COMPUTER VISION BASED APPLICATION DEVELOPMENT FOR SCREEN TEXT TRANSLATION

Raka Wirabuana Ninagan
School of Electrical Engineering and Informatics
Bandung Institute of Technology
Bandung, Indonesia
rkvilena11@gmail.com

Rinaldi Munir
School of Electrical Engineering and Informatics
Bandung Institute of Technology
Bandung, Indonesia
rinaldi@staff.stei.itb.ac.id

***Abstract*— A pre-rendered foreign language text on a desktop is hard to translate because the text can't be copied directly to the translator. This can be solved by using scene text detection and recognition to acquire the text into machine-readable text and integrate it with a translator. A scene text detection and recognition framework is used to directly integrate the detection and recognition process and use its pre-trained model. The application uses the Google Cloud Translation service as its translator to support multiple text translations. It has a GUI module to capture a particular area of the screen and put the translation result to the location of the original text using TKinter. This application managed to translate a text in many conditions, with some downsides like unique text and tilted text. Its execution time range is 0.45–3.0 sec. While some inconsistencies result in a bad situation (noised visuals, high usage of memory), the application manages to put the text directly next to each original text on the screen.**

Keywords—scene text detection; scene text recognition; translation; screen capture; text overlays;

I. INTRODUCTION

One of the most common forms of information is text. The variation of culture in this world makes the text have so many character scripts and languages. An example is a video game that was developed by a Japanese studio, which often only developed the game in Japanese. While learning a new language is an option, not everyone wants to or can learn every language they need to understand. The translation machine plays a big role in this, converting the original text into one that the user understands.

In some cases, not every text can be translated by the translator, as the text can't be copied by the machine. A text like image text or video game text can't be translated by copying the text to the translator, as seen in Fig. 1. There's an option to write back every character of the text to the translation machine, but this will take a lot of time for a larger text count, and there's a possibility that the user didn't know how to write it. Based on this, there's a need for a tool that can acquire these texts into machine-readable text and make it possible to directly integrate their results into the translation machine.

Some developers have already tried to build an application to solve this problem, whose application name is DeskTranslate [1]. It asks the user to select the region to be translated, and a translation result is shown in a special window. Another option is Google Lens, which is able to put the translation result directly on the smartphone screen by using its camera. Both solve some issues with each of its downsides, as the DeskTranslate result will be stacked into that special window, making it hard to read the translation result for a larger text count and an impractical use of Google Lens for a repeating task like playing heavy-text video games as the user should keep capturing the screen using their phone every time. Based on that, there's a need for an application that manages to convert every text selected by the user into machine-readable text, translate the text, and output the translation result efficiently.

Fig. 1. A video game that contain a text with Cyrilic script

II. LITERATURE REVIEW

A. Computer Vision

Computer vision is one of the artificial intelligence disciplines that focuses on utilizing a computational system to extract useful information from a visual component [2]. Computer vision replicates a human way of processing vision information, for instance, by using sensors to receive an input and then processing it using a trained pattern recognizer. The pattern recognizer was already trained using certain training data that related to the information target. Recently, computer vision technology has already been applied to many fields, such as medical diagnosis, automatic manufacturing and surveillance, autonomous vehicles, and many more [3]. Another example of a computer vision application is the use of a camera to understand human writing, object detection, mechanical inspection, etc.

B. Deep Learning

Deep learning is a computer model that consists of many processing layers of an artificial neural network to understand data at a certain abstraction level. Deep learning can be used to classify any kind of input, like images or texts. This classification ability is gained from model training with certain datasets.

An artificial neural network (ANN) replicates a biological neural system by processing information in a parallel way that is distributed to many neurons [4]. It has three types of layers: the input layer, the hidden layer, and the output layer. These neurons are functions that will define specific features or characteristics of an input. For example, there's two layers consisting of neurons that represent an edge and texture processing in an object detection context. Every neuron in one layer is connected to another neuron in the next layer with a certain coefficient called weight. In every neuron, the sum of weights is done and then processed by an activation function and bias as a decision to activate or deactivate that particular neuron. These activations hold a great role in influencing the results in the next layers. All of those calculations are done in the hidden layer. The result will be delivered to an output layer, producing a result of the classification process.

C. *Convolutional Neural Network*

A convolutional neural network (CNN) is a type of artificial neural network that focuses on processing a grid-like input. The CNN architecture is designed to extract specific features from an input. ANN can actually process a grid-like input with very low performance due to the complexity of the input, which has a very large weight, making it very hard to process. CNN has three main steps: feature extraction, followed by classification, and a probabilistic distribution to complete the steps. The architecture visualization shown in Fig. 2.

1. Convolutional Layer

This layer is responsible for extracting all input features. The extraction is done using a convolution operation. This operation is focused on region calculation based on a certain hyperparameter called the kernel. The kernel will cover parts of the input. The operation will change the middle value of a covered part with the calculation result after the result is processed by an activation function. The activation function used in this part is ReLU, which will convert every negative number to 0 and let another value keep the value.

2. Pooling Layer

This layer will reduce the input dimensions in order to make the process lighter [5]. It uses max pooling or average pooling to keep the essential value while reducing the dimensions. Max pooling will take the highest value as the essential value, while average pooling will take the average value as the essential value.

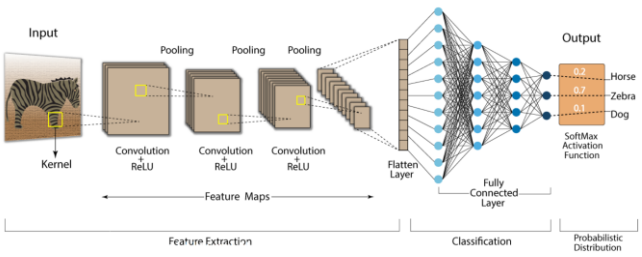

Fig. 2. General CNN architecture

The classification steps will classify the extracted features into a certain class through the fully connected layer. It will process a flattened set of features to get the classification result at the output layer. In the output layer, a probabilistic distribution is done using a softmax activation function to normalize the classification result.

D. *Scene Text Detection & Recognition*

Scene text detection and recognition is a task that detects the existence of texts from an image input and recognizes every detected text as machine-readable text. Scene text detection generally predicts pixels if a text exists and simultaneously produces a bounding box. Scene text recognition predicts the text of the image parts that are predicted to have text. This task can be solved using traditional or deep learning approaches, but much research in recent years has been done using deep learning [6]. Many architectures have been proposed to do these tasks, for instance, the CRAFT [7] and AF-RPN [8] as detectors and the CRNN [9] and CNN-DBLSTM [10] as recognizers.

With the many architectures proposed by researchers, some developers decided to build a framework to facilitate other users or developers using those architectures (in this case, as trained models) easier. One example of these frameworks is EasyOCR [11], which provides scene text detection and recognition model integration, a tool that enables the user to easily train and/or use a model. These frameworks usually come with their own pre-trained models in various languages.

E. *Google Translate*

Google Translate is an online service developed by Google to translate a text, document, image, or website from one language to another. It is available in many devices and shapes, such as websites, smartphones, and APIs for developers.

Google Translate has been using a deep learning version of translation named neural machine translation (NMT) since 2016. The method used in that update is example-based machine translation, which makes the model "learn from a million examples." The accuracy of it increased because it translated the text for every word simultaneously, giving the translation a better understanding of the context of the text [12].

III. Proposed Solution

We propose an application that integrates scene text detection and recognition with the translator so that the text can be translated without further work from the user. The translation result will be placed on the screen exactly at each location of the original text, making the translation result "replace" the original text.

The application will have a general flow that starts with a translation configuration, as shown in Fig. 3. This step asks users to choose the text language and the translation language. For example, an English text that wants to be translated to Indonesian will be both English and Indonesian. A text detection process is followed to detect every text in the input. It will produce a bounding box that shows where the text exists in the input. There's a need to crop every part of the text as a standalone image so the text recognizer can start predicting its machine-readable text version. The results can now be translated for the translator. On the screen, the application will draw a bounding box from the detection

result as a container for the translation result. All translated text will be placed in each corresponding box.

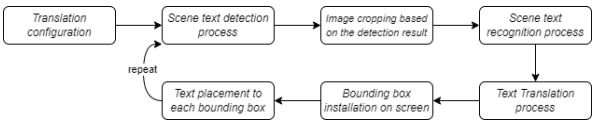

Fig. 3. General flow of the solution

This general flow is a rough idea of how to process this problem. From the application perspective, there's some difference as constraints exist like execution time, memory usage, and library-dependent tasks. Based on the general flow, we decided to create the architecture of the application as shown in Fig. 4. There are three core modules: the Graphical User Interface module, the Text Detection and Recognition module, and the Text Translation module. These three cores can't directly communicate due to the existence of threads to run the text detection, recognition, and translation modules. A shared memory and a queue are placed between these three modules to communicate properly without any blocking.

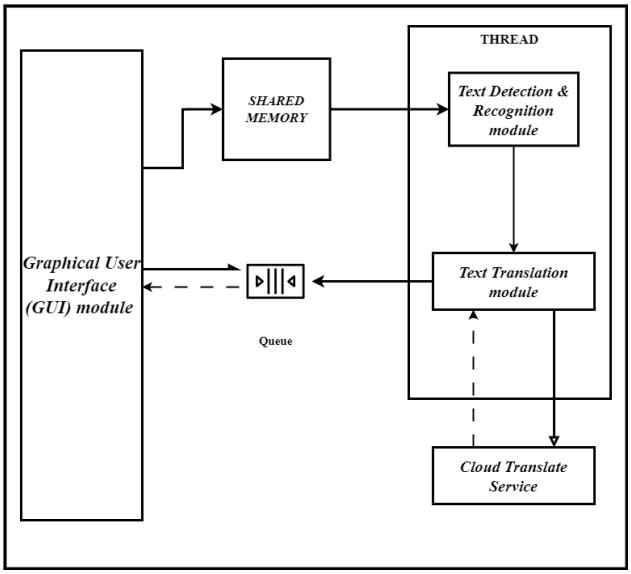

Fig. 4. Proposed architecture the TexTranslator application

A. *Text Detection & Recognition Module*

The application idea has the ability to change the language, resulted in the need of model flexibility. With so many language requirements, the existence of scene text detection & recognition frameworks become very important. For example, EasyOCR claim that a 80 language is ready to be processed by only change the configuration to the preferred language. This makes the framework to hold a responsibility in recognizing a text into a machine readable text. Another strong point of the frameworks utilitization is its integration processing. Generally, the bounding box resulted from the scene text detection model shown by a marked pixel with a text existence prediction, makes the application need to process it into a bounding box format (for instance, 4-points coordinates). A framework handle its processing, removing the needs to produce a bounding box format. The recognition process also got its benefit by removing the need for manual image cropping as recognizer input. Those two points makes the utilization of frameworks very important.

The proposed solution uses 2 frameworks, that is EasyOCR and WinOCR. These 2 frameworks use a different model for every step. EasyOCR uses CRAFT and CRNN as their default detection & recognition model. On the other hand, WinOCR uses Windows OCR Engine with AF-RPN and CNN-DBLSTM as their models. EasyOCR implementation has more modifications freedom compared to WinOCR as WinOCR only integrate Windows OCR Engine with an application that developed in Python. EasyOCR provided a threshold argument that controls the limit of box merging like *width_ths* and *height_ths*. It also offers a detection & recognition in paragraph levels, makes the text merged if *x_ths* and *y_ths* limitation permit those merging. As this module will took quite a while in processing an input, it will be running in a thread to make the main thread free from blocking.

The module will receive an image as an input from the shared memory. The image format is in numpy array format. The module will put the image as an input for the used framework. A produced result from the framework processing is a machine readable text and its bounding box. The module will convert all of the bounding box format from 4-points coordinates to a format with the specification of height, width, x and y of the left top corner (h, w, x, y). Lastly, the recognized text will be delivered to the Text Translation Module. These process visualizations shown in Fig. 5.

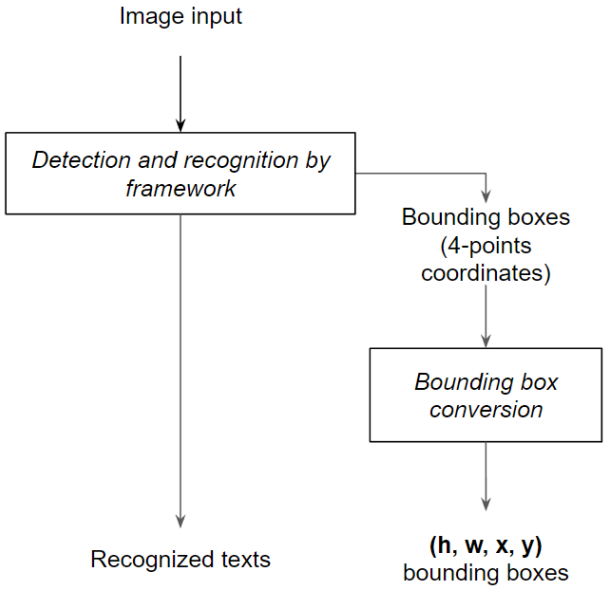

Fig. 5. Text detection & recognition module process visualization

B. *Text Translation Module*

This module will translate all of recognized text from the detection and recognition module with online translator, specifically Google Translate. There are an options to use a local translator and even both translator. The local translator offer an offline use of the application, as the translator is running in the device. The problem is that the device will need to load and run the model together with the framework models. This will take so many memory resulting in

application lag and/or crash for a mid-end device. Another downside is that the quality of the translation result is not as good as online service because the online service has a development and maintenance guaranteed.

In Python, a numerous of library developed to connect a python program with Google Translate API. A program can only request a translation for one strings, makes the translation process longer when the texts are two or more strings. Multiple texts translation used to handle such problems. It can be done using Google Cloud Translation with v2 library. The limit of the strings now up to 128, which make translation process faster.

The process that happen in this module shown in Fig. 6. The recognized texts will be translated to Google Cloud Translation. For a texts with more than 128 strings, there will be some iterations to translate all of it. The translated text will be merged back with the newly-formatted bounding boxes. A queue will become the communication path for the texts and bounding boxes to the Graphical User Interface module.

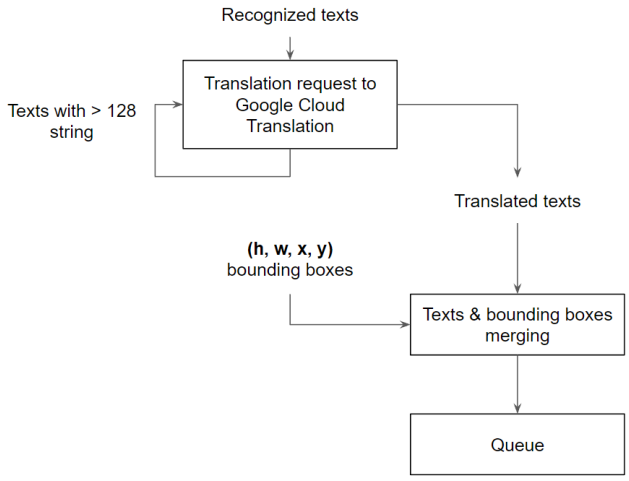

Fig. 6. Text Translation module process visualization

C. *Graphical User Interface module*

Graphical User Interface (GUI) module will be responsible for capturing a screen as an input. The application uses tkinter as its GUI. The GUI can draw a rectangle on the screen to determine the area of the screen that want to be captured and processed. This rectangle is called *screenbox*. Screenbox has a transparent interface so the user still can interact with the screen while an overlay drawn there. Tkinter can't have a root window more than one, so the screenbox window type is a child of the root. The root will only consist of language configuration, paragraph mode and fullscreen mode button, and delete button. The delete button exist to give the user an option to delete some of the translated result.

The screen capture process can be executed from a direct command by the user. There's two options to do this, by clicking a green-colored button on the screenbox or by clicking '*c*' in the keyboard. It uses mss library in Python to capture the screen based on the screenbox position.

Its another responsibility is to place the translated text into the screenbox. This process has 4 main subprocess, that is text size adjustment, text background blurring, text coloring, and text placement.

1. Text Size Adjustment: The translated text has possibilities to produce a larger or smaller width compared to the original text. Tkinter Canvas provided a text size calculation when the canvas already placed in the screen. The module will calculate a text size prediction based on the ratio of the screenbox height to the screen height. The predicted result will be simulated by placing the translated text to the screen with that size. If the text canvas becomes too large compared to the original bounding box, the module will simulate the text again with a smaller size. The process will be repeated until the simulated size has a smaller size compared to the bounding box.

2. Text Background Blurring: The idea is to place the translated text into the original text location. Without any preparation, the translated text won't be visible too much because of the conflict with the original text. An image contain the original text is blurred to make the original text less visible while still keep the general visual color of the background. The blurring function used is median blur as other blur have a tendency to make the original text blurry but with a larger spread.

3. Text Coloring: A blurred background has a certain degree of brightness. To make sure that the translated text visible, the text color should has a brightness level that didn't look similar to the background. The blurred image is converted to the HSV to take the V (*brightness*) value. If the image has V above 128, the text will get a black color, and vice versa (text with white color for a darker brightness).

4. Text Placement: Each translated text will have its own Tkinter Canvas. Each canvas will have a different background color and bounding box size.

IV. Testing and Evaluation

Testing is conducted with the aim to check the integration quality, the best models as a whole application, the output consistency, the accuracy and executing time of the application in various situations, and compare the result quality with a similar application.

A. *Test Case*

There will be two cases types, the first one is a text in an image/still visual and the second one will be a text in a video game. An image/still consume small memory while video game consume large amounts of memory, makes the application can be tested in an optimal and not optimal situation.

B. *Testing Scenario*

An image will be used to check the integration quality. This screnario aims to check if the existence/nonexistence of text can be checked correctly. The second scenario uses an image to check the best models, with the aims to know which models are better between EasyOCR models and WinOCR models. Consistency of the output will be checked by running the application for a video game with english and japanese language model and three iterations for each language. This setup aims to check if a high usage of memory and unstable condition (the video game that used

has a noise as its visual effect) will produce a different recognition result. Another scenario is to check the application capabilities in many situations. This aims to check if the application is flexible enough to be used in various conditions. The last scenario uses a video game and a document as a cases to check the application quality in terms of accuracy, execution time, and text placing efficiency compared to its predecessor, DeskTranslate. Every scenario besides the second will use EasyOCR as the framework.

C. Testing Results

The application tested to 4 images to test the first scenario. It managed to detect the existence/nonexistence of the text properly, as when the text exist, there's a translation result and vice versa. The application hit 100% accuracy for two cases while another one has 92,8% accuracy (the image is blurry).

The second scenario performed with 2 images. The result shows that EasyOCR produce a better accuracy of the bounding box compared to WinOCR. EasyOCR managed to show a better flexibility as a very large of characters can be detected by EasyOCR while WinOCR couldn't. That being said, WinOCR perform extremely faster compared to EasyOCR due to the engine exist the machine compared to EasyOCR that need to load the model every initialization as shown in Table I.

TABLE I. EASYOCR & WINOCR MEMORY USAGE

Framework	**Profiling Result**	
	Memory Usage	***Command***
EasyOCR	1928,1 MiB	CUDA load model to the GPU
	1616,5 MiB	Detection & Recognition
WinOCR	0,0 MiB	WinOCR initialization
	5,4 MiB	Detection & Recognition

The application couldn't produced a consistent result as each iteration for both languages has a different output. The recognized text in english are generally more consistent compared to the japanese. The output sometimes produces a nonexistent character or number. The translation result quality is linear with the detection & recognition output.

The fourth scenario is performed in 9 situations. The application managed to translate every text in every scenario, with some exception for a tilted text and unique font text that produced a bad result. The text can't be placed with the same slope with the original and a unique font text case made the application unable to recognize each letter properly. The execution time of the application is in a range of 0,45 – 3,0 second. The more text existed in a captured image, the longer the application to process it. Each cases execution time shown in Table II.

TABLE II. APPLICATION EXECUTION TIME FOR THE 4TH SCENARIO

Situations	**Table Column Head**	
	Normal Mode	***Paragraph Mode***
Image text	0,469	0,667
Website text	0,833	0,934
Document text	1,329	1,248
Zoomed document text	1,526	1,453
Tilted text	0,494	0,516
Video game text	1,204	1,275
Non-latin text	1,515	1,042
Right to left text	2,767	2,974
Unique font text	0,799	1,071

The last scenario result shows that the implemented application produces a more efficient text placement compared to DeskTranslate. DeskTranslate process the text slightly faster than the implemented application because of its automatic capture, but the speed is actually similar in terms of each execution. The complete test result can be seen in the following page:

https://docs.google.com/document/d/1b0Oz2Fo8HsmctwOM2r4dO7jnMJv8FQKJ/edit?usp=sharing&ouid=113435573993133639499&rtpof=true&sd=true

V. CONCLUSION AND FUTURE WORKS

A. Conclusion

1. The integration is a success with using a framework that focuses on integrating the detection & recognition part and completed the integration with the translator. The used framework is EasyOCR with CRAFT & CRNN as its models and WinOCR with AF-RPN and CNN-DBLSTM as its models. Google Cloud Translation is the translator used in this integration.

2. Text processing from a screen can be translated by using a GUI to configure the language properties, capturing the text on the screen, delivering the image to the text detection & recognition followed by the translation, and placing the translated text from the translation module. The text placement consist of text size adjustments, text background blurring, text coloring, and text placement on the screenbox.

3. Application managed to translate a text from various cases such as images, websites, documents, video games, non-latin texts, and right to left texts, for the exception on tilted text and unique font text. The application can't produced a consistent result in a bad condition such as high usage memory and noised images. It can produces a better text placement with normal mode in a low density condition and paragraph mode in a high density condition. Execution time of the application to various situation is in range of 0,45 – 3,0 seconds.

4. Application can produces more efficient text placement compared to another similar application by placing it in each of the original text.

B. Future Works

For future research, the following ideas and suggestion that can be implemented:

1. Add another module to fix a recognition result to produce a better recognized and translated text, for instance a grammar checker.

2. Change the application flow into an automatic process. Need to use a better framework/models or optimized EasyOCR.

3. Increase the application speed to a real-time state. WinOCR that run very fast can be optimized in terms of the bounding box result.

4. Change the text placement method by using a 4-points format instead of (h, w, x, y) for a better flexibility.

ACKNOWLEDGMENT

The researcher expresses gratitude to God for the blessings received during the completion of this research. The researcher would like to thank Mr. Dr. Ir. Rinaldi Munir, M.T., for providing guidance throughout the research process. The researcher receives much help from family members and many individuals and wants to appreciate all the help and assistance. The researcher hopes that this research will contribute to the advancement of screen text translation usage.

REFERENCES

[1] Amelia, H. Lun, C. J. Hao, and GerardTWK, "DeskTranslate," DeskTranslate, 2021. https://desktranslate.github.io/DeskTranslate/

[2] F. Alsakka, I. El-Chami, H. Yu, and M. Al-Hussein, "Computer vision-based process time data acquisition for offsite construction," Automation in Construction, vol. 149, p. 104803, May 2023, doi: 10.1016/j.autcon.2023.104803.

[3] K. K. Patel, A. Kar, S. N. Jha, and M. A. Khan, "Machine vision system: a tool for quality inspection of food and agricultural products," Journal of Food Science and Technology/Journal of Food Science and Technology, vol. 49, no. 2, pp. 123–141, Apr. 2011, doi: 10.1007/s13197-011-0321-4.

[4] T. M. Mitchell, Machine learning. McGraw-Hill Science/Engineering/Math, 1997.

[5] K. O'Shea and R. Nash, "An introduction to convolutional neural networks," arXiv (Cornell University), Jan. 2015, doi: 10.48550/arxiv.1511.08458.

[6] F. Naiemi, V. Ghods, and H. Khalesi, "Scene text detection and recognition: a survey," Multimedia Tools and Applications, vol. 81, no. 14, pp. 20255–20290, Mar. 2022, doi: 10.1007/s11042-022-12693-7.

[7] Y. Baek, B. Lee, D. Han, S. Yun, and H. Lee, "Character region awareness for text detection," arXiv.org, Apr. 03, 2019. https://arxiv.org/abs/1904.01941

[8] Z. Zhong, L. Sun, and Q. Huo, "An Anchor-Free Region Proposal Network for Faster R-CNN based Text Detection Approaches," arXiv (Cornell University), Jan. 2018, doi: 10.48550/arxiv.1804.09003.

[9] B. Shi, X. Bai, and C. Yao, "An End-to-End trainable neural network for image-based sequence recognition and its application to scene text recognition," arXiv (Cornell University), Jan. 2015, doi: 10.48550/arxiv.1507.05717.

[10] H. Ding et al., "A Compact CNN-DBLSTM Based Character Model for Offline Handwriting Recognition with Tucker Decomposition," 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), Nov. 2017, doi: 10.1109/icdar.2017.89.

[11] JaidedAI, "Jaided AI: EasyOCR." https://www.jaided.ai/easyocr/

[12] B. Turovsky, "Found in translation: More accurate, fluent sentences in Google Translate," Google, Nov. 16, 2016. [Online]. Available: https://blog.google/products/translate/found-translation-more-accurate-fluent-sentences-google-translate/