CSCI 1430 Final Project Report: Google Tran-SLAY-te

Team name: Gavin Dhanda, Karim Mouline, Grace Marshburn.
TA name: Anh Duong. Brown University

Abstract

Our project functions as a visual translator, projecting translated words and phrases that have been detected in the image by a convolutional neural network. This project addresses the issue of language barriers, providing real-time translation for signs, menus, and any other type of written text. We acquired a sufficiently sized data set of letters with which to train the convolutional neural network, used ChatGPT and Google Translate APIs to create coherent text and translate the text respectively, and projected the translated text back onto the original image in a formatted display.

1. Introduction

Our project focuses on implementing a version of Google Translate's live translate feature, where the input is an image with text on it, and the output is the same image with text translated into the target language overlayed. This is challenging for multiple reasons; firstly, one must identify any letters in the image. Then, one must be able to accurately predict what letters each set of pixels corresponds to. Finally, after translation, one must be able to use the information from the input image to correctly overlay the new translated text onto the original image. For letter detection, we used CV2's contour detection method, which finds sharp edges in images and draws bounding boxes around them. This made it simple to extract letter-by-letter information, but also made our program more prone to noise. For letter prediction, we decided to take the approach of using a CNN trained on the EMNIST dataset, which is a set of handwritten character digits derived from the NIST Special Database 19 and converted to a 28x28 pixel image format. This enabled us to have accurate letter detection. After passing the sentence to the ChatGPT and Google Translate API to refine + translate the neural network's outputted string, we finally project an image of the final translated sentence back onto the final, which is generated using the bounding box information given from the original input image.

2. Related Work

We utilized the neural network setup from Homework 5: Convolutional Neural Networks in order to get us started, as we found ourselves to be comfortable with the Tensorflow library after completing that assignment. The CV2 library's findContour method was suggested by the internet as a simple way to perform letter detection in simple images, and our Mentor TA Anh Duong, who did a similar project the previous year, suggested the dataset and that we focus on letter detection as opposed to word detection in order to make the deep learning process more effective - it is much easier to train a network on 26 letters (or 52 if we consider a difference between upper and lowercase) than it is to train it on the incredibly large English Language dictionary - nevermind if we want to have support for multiple languages.

3. Method

The architecture of our Sequential neural network is as below:

```
1  model.add(tf.keras.layers.Conv2D(
       filters=32, kernel_size=(5,5),
       padding='same', activation='relu
       ', input_shape=(28, 28, 1)))
2  model.add(tf.keras.layers.MaxPool2D
       (strides=2))
3  model.add(tf.keras.layers.Conv2D(
       filters=48, kernel_size=(5,5),
       padding='valid', activation='
       relu'))
4  model.add(tf.keras.layers.MaxPool2D
       (strides=2))
5  model.add(tf.keras.layers.Flatten()
       )
6  model.add(tf.keras.layers.Dense
       (256, activation='relu'))
7  model.add(tf.keras.layers.Dense(84,
        activation='relu'))
8  model.add(tf.keras.layers.Dense(26,
        activation='softmax'))
```

This was provided alongside the EMNIST dataset on the Kaggle website. The 26 classes in the final dense layer signifies the number of labels, since the dataset classifies both upper and lowercase lettes with the same label (i.e. A and a are both labelled with a 0). This classification did lead to some issues where an uppercase I in a sans-serifed font would be recognized as a lowercase L, but fortunately our calls to the GPT API would smooth any misclassifications out.

As mentioned above, Chat-GPT's API was used to convert the sequence of letters into coherent text. It was most helpful for adding proper capitalization and controlling for any typos and intricacies of the model (such as the capital I and lowercase L discrepancies). Here is the provided prompt:

"Your task is to assist in formatting text extracted from images for further processing. Convert sequences of characters into coherent English text. Ensure to maintain punctuation, correct typos, and do not add extraneous comments or questions. Process all input accordingly."

The text is then fed into the Google Cloud-Translate API with the desired output language (i.e. "en", "es", "ru").

Lastly, the text is projected back onto the region of the image that it originated from. This involves first using the original bounding boxes to find the smallest rectangle that covers all input letters. With the ouput region defined, we iterate through multiple font sizes (starting from approximately 100px down to 10px), and decrease the font size until the text can all fit properly within the box. This accounts for any changes in length that arise during translation, and also simultaneously makes sure to wrap text onto new lines to accommodate the height of the output region.

4. Results

The final version of our convolutional neural network consists of two conv2D layers each followed by a maxpool layer, a flatten layer, and three dense layers. With this structure and 15 epochs, we were able to properly train the neural network to avoid both under and overfitting. With this structure, we are able to accurately detect letters, calculate the location of each letter in the image, and determine the language of the inputted text.

However, letter detection fails more significantly when there are multiple lines of text in the inputted image. As a result, not all letters are detected and the returned translation can be complete. This stands for multiple lines of text in one area of the image as well as singular lines of text that may be spread across the image.

Our program runs statically, but converting it to real-time

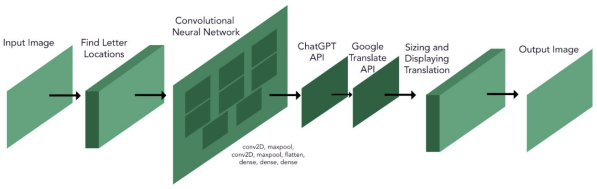

Figure 1. Our pipeline.

is not a difficult goal given our implementation. Currently, a file path and output language are specified within the program, so these are the two things, along with a small UI, that must be changed in order to make the program real-time. The program's current runtime would also most likely be sufficient for real-time translation.

4.1. Technical Discussion

Our method did require a few tradeoffs. We went for the simplest route of letter detection via contour detection, which unfortunately comes at the cost of not really being able to do a good job of letter detection in images with a lot of noise. We did try to couteract this slightly, however, by adding in a confidence threshold from the neural network's prediction, and if a contour gets predicted below that threshold, its "letter" gets tossed out from the final result. This allows different, noisier images (like the one in Figure 2 with the Classroom text) to correctly be recognized.

For the network itself, the architecture choice was a mix of trial and error. The dataset actually was very conducive to a network - epoch 1 would already produce 83 percent accuracy - but there were a few bumps along the way. Firstly, the data in the EMNIST dataset is inverted (white on black background), horizontally flipped, and rotated 90 degrees. Rather than preprocess our input images to match that, we preprocessed the image from the dataset in order to remedy that.

Additionally, we decided to simply project back a non-affine translated rectangle back onto the original image instead of taking the translation of the original image into account for simplicity. We may have been able to use the bounding boxes and detect the offset of their corners in order to add a translation to the projected image box, but we instead chose to focus our efforts on getting the spacing of the text within the box to better match the area.

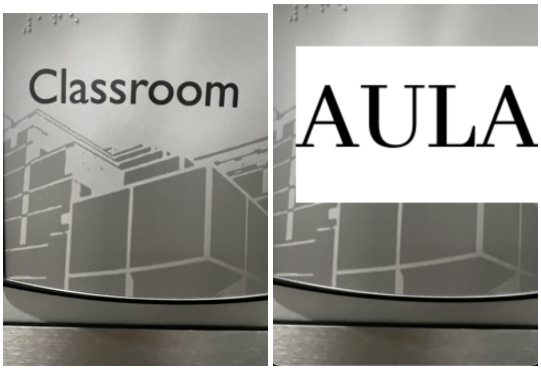

Figure 2. Image with more noise. *Left:* Before translation. *Right:* After successful translation.

Figure 3. Our best result. *Left:* Before translation. *Right:* After successful translation.

5. Conclusion

While stopping short of Google's Live Video translation, our photo-based implementation succeeded in our goals to translate images to and from multiple languages. We were able to segment letters within images with a high success rate, detect which letter each segment represented, and reconstruct the original sentence with relative success using GPT. With this text, we relied on the accuracy of Google Cloud Translate's API to generate the output text, and integrated spatial information from the image segments to adaptively project the translated text into the original region of text.

Further development would hope to improve our ability to detect words and avoid scrambling letters on multi-line inputs, and better adapt to inputs with complicated scenery / background content. We also might hope to add support for languages that reach beyond the extended ASCII alphabet used in English, Spanish, French, etc. Lastly, we might use adaptive detection techniques to add support for separate chunks of text (for instance, if there is text in both the foreground and background).

Why it Matters

The primary importance of this project lies in its potential to assist non-native speakers and foreigners in navigating environments where they do not understand the local language. This system can be particularly useful in various scenarios:

1. Travel and Tourism: Tourists can understand signs, menus, and other written information in foreign countries without needing to rely on human translators or dictionaries.

2. Education: Students and educators can use this tool to facilitate language learning and comprehension of foreign-language materials.

3. Accessibility: Non-native speakers and immigrants can better access and understand information, enhancing their ability to participate in daily activities and access services.

Impact Going Forward

1. Improved Accessibility and User Experience: By making information accessible in multiple languages, this system can bridge communication gaps and promote inclusivity in global contexts. Users can interact with their environment more seamlessly, reducing the cognitive load and frustration associated with language barriers.

2. Potential for Integration: The technology can be integrated into various applications, such as mobile apps, augmented reality (AR) devices, and smart glasses, broadening its usability and reach.

3. Future Developments: The project lays the groundwork for further advancements in real-time translation. As letter recognition, translation algorithms, and projection techniques improve, the accuracy and naturalness of the translations will continue to enhance.

Overall, this project represents a meaningful step towards a more interconnected and accessible world, where language barriers are less of an obstacle for communication and understanding.

Figure 4. Our program can handle any set of latin-alphabet languages!

References

EMNIST Dataset on Kaggle: Link Google Cloud Translate API: Link Image Dataset: Link

Appendix

Team contributions

Please describe in one paragraph per team member what each of you contributed to the project.

Gavin Dhanda Worked on using spatial information about letters to add spaces between predicted letters. Then, implemented Chat-GPT's API, with a prompt designed to convert the letters into coherent text. Then implemented Google's Cloud-Translate API to translate to a desired output language. Also assisted with mapping text back onto the image.

Grace Marshburn Focused on using spatial data from the original text to project the output text back onto the original image. Generating the output region and adaptively formatting the text, such that it accounts for different line lengths after translation, and choosing an appropriate font size to maximize readability of output.

Karim Mouline Focused on image processing, which included implementing the detecting of each letter's contours and creation of bounding boxes, and then implementing and training a neural network on the EMNIST dataset in order to predict each letter recognized by the detection method, outputting a string of letters.

All Members Debugging in all three stages of the project; creating the project poster, write-up, and progress checks; initial research and pipeline design; etc.

Thank you for a great semester!!!