5 error rates of 37.5% and 17.0% which is considerably better than the previous state-of-the-art. The neural network, which has 60 million parameters and 650,000 neurons, consists of five convolutional layers, some of which are followed by max-pooling layers, and three fully-connected layers with a final 1000-way softmax. To make training faster, the researchers used non-saturating neurons and a very efficient GPU implementation of the convolution operation. To reduce over fitting in the fully connected layers they employed a regularization method called “dropout” that proved to be very effective.

The results presented by these researchers changed the way convolutional neural networks are designed, they introduced the idea of using deep networks with varying kernel sizes, and the idea of implementing dropout layers to reduce overfitting and to complement undependability among the network kernels, which led to the high performance they were able to achieve.

Very deep convolutional networks for large-scale image recognition VGG (Karen Simonyan, Andrew Zisserman):

In this research, the effect of the convolutional network depth on its accuracy in the large-scale image recognition setting is investigated. The main contribution provided is a thorough evaluation of networks of increasing depth using an architecture with very small (3 × 3) convolution filters, which shows that a significant improvement on the prior-art configurations can be achieved by pushing the depth to 16–19 weight layers. These findings were the basis of the authors' ImageNet Challenge 2014 submission, where their team secured the first and the second places in the localization and classification tracks respectively. They also show that their representations generalize well to other datasets, where they achieve state-of-the-art results. They have made their two best-performing ConvNet models publicly available to facilitate further research on the use of deep visual representations in computer vision.

This research led to a large improvement in the design of convolutional neural networks, where they recommended the use of deeper neural networks with smaller kernel sizes, leading to more context being considered in later stages of the network.

Multi-Dimensional Recurrent Neural Networks (Alex Graves, Santiago Fernandez, J ´ urgen Schmidhuber):

This research is built upon the idea that recurrent neural networks (RNNs) have proved effective at one dimensional sequence learning tasks, such as speech and online handwriting recognition. Whereas, some of the properties that make RNNs suitable for such tasks, for example robustness to input warping, and the ability to access contextual information, are also desirable in multidimensional domains. However, at that point of time, there has been no direct way of applying RNNs to data with more than one spatio-temporal dimension. Hence, in this research, the idea of multi-dimensional recurrent neural networks (MDRNNs) is introduced, which thereby extends the potential applicability of RNNs to vision, video processing, medical imaging and many other areas, while avoiding the scaling problems that have degraded the performance of other multi-dimensional models.

The idea of using multidimensional recurrent neural networks that were introduced in this research turned out to carry lots of benefits, especially in the case of offline handwritten text recognition, where the technique of scanning the text line from one side to the other could led to some context being lost, which in turn results in some performance breaches.

Arabic Text Recognition:

Urdu Nasta'liq text recognition system based on multi-dimensional recurrent neural network and statistical features (Saeeda Naz, Arif I. Umar, Riaz Ahmad, Saad B. Ahmed, Syed H. Shirazi, Muhammad I. Razzak):

In this research, the recognition of Urdu Nasta'liq text is investigated, where it's known that the character recognition task for cursive script like Arabic, handwritten English and French is challenging, which becomes even more complicated for Urdu Nasta'liq text due to complexity of this script over the others. Whereas, recurrent neural networks (RNN) has proved excellent performance for English, French as well as cursive Arabic script due to sequence learning property. But although that most of the approaches perform segmentation-based character recognition, the complexity of the Nasta'liq script

makes this unfeasible, where the segmentation error is quite high as compared to Arabic Naskh script. RNN has provided promising results in such scenarios. In this research, high accuracy for Urdu Nasta'liq using statistical features and multi-dimensional long short-term memory was achieved. Where robust feature extraction approach is presented, it extracts feature based on right-to-left sliding window. Results showed that selected features significantly reduce the label error.

This research explained the benefits of using advanced feature extraction methods instead of using raw text as input to network model, this is particularly useful in cases where the input text cannot be easily segmented using regular methods.

Convolutional Neural Network Model for Arabic Handwritten Characters Recognition (Murtada Khalafallah Elbashir, Mohamed Elhafiz Mustafa):

In this research, a Convolutional Neural Network (CNN) model for off-line Arabic handwritten character recognition is presented. The proposed CNN model used the dataset which prepared by Sudan University of Science and Technology- Arabic Language Technology group. The dataset is pre-processed before feeding it to the CNN model. In the pre-processing, all the characters images are size normalized to fit in a 20 by 20 pixel and then centered in a scaled images of size 28×28 pixel using the center of mass then all the images are converted to be having a black background and white foreground colors. The pre-processed images are fed to the CNN model, which is constructed using the sequential model of the Keras library under Tensorflow environment. The accuracy obtained varied from 93.5% as test accuracy to 97.5% as training accuracy showing better results than other methods that used the same dataset.

This research provided the basis of analyzing the ability of CNN models to recognize Arabic characters accurately, taking into account what preprocessing techniques helps in yielding better performance.

A Deep Learning based Arabic Script Recognition System: Benchmark on KHAT (Riaz Ahmad, Saeeda Naz, Muhammad Afzal, Sheikh Rashid, Marcus Liwicki, and Andreas Dengel):

This research presents a deep learning benchmark on a complex dataset known as KFUPM Handwritten Arabic TexT (KHATT). The KHATT data-set consists of complex patterns of handwritten Arabic text-lines. This research contributes mainly in three aspects i.e., (1) pre-processing, (2) deep learning based approach, and (3) data-augmentation. The pre-processing step includes pruning of white extra spaces plus de-skewing the skewed text-lines. The authros deploy a deep learning approach based on Multi-Dimensional Long Short-Term Memory (MDLSTM) networks and Connectionist Temporal Classification (CTC). The MDLSTM has the advantage of scanning the Arabic text-lines in all directions (horizontal and vertical) to cover dots, diacritics, strokes and fine inflammation. The data-augmentation with a deep learning approach proves to achieve better and promising improvement in results by gaining 80.02% Character Recognition (CR) over 75.08% as baseline.

This research represents the state-of-the-art in Arabic text recognition, it explains the applicability of new techniques such as MDLSTM networks and CTC in the recognition of Arabic text, it also provides some ideas on how to carry out preprocessing in order to clean up and prepare the dataset, and how data augmentation can be done to increase the size of the dataset.

Text Localization:

EAST: An Efficient and Accurate Scene Text Detector (Xinyu Zhou, Cong Yao, He Wen, Yuzhi Wang, Shuchang Zhou, Weiran He, and Jiajun Liang):

Previous approaches for scene text detection have already achieved promising performances across various benchmarks. However, they usually fall short when dealing with challenging scenarios, even when equipped with deep neural network models, because the overall performance is determined by the interplay of multiple stages and components in the pipelines. In this research, a simple yet powerful pipeline is presented, which yields fast and accurate text detection in natural scenes. The pipeline directly predicts words or text lines of arbitrary orientations and quadrilateral shapes in full images, eliminating unnecessary intermediate steps (e.g., candidate aggregation and word partitioning), with a single neural network. The simplicity of the pipeline allows concentrating efforts on designing loss functions and neural network architecture. On the ICDAR 2015 dataset, the proposed algorithm achieves an F-score of 0.7820 at 13.2fps at 720p resolution.

This research represents the state-of-the-art in scene text localization, it introduces a new technique for detecting text in images even in arbitrary orientations and distortions.

Arabic Cursive Text Recognition from Natural Scene Images (Saad Bin Ahmed , Saeeda Naz, Muhammad Imran Razzak and Rubiyah Yusof):

This research presents a comprehensive survey on Arabic cursive scene text recognition. The recent years' publications in this field have witnessed the interest shift of document image analysis researchers from recognition of optical characters to recognition of characters appearing in natural images. Scene text recognition is a challenging problem due to the text having variations in font styles, size, alignment, orientation, reflection, illumination change, blurriness and complex background. Among cursive scripts, Arabic scene text recognition is contemplated as a more challenging problem due to joined writing, same character variations, a large number of ligatures, the number of baselines, etc. Surveys on the Latin and Chinese script-based scene text recognition system can be found, but the Arabic like scene text recognition problem is yet to be addressed in detail. In this manuscript, a description is provided to highlight some of the latest techniques presented for text classification. The presented techniques following a deep learning architecture are equally suitable for the development of Arabic cursive scene text recognition systems. The issues pertaining to text localization and feature extraction are also presented. Moreover, this article emphasizes the importance of having benchmark cursive scene text dataset. Based on the discussion, future directions are outlined, some of which may provide insight about cursive scene text to researchers.

State of the art:

Here, we will include a review of the current state-of-the-art.

Arabic text recognition:

The current state-of-the-art method of Arabic text recognition is the method studied by the research: “A Deep Learning based Arabic Script Recognition System: Benchmark on KHAT”, this method yields 80.02% Character Recognition (CR) over 75.08% as baseline.

Localization:

The state-of-the-art method for the localization of scene text is the one considered in the research paper: “EAST: An Efficient and Accurate Scene Text Detector”, On the ICDAR 2015 dataset, the proposed algorithm achieves an F-score of 0.7820 at 13.2fps at 720p resolution

METHODOLGY

Chapter contents:

This chapter generally contains the following parts:

Design of methodology:

Contains definitions for the methods used, the name of the general methodology, and a description for why is it a good match.

Steps to conduct the study:

Contains a description for the general stages followed for the conduction of the study including: literature review, data collection, problem definition, brainstorming, planning, implementation, testing.

System design:

Contains a description of the implementation of the system, taking into account the details of each of the three main systems (recognition, localization, and replacement).

Data analysis:

Contains details on the process used for analyzing data.

Validation of findings:

Contains a description on how the findings of the research are validated, so that they can be considered reliable.

Design of methodology:

The main methodology scheme followed for this research was an agile software development scheme, where the work was carried out in the shape of sprints, each of which containing: planning, literature review, implementation, and testing.

This allowed for the system to be robust and easily modified to make up for the development in the field of research.

Each sprint starts with a planning phase, in which the present parameters of the research are considered, and the next required improvements are decided. After planning for the sprint is complete, a literature review stage takes place, where the data and information required for carrying out the requirements of the sprint are acquired, this step creates the base of knowledge for the sprint. Once the literature review is complete, the implementation phase takes place, where the required components for the system are developed and are added to the main system. Finally, the added components and the general system are thoroughly tested in order to validate the robustness and reliability of the new system.

The name of the design of methodology used in this research is long term agile methodology, the general stages of methodology will be considered next.

General construction of sprints:

The general sprints considered in carrying out the study are the following:

The first sprint:

This sprint was for acquiring a general understanding of the text recognition field, the main purpose of this sprint was to create a simple system that can perform text recognition for English handwritten text.

Various methods for handwritten text recognition was studied throughout this stage, including text recognition through segmentation and character by character recognition, text recognition using convolutional neural networks, and text recognition using support vector machines.

This stage was the main building part of the rest of the research.

The second sprint:

This sprint was for improving the accuracy and robustness of the systems considered in the previous sprint, where more complex models and methods were considered including recurrent neural networks, vanilla long short-term memory networks, and bidirectional long short-term memory networks.

At this stage, the general systems for recognition of text were ready to be tested and implemented for Arabic language.

The third sprint:

This sprint was for adopting the recognition methods for Arabic text, previous research on the field of Arabic text recognition was reviewed, and knowledge on available datasets was acquired.

At this sprint, the dataset KHATT was studied and tested with different models, the main fallbacks of this dataset was discovered, and a more robust version of it was created (namely: KHATTX).

By this time, the recognition system was able to compete against state-of-the-art systems for Arabic text recognition, the final system used gated convolutional neural networks, which was able to achieve very accurate results. More details will be provided later.

The fourth sprint:

This sprint was for studying the localization of text, where previous methods on text localization for English text were considered and deeply studied. Previous datasets used in the field were identified, acquired, and organized to fit the needs of the study.

Once such systems and datasets were acquired, the possibility of using these same methods for Arabic text localization were considered, the lack for Arabic text localization datasets was the main issue, though, a technique for reusing text localization models created for other languages in the localization of Arabic text was discovered and implemented. More details will be provided later.

At this sprint, various datasets were acquired and tested on a number of different systems, the system with the highest accuracy was to be considered in building the general final system.

The fifth sprint:

This sprint was for creating a merged system that contained both the recognition and the localization systems.

After feeding the image to the merged system and extracting the locations of the text boxes, each of the extracted boxes needs to be processed before being fed into the recognition system, the processing of the extracted text boxes was the main issue considered in this sprint.

At this sprint, the merged system was constructed and the processing of the extracted text boxes was studied and implemented.

The sixth sprint (the final sprint):

This sprint was for adding the replacement module to the merged system.

The main purpose of the replacement module is to carry out color estimation for both the text and the background of the text box in the original image, each box of extracted text is later replaced by its counterpart recognized and color estimated text box.

By the end of this sprint, the system was complete and ready for further testing and evaluation.

Validation of methodology:

In this part, we will consider why is this design of methodology a good match to answer the research question.

Mainly, this design of methodology allows the system to be built in a block by block manner where each time a block is added further improvements are added to the previous blocks as well. Where at each stage, the system is fully tested and verified before moving to the next stage, which allows for added robustness in the resulting system.

This methodology also allows the team to gain more insight and experience into the article of research the further they go deeper within it, which helps in building the experience needed for the more complex issues faced in the later stages of the research.

Generally, this methodology allows for a flexible and adaptive work scheme.

Steps to conduct the study:

Here, we will include general description for the main steps of conducting the study.

Literature review:

In this stage, the general material for the research was reviewed, previous work on the field was carefully studied in order to find loopholes or areas that lack investigation.

The task of recognizing Arabic text was considered in previous research, but it was still considerably immature compared to the recognition of text written in other languages. And so was the case for the detection of Arabic text in scene images.

This stage helped in creating a general understanding for the area of knowledge, which helped later in deciding what can be achieved, and also, what can be improved.

Data collection:

In this stage, knowledge and data specific to the field was acquired, which was done in preparation for the further steps of the study.

Problem definition:

In this stage, the current technologies were studied in-depth to discover possible improvements for their current state or for finding areas that require solutions.

The area of scene text recognition for the Arabic language was found to carry a number of different benefits and possible inventions. Each of which were considered and studied deeply in the next stage.

Brainstorming:

In this stage, ideas on improving or adding inventions to the field of scene text recognition for the Arabic language were developed.

The general idea of the research was agreed on to be idea of solving the particular problem of the difficulty of reading Arabic text written in road or market signs from a distance, which was hard for the visually impaired to read it. Also, if a solution can be found for this particular issue, we found it to be also useful for situations where an unfamiliar tourist tries to read such text and translate it digitally, where the tourist has to first read the letters before writing it into a translation service. The system can be used for translating the detected text into the required language using an embedded translation module.

Planning:

In this stage, the general plan for conducting the study was developed, where different methodology schemes were considered and differentiated, the most suiting scheme was selected and the general steps for implementing the scheme were defined, providing the details on how much each step should cover out of the full system, and also, how much time would each step take to be completely implemented, tested, and added into the full system.

Implementation:

In this stage, the implementation of the main system took place, where, as described previously, the work was carried out in the shape of sprints, each of which builds upon the ones previous to it. More details on the implementation of the system will be provided later.

Testing and evaluation:

In this stage, the full system was tested and evaluated against a variety of test scenarios, this stage is different that the testing step that is carried out during individual sprints in that here the full system as a whole is tested and evaluated for performance and accuracy bottleneck. More details on the evaluation and results are to be provided later.

System design:

This part will contain description of the general design and implementation of the system, code samples and figures will be included to further assist understanding. Each individual subsystem will be described first, which will be followed by the description of the full merged system.

The recognition subsystem:

A fully robust system for the recognition of text was created, it allows for using different datasets and different machine learning models using the same code.

Datasets:

The recognition subsystem can be used with a variety of datasets including: IAM, Saint Gall, Rimes, Bentham and KHATT.

Each dataset is constituted of three parts: Training, validation, and testing.

Each part is stored as a folder that contains the images and a file that contains the ground truth for providing the correct labels for the images.

The required dataset is selected by the user and the path to the dataset is input to the system as well. The system reads each image of the dataset and performs preprocessing on it, once its preprocessed, the image is then stored in a dictionary. The labels are also read from their corresponding file (.txt , .csv, or .xml), they are then also stored in a dictionary.

Once the reading and preprocessing are done, the dictionaries are then saved into an hdf5 file, which allows for both better performance when reading the dataset (can be read to the RAM directly) and for shortening the time by removing the need for preprocessing the whole dataset each time training is done.

As mentioned earlier, an improved version of the KHATT dataset was introduced which yields better accuracy. The dataset reader has a special portion for reading the new dataset due to its difference in construction compared to the vanilla version.

Here is a code sample from the dataset reader of the text recognition subsystem:

```
def _khattx(self):

        """KHATTX dataset reader"""

        symbol_dict = {'hh': "□",'am':"□",'ae': "□",'ah': "·",'aa': "□",'ba': "Ω",
               'ta': "E",'teE': "T",'th': "I",'ja': "N",'ha': "P",'kh': "Y",
               'da': "Ω",'dh': "Ÿ",'ra': "έ",'za': "ί",'se': "α",'sh': "ε",
               'sa': "ι",'de': "ν",'to': "ρ",'zha': "υ",'ay': "ω",'gh': "ύ",
               'fa': "□",'ka': "□",'ke': "□",'la': "□",'ma': "□",'na': "□",
               'he': "□",'wa': "□",'wl': "ΰ",'ya': "□",'ee': "□",'al': "H",
               'n0': "0",'n1': "1",'n2': "2",'n3': "3",'n4': "4",'n5': "5",
               'n6': "6",'n7': "7",'n8': "8",'n9': "9",'atr': "@",'col': ":",
               'dbq': '"','com': ",",'qts': "?",'exc': "!",'dot': ".",'bro': "(",
               'brc': ")",'fsl': "/",'bsl': "\\",'equ': "=",'hyp': "-",'usc': "_",
               'scr': "#",'per': "%", 'sp': " ", 'te': "T"
              }
        def to_symbs(inp):
            temp = inp.split()
            out = ""
            for c in temp:
                out += symbol_dict[c]
            out.replace(" ","")
            return out
        pt_path = os.path.join(self.source, "KHATTX") # os.path.join(self.source, "sets")

        paths = {"train": open(os.path.join(pt_path, "Training-
Groundtruth.txt")).read().splitlines(),
                 "valid": open(os.path.join(pt_path, "Validation-
Groundtruth.txt")).read().splitlines(),
                 "test": open(os.path.join(pt_path, "Test-Groundtruth.txt")).read().splitlines()}

        image_paths = {"train": os.path.join(pt_path, "Training"),
                       "valid": os.path.join(pt_path, "Validation"),
                       "test": os.path.join(pt_path, "Test")}

        dataset = dict()

        for i in self.partitions:
            dataset[i] = {"dt": [], "gt": []}
            image_path = image_paths[i]
            img_list = sorted(os.listdir(image_path))
            txt_file = sorted(paths[i])
            j = 0
            JJ = 0
            for img in img_list:
                found = False
                while j<len(txt file):
                    line = txt_file[j]
                    if(line.find(img)!=-1):
                        found = True
                        break
                    j+=1
                if not found:
                    print(img,i)
                    j = JJ
                    continue
                line = line.replace(img,"")
                line = to_symbs(line)
                dataset[i]['dt'].append(os.path.join(image path, img))
                dataset[i]['gt'].append(line)
                j+=1
                JJ = j
        return dataset
```

Figure (3.1): Recognition dataset reader code

Models:

As for the models, the system allows for the use of various models which are predefined, each of the models was written using the Tensorflow Keras library. The user can select which model to be used as well as setting the learning rate that will be used by the model.

The predefined models make use of the Keras callback functions for saving the best checkpoint weights and for allowing the learning rate to be decreased throughout the training session, this results in higher accuracy and robustness for the trained models.

The code for the models was written in a fashion that makes it highly reusable and modifiable so that future research can build upon it easily.

The code includes definitions for the following architectures: Blueche's architecture, Pugigcerver architecture, and Yazeed's architecture (created by our team) .

Here are code samples from the models section of the text recognition subsystem (only certain parts of the code are shown):

```
class NNModel:

    def __init__(self,
                 architecture,
                 input_size,
                 vocab_size,
                 greedy=False,
                 beam_width=10,
                 top_paths=1):
        """
        Initialization of a NN Model.

        parameters:
            architecture: option of the architecture model to build and compile
            greedy, beam_width, top_paths: Parameters of the CTC decoding
        """

        self.architecture = globals()[architecture]
        self.input_size = input_size
        self.vocab_size = vocab_size

        self.model = None
        self.greedy = greedy
        self.beam_width = beam_width
        self.top_paths = max(1, top_paths)

    def load_checkpoint(self, target):
        """ Load a model with checkpoint file"""

        if os.path.isfile(target):
            if self.model is None:
                self.compile()

            self.model.load_weights(target)

    def compile(self, learning_rate=None):
        """
        Configures the NN Model for training/predict.

        optimizer: optimizer for training
        """

        # define inputs, outputs and optimizer of the chosen architecture
        outs = self.architecture(self.input_size, self.vocab_size + 1, learning_rate)
        inputs, outputs, optimizer = outs

        # create and compile
        self.model = Model(inputs=inputs, outputs=outputs)
        self.model.compile(optimizer=optimizer, loss=self.ctc_loss_lambda_func)
```

Figure (3.1): Recognition models code

Preprocessing:

A variety of preprocessing techniques were defined, each of which are used for both training the models and for acquiring predictions from them.

The preprocessing techniques used includes the following:

Image normalization:

The average of all the images is first calculated along with their standard deviation, then, the value of each pixel in the image is replaced by its value minus the average and then divided by the standard deviation.

It is implemented using the following code:

```
def normalization(imgs):
    """Normalize list of images"""

    imgs = np.asarray(imgs).astype(np.float32)
    _, h, w = imgs.shape

    for i in range(len(imgs)):
        m, s = cv2.meanStdDev(imgs[i])
        imgs[i] = imgs[i] - m[0][0]
        imgs[i] = imgs[i] / s[0][0] if s[0][0] > 0 else imgs[i]
    print("M & S"+str(m[0][0])+" "+str(s[0][0]))
    return np.expand_dims(imgs, axis=-1)
```

Figure (3.1): Image normalization code

Augmentation:

A preprocessing method used usually with smaller datasets (but is also used for larger ones), whereby from each image a set of new images are created which are scaled (resized), dilated (stretched), rotated, eroded, or shifted versions of the original image. This helps increasing the size of the training set for the dataset.

It is implemented using the following code:

```
def augmentation(imgs,
                 rotation range=0,
                 scale range=0,
                 height shift range=0,
                 width shift range=0,
                 dilate range=1,
                 erode_range=1):
    """Apply variations to a list of images (rotate, width and height shift, scale, erode, dilate
)"""

    imgs = imgs.astype(np.float32)
     , h, w = imgs.shape

    dilate kernel = np.ones((int(np.random.uniform(1, dilate range)),), np.uint8)
    erode kernel = np.ones((int(np.random.uniform(1, erode range)),), np.uint8)
    height shift = np.random.uniform(-height shift range, height shift range)
    rotation = np.random.uniform(-rotation range, rotation range)
    scale = np.random.uniform(1 - scale range, 1)
    width_shift = np.random.uniform(-width_shift_range, width_shift_range)

    trans map = np.float32([[1, 0, width shift * w], [0, 1, height shift * h]])
    rot map = cv2.getRotationMatrix2D((w // 2, h // 2), rotation, scale)

    trans_map_aff = np.r_[trans_map, [[0, 0, 1]]]
    rot map aff = np.r [rot map, [[0, 0, 1]]]
    affine mat = rot map aff.dot(trans map aff)[:2, :]

    for i in range(len(imgs)):
        imgs[i] = cv2.warpAffine(imgs[i], affine mat, (w, h), flags=cv2.INTER NEAREST, borderValu
e=255)
        imgs[i] = cv2.erode(imgs[i], erode kernel, iterations=1)
        imgs[i] = cv2.dilate(imgs[i], dilate kernel, iterations=1)

    return imgs
```

Figure (3.1): Augmentation code

Illumination compensation:

Whereby the uneven illumination of images is compensated and images in normal lighting conditions are reconstructed. The reconstructed images are then used for classification. An illumination compensation scheme includes the following modules: lighting category evaluation, shape normalization, and lighting compensation.

It is implemented using the following code:

```
def illumination compensation(img):
    """Illumination compensation technique for text image"""

    def scale(img):
        s = np.max(img) - np.min(img)
        res = img / s
        res -= np.min(res)
        res *= 255
        return res

    img = img.astype(np.float32)
    height, width = img.shape
    sqrt hw = np.sqrt(height * width)

    bins = np.arange(0, 300, 10)
    bins[26] = 255
    hp = np.histogram(img, bins)
    for i in range(len(hp[0])):
        if hp[0][i] > sqrt_hw:
            hr = i * 10
            break

    np.seterr(divide='ignore', invalid='ignore')
    cei = (img - (hr + 50 * 0.3)) * 2
    cei[cei > 255] = 255
    cei[cei < 0] = 0

    m1 = np.asarray([-1, 0, 1, -2, 0, 2, -1, 0, 1]).reshape((3, 3))
    m2 = np.asarray([-2, -1, 0, -1, 0, 1, 0, 1, 2]).reshape((3, 3))
    m3 = np.asarray([-1, -2, -1, 0, 0, 0, 1, 2, 1]).reshape((3, 3))
    m4 = np.asarray([0, 1, 2, -1, 0, 1, -2, -1, 0]).reshape((3, 3))

    eg1 = np.abs(cv2.filter2D(img, -1, m1))
    eg2 = np.abs(cv2.filter2D(img, -1, m2))
    eg3 = np.abs(cv2.filter2D(img, -1, m3))
    eg4 = np.abs(cv2.filter2D(img, -1, m4))

    eg avg = scale((eg1 + eg2 + eg3 + eg4) / 4)

    h, w = eg_avg.shape
    eg bin = np.zeros((h, w))
    eg bin[eg avg >= 30] = 255

    h, w = cei.shape
    cei bin = np.zeros((h, w))
    cei bin[cei >= 60] = 255

    h, w = eg_bin.shape
    tli = 255 * np.ones((h, w))
    tli[eg bin == 255] = 0
    tli[cei_bin == 255] = 0

    kernel = np.ones((3, 3), np.uint8)
    erosion = cv2.erode(tli, kernel, iterations=1)
    int_img = np.asarray(cei)

    estimate_light_distribution(width, height, erosion, cei, int_img)

    mean_filter = 1 / 121 * np.ones((11, 11), np.uint8)
    ldi = cv2.filter2D(scale(int_img), -1, mean_filter)

    result = np.divide(cei, ldi) * 260
    result[erosion != 0] *= 1.5
    result[result < 0] = 0
    result[result > 255] = 255

    return np.asarray(result, dtype=np.uint8)
```

Figure (3.1): Illumination compensation code

Cursive hand written text normalization:

A preprocessing method used for the recognition of cursive handwritten text [8], whereby the cursive handwriting style of the writer is estimated using different methods, and then the text image is reconstructed without the cursive style which improves the recognition accuracy. In this technique, the input image is rotated and text lines are eroded or dilated as required, so that all the characters are vertically separated.

It is implemented using the following code:

```
def remove_cursive_style(img):
    """Remove cursive writing style from image with deslanting algorithm"""

    def calc_y_alpha(vec):
        indices = np.where(vec > 0)[0]
        h_alpha = len(indices)

        if h_alpha > 0:
            delta_y_alpha = indices[h_alpha - 1] - indices[0] + 1

            if h_alpha == delta_y_alpha:
                return h_alpha * h_alpha
        return 0

    alpha_vals = [-1.0, -0.75, -0.5, -0.25, 0.0, 0.25, 0.5, 0.75, 1.0]
    rows, cols = img.shape
    results = []

    ret, otsu = cv2.threshold(img, 0, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)
    binary = otsu if ret < 127 else sauvola(img, (int(img.shape[0] / 2), int(img.shape[0] / 2)),
127, 1e-2)

    for alpha in alpha_vals:
        shift_x = max(-alpha * rows, 0.)
        size = (cols + int(np.ceil(abs(alpha * rows))), rows)
        transform = np.asarray([[1, alpha, shift_x], [0, 1, 0]], dtype=np.float)

        shear_img = cv2.warpAffine(binary, transform, size, cv2.INTER_NEAREST)
        sum_alpha = 0
        sum_alpha += np.apply_along_axis(calc_y_alpha, 0, shear_img)
        results.append([np.sum(sum_alpha), size, transform])

    result = sorted(results, key=lambda x: x[0], reverse=True)[0]
    warp = cv2.warpAffine(img, result[2], result[1], borderValue=255)

    return cv2.resize(warp, dsize=(cols, rows))
```

Figure (3.1): Cursive hand written text normalization code

Main panel:

This part is where the operations entered by the user are interpreted into actions by the system. The main function takes as input a number of parameters that are used to determine the functionality required.

Here is a code that includes the description of usage of the main panel:

```
"""
* source: dataset/model name (iam, rimes, khatt, khattx)
* arch: network to be used (puigcerver, bluche, yazeed)
* transform: transform dataset to the HDF5 file (hdf5 files are better for processing and trainin
g)
* cv2: visualize sample from transformed dataset
* image: predict a single image with the source parameter
* train: train model with the source argument
* test: evaluate and predict model with the source argument
* epochs: number of epochs
* batch_size: number of batches
"""
# NOTE: KHATTX is the cleaned version of KHATT that I created, use KHATT without an "X" for the v
anilla KHATT.
Main(source="khattx",image="/content/drive/My Drive/data/demo/obal.png")
```

Figure (3.1): Main panel for text recognition code

The localization subsystem:

A fully robust system for the localization of text was created, too, it allows for using different datasets and different machine learning models using the same code.

Datasets:

The localization subsystem can be used with a variety of datasets including: SVT, ICPR, ICDAR, and Total Text.

As was the case for text recognition, each dataset is also constituted of three parts: Training, validation, and testing.

Each part is stored as a folder that contains the images and a file that contains the ground truth for providing the correct labels for the images.

The same operations are provided for the text localization datasets as that for text recognition ones, where they are also read from different folders for each of the three main parts (training validation and testing) and are stored at the end in an hdf5 file for the same reasons described in the recognition section.

Though, the main difference between the text recognition and the localization parts is the processing of the ground truth values found in the their corresponding files, where, instead of tokenizing each ground truth instance and saving them as a coded sequence, the values are treated in a different manner.

Each instance of the ground truth for text localization contains the coordinates of the four (or more) corners that constitute the text box, where each image can contain more than one text box. The values required to be output for the network are estimated (based on the EAST network architecture), where first, the activation layer values are calculated, followed by the value that describes whether the pixel is a part of the border area, the values for whether the pixel constitutes an ending or a beginning of a text box, and finally, the scores for each of the 4 geos constituting each box.

Here are code samples from the dataset reader of the text localization subsystem:

```
class Dataset():
    """Dataset class to read images and annonations from base (raw files)"""

    def   init  (self, source, name):
        self.source = source
        self.name = name
        self.dataset = None
        self.partitions = ['train', 'valid','test']

    def read partitions(self):
        """Read images and annonations from dataset"""

        self.dataset = getattr(self, " "+str(self.name))()

    def  icpr(self):

        data dir = os.path.join(self.source, "icpr")
        img_h, img_w = cfg.max_train_img_size, cfg.max_train_img_size
        pixel num h = img h // cfg.pixel size
        pixel_num_w = img_w // cfg.pixel_size
        f_list = dict()
        dataset = dict()
        origin_image_dir = "/content/train_1000/image_1000"
        origin_txt_dir = "/content/train_1000/txt_1000"

        o img list = os.listdir(origin image dir)
        print('found %d origin images.' % len(o_img_list))

        val_count = int(cfg.validation_split_ratio * len(o_img_list))

        dataset['train'] = {"dt": [], "gt": []}
        dataset['valid'] = {"dt": [], "gt": []}
        dataset['test'] = {"dt": [], "gt": []}
        for o_img_fname, I in zip(o_img_list, tqdm(range(len(o_img_list)), position=0, leave=True
)):
            with Image.open(os.path.join(origin_image_dir, o_img_fname)) as im:

                d_wight, d_height = cfg.max_train_img_size, cfg.max_train_img_size
                scale ratio w = d wight / im.width
                scale_ratio_h = d_height / im.height
                im = im.resize((d wight, d height), Image.NEAREST).convert('RGB')

                with open(os.path.join(origin txt dir, o img fname[:-4] + '.txt'), 'r') as f:
                    anno_list = f.readlines()
                xy list array = np.zeros((len(anno list), 4, 2))
                for anno, i in zip(anno_list, range(len(anno_list))):
                    anno_colums = anno.strip().split(',')
                    anno array = np.array(anno colums)
                    xy_list = np.reshape(anno_array[:8].astype(float), (4, 2))
                    xy list[:, 0] = xy list[:, 0] * scale ratio w
                    xy_list[:, 1] = xy_list[:, 1] * scale_ratio_h
                    xy list = reorder vertexes(xy list)
                    xy_list_array[i] = xy_list
                     , shrink xy list,   = shrink(xy list, cfg.shrink ratio)
                    shrink_1, _, long_edge = shrink(xy_list, cfg.shrink_side_ratio)
                gt = np.zeros((d height // cfg.pixel size, d wight // cfg.pixel size, 7))
                for xy_list in xy_list_array:
                   , shrink xy list,   = shrink(xy list, cfg.shrink ratio)
                  shrink_1, _, long_edge = shrink(xy_list, cfg.shrink_side_ratio)
                  p_min = np.amin(shrink_xy_list, axis=0)
                  p max = np.amax(shrink xy list, axis=0)
                  # floor of the float
                  ji min = (p min / cfg.pixel size - 0.5).astype(int) - 1
                  # +1 for ceil of the float and +1 for include the end
                  ji max = (p max / cfg.pixel size - 0.5).astype(int) + 3
                  imin = np.maximum(0, ji_min[1])
                  imax = np.minimum(d height // cfg.pixel size, ji max[1])
                  jmin = np.maximum(0, ji_min[0])
                  jmax = np.minimum(d_wight // cfg.pixel_size, ji_max[0])
                  # OMMITTED CODE FOR SPACE
        return dataset
```

Figure (3.1): Localization dataset reader code

Models:

As for the models, the system primarily uses the EAST model for localization of text, the model is defined and written using the Tensorflow Keras library. The user can select learning rate that will be used by the model along with other useful features that are included in the main panel part.

The predefined model make use of the Keras callback functions for saving the best checkpoint weights and for allowing the learning rate to be decreased throughout the training session, this results in higher accuracy and robustness for the trained models.

General architecture for the EAST model:

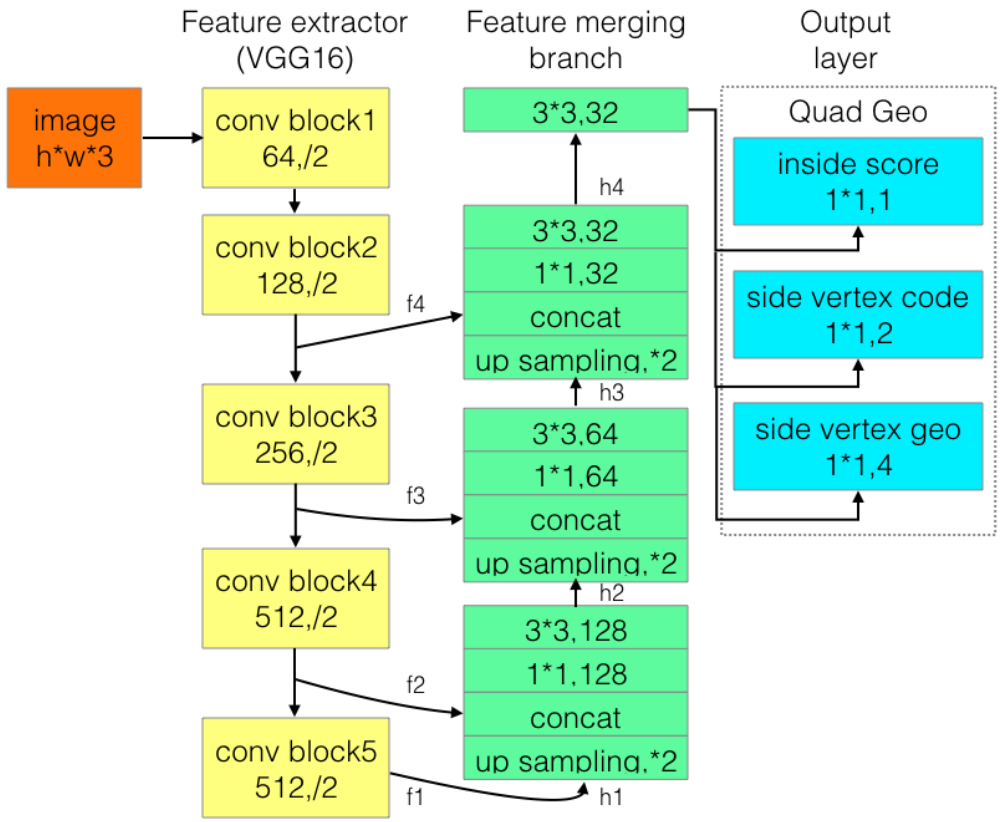


Figure (3.1): EAST model architecture

Here are code samples from the models section of the text localization subsystem (some parts has been omitted):

```
class NNModel:

    def   init  (self,
                 architecture="east",
                 num channels=3,
                 locked layers=False,
                 feature_layers_range=range(5, 1, -1),
                 feature_layers_num=4,
                 learning_rate=1e-3
                ):
        """
        Initialization of a NN Model.

        parameters:
            architecture: option of the architecture model to build and compile.
        """

        self.architecture = globals()[architecture]
        self.num_channels = num_channels
        self.locked_layers = locked_layers
        self.feature_layers_range = feature_layers_range
        self.feature layers num = feature layers num
        self.learning rate = learning rate

        self.model= None

    def compile(self, learning rate=None):
        """
        Configures the NN Model for training/predict.

        optimizer: optimizer for training
        """

        # define inputs, outputs and optimizer of the chosen architecture
        outs = self.architecture(self.num channels, self.locked layers, self.feature layers range
, self.feature layers num, self.learning rate)
        inputs, outputs, optimizer = outs

        # create and compile
        self.model = Model(inputs=inputs, outputs=outputs)
        self.model.compile(optimizer=optimizer, loss=self.quad loss)

    def fit(self,
            x=None,
            y=None,
            batch_size=None,
            epochs=1,
            verbose=1,
            callbacks=None,
            validation_split=0.0,
            validation_data=None,
            shuffle=True,
            class weight=None,
            sample weight=None,
            initial epoch=0,
            steps_per_epoch=None,
            validation_steps=None,
            validation_freq=1,
            max queue size=10,
            workers=1,
            use multiprocessing=False,
            checkpoint=""):
        out = self.model.fit_generator(generator=x,
                                       steps_per_epoch=steps_per_epoch,
                                       epochs=epochs,
                                       validation data=validation data,
                                       validation steps=validation steps,
                                       verbose=1,
                                       initial_epoch=initial_epoch,
                                       callbacks=callbacks)
        return out
```

Figure (3.1): Localization models code

Here is the code for the loss function used for training the model:

```
@staticmethod
def quad loss(y true, y pred):

    epsilon = 1e-6
    lambda_side_vertex_coord_loss = 1.0
    lambda side vertex code loss = 1.0
    lambda_inside_score_loss = 4.0
    def smooth l1 loss(prediction tensor, target tensor, weights):
        n_q = tf.reshape(quad_norm(target_tensor), tf.shape(weights))
        diff = prediction tensor - target tensor
        abs_diff = tf.abs(diff)
        abs diff lt 1 = tf.less(abs diff, 1)
        pixel_wise_smooth_l1norm = (tf.reduce_sum(
            tf.where(abs diff lt 1, 0.5 * tf.square(abs diff), abs diff - 0.5),
            axis=-1) / n_q) * weights
        return pixel wise smooth l1norm


    def quad norm(g true):
        shape = tf.shape(g_true)
        delta xy matrix = tf.reshape(g true, [-1, 2, 2])
        diff = delta_xy_matrix[:, 0:1, :] - delta_xy_matrix[:, 1:2, :]
        square = tf.square(diff)
        distance = tf.sqrt(tf.reduce_sum(square, axis=-1))
        distance *= 4.0
        distance += epsilon
        return tf.reshape(distance, shape[:-1])


    # loss for inside_score
    logits = y pred[:, :, :, :1]
    labels = y_true[:, :, :, :1]
    # balance positive and negative samples in an image
    beta = 1 - tf.reduce_mean(labels)
    # first apply sigmoid activation
    predicts = tf.nn.sigmoid(logits)
    # log +epsilon for stable cal
    inside_score_loss = tf.reduce_mean(
        -1 * (beta * labels * tf.math.log(predicts + epsilon) +
              (1 - beta) * (1 - labels) * tf.math.log(1 - predicts + epsilon)))
    inside score loss *= lambda inside score loss

    # loss for side vertex code
    vertex_logits = y_pred[:, :, :, 1:3]
    vertex labels = y true[:, :, :, 1:3]
    vertex_beta = 1 - (tf.reduce_mean(y_true[:, :, :, 1:2])
                       / (tf.reduce mean(labels) + epsilon))
    vertex_predicts = tf.nn.sigmoid(vertex_logits)
    pos = -1 * vertex beta * vertex labels * tf.math.log(vertex predicts +
                                                         epsilon)
    neg = -1 * (1 - vertex beta) * (1 - vertex labels) * tf.math.log(
        1 - vertex_predicts + epsilon)
    positive weights = tf.cast(tf.equal(y true[:, :, :, 0], 1), tf.float32)
    side_vertex_code_loss = \
        tf.reduce sum(tf.reduce sum(pos + neg, axis=-1) * positive weights) / (
                tf.reduce_sum(positive_weights) + epsilon)
    side vertex code loss *= lambda side vertex code loss

    # loss for side vertex coord delta
    g_hat = y_pred[:, :, :, 3:]
    g true = y true[:, :, :, 3:]
    vertex_weights = tf.cast(tf.equal(y_true[:, :, :, 1], 1), tf.float32)
    pixel wise smooth l1norm = smooth l1 loss(g hat, g true, vertex weights)
    side_vertex_coord_loss = tf.reduce_sum(pixel_wise_smooth_l1norm) / (
            tf.reduce sum(vertex weights) + epsilon)
    side_vertex_coord_loss *= lambda_side_vertex_coord_loss

    return inside_score_loss + side_vertex_code_loss + side_vertex_coord_loss
```

Figure (3.1): The quad loss function code

And, here is the code for defining the EAST model:

```
def east(num_channels, locked_layers, feature_layers_range, feature_layers_num, learning_rate):
    input_img = Input(name='input_img',
                              shape=(None, None, num_channels),
                              dtype='float32')
    vgg16 = VGG16(input_tensor=input_img,
                  weights='imagenet',
                  include_top=False)
    if locked layers:
        # locked first two conv layers
        locked layers = [vgg16.get layer('block1 conv1'),
                         vgg16.get_layer('block1_conv2')]
        for layer in locked_layers:
            layer.trainable = False
    f = [vgg16.get_layer('block%d_pool' % i).output
              for i in feature layers range]
    f.insert(0, None)
    diff = feature_layers_range[0] - feature_layers_num

    def g(i):
        # i+diff in cfg.feature layers range
        assert i + diff in feature_layers_range, \
            ('i=%d+diff=%d not in ' % (i, diff)) + \
            str(feature_layers_range)
        if i == feature_layers_num:
            bn = BatchNormalization()(h(i))
            cn = ConvZD(32, 3, activation='relu', padding='same')(bn)
            return cn
        else:
            return UpSamplingZD((2, 2))(h(i))

    def h(i):
        # i+diff in cfg.feature_layers_range
        assert i + diff in feature_layers_range, \
            ('i=%d+diff=%d not in ' % (i, diff)) + \
            str(feature layers range)
        if i == 1:
            return f[i]
        else:
            concat = Concatenate(axis=-1)([g(i - 1), f[i]])
            bn1 = BatchNormalization()(concat)
            conv_1 = ConvZD(128 // 2 ** (i - 2), 1,
                            activation='relu', padding='same',)(bn1)
            bn2 = BatchNormalization()(conv_1)
            conv_3 = ConvZD(128 // 2 ** (i - 2), 3,
                            activation='relu', padding='same',)(bn2)
            return conv_3

    before_output = g(feature_layers_num)
    inside_score = ConvZD(1, 1, padding='same', name='inside_score'
                          )(before output)
    side_v_code = ConvZD(2, 1, padding='same', name='side_vertex_code'
                         )(before output)
    side_v_coord = ConvZD(4, 1, padding='same', name='side_vertex_coord'
                          )(before_output)
    east_detect = Concatenate(axis=-1,
                              name='east_detect')([inside_score,
                                                   side_v_code,
                                                   side_v_coord])

    optimizer = Adam(lr=learning rate, decay=5e-4)

    return (input_img, east_detect, optimizer)
```

Figure (3.1): EAST model code

Preprocessing:

The same set preprocessing techniques were defined as for the text recognition subsystem.

Post-processing:

The post-processing techniques used includes the following:

NMS:

Non-Maximum Suppression is a post-processing technique used in text localization for the EAST model, where the activation layer together with the geometry layers of the model output is used to calculate the coordinates of the bounding boxes.

It is implemented using the following code (this is the simplified version, a more sophisticated method was used in the actual code to yield better results, and the simplified version is shown instead for space):

```
def nms2(predict, activation_pixels, threshold=cfg.side_vertex_pixel_threshold):
    region list = []
    for i, j in zip(activation pixels[0], activation pixels[1]):
        merge = False
        for k in range(len(region list)):
            if should merge(region list[k], i, j):
                region_list[k].add((i, j))
                merge = True
                # Fixme
                # break
        if not merge:
            region list.append({(i, j)})
    D = region_group(region_list)
    quad list = np.zeros((len(D), 4, 2))
    score list = np.zeros((len(D), 4))
    for group, g th in zip(D, range(len(D))):
        total score = np.zeros((4, 2))
        for row in group:
            for ij in region_list[row]:
                score = predict[ij[0], ij[1], 1]
                if score >= threshold:
                    ith score = predict[ij[0], ij[1], 2:3]
                    if not (cfg.trunc threshold <= ith score < 1 -
                            cfg.trunc_threshold):
                        ith = int(np.around(ith score))
                        total score[ith * 2:(ith + 1) * 2] += score
                        px = (ij[1] + 0.5) * cfg.pixel size
                        py = (ij[0] + 0.5) * cfg.pixel size
                        p v = [px, py] + np.reshape(predict[ij[0], ij[1], 3:7],
                                                    (2, 2))
                        quad list[g th, ith * 2:(ith + 1) * 2] += score * p v
        score list[g th] = total score[:, 0]
        quad list[g th] /= (total score + cfg.epsilon)
    return score_list, quad_list
```

Figure (3.1): NMS code

Box geometry manipulation:

Used for shrinking and expanding the text boxes detected in the image.

Code samples:

```
def shrink(xy_list, ratio=cfg.shrink_ratio):
    if ratio == 0.0:
        return xy_list, xy_list
    diff_1to3 = xy_list[:3, :] - xy_list[1:4, :]
    diff_4 = xy_list[3:4, :] - xy_list[0:1, :]
    diff = np.concatenate((diff_1to3, diff_4), axis=0)
    dis = np.sqrt(np.sum(np.square(diff), axis=-1))
    # determine which are long or short edges
    long_edge = int(np.argmax(np.sum(np.reshape(dis, (2, 2)), axis=0)))
    short_edge = 1 - long_edge
    # cal r length array
    r = [np.minimum(dis[i], dis[(i + 1) % 4]) for i in range(4)]
    # cal theta array
    diff_abs = np.abs(diff)
    diff_abs[:, 0] += cfg.epsilon
    theta = np.arctan(diff_abs[:, 1] / diff_abs[:, 0])
    # shrink two long edges
    temp_new_xy_list = np.copy(xy_list)
    shrink_edge(xy_list, temp_new_xy_list, long_edge, r, theta, ratio)
    shrink_edge(xy_list, temp_new_xy_list, long_edge + 2, r, theta, ratio)
    # shrink two short edges
    new_xy_list = np.copy(temp_new_xy_list)
    shrink_edge(temp_new_xy_list, new_xy_list, short_edge, r, theta, ratio)
    shrink_edge(temp_new_xy_list, new_xy_list, short_edge + 2, r, theta, ratio)
    return temp_new_xy_list, new_xy_list, long_edge


def shrink_edge(xy_list, new_xy_list, edge, r, theta, ratio=cfg.shrink_ratio):
    if ratio == 0.0:
        return
    start_point = edge
    end_point = (edge + 1) % 4
    long_start_sign_x = np.sign(
        xy_list[end_point, 0] - xy_list[start_point, 0])
    new_xy_list[start_point, 0] = \
        xy_list[start_point, 0] + \
        long_start_sign_x * ratio * r[start_point] * np.cos(theta[start_point])
    long_start_sign_y = np.sign(
        xy_list[end_point, 1] - xy_list[start_point, 1])
    new_xy_list[start_point, 1] = \
        xy_list[start_point, 1] + \
        long_start_sign_y * ratio * r[start_point] * np.sin(theta[start_point])
    # long edge one, end point
    long_end_sign_x = -1 * long_start_sign_x
    new_xy_list[end_point, 0] = \
        xy_list[end_point, 0] + \
        long_end_sign_x * ratio * r[end_point] * np.cos(theta[start_point])
    long_end_sign_y = -1 * long_start_sign_y
    new_xy_list[end_point, 1] = \
        xy_list[end_point, 1] + \
        long_end_sign_y * ratio * r[end_point] * np.sin(theta[start_point])
```

Figure (3.1): Shrink box / shrink edge code

```
def batch_reorder_vertexes(xy_list_array):
    reorder xy list array = np.zeros like(xy list array)
    for xy_list, i in zip(xy_list_array, range(len(xy_list_array))):
        reorder_xy_list_array[i] = reorder_vertexes(xy_list)
    return reorder_xy_list_array


def reorder_vertexes(xy_list):
    reorder xy list = np.zeros like(xy list)
    # determine the first point with the smallest x,
    # if two has same x, choose that with smallest y,
    ordered = np.argsort(xy_list, axis=0)
    xmin1_index = ordered[0, 0]
    xmin2_index = ordered[1, 0]
    if xy_list[xmin1_index, 0] == xy_list[xmin2_index, 0]:
        if xy_list[xmin1_index, 1] <= xy_list[xmin2_index, 1]:
            reorder_xy_list[0] = xy_list[xmin1_index]
            first_v = xmin1_index
        else:
            reorder_xy_list[0] = xy_list[xmin2_index]
            first_v = xmin2_index
    else:
        reorder_xy_list[0] = xy_list[xmin1_index]
        first_v = xmin1_index
    # connect the first point to others, the third point on the other side of
    # the line with the middle slope
    others = list(range(4))
    others.remove(first_v)
    k = np.zeros((len(others),))
    for index, i in zip(others, range(len(others))):
        k[i] = (xy_list[index, 1] - xy_list[first_v, 1]) \
                    / (xy list[index, 0] - xy list[first v, 0] + cfg.epsilon)
    k_mid = np.argsort(k)[1]
    third_v = others[k_mid]
    reorder_xy_list[2] = xy_list[third_v]
    # determine the second point which on the bigger side of the middle line
    others.remove(third_v)
    b_mid = xy_list[first_v, 1] - k[k_mid] * xy_list[first_v, 0]
    second_v, fourth_v = 0, 0
    for index, i in zip(others, range(len(others))):
        # delta = y - (k * x + b)
        delta y = xy list[index, 1] - (k[k mid] * xy list[index, 0] + b mid)
        if delta_y > 0:
            second_v = index
        else:
            fourth_v = index
    reorder_xy_list[1] = xy_list[second_v]
    reorder_xy_list[3] = xy_list[fourth_v]
    # compare slope of 13 and 24, determine the final order
    k13 = k[k_mid]
    k24 = (xy_list[second_v, 1] - xy_list[fourth_v, 1]) / (
                xy_list[second_v, 0] - xy_list[fourth_v, 0] + cfg.epsilon)
    if k13 < k24:
        tmp_x, tmp_y = reorder_xy_list[3, 0], reorder_xy_list[3, 1]
        for i in range(2, -1, -1):
            reorder_xy_list[i + 1] = reorder_xy_list[i]
        reorder_xy_list[0, 0], reorder_xy_list[0, 1] = tmp_x, tmp_y
    return reorder_xy_list
```

Figure (3.1): Reorder vertices code

Main panel:

This part is where the operations entered by the user are interpreted into actions by the system. The main function takes as input a number of parameters that are used to determine the functionality required.

Here is a description that includes a code line example of usage of the main panel:

Main(source,arch,best,lr,transform,image,train,epochs,batch_size**)**

source: dataset/model name (*totaltext*, *icpr*, *icpr2*, *svt*).

arch: network to be used (*east*).

transform: transform dataset to the HDF5 file (hdf5 files are better for processing and training).

cv2: visualize sample from transformed dataset.

image: predict a single image with the source parameter, Special inputs: "**sample**" -> random sample from dataset, "**sampleOut**" -> random sample from dataset with its groundtruth as output.

train: train model with the source argument.

epochs: number of epochs.

batch_size: number of examples per batch.

```
Main(source="icpr", image="/content/drive/My Drive/data/demo/sign1.jpg")
```

Figure (3.1): Main panel for text localization

The replacement subsystem:

This subsystem generally provides two functionalities, the translation of the recognized text, and the replacement of the text in the general image with that recognized text.

Translation:

Translation of the recognized text is performed through an HTTP request to the google translation API, the user is able to specify if translation is to be applied or not, in addition to the output language.

It is implemented using the following code:

```
agent = {'User-Agent':
"Mozilla/4.0 (\
compatible;\
MSIE 6.0;\
Windows NT 5.1;\
SV1;\
.NET CLR 1.1.4322;\
.NET CLR 2.0.50727;\
.NET CLR 3.0.04506.30\
)"}


def unescape(text):
    if (sys.version_info[0] < 3):
        parser = HTMLParser.HTMLParser()
    else:
        parser = html.parser.HTMLParser()
    return (html.unescape(text))


def translate(to_translate, to_language="auto", from_language="auto"):
    to_language = cfg.translation_lang
    base_link = "http://translate.google.com/m?hl=%s&sl=%s&q=%s"#"trans-
late.google.com/m?tl=%s&sl=%s&q=%s&hl=%s"
    if (sys.version_info[0] < 3):
        to_translate = urllib.quote_plus(to_translate)
        link = base_link % (to_language, from_language, to_translate, from_language)
        request = urllib2.Request(link, headers=agent)
        raw_data = urllib2.urlopen(request).read()
    else:
        to_translate = urllib.parse.quote(to_translate)
        link = base_link % (to_language, from_language, to_translate)
        request = urllib.request.Request(link, headers=agent)
        raw_data = urllib.request.urlopen(request).read()
    data = raw_data.decode("utf-8")
    expr = r'class="t0">(.*?)<'
    re_result = re.findall(expr, data)
    if (len(re_result) == 0):
        result = ""
    else:
        result = unescape(re_result[0])
    return (result)
```

Figure (3.1): Translation code

Replacement:

An estimation of the text color and that of the background is obtained and returned by the replacement module, where then, text boxes in the original image are replaced with blank boxes with the estimated color background, each box containing the recognized text line instead of the original one, translation can be applied before this stage to obtain translated text.

It is implemented using the following code:

```
!pip install --upgrade arabic-reshaper
from PIL import Image
from PIL import ImageFont
from PIL import ImageDraw
import arabic_reshaper
from keras.preprocessing import image as image

def replace_all(img_path, text_colors, back_colors, geo_list, text_list):
    img = image.load_img(img_path)

    d_wight, d_height = resize_image(img, cfg.max_predict_img_size)
    img = img.resize((d_wight, d_height), Image.NEAREST).convert('RGB')
    img = image.img_to_array(img)
    b, g, r = img[:,:,0],img[:,:,1],img[:,:,2]
    img = np.dstack([r,g,b])

    for geo, text, tclr, bclr in zip(geo_list, text_list, text_colors, back_colors):
        img = replace_box(img, geo, text, tclr, bclr)
    cv2_imshow(img)
def replace_box(img,geo,text, bclr, tclr):
    if(text == "#" or text == " " or text == ""):
        return img
    _, geo, _ = shrink(geo,ratio = cfg.expand_ratio*0.5)
    geo = square_geo(geo)
    p_min = np.amin(geo, axis=0)
    p_max = np.amax(geo, axis=0)
    min_xy = p_min.astype(int)
    max_xy = p_max.astype(int) + 2
    sub_im_arr = img
    for m in range(min_xy[1], max_xy[1]):
        for n in range(min_xy[0], max_xy[0]):
            if point_inside_of_quad(n, m, geo, p_min, p_max):
                #print("SIZE",sub_im_arr.shape[0],sub_im_arr.shape[1])
                if sub_im_arr.shape[0] > 0 and sub_im_arr.shape[1] > 0 and (m - min_xy[1]) < sub_
im_arr.shape[0] and (n - min_xy[0]) < sub_im_arr.shape[1]:
                    sub_im_arr[m, n, :] = [tclr[0],tclr[1],tclr[2]]
    img = sub_im_arr
    ## FOR ENGLISH TEXT
    Arabic = CheckLang(text)
    if(not Arabic):
      box_height = max_xy[1]-min_xy[1]
      font                   = cv2.FONT_HERSHEY_SIMPLEX
      bottomLeftCornerOfText = (int(min_xy[0]+0.2*box_height),int(max_xy[1]-0.25*box_height))
      fontScale              = box_height /60
      print(tclr, bclr)
      fontColor              = (int(bclr[0]),int(bclr[1]),int(bclr[2]))
      thickness              = box_height//20
      lineType               = cv2.LINE_AA
      cv2.putText(img,translate(text),
      bottomLeftCornerOfText,
      font,
      fontScale,
      fontColor,thickness)
    ## FOR ARABIC TEXT
    else:
      b,g,r,a = int(bclr[0]),int(bclr[1]),int(bclr[2]),0
      fontpath = "/content/drive/My Drive/data/fonts/tradbdo.ttf"
      box_height = max_xy[1]-min_xy[1]
      font = ImageFont.truetype(fontpath, int(box_height*0.8))
      img_pil = Image.fromarray(img.astype(np.uint8)).convert('RGB')
      #img_pil = Image.fromarray(img)
      draw = ImageDraw.Draw(img_pil)
      text_to_be_reshaped = text
      reshaped_text = arabic_reshaper.reshape(text_to_be_reshaped)
      print_text = ''.join(reversed(reshaped_text))
      draw.text((min_xy[0]+0.25*box_height,min_xy[1]-
0.2*box_height),  print_text, font = font, fill = (b, g, r, a))
      img = np.array(img_pil)
    return img
def CheckLang(text):
    return not cfg.translate_result
```

Figure (3.1): Replacement code

The merged system (the full system):

This system generally contains the previously described systems, where each system contributes to the final result of full system, the user is also provided with the ability to select a variety of different optional settings.

Localization:

The localization part contains a function that allows for reading a pre-trained text localization model, the model is then used to extract the locations of the text lines which are then cut into smaller boxes. The smaller boxes are then manipulated in geometry in order to expand them to cover the full text line or word.

Here are some code samples:

```
def predict_txt(east_detect, img_path, txt_path, pixel_threshold, quiet=False):
    img = image.load img(img path)
    d wight, d height = resize image(img, cfg.max predict img size)
    scale ratio w = d wight / img.width
    scale ratio h = d height / img.height
    img = img.resize((d wight, d height), Image.NEAREST).convert('RGB')
    img = image.img to array(img)
    img = preprocess input(img, mode='tf')
    x = np.expand dims(img, axis=0)
    y = east detect.predict(x)

    y = np.squeeze(y, axis=0)
    y[:, :, :3] = sigmoid(y[:, :, :3])
    cond = np.greater equal(y[:, :, 0], pixel threshold)
    activation pixels = np.where(cond)
    quad scores, quad after nms = nms(y, activation pixels)

    txt items = []
    for score, geo in zip(quad scores, quad after nms):
        if np.amin(score) > 0:
            rescaled_geo = geo / [scale_ratio_w, scale_ratio_h]
            rescaled geo list = np.reshape(rescaled geo, (8,)).tolist()
            txt item = ','.join(map(str, rescaled geo list))
            txt items.append(txt item + '\n')
        elif not quiet:
            print('quad invalid with vertex num less then 4.')
    if cfg.predict write2txt and len(txt items) > 0:
        with open(txt path, 'w') as f txt:
            f txt.writelines(txt items)
def square geo(geo):
    result = geo.copy()
    p_min = np.min(geo,axis=0)
    p max = np.max(geo,axis=0)
    result = [[p_min[0],p_min[1]],[p_min[0],p_max[1]],[p_max[0],p_max[1]],[p_max[0],p_min[1]]]
    return np.asarray(result)
def cut text line(geo, scale ratio w, scale ratio h, im array, img path, s):
    geo /= [scale ratio w, scale ratio h]

     , geo,   = shrink(geo,ratio = cfg.expand ratio)
    geo orig = geo.copy()
    geo = square geo(geo)
    #print("GEO: ",geo)
    p min = np.amin(geo, axis=0)
    p_max = np.amax(geo, axis=0)
    min xy = p min.astype(int)
    max xy = p max.astype(int) + 2
    sub im arr = im array[min xy[1]:max xy[1], min xy[0]:max xy[0], :].copy()
    return binarize(sub_im_arr,geo_orig)
```

Figure (3.1): Main system localization code, Main system localization code, cutting the text line

```
def predict_image(detector, img_path, pixel_threshold, quiet=True,gen=None):
    results = []
    text_colors = []
    back_colors = []
    geos = []
    d wight, d height = resize image(img, cfg.max predict img size)
    img = img.resize((d_wight, d_height), Image.NEAREST).convert('RGB')
    img = image.img to array(img)
    img = preprocess input(img, mode='tf')
    b, g, r = img[:,:,0],img[:,:,1],img[:,:,2]
    img = np.dstack([r,g,b])
    x = np.expand dims(img, axis=0)
    y = detector.predict(x)
    y = np.squeeze(y, axis=0)
    y[:, :, :3] = sigmoid(y[:, :, :3])
    opt = np.get printoptions()
    """np.set_printoptions(threshold=np.inf)
    #print(np.asarray(y[:,:,1]))
    print(np.asarray(y[:,:,2]))
    np.set_printoptions(**opt)"""
    cond = np.greater_equal(y[:, :, 0],sigmoid(pixel_threshold))
    activation pixels = np.where(cond)
    quad_scores, quad_after_nms = nms(y, activation_pixels)
    im = image.array to img(img)
    im array = image.img to array(im.convert('RGB'))
    d wight, d height = resize image(im, cfg.max predict img size)
    scale ratio w = d wight / im.width
    scale ratio h = d height / im.height
    im = im.resize((d wight, d height), Image.NEAREST).convert('RGB')
    quad_im = im.copy()
    draw = ImageDraw.Draw(im)
    for i, j in zip(activation_pixels[0], activation_pixels[1]):
        px = (j + 0.5) * cfg.pixel_size
        py = (i + 0.5) * cfg.pixel_size
        line_width, line_color = 1, 'red'
        if y[i, j, 1] >= sigmoid(cfg.side_vertex_pixel_threshold):
            if y[i, j, 2] < sigmoid(cfg.trunc threshold):
                line width, line color = 2, 'yellow'
            elif y[i, j, 2] >= sigmoid(1-cfg.trunc threshold):
                line width, line color = 2, 'green'
        draw.line([(px - 0.5 * cfg.pixel size, py - 0.5 * cfg.pixel size),
                   (px + 0.5 * cfg.pixel size, py - 0.5 * cfg.pixel size),
                   (px + 0.5 * cfg.pixel size, py + 0.5 * cfg.pixel size),
                   (px - 0.5 * cfg.pixel size, py + 0.5 * cfg.pixel size),
                   (px - 0.5 * cfg.pixel size, py - 0.5 * cfg.pixel size)],
                  width=line_width, fill=line_color)
    quad draw = ImageDraw.Draw(quad im)
    txt_items = []
    for score, geo, s in zip(quad_scores, quad_after_nms,
                             range(len(quad_scores))):
        if np.amin(score) > sigmoid(0):
             ,geo2,  = shrink(geo,ratio = cfg.expand ratio)
            quad draw.line([tuple(geo2[0]),
                            tuple(geo2[1]),
                            tuple(geo2[2]),
                            tuple(geo2[3]),
                            tuple(geo2[0])], width=2, fill='red')
            if cfg.predict cut text line:
                x, txt clr, back clr = cut text line(geo, scale ratio w, scale ratio h, im array,
                                img path, s)
                if x is not None:
                    results.append(x)
                    text_colors.append(txt_clr)
                    back_colors.append(back_clr)
                    geos.append(geo)
            rescaled_geo = geo / [scale_ratio_w, scale_ratio_h]
            rescaled_geo_list = np.reshape(rescaled_geo, (8,)).tolist()
            txt item = ','.join(map(str, rescaled geo list))
            txt items.append(txt item + '\n')
    return results, text_colors, back_colors, geos, image.img_to_array(quad_im.convert('RGB'))
```

Figure (3.1): Main system localization code, Image prediction part

Binarization:

In this section, each of the extracted text boxes is introduced into the text binarization algorithm. Which is constituted of the following steps:

1) Converting the image to gray scale
2) Applying canny edge detection
3) Filling the blobs in the image with predefined colors
4) Sharpening the image
5) Applying a median filter to the image

Code samples for functions used in the binarization process:

```
def thresholder(img, w_size=15,k=0.5):
    rows, cols = img.shape
    i_rows, i_cols = rows + 1, cols + 1
    integ = np.zeros((i_rows, i_cols), np.float)
    sqr_integral = np.zeros((i_rows, i_cols), np.float)

    integ[1:, 1:] = np.cumsum(np.cumsum(img.astype(np.float), axis=0), axis=1)
    sqr_img = np.square(img.astype(np.float))
    sqr_integral[1:, 1:] = np.cumsum(np.cumsum(sqr_img, axis=0), axis=1)
    x, y = np.meshgrid(np.arange(1, i_cols), np.arange(1, i_rows))
    hw_size = w_size // 2
    x1 = (x - hw_size).clip(1, cols)
    x2 = (x + hw_size).clip(1, cols)
    y1 = (y - hw_size).clip(1, rows)
    y2 = (y + hw_size).clip(1, rows)
    l size = (y2 - y1 + 1) * (x2 - x1 + 1)
    sums = (integ[y2, x2] - integ[y2, x1 - 1] -
            integ[y1 - 1, x2] + integ[y1 - 1, x1 - 1])
    sqr_sums = (sqr_integral[y2, x2] - sqr_integral[y2, x1 - 1] -
                sqr integral[y1 - 1, x2] + sqr integral[y1 - 1, x1 - 1])
    means = sums / l_size
    stds = np.sqrt(sqr sums / l size - np.square(means))
    max_std = np.max(stds)
    min_v = np.min(img)
    thresholds = ((1.0 - k) * means + k * min v + k * stds /
                  max_std * (means - min_v))

    return thresholds

def imFill(im_in):
    th, im th = cv2.threshold(im in, 0, 128, cv2.THRESH BINARY)
    im_floodfill = im_th.copy()
    h, w = im_th.shape[:2]
    mask = np.zeros((h+2, w+2), np.uint8)
    cv2.floodFill(im_floodfill, mask, (0,0), 255)
    im_floodfill_inv = cv2.bitwise_not(im_floodfill)
    im_out = im_th | im_floodfill_inv

    return im_out
def apply threshold(img, threshold=128, wp val=255):
    return ((img >= threshold) * wp_val).astype(np.uint8)
```

Figure (3.1): Binarization code

Binarization is implemented using the following code (some parts are omitted for simplicity):

```
def binarize (img_path, geo):
    Use MedianFilter = cfg.Use MedianFilter
    raw_image = img_path
    if raw image.shape[0] == 0 or raw image.shape[1] == 0:
        return None ,None,None
    if(len(raw image.shape)!=2):
        grayScale=cv2.cvtColor(raw_image,cv2.COLOR_BGR2GRAY).astype('uint8')
    else:
        grayScale=raw_image.astype('uint8')
    m,n = grayScale.shape
    high thresh, thresh im = cv2.threshold(grayScale, 0, 255, cv2.THRESH BINARY+ cv2.THRESH OTSU)
    lowThresh = 0.5*high_thresh
    canny Image = cv2.Canny(grayScale,200,100)
    kernel = np.ones((3,3),np.uint8)
    p min = np.amin(geo, axis=0)
    p_max = np.amax(geo, axis=0)
    min_xy = p_min.astype(int)
    max xy = p max.astype(int) + 2
    sub_im_arr = canny_Image.copy()
    for m in range(min xy[1], max xy[1]):
        for n in range(min_xy[0], max_xy[0]):
            if not point inside of quad(n, m, geo, p min, p max):
                if sub_im_arr.shape[0] > 0 and sub_im_arr.shape[1] > 0 and (m - min_xy[1]) < sub_
im arr.shape[0] and (n - min xy[0]) < sub im arr.shape[1]:
                    sub_im_arr[m - min_xy[1], n - min_xy[0]] = 0
    canny_Image = sub_im_arr
    closing = imFill(canny Image)
    if(cfg.show_process):
      print("Fill")
      cv2_imshow(closing)
    inds = np.where(closing==255)
    inds2 = np.where(closing==0)
    pix val = grayScale[inds]
    pix3_valR = np.median(raw_image[:,:,0][inds])
    pix3 valG = np.median(raw image[:,:,1][inds])
    pix3_valB = np.median(raw_image[:,:,2][inds])
    pix_val_back = grayScale[inds2]
    pix3 val backR = np.median(raw image[:,:,0][inds2])
    pix3_val_backG = np.median(raw_image[:,:,1][inds2])
    pix3 val backB = np.median(raw image[:,:,2][inds2])
    median_text=np.median(pix_val,axis=0)
    median back=np.median(pix val back,axis=0)
    if median text>median back:
        pre_binarize = cv2.bitwise_not(grayScale)
    else:
        pre_binarize=grayScale
    sharpen kernel = np.array([[0,-0.9,0], [-1,4.9,-1], [0,-1,-0]])
    new_img = cv2.filter2D(pre_binarize, -1, sharpen_kernel)
    img= apply threshold(new img,thresholder(new img))
    if Use_MedianFilter:
        img = cv2.medianBlur(img,3)
    sub_im_arr = img.copy()
    for m in range(min_xy[1], max_xy[1]):
        for n in range(min xy[0], max xy[0]):
            if not point_inside_of_quad(n, m, geo, p_min, p_max):
                if sub im arr.shape[0] > 0 and sub im arr.shape[1] > 0 and (m - min xy[1]) < sub
im_arr.shape[0] and (n - min_xy[0]) < sub_im_arr.shape[1]:
                    sub im arr[m - min xy[1], n - min xy[0]] = 255
    img = sub_im_arr
    if(cfg.show process):
        print("Final: ")
    wt, ht, _ = (1024,64,0)
    h, w = np.asarray(img).shape[:2]
    f = max((w / wt), (h / ht))
    new size = (max(min(wt, int(w / f)), 1), max(min(ht, int(h / f)), 1))
    img = cv2.resize(img, new_size)
    target = np.ones([ht, wt], dtype=np.uint8) * 255
    target[0:new_size[1], 0:new_size[0]] = img
    img = cv2.transpose(target)
    img = cv2.flip(img,0)
    return img, (pix3_valR,pix3_valG,pix3_valB), (pix3_val_backR,pix3_val_backG,pix3_val_backB)
```

Figure (3.1): Binarization code

Recognition:

The recognition part contains a function that allows for reading a pre-trained text recognition model, the user must describe what type of model is the pre-trained model. This is done to assure that the model reading function is compatible with the pre-trained model and that it can be read without issues.

The model is then used to recognize the text that was extracted by the localization model and processed by the binarization, the output of this stage is two arrays, one is for the recognized text lines and the other is for the locations of those text lines.

It is implemented using the following code:

```
def predict text(model,
                 x,
                 batch size=None,
                   verbose=0,
                   steps=1,
                   callbacks=None,
                   max_queue_size=10,
                   workers=1,
                   use multiprocessing=False,
                   ctc decode=True):

        x = normalization(x)
        out = model.predict(x=x, batch_size=batch_size, verbose=verbose, steps=steps,
                                 callbacks=callbacks, max_queue_size=max_queue_size,
                                 workers=workers, use_multiprocessing=use_multiprocessing)

        if not ctc decode:
            return np.log(out.clip(min=1e-8))

        steps done = 0
        if verbose == 1:
            print("CTC Decode")
            progbar = tf.keras.utils.Progbar(target=steps)

        batch size = int(np.ceil(len(out) / steps))
        input length = len(max(out, key=len))

        predicts, probabilities = [], []
        while steps_done < steps:
            index = steps_done * batch_size
            until = index + batch size

            x test = np.asarray(out[index:until])
            x test len = np.asarray([input length for   in range(len(x test))])

            decode, log = K.ctc_decode(x_test,
                                       x_test_len,
                                       greedy=cfg.greedy,
                                       beam width=cfg.beam width,
                                       top paths=cfg.top paths)

            probabilities.extend([np.exp(x) for x in log])
            decode = [[[int(p) for p in x if p != -1] for x in y] for y in decode]
            predicts.extend(np.swapaxes(decode, 0, 1))

            steps_done += 1
            if verbose == 1:
                progbar.update(steps done)
        return (predicts, probabilities)
```

Figure (3.1): Main system recognition code

Replacement:

The replacement part takes as input the two arrays that were produced from the recognition stage, where translation is applied first if specified to, and then the location for each of the text lines is used to determine which part of the image does it belong to, once that's done, the output from the binarization stage is used for determining the colors for both the background and the foreground of the original text line, which are used for replacing them with the recognized text lines.

Main panel:

This part is where the operations entered by the user are interpreted into actions by the system. The main function takes as input a number of parameters that are used to determine the functionality required.

Main(image, localizer, loc_source, recognizer, rec_source, replacement, translation)

image: Path to the original image.

localizer: Localization model architecture (*east*)

loc_source: The dataset on which the localizer was trained (*icpr*, *icpr2*, *totalText*, *svt*)

recognizer: Recognition model architecture (*yazeed*, *puigcerver*, *bluche*)

rec_source: The dataset on which the localizer was trained (*khatt*, *khattx*, *rhimes*, *IAM*)

replacement: Replace orignal text lines with recognized ones.

translation Translate to selected language (*en*, *fr*, etc)

```
Main(image="/content/drive/My Drive/data/demo/ph3.jpg", localizer = "east",
recognizer = "yazeed", replacement=False, translation = "")
```

Figure (3.1): Main panel for merged system code

Data analysis:

The main process used for analysis of data was by comparing the results of our system against those of other systems, the same test methods were used to validate the performance of the system. And also,

real life examples were considered as well, the process of data analysis and the validation of findings will be considered in detail in the next chapter.

RESULTS AND DATA ANALYSIS

Organization of the data and analysis:

…

Findings in the context of the setting:

In this part, we will discuss the general findings from the study.

MDLSTM networks are not reliable and slow to train:

As was thoroughly tested and reviewed by other studies, multidimensional long short-term memory networks were found to be much slower to train compared to other methods.

We tested the usage of MDLSTM in the task of Arabic text recognition and they were found to provide lesser performance and accuracy when compared to other network models such as bi-dimensional LSTM networks.

This is generally due to the complexity of MDLSTM network models and the difficulty in training them, such complexity usually also leads to slower inference times and unreliability when it comes to usage in real life scenarios. Especially, when they're used for the detection of text in a stream of images instead of a static image, whereby, the network has to operate in a fast enough pace as to accommodate for the high refresh rate of the image stream.

Comparison between KHATT and KHATTX:

As was explained earlier, the KHATT dataset is a very large dataset used for the task of recognizing handwritten Arabic text. Whereas, in the field of Arabic text recognition, finding such a dataset is considered rare due to the unavailability of Arabic datasets for public use (usually restricted to certain universities or research institutes).

This dataset is considered to be a good one with a considerable amount of variability in its samples, but, it was found to contain some outliers and invalid samples. Such samples was present due to the fact that the text was written by various writers with different ages.

Some of the writings were found to be out of line, as in, they were not following a straight horizontal alignment, where they were sometimes following a rather diagonal alignment. Such samples will introduce issues when it comes to training the model, this is mainly because all samples must be resized to the same size before being introduced as training examples for the network, which in this case leads to squished text with abnormal alignment.

The same can be said for some of the other samples where the text lines were not cut correctly, as in, there was a large white padding below the text line, this will also result in squished text lines once they are resized for usage on training the network, which will lead to abnormal behavior by the network.

And finally, some samples were found to contain plain scratches, the contained no text at all, instead, the contained some strange line artifacts that were clearly due to error by the authors of the dataset.

All such issues were removed in the new dataset KHATTX that was created by our team, this led to much higher accuracies and reliability when compared to the older one.

Here are samples from the previous dataset that were discarded in the new one:

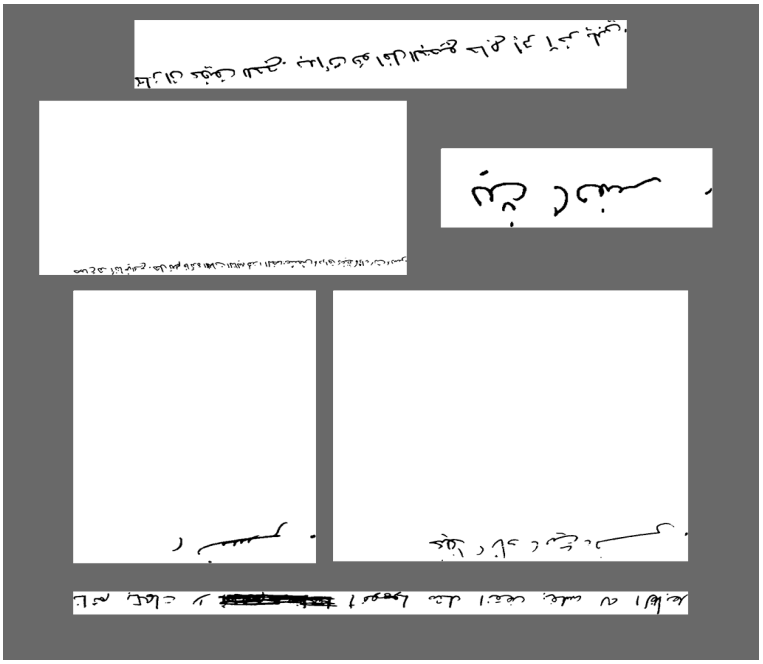


Figure (4.1): Invalid samples from the old KHATT dataset

Recognition models results:

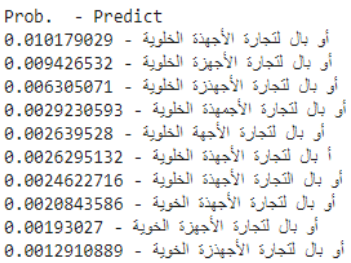
The best recognition model was found to be the one based on Yazeed's architecture (created by one of the members of the team based on the Arthur Flur's architecture), it was able to achieve the following results when tested using images from the KHATTX dataset:

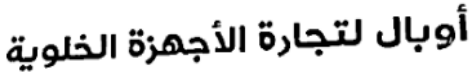
Sentence error rate (SER): 5.626902%

Word error rate (WER): 17.166011%

Character error rate (CER): 69.380204%

Some figures for the recognition model:





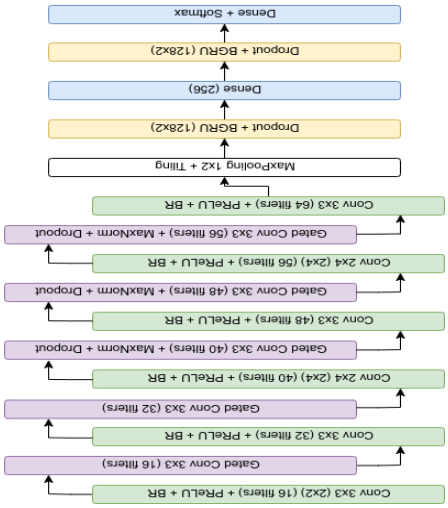


Figure (4.1): Recognition model results sample

Figure (4.1): Recognition model architecture

Localization results:

The main model used for localization was the EAST model, it is an advanced localization model that makes use of seven output layers for localizing text.

Here are graphical representations for some of the outputs of the model which indicate its performance:

Final Result After Applying NMS

Neural Network Activations

Original Image

Figure (4.1): Localization example

Image binarization results:

The method used for binarization was a multistep method, here is a list of the steps followed:

6) Converting the image to gray scale
7) Applying canny edge detection
8) Filling the blobs in the image with predefined colors
9) Sharpening the image
10) Applying a median filter to the image

This figure shows the results of the different stages of binarization:

Figure (4.1): Binarization stages

Replacement method results:

As described in the previous chapter, the steps of replacement are as follows:

1. Extraction of the text box coordinates (provided by the localization step)
2. Estimation of the background color of the text box (done within the binarization step to reduce redundancy.
3. Estimation of the text color of the text box (the foreground color, also estimated during the binarization step).
4. Replacing the box in the original image with a blank box with the estimated color.
5. Translation to the selected language (optional)
6. Placement of the text on the blank box, giving it the pre-estimated color.

This figure shows the results of the different stages of replacement:

Original

Without Translation

Translated to English

Figure (4.1): Replacement results

Merged system results:

The merged system was tested using a variety of images to evaluate bottlenecks in the system performance, here are the latency results of each of the system stages:

Localization latency: 00 milliseconds

Recognition latency: 00 milliseconds

Replacement latency (with translation): 00 milliseconds

Replacement latency (without translation): 00 milliseconds

Full system latency (with translation): 00 milliseconds

Full system latency (without translation): 00 milliseconds

Explanation of analysis:

In this part, an explanation for the provided analysis will be covered, as in, explaining why the provided results were achieved. In addition, an explanation for what could've gone better will be provided as well.

Recognition model:

The recognition model was able to achieve such scores mainly due to the sophistication of the model architecture used, the gated convolutional network is able to memorize context throughout the recognition of the text line, this is indeed necessary in the recognition of Arabic text due to the connectivity between each character and the ones after and before it.

Another point of power for the model was the fact that it was trained on equal size samples which only contains binary colors, the equality of size was achievable by acquiring the maximum possible size and then padding all the samples with blank white spaces as required. Whereby, the removal of outliers came in handy for such a scenario.

Despite that the results achieved using the recognition model were indeed high for handwritten text, the same cannot be said for printed text, this is due to the fact that printed text is generally different in formation when compared to handwritten text, though, this can be easily remedied by training the model on a printed Arabic text dataset.

Localization model:

As for the localization model, its main source of power was found to be related to two factors.

The first factor was the shape and the architecture of the network used, that is, the number of layers and the amount of units present in each layer, in addition, the type of activations and the hyper parameters used when training the network.

The model used (the EAST model), has a very important property, that is, its ability to detect boxes of different sizes in the same image. This is generally due to the usage of concatenation layers which takes the output from the first few layers and concatenates them to those of the last layers, the first layers are responsible for the detection of larger text boxes, whereas, the ones at the end are responsible for the smaller boxes. This is due to the fact that the deeper the sample goes through the model, the smaller will be the details that are considered.

The second main factor is simply the fact that the model was trained on a wide variety of datasets as was described in the previous chapters, this provided it with the modularity property which is important for generalization to a large amount of scenarios (whereas, it must be able to detect text both in book papers and in scenery pictures, which are two very different case scenarios).

Image binarization:

As for image binarization, it was found to work the best when there is a high amount of contrast between the foreground text color and the background color. This is mainly due to the way in which the binarization itself works, as in, it must first apply a blur effect in order to discard high frequency changes in the image and to give the background and the foreground more stable uniform colors. But in the case where an image doesn't have enough contrast between the text and the background colors, the blur effect will lead to them having even closer color values, which will present an issue in separating them.

In the other cases, the binarization works well due to its ability to discriminate between the background and the foreground portions of the image.

Image replacement:

Due to its dependence on both the localization and the binarization modules, the performance of the replacement system is highly tied to the performance of these modules. As in, it only performs well when the localization module has given it correct coordinates for the text box corners, otherwise, the placement of the new text boxes will be invalid. Also, the result of the binarization module is important due to the fact that it is used by the replacement system to estimate the colors of the foreground and the background of the text box (using the separation of those areas provided by the binarization module). That is, the better the separation between the areas, the more accurate the color estimation for each of the areas will be, that's because the replacement module actually calculates the average of the colors of the pixels in those areas (whereas, if the pixel is black in the binarization stage, then it is from the foreground part, and its original color is used for calculating the average for the foreground color).

How sense was made of the findings:

In this part, the outcomes of the research will be summarized and interpreted in relation to their importance to the research questions.

Firstly, the localization model used was found to yield high accuracy when tested with high resolution images, the lower the resolution the faster the interpretation but the lower the accuracy, this leads to the conclusion that this model can indeed be used with static images with standard phone camera quality, which is exactly what's needed by the study.

Secondly, the recognition model was found to be more accurate the better the binarization of text is. Whereas, the binarization of text depends mainly on the image contrast and resolution. For the use case that is focused upon in the research (recognition of text found in street/ market signs), this is usually the case, but it is found to be usually better if the image was taken from a close distance and with a high resolution image quality.

Thirdly, the translation and the box replacement modules were found to be completely dependent upon the performance of the localization and the recognition subsystems, and are found to yield better results with high resolution images that contain high contrast ratios. One good property of the translation module is that it is able to fix typos found in the recognized text, that is, as long as the typo does not cover too many characters inside the same word.

CONCLUSION

Results:

The developed system has the ability to localize text ilnes written in Arabic from a scenery image and cut those text lines, it then introduces those text lines to the binarization algorithm which converts them into black and white images, the images are then fed to the recognition module which carries out image to text conversion, the output texts are used along with the text box locations (provided by the localizer) for replacing them in the original image with the new values using color estimation, the system also allows for the translation of text before replacement.

This system can be applied to an android application which takes images from the camera, feeds it to the system, and the present the user with the results, which is indeed the main purpose of the study (to develop a system that allows the visually impaired and the unfamiliar tourist to view text in an image and perhaps either translate it or use it for voice pronunciation).

Discussion of findings:

The overall system was found to work well when considered as separated systems, though, the merged system has its limitations due its high dependency on the image quality.

The recognition subsystem was found to work well especially when introduced to images that contains text that is close enough in format to that of the KHATT dataset, this is due to the fact that the model was not trained on other datasets which lead to its inability to generalize for more scenarios.

The overall system was found to work the best on high resolution images that has high contrast ratios, this is due to the following reasons:

1. The fact that the localization algorithm can work the best when enough detail is present, that is, the less detailed the image is, the harder it will be for the localization algorithm to extract text lines.
2. The binarization algorithm works the best when the extracted boxes has both high detail and high contrast, whereas, the lower the contrast, the harder it is will be to separate between the color of the text and that of the background. The same can be said for the resolution part, whereby, the lower resolution images usually result in blurry edges between the text and the background, which leads to difficulties in discriminating between them.
3. The recognition model has also been evaluated to have its highest accuracy when used with images of text that has high resolution.

Past and future research:

Whereas the recognition of Arabic text is indeed not new, it can be easily described as immature.

Most of the newer algorithms that were applied to the recognition of other languages has not yet been tested on the Arabic language, this leads to a huge lag between the performance of Arabic OCR systems and those used for other languages.

The same can be said for the localization of Arabic text in scene images, whereas, there were only a few attempts at developing accurate and reliable Arabic text localization systems. Previous to this research, the most highlighted research in the localization of Arabic text was the one that focused upon the detection of Arabic subtitles in television shows (a dataset that contains images from the BBC channel was the one used).

Recommendations and lessons learned:

In this part, we will provide some of the lessons learned from the research including recommendations on future work.

More research on Arabic text recognition is required, that is, the field of Arabic text recognition is indeed very prone to improvements and discoveries, so we feel that more attention should be paid to this field mainly because it has a plethora of useful applications and use cases.

More Arabic datasets should be public and available to researchers, this in particular was one of the main issues we faced throughout the research, most of the Arabic datasets are restricted and very difficult to acquire, which leads to the inability of the students and researchers to carry out experiments and to train more sophisticated systems. We feel that there must be a way to allow all the students and the researchers (at least those from Arab countries) to use those datasets without any difficulties or conditions. Perhaps, a website can be created which represents an association between Arabic universities and research institutes, this website should allow any verified Arabic student or researcher to access all the previous research papers and datasets.

If a dataset cannot be found, one can always be created, that is, one should not quit the research or change its subject if no dataset can be found, there is always the ability to create one from scratch even if one would have to make creating this dataset the main purpose of their research. This leads to a healthier research community and allows for more and more datasets to be present for future researchers, we believe that creating a dataset is as important as doing the researching and experimenting.

…

.

.

.

.

.

The future of the research:

In this part, we will provide some of the possible improvements that can be applied to this system in the future, in addition, some use cases will be recommended for the current system.

Improvements to the recognition model:

We recommend the following to be implemented in the future:

- **Better models.** That is, improvements to the current model can be applied, we recommend the usage of newer technologies which were tested on other languages and generally yield higher performance.
- **More training.** The current model was only trained on the KHATTX dataset, which is a portion of the original KHATT dataset. More training can be done using other datasets which will help the model to generalize better.
- **New datasets.** The future researchers are recommended to create new datasets for the Arabic text recognition task, whereas, we recommend creating a new dataset for printed Arabic text, this can be easily achieved using third party libraries (creating random sentences from the Wikipedia and converting them to image format using openCV, segmenting them, and organizing them in a form that is usable for training by the system).

Improvements to the localization model:

We recommend the following to be implemented in the future:

- **Better models.** As the case was for recognition, improvements to the current model can be applied, we recommend the usage of newer state-of-the-art technologies which were tested on other languages and generally yield higher performance.
- **More training.** The current model was indeed trained on a variety of datasets, though, we believe that it can yield even better results if it is trained on more datasets (the COCO dataset for instance).
- **New datasets.** The future researchers are recommended to create new datasets for the detection Arabic text in scenery images, whereas, this can be easily achieved using images from the google street view service, where the screenshots can be taken for different scene images that contain signs and other forms of written text, this dataset will provide a performance boost for the current system in that it will allow the system to generalize more and detect Arabic text in particular more easily. The ground truth for the dataset can be generated by using any third party image editing software that presents the user with the ability to read the current coordinates of the mouse pointer, whereas, this info can be used to figure out the locations of the corners of the text boxes.

Improvements to the preprocessing techniques:

We recommend the following to be implemented in the future:

- **Super resolution image processing.** Whereby, as described previously, the most critical point in the performance of the system was found to be the resolution of the image. Using super resolution, the quality of images can be greatly enhanced before feeding them to the system, which will lead to a large boost in the accuracy of the system.
- **Contrast enhancement.** As described earlier, the better contrast ratio the image possess, the higher will be the performance of the binarization algorithm, which then means, the higher the accuracy of the recognition of the text inside the text box.

Other improvements and additions:

We recommend the following to be implemented in the future:

- **Faster models and more optimized codes.** Which will be very critical in the deployment of this system to mobile devices which generally has much less power to offer compared to personal computers.
- **Language discovery.** A language discovery system can be implemented, where the language of the written text is first identified, and then it is fed to the correct recognition module, this allows for the use of this system with multiple languages.
- **Android / iOS implementation.** A mobile implementation for this system can be developed in the future, this will ensure the usability of the system by a wide variety of users due to the ease of use that comes with mobile software. The input to the system can be fetched from the device camera, and the rest of processing can be either done within the device itself or handled by a server computer that is connected to the internet (web sockets will be a good method of achieving this).