Fall 2021

Senior Project Report

Computer Science Department

California State University, **Dominguez Hills**

ImgTrans: Image Text Detection and Translation

Prepared by:
Shaun Froseth

In
Partial Fulfillment of the requirements
For
Senior Project – CSC 492

Department of Computer Science
California State University, Dominguez Hills
Fall 2021

Committee Members/Approval

Faculty Advisor	*Signature*	*Date*
Committee Member	*Signature*	*Date*
Committee Member	*Signature*	*Date*
Department Chair	*Signature*	*Date*

I. Acknowledgements

I would like to thank all my professors and the staff at California State University Dominguez Hills and El Camino Community College. They were crucial in building my knowledge and skill sets related to Computer Science and overall shaped me into a better and more rounded person. I would also like to thank my family and most importantly my girlfriend, Ashley. She has been a huge driving force and motivation to pursue my degree and without the support, I am not sure what path I would be on today.

II. Abstract

The purpose of this web application is to use Optical Character Recognition to analyze images and documents and serve the retrieved data to the user in the form of text. More specifically, the user will be able to upload an image or text document from their device, select its original language and target language, and it will return the text detected in the file translated to the targeted language.

This will be achieved using Google's Cloud Vision API to process the files and retrieve the text, Translation API to handle the translation, Vue.js to serve the frontend, and Express to handle the backend API calls. While the API is handling all the heavy lifting of analyzing the files, the primary goal is to provide a highly responsive web app with a seamless user interface and user experience across multiple devices.

Table of Contents

III. List of Figures

IV. Introduction

I. What is Optical Character Recognition (OCR)

OCR is a way of automating data (text) extraction from images. Typically, this can be done on any image file, PDF, or scanned documents containing typed and even handwritten text. The most common utilization of this technology comes in the form of electronic data entry that derives from printed material including receipts, bank statements, invoices, mail, and more. OCR is even used by traffic cameras to scan license plates and issue red light tickets.

There are several pre-processes used in modern OCR that help boost successful recognition and accuracy of text detection in images. Some of these pre-processes include de-skewing and straightening of the document, removing speckles or other “non-glyph” artifacts, and converting the images to greyscale to provide a binary image of black and white.

The two basic OCR algorithms include matrix matching and feature extraction. Matrix matching, being one of the earlier utilized methods, will analyze each character one-by-one, pixel-by-pixel, and compare each against a stored glyph and try to match its patterns. A significant downside of using this method is that it relies on the text being isolated from the rest of the image, so it works best on typed documents with consistent text and font. Feature extraction, which is a more modern solution, will breakdown each character into “features” such as “lines, closed loops, line direction, and line intersections”. The extracted features are then compared with an “abstract vector-like representation of a character” which reduces the amount of needed stored glyphs to compare against, boosting overall efficiency. The k-nearest neighbors algorithm is also used to in this process to help with said comparisons.

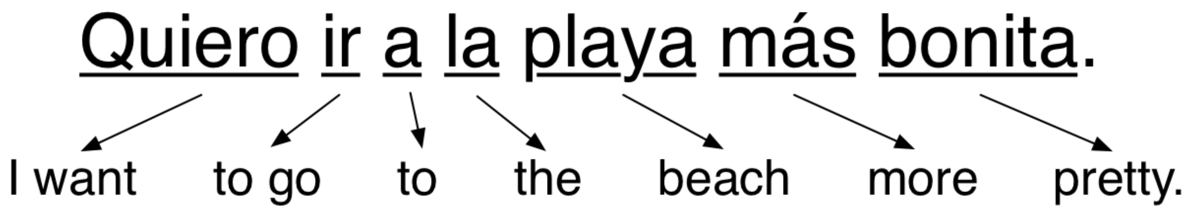
II. History of OCR

Ideas of OCR were conceived as early as the 1870s, but the first usage of OCR was credited to Fournier d'Albe for his invention of the Optophone in 1913, which was a device to help blind people read. The Optophone applied sonification to read black print and generate chords and tones used to identify specific characters. While this invention was not popular by any means and only a few units were made, it still marked the first application of OCR and paved the way for this highly useful technology. Between the years 1920 and 1931 a couple more inventions were produced that used OCR, except instead of converting text to audible tones, these machines read text and produced telegraph code, which was highly useful as telegraphs were the fastest and most reliable way of long-distance communication at the time. Fast forward through several years and commercially successful inventions to the present day, where there are several applications and services that allow anyone to easily access this technology, many of which are free to use.

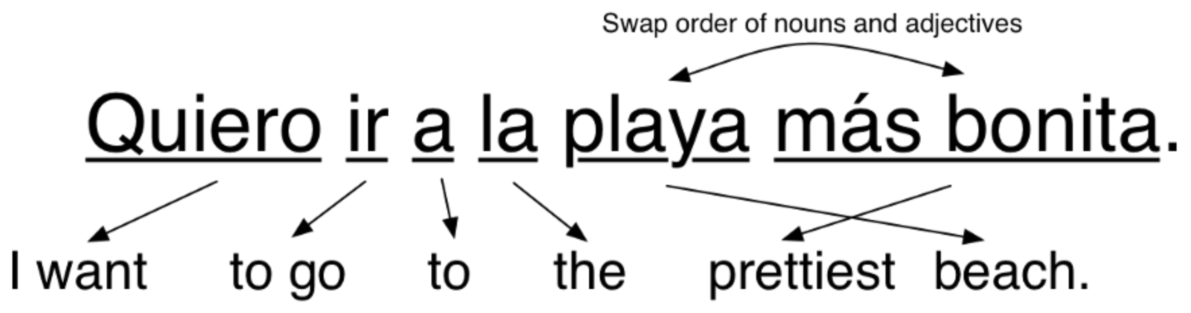
III. How Computers Translate Text

The following segment will explain on a simplistic level how computers handle text translation. Below we will be working to translate the following sentence from Spanish to English: "Quiero ir a la playa más bonita"

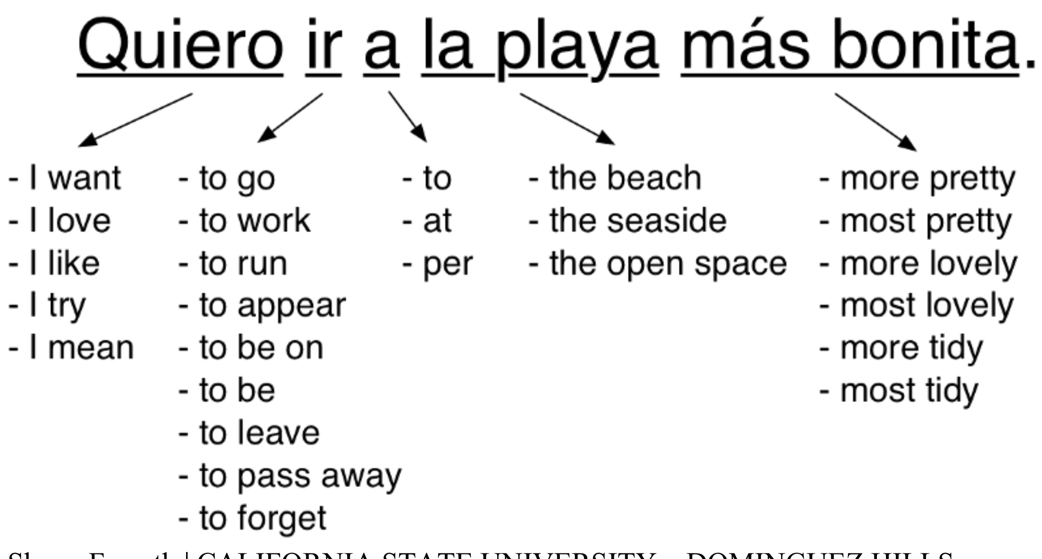
Step 1: Break up the sentence for a word-by-word translation.



Step 2: Swap the order of the words to match grammar rules of the target language.


Swap order of nouns and adjectives
Quiero ir a la playa más bonita.
I want
to go
to
the
prettiest
beach.

Step 3: Find all possible translations for each word/segment.



Step 4: Generate possible sentences to find most likely candidate. Here are some examples of how the sentence could read:

I love | to leave | at | the seaside | more tidy.

I mean | to be on | to | the open space | most lovely.

I like | to be |on | per the seaside | more lovely.

I mean | to go | to | the open space | most tidy.

I try | to run | at | the prettiest | open space.

I want | to run | per | the more tidy | open space.

I mean | to forget | at | the tidiest | beach.

I try | to go | per | the more tidy | seaside.

Step 5: These sentences are then compared against millions of possible phrases and sentences to produce a candidate that sounds the most “human-like”. Eventually the highest-ranked candidate is chosen which reads: “I want to go to the most beautiful beach.”

While this method is the simplest and is somewhat dated, it still requires a tremendous amount of training data to use statistics to rank the most likely candidates. For brevity’s sake, I will not cover the new and improved algorithms that are much more complicated to explain and comprehend. The above gives enough of a foundation to explain the foundations of text translation.

V. Tools/Software

***Languages/Frameworks*:**

Javascript/HTML/CSS

Vue.js 2 – An open-source JavaScript front end framework - https://vuejs.org/

Express – Back end framework for Node.js - https://expressjs.com/

Vuetify – UI framework built on Vue.js - https://vuetifyjs.com/en/

***Node Packages*:**

Axios – Promise based HTTP client for the browser and Node.js -

npmjs.com/package/axios

Multer – Node.js middleware for handling file uploads - www.npmjs.com/package/multer

Cors – Node.js middleware for handling Cross-Origin Resource Sharing -

https://www.npmjs.com/package/cors

Sharp – Node.js module for image processing - https://www.npmjs.com/package/sharp

***APIs*:**

Cloud Vision – uses pre-trained machine learning models to analyze images -

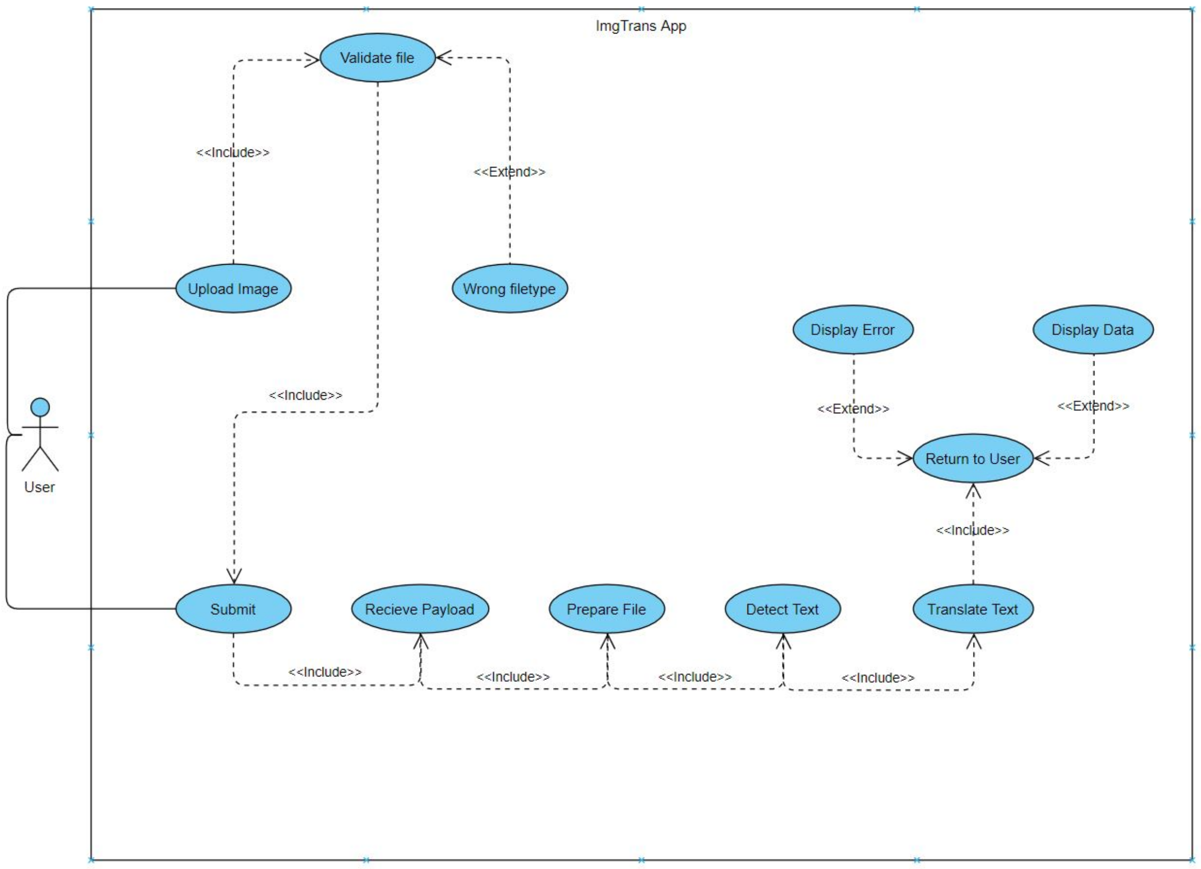
https://cloud.google.com/vision

Translation API – uses AutoML to handle translations -

https://cloud.google.com/translate/docs

VI. Project Structure/Diagrams

I. Use Case Diagram



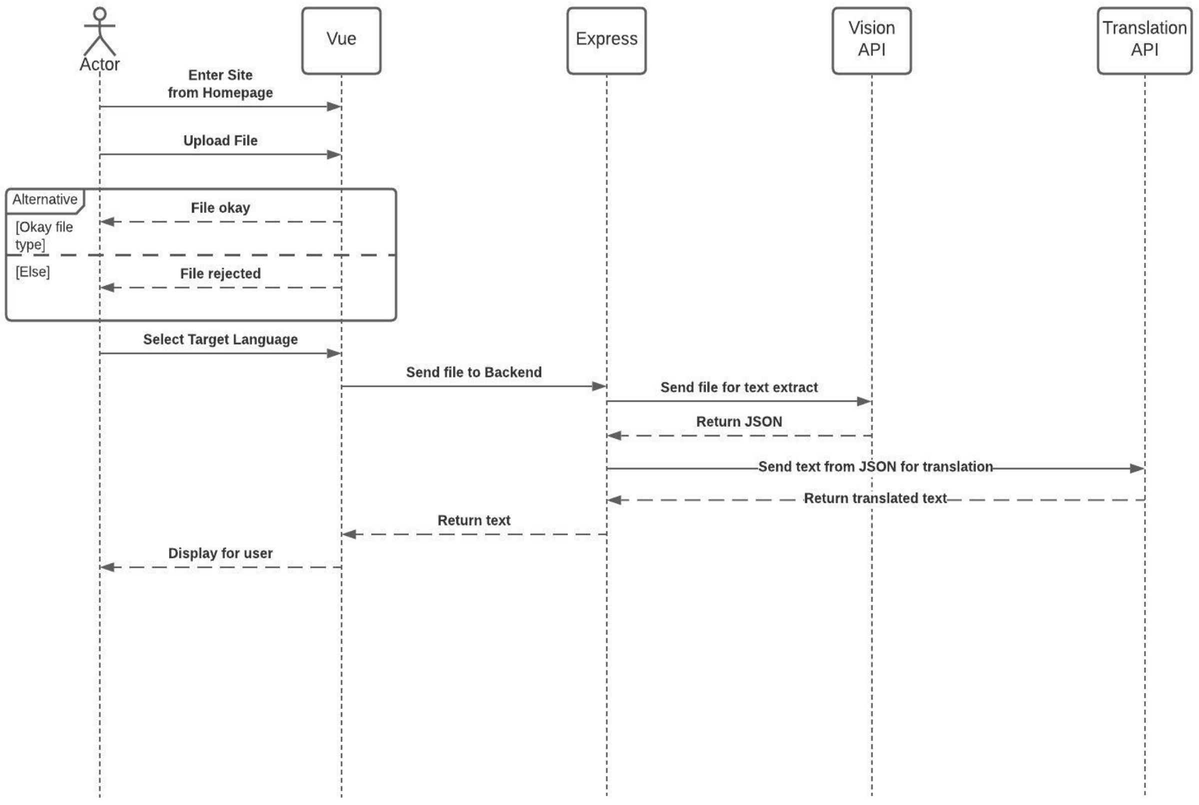
1. Use Case: Upload Image, Actor: User

Triggered by the user selecting an image to upload. The user will be prompted to upload an image of formats: JPG, JPEG, or PNG. Assuming the user selects the correct file format for upload, the program will resume with no incident. If the user selects an unaccepted file type, such as a pdf, the user will be prompted that they have selected the wrong file type and to try again. The input field will then clear, and the user will be able to retry their upload. Upon successful upload, a submit button will appear, indicating the selected the correct file type. This will allow for the next use case to be accessible.

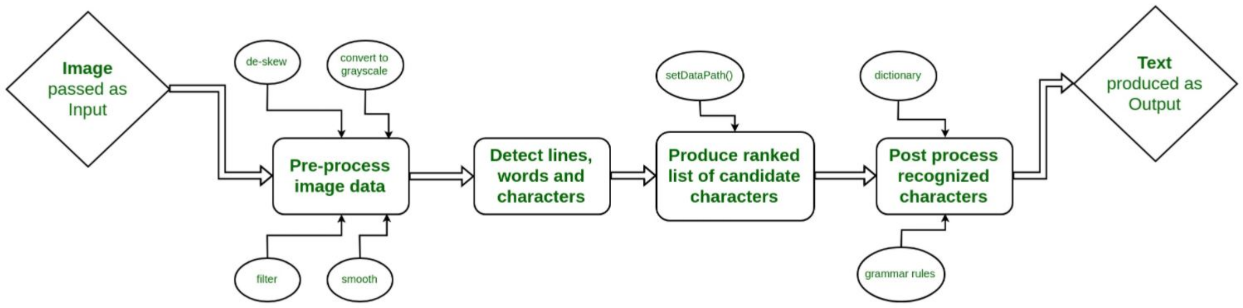
2. Use Case: Submit, Actor: User

When the user presses the submit button, the file will be sent to the backend, where the image processing takes place. The file is then prepared by Multer, which assigns a storage location and converts the raw file back into an image that can be retrieved. The image is then passed into a function that uses Cloud Vision API where the text is detected and extracted into an array. This array of text is then sent to the Translation API, where the array of text is parsed and produces a new array of translated text. Finally, the result is returned to the frontend where the user will either be met by an error, or the desired output of the original text that has been extracted from the image and the desired translated text.

II. Sequence Diagram

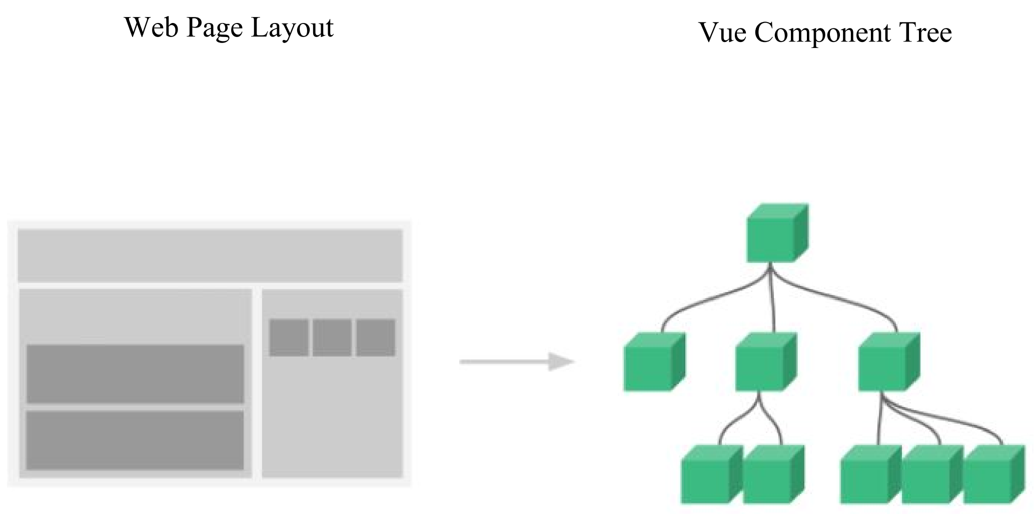


*III. Basic Diagram of OCR**



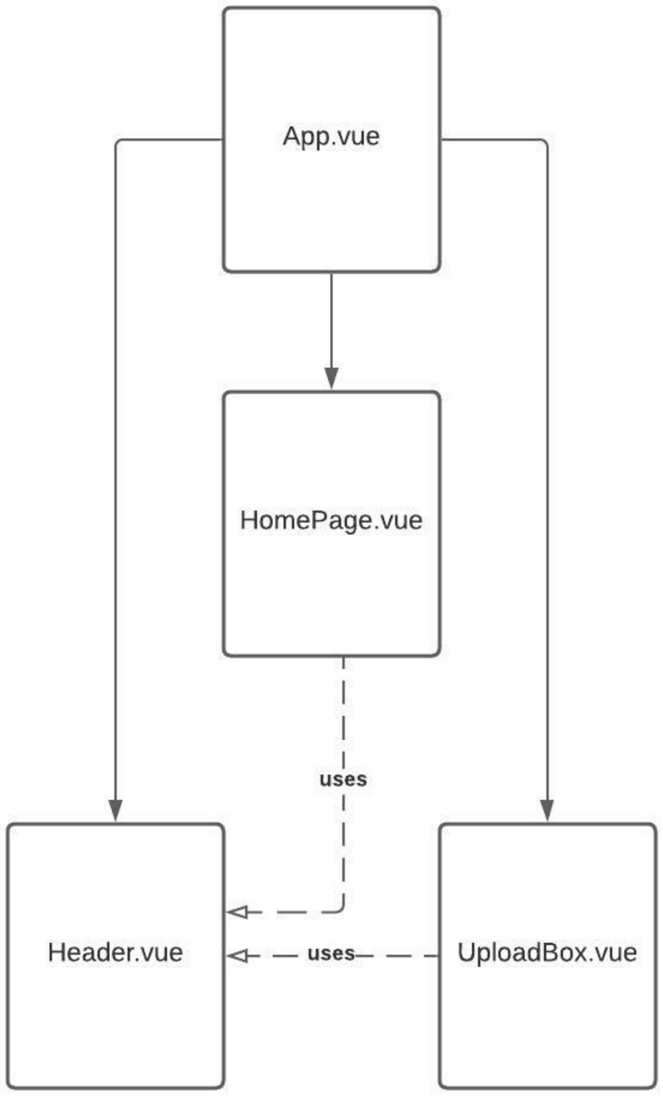
**This is just a general diagram of how OCR works. Google's Cloud Vision seems to keep their algorithms obscure for obvious reasons. While Cloud Vision may follow a similar flow, Google also has massive amounts of data at their disposal to compare against during its execution.*

IV. General Component Diagram



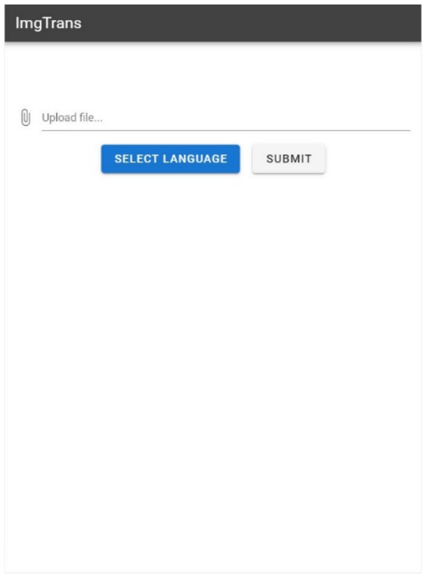
Vue is especially convenient for its implementation of components. Vue components are exactly what they sound like, they allow the developer to break down large-scale applications into small, compartmentalized components that can be reused across the application. For example, in ImgTrans, the UploadBox component could be made into a simpler, more generalized component that is stripped of its specific functionality. Developers could then use that barebones component to serve some other purpose. The reusability of components makes frontend development more streamlined and can significantly boost production, which is why frontend JavaScript frameworks like Vue, React, and Angular are currently so popular.

V. Diagram of Vue Components for ImgTrans

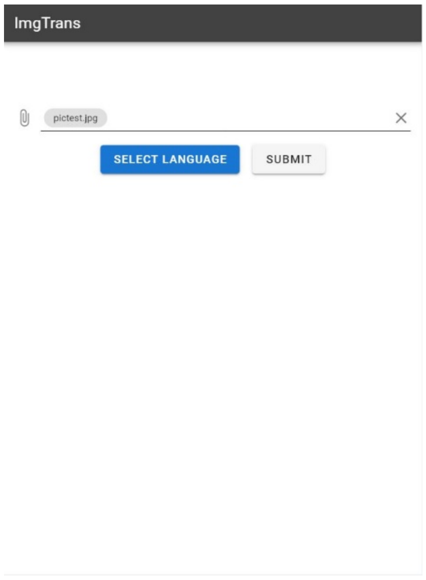


VI. Basic application flow

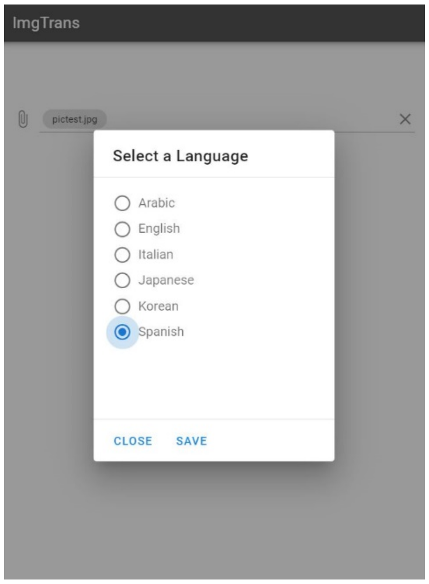
1. User Interface



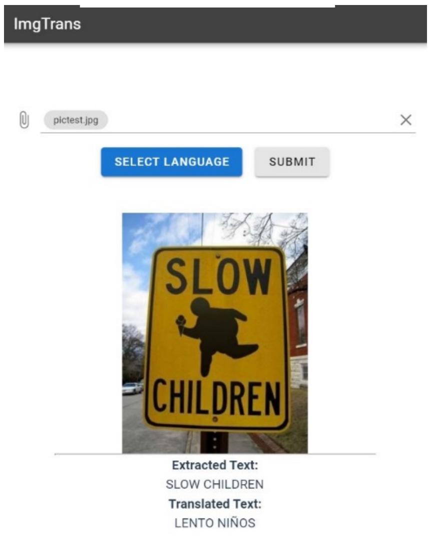
2. Selected file



3. Select Target Language



4. Output Results



Extracted Text:
SLOW CHILDREN
Translated Text:
LENTO NIÑOS

Barebones working example of ImgTrans, not final product.

VII. Current Work and Implementation

I. Working Functions/Components:

- Vue
    - UploadBox.vue:
        - errorHandler(): Gives user feedback if there is a problem with upload.
        - fileSelected(): Sets uploaded file to local variable.
        - submitFile(): Bulk of frontend processing. Sets user-defined data and bundles everything into a FormData object which is sent as a request to the backend. Receives response from backend and sets variables to data returned from API calls.
    - HomePage.vue
        - redirectToUpload(): Handles browser redirection to main page.
    - Header.vue
        - redirectToHome(): Handles browser redirection to welcome page.
- Express
    - Index.js
        - detectText()
        - translateText()
        - app.post(/upload)
        - All middleware working
- CSS/Styling – so far, all styling/UI is working as intended. Responsive for mobile devices as well.

II. Non-working Functions:

- Vue
    - UploadBox.vue
        - draw(): This function is responsible for drawing boxes around the selected text. The function does work assuming the image is rendered in the same resolution and size as the original upload, but this is usually not the case. Most of the time the image must be scaled down, especially for mobile devices to maintain the UI, which throws off the returned vertices from the API call.

III. Test Cases

1. errorHandler(): Current max size for files is 200Kb but can be adjusted. Here I will show one example of the browser view but going forward will only be showing mobile view where appropriate for ease of viewing. Notice that the buttons are currently locked, and can only be unlocked if the user uploads a file that follows the parameters that are set.

Browser view:

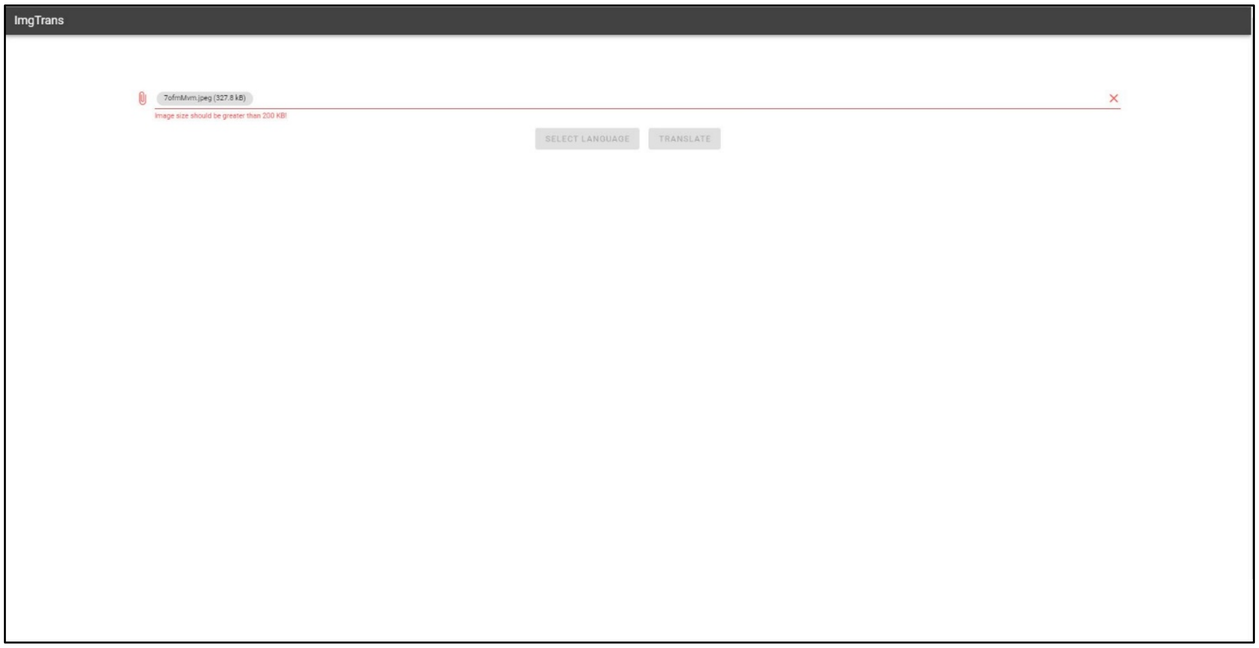


Image size should be greater than 200 KB!

Mobile view:

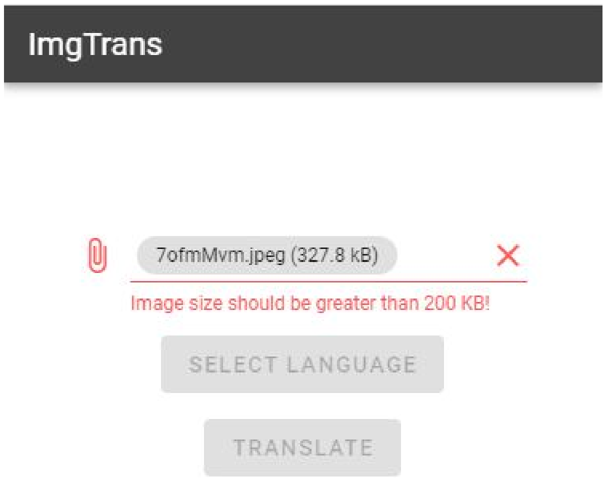
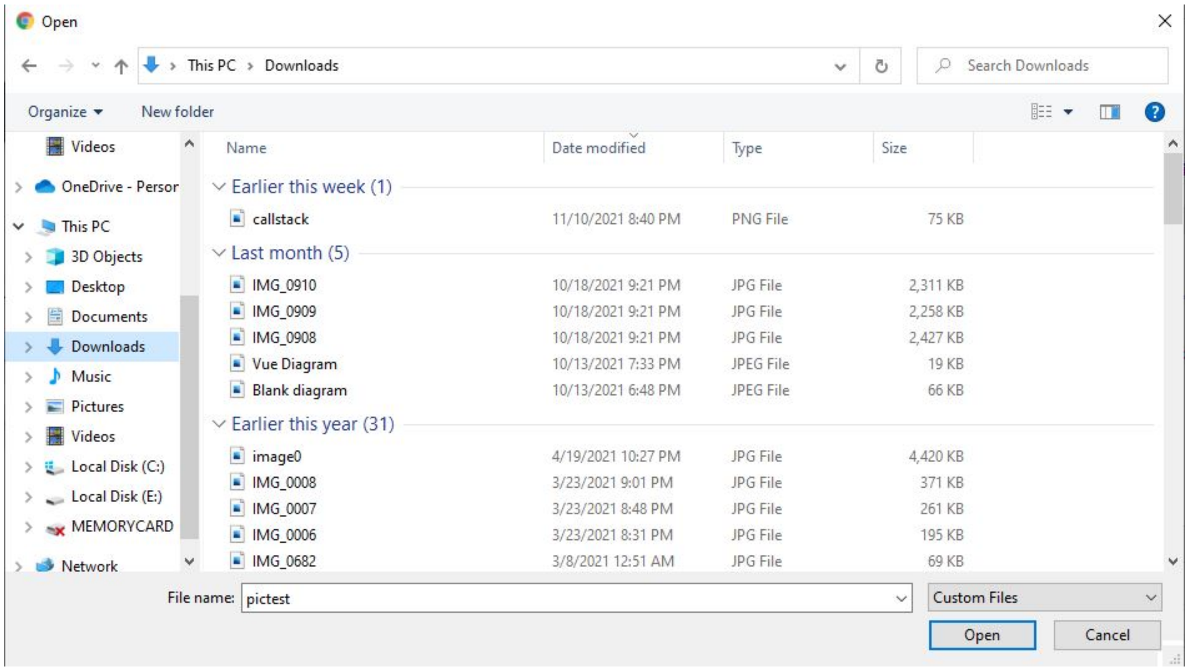


Image size should be greater than 200 KB!

2. Below demonstrates an important aspect of file validation. Vuetify allows developers to set restrictions for their upload components. This is to show that only images of jpeg and png format are viewable in the file selector when uploading a file, even though my download folder contains numerous other file types that would normally be visible. If the user were to manage to upload a file that is not supported, it would also be caught by Multer, which also handles file validation, in the backend. The frontend validation is a weak frontline defense while the backend validation will reject any requests made by an unsupported file type and throw an error to the user as the response.



Name	Date modified	Type	Size
Earlier this week (1)			
callstack	11/10/2021 8:40 PM	PNG File	75 KB
Last month (5)			
IMG_0910	10/18/2021 9:21 PM	JPG File	2,311 KB
IMG_0909	10/18/2021 9:21 PM	JPG File	2,258 KB
IMG_0908	10/18/2021 9:21 PM	JPG File	2,427 KB
Vue Diagram	10/13/2021 7:33 PM	JPEG File	19 KB
Blank diagram	10/13/2021 6:48 PM	JPEG File	66 KB
Earlier this year (31)			
image0	4/19/2021 10:27 PM	JPG File	4,420 KB
IMG_0008	3/23/2021 9:01 PM	JPG File	371 KB
IMG_0007	3/23/2021 8:48 PM	JPG File	261 KB
IMG_0006	3/23/2021 8:31 PM	JPG File	195 KB
IMG_0682	3/8/2021 12:51 AM	JPG File	69 KB

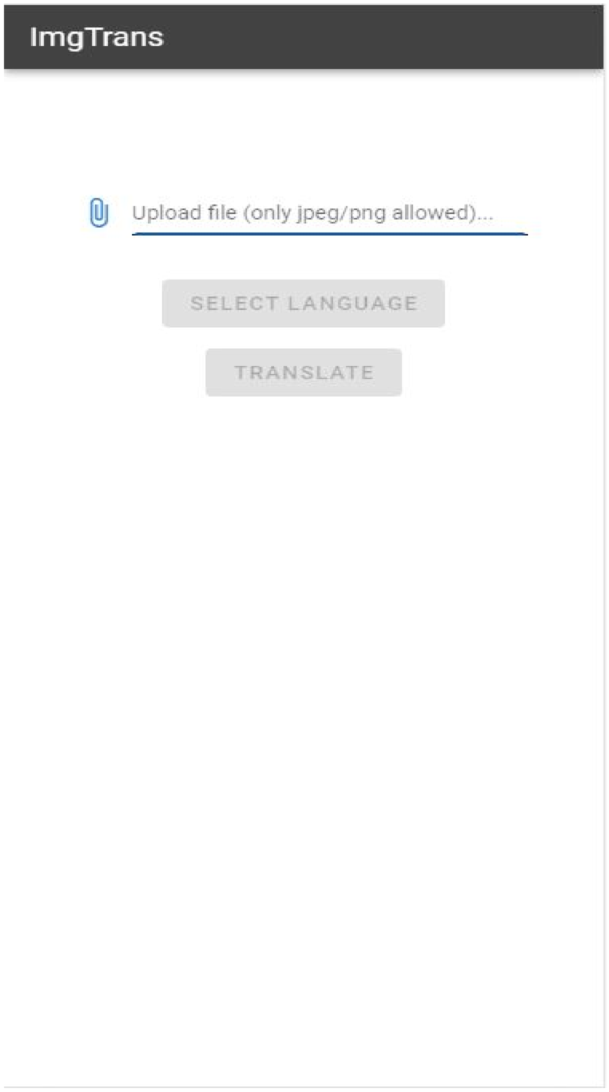
File name: pictest

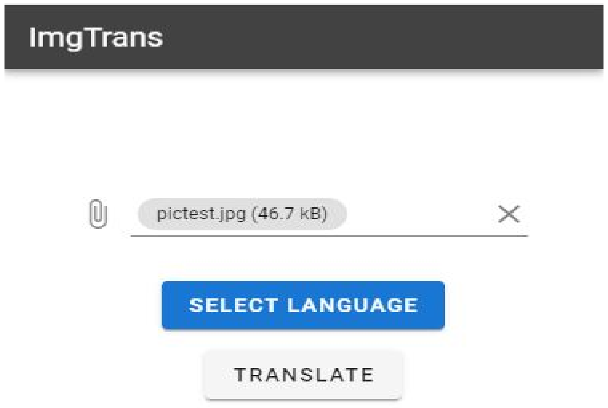
Custom Files

3. To prevent empty requests being sent to Express, there are several checks in place to ensure the FormData cannot be submitted until all appropriate data is received from the user. Primarily, The "TRANSLATE" button is disabled until a few conditions are met:

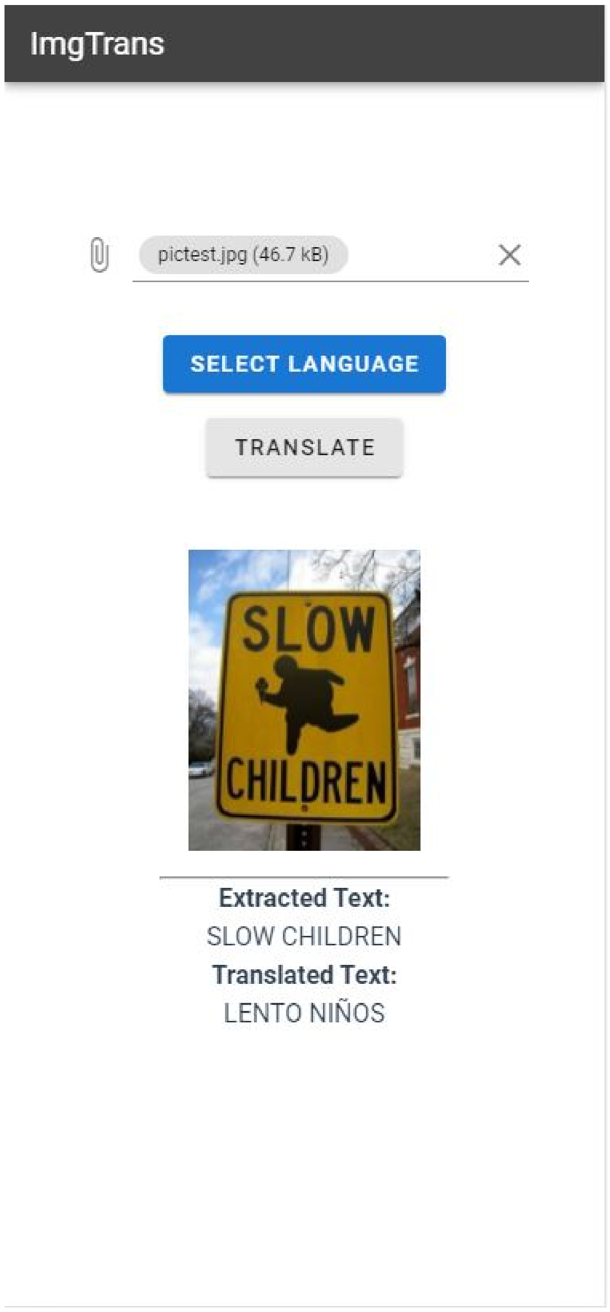
1. A valid file type must be selected.
2. A target language must be selected.
3. The errorHandler() returns a truthy value, saying the size is acceptable.

Below are examples of before and after the buttons are unlocked. Will also revert to left image once the file is cleared from the upload box.



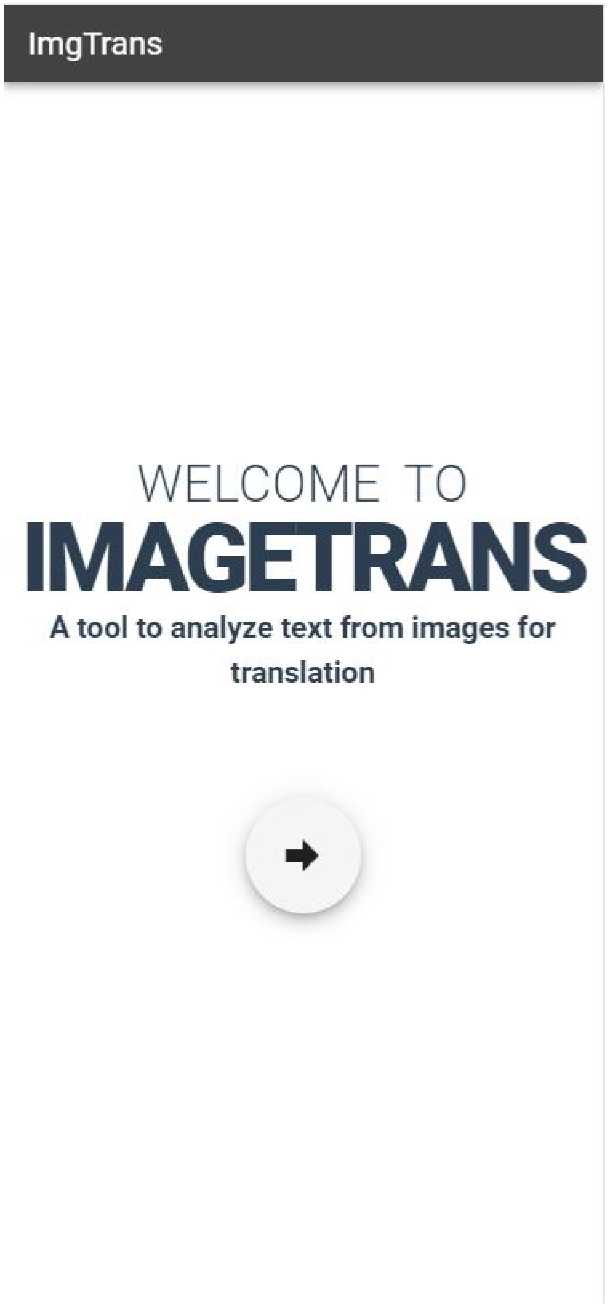


4. Below is an example of when the "TRANSLATE" button is finally pressed. This will execute most of the code that has not been mentioned yet. The resulting window, below the buttons, is rendered conditionally and will also be removed from the page if the file is cleared from the page by the user. This is where the user would also have the option to visualize where the detected text is on the image, but since that feature is not fully working, it is excluded for the time being.



Extracted Text:
SLOW CHILDREN
Translated Text:
LENTO NIÑOS

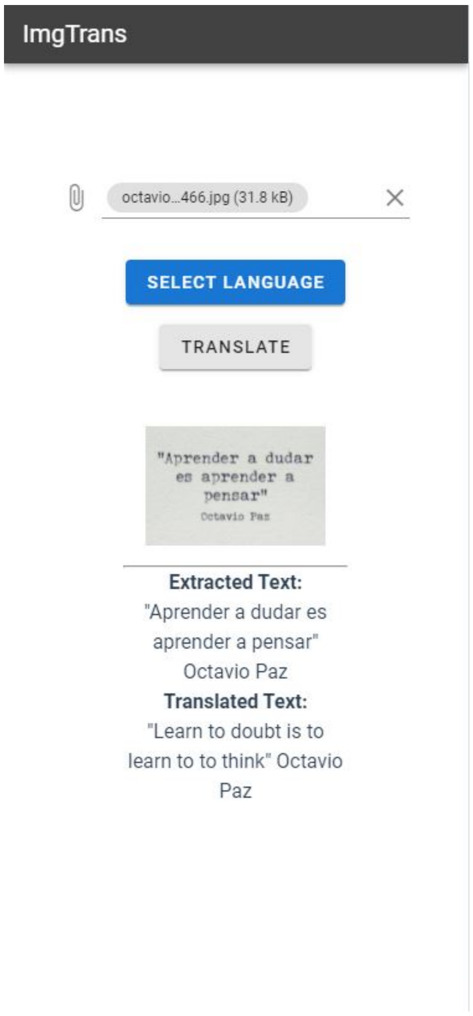
5. This is a barebones homepage to greet the user. It should be populated with more information but for now serves as a placeholder to demonstrate the ability to navigate between multiple pages using a VueRouter. The button with the arrow serves as the entrance to the actual functionality of the application. Another feature added are transitions between pages, which are obviously not capturable on still images. Instead of a sudden snap to the next page there are smooth fade outs to ins between pages.



IV. Current Problems:

Translation API:

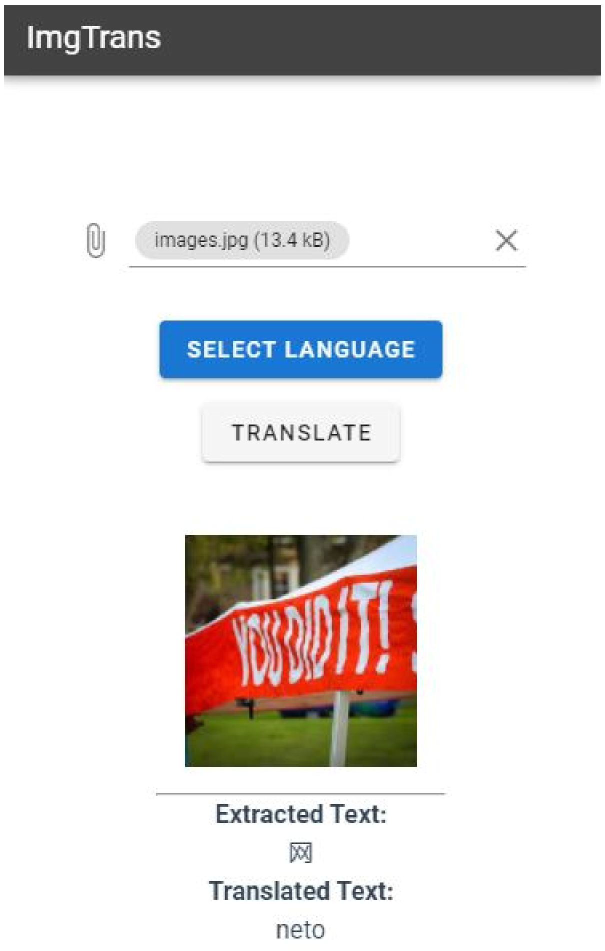
The Translation API seems to give mixed results when matched against the translator Google uses on their browser. The browser translator seems to be significantly more accurate than ImgTrans, even though this app is using Google's APIs. The reason for this is not apparent, but perhaps the APIs that Google provides are not nearly as accurate for intentional reasons, which most like are to protect their business against other companies having an equally reliable implementation. Below is an example of the shortcomings of the Translation API: The translated text should read: "Learning to doubt is learning to think" Octavio Paz.


ImgTrans
octavio...466.jpg (31.8 kB)
SELECT LANGUAGE
TRANSLATE
"Aprender a dudar es aprender a pensar"
Octavio Paz

Extracted Text:
"Aprender a dudar es aprender a pensar" Octavio Paz
Translated Text:
"Learn to doubt is to learn to to think" Octavio Paz

Vision API:

While it is impossible to expect perfection from OCR algorithms, there are still occasional issues with the text detection. For example:



Extracted Text:

冈

Translated Text:

neto

This is obviously an extreme example, but it proves the above point. There is something that is readable to English speakers in the image, but there are still problems that exist in OCR technology that makes it extremely difficult for a computer to process this accurately. Not only is the extracted text garbage data, but so is the translation. Ideally, there would be a check to see the confidence of the algorithm to determine if it even returns useable data, and instead tells the user to resubmit a better image.

VIII. Prior Related Work

I. Work done by others

1. Online OCR - https://www.onlineocr.net/

Online OCR is a website that allows a user to convert a few specific file types into a Word, Excel, or a plaintext document. It is the first non-ad placed result from Google for an "OCR website". Upon further inspection, it appears Online OCR has its own proprietary API that developers can use to access their OCR technology. While this website works fine if a user casually needs to convert a simple PDF to an editable document, there are a few problems discovered listed below:

- Highly inaccurate results from text pulled directly from a real-world photo. For example, it could not read a simple street sign with highly uniform text that only contained the two words "SLOW CHILDREN". It returned completely random text with a 0% accuracy. Some photos did slightly better, but it seems that any image or document with a non-plain background will return inaccurate results.
- Users are unable to convert more than 15 documents an hour and they implement several other restrictions without first creating an account. Once an account is created the user gains more features, but only gets 50 uses until they are forced to pay. Cost is dependent on how many documents you pay for, but it seems unlikely a user would want to pay given the accuracy and limits of their own API.
- If a developer wants to use their API to make HTTP calls for their own application, they require you pay a monthly fee depending on daily request limits that the developer can choose from, and the fees seem rather high for the quality and accuracy of said API.

2. Yandex Translate - https://translate.yandex.com/ocr

Yandex Translate is a web application that allows users to translate their own inputted text, images, websites, and even documents. From a design standpoint the site is extremely well put together and provides a clean interface and plenty of functionality. From what I can gather, it appears the service is completely free to use, but may not be the case considering my limited usage. Yandex also offers their own API for developers but instead of charging per request, they only charge per million characters translated which is 15$ and is the minimum monthly payment a developer can make. This might be great to use for an application where its intended use isn't reliant on translating overly dense documents. Yandex has a similar downfall in whatever OCR technology they are using as Online OCR. While it appears their accuracy is significantly better for images (compared to Online OCR), it is still not up to par with other services, such as Google's Vision API.

3. TranslatePhoto – Mobile Application

TranslatePhoto serves all the functionality of Yandex, minus web page translation, in the form of a mobile application. It has great functionality, user interface, and surprisingly good OCR. A major caveat to this is the extremely intrusive advertisements and that are served at a frequency that is highly off-putting. Only 5 requests are able to be made per 24 hours if the user does not sign up for their monthly membership which is quite cheap at 8$ a month for unlimited translations. One feature that this app lacks is that it does not show the which words are being translated. For instance, if there is an image with text that is scrambled across on the image, it is impossible for the user to know which words correspond with the translation they provide. This could be crucial for someone who is either trying to learn new language or simply just gather information, say if they are trying to read a sign in a foreign country.

II. Work done by me

The purpose of this web application is for users of any computer or smart device to be able to upload an image or document and receive any text found in an image translated to the desired language. To do the text detection and translation, I will be using Google's Cloud Vision and Translation APIs. The bulk of the work I will be doing is as follows:

- Setting up the frontend using Vue.js and connecting it to my backend, in which I will be using Express.js.
- Handling file upload on the front end and using Axios to send the uploaded file to my back end.
- Perform file validation on uploaded files using Multer to make sure the user is uploading an applicable file format.
- Converting the raw File object back to the specified file type using Sharp so it is able to be passed to the API.
- Making a request to Vision API on the file and extracting the text
- Make another request to Translation API to translate the extracted text
- Send the data back to the front end to display for the user

As time permits:

I would like to set up a registration process to allow user log-in and save their information using MongoDB. Additionally, I would like my site to be deployable in its final state and would use a service like Heroku for this process.

Here are some issues/ideas I wanted to address in relation to previous work by others, but not all were completed due to time restraints:

- Visually outline each word and block of words and show the relation between the translated text to the user so they know what is being translated to what.
- Make the user experience enjoyable for a user who is using the application for free.
- Provide a clean user interface that makes navigating the application simple and aesthetically pleasing.
- Use APIs that are arguably the best at what they do to provide the most accurate translations, even if it means sacrificing profits to pay for each request.

IX. References

David Shepard Invents the First OCR System "GISMO" : History of Information. (n.d.). History of Information. Retrieved September 16, 2021, from https://www.historyofinformation.com/detail.php?entryid=885

Geitgey, A. (2020, September 24). *Machine Learning is Fun Part 5: Language Translation with Deep Learning and the Magic of Sequences*. Medium. Retrieved December 1, 2021, from https://medium.com/@ageitgey/machine-learning-is-fun-part-5-language-translation-with-deep-learning-and-the-magic-of-sequences-2ace0acca0aa

Gangele, J. (2018, June 7). *How does AutoML works? - jeetendra gangele*. Medium. https://medium.com/@gangele397/how-does-automl-works-b0f9e45fbb24

Hyland Software Inc. (n.d.). *What is Optical Character Recognition? | OCR Technology | Hyland*. Hyland. Retrieved September 16, 2021, from https://www.hyland.com/en/resources/terminology/data-capture/what-is-optical-character-recognition-ocr

Karandish, F. (2019, November 25). *The Comprehensive Guide to Optical Character Recognition (OCR)*. Moov AI. https://moov.ai/en/blog/optical-character-recognition-ocr/

Schantz, Herbert F. (1982). *The history of OCR, optical character recognition*. Recognition Technologies Users Association. ISBN 9780943072012.

"How OCR Software Works". OCRWizard. Archived from the original on August 16, 2009. Retrieved June 16, 2013.

Technostacks. (2020, December 8). *How Cloud Vision API is utilized to integrate Google Vision Features*. Technostacks Infotech. https://technostacks.com/blog/google-cloud-vision-api-features-use-cases/

X. Appendix of Code

Note: I will only be attaching code from the main driving files of the application. The code in full can be found at: https://github.com/shaunfroseth/imgtrans

UploadBox.vue:

```
<template>
  <div class="hello">
    <!-- File Upload Container -->
    <div background="grey darken-3" class="fileBox">
      <v-file-input
        class="my-4"
        accept="image/png, image/jpeg"
        name="file"
        small-chips
        show-size
        :rules="rules"
        truncate-length="15"
        placeholder="Upload file (only jpeg/png allowed)..."
        @change="fileSelected"
        @click:clear="
          message = '';
          resultText = '';
          translatedText = '';
          returnFile = null;
          uploadError = false;
          submitPressed = false;
        "
        @update:error="errorHandler"
      >
      </v-file-input>
      <!-- Buttons (Select/Translate) -->
      <v-row justify="center">
        <v-dialog v-model="dialog" scrollable max-width="300px">
          <template v-slot:activator="{ on, attrs }">
            <v-btn
              class="ma-2"
              color="primary"
              :disabled="!file || uploadError"
              v-bind="attrs"
              v-on="on"
              @click="message = ''"
            >
              Select Language
            </v-btn>
            <v-btn
              @click="submitFile"
              :disabled="!file || dialogm1 == '' || uploadError"
              class="ma-2"
              >Translate</v-btn
            >
          </template>
          <!-- Language Selector -->
          <v-card>
            <v-card-title>Select a Language</v-card-title>
            <v-divider></v-divider>
            <v-card-text style="height: 300px;">
              <v-radio-group v-model="dialogm1" column>
                <v-radio label="Arabic" value="ar"></v-radio>
                <v-radio label="Czech" value="cs"></v-radio>
                <v-radio label="Danish" value="da"></v-radio>
                <v-radio label="Dutch" value="nl"></v-radio>
                <v-radio label="English" value="en"></v-radio>
                <v-radio label="French" value="fr"></v-radio>
                <v-radio label="German" value="de"></v-radio>
                <v-radio label="Greek" value="el"></v-radio>
                <v-radio label="Hindi" value="hi"></v-radio>
                <v-radio label="Italian" value="it"></v-radio>
                <v-radio label="Japanese" value="ja"></v-radio>
                <v-radio label="Korean" value="ko"></v-radio>
                <v-radio label="Persian" value="fa"></v-radio>
                <v-radio label="Polish" value="pl"></v-radio>
                <v-radio label="Portuguese" value="pt"></v-radio>
                <v-radio label="Russian" value="ru"></v-radio>
                <v-radio label="Serbian" value="sr"></v-radio>
                <v-radio label="Spanish" value="es"></v-radio>
                <v-radio label="Swedish" value="sv"></v-radio>
                <v-radio label="Vietnamese" value="vi"></v-radio>
              </v-radio-group>
            </v-card-text>
            <v-divider></v-divider>
            <v-card-actions>
              <v-btn color="blue darken-1" text @click="dialog = false">
                Close
              </v-btn>
              <v-btn color="blue darken-1" text @click="dialog = false">
                Save
              </v-btn>
            </v-card-actions>
```

```
        </v-card>
      </v-dialog>
    </v-row>
    <!-- Output -->
    <div v-show="returnFile" class="outputResult">
      <figure>
        <img class="centered my-4" :src="returnFile" alt="" />
        <hr />
        <h4>Extracted Text:</h4>
        <figcaption>{{ resultText }}</figcaption>
        <h4>Translated Text:</h4>
        <figcaption>{{ translatedText }}</figcaption>
      </figure>
    </div>
  </div>
  <!-- Progress Bar -->
  <div v-if="submitPressed" class="progress-bar">
    <v-col>
      <p>Fetching text ...</p>
      <v-progress-linear
        class="my-4"
        color="blue darken-1"
        indeterminate
        rounded
        height="6"
      ></v-progress-linear>
    </v-col>
  </div>

  <!-- TO-DO:  Implement text outliner
    <div>
    <div id="imgContainer">
      <img src="@/assets/pictest.jpg" />
      <canvas
        width="500"
        height="649"
        ref="myCanvas"
        class="canvas-overlay"
      ></canvas>
    </div>
    <button @click="draw">Draw</button>
  </div> -->
 </div>
</template>

<script>
import axios from "axios";
```

```
export default {
 name: "UploadBox",
 data() {
  return {
   file: null,
   uploadError: false,
   returnFile: null,
   resultText: "",
   uploadedFiles: [],
   translatedText: "",
   dialogm1: "",
   dialog: false,
   message: "",
   error: false,
   submitPressed: false,
   rules: [
    (value) =>
     !value ||
     value.size < 200000 ||
     "Image size should be greater than 200 KB!",
   ],
  };
 },
 props: {
  msg: String,
 },
 methods: {
  errorHandler(value) {
   if (value) this.uploadError = true;
   console.log(value);
  },
  draw() {
   let c = this.$refs["myCanvas"];
   const canvas = c.getContext("2d");
   canvas.beginPath();
   canvas.rect(118, 113, 401 - 118, 222 - 113);
   canvas.stroke();
   canvas.beginPath();
   canvas.rect(86, 435, 427 - 86, 555 - 435);
   canvas.stroke();
  },
  fileSelected(event) {
   this.file = event;
   this.error = false;
  },
  async submitFile() {
   this.submitPressed = true;
```

```
      const fd = new FormData();
      fd.append("file", this.file, this.file.name);
      fd.append("dialog", this.dialogm1);

      try {
        const response = await axios.post(
          "http://192.168.0.13:5000/upload",
          fd
        );
        this.resultText = response.data.results;
        this.translatedText = response.data.translatedResults;
        this.returnFile = response.data.file;
        console.log(response.data.file);
        this.error = false;
      } catch (err) {
        this.message = err.response.data.error;
        this.error = true;
      }
      this.submitPressed = false;
    },
  },
};
</script>

<!-- Add "scoped" attribute to limit CSS to this component only -->
<style scoped>
.hello {
  width: 100%;
}
.fileBox {
  width: 80%;
  margin: auto;
}

.v-input {
  font-size: 14px;
}

.v-text-field {
  left: 100px;
}
.outputResult {
  margin: 50px;
}
.centered {
  display: block;
  margin-left: auto;
  margin-right: auto;
}

h3 {
  margin: 40px 0 0;
}
ul {
  list-style-type: none;
  padding: 0;
}
li {
  display: inline-block;
  margin: 0 10px;
}
a {
  color: #42b983;
}
img {
  height: auto;
  width: 400px;
}
figcaption {
  margin-top: 0px;
}
.canvas-overlay {
  position: absolute;
  top: 0;
  left: 0;
  right: 0;
  bottom: 0;
  pointer-events: none;
  width: 100%;
  height: 100%;
}

.canvas-wrapper {
  position: relative;
}
.progress-bar {
  height: 500px;
  width: 75%;
  margin: auto;
  display: flex;
  align-items: center;
}
div#imgContainer {
  /* min-width: 200px;
  min-height: 200px; */
  display: inline-block;
  position: relative;
}
@media (max-width: 500px) {
  img {
    height: auto;
    width: 80%;
  }
}
</style>
```

Header.vue:

```
<template>
  <v-app-bar dense fixed dark color="grey darken-3">
    <v-app-bar-title>
      <v-btn plain left class="title pl-0" @click="redirectHome"
        >ImgTrans</v-btn
      >
    </v-app-bar-title>
  </v-app-bar>
</template>

<script>
export default {
  methods: {
    redirectHome() {
      this.$router.push({ path: "/" });
    },
  },
};
</script>
<style>
.v-toolbar__content {
  margin-bottom: 110px;
}
</style>
```

HomePage.vue

```
<template>
  <div class="welcome">
    <h1>WELCOME TO</h1>
    <h2>IMAGETRANS</h2>
    <h3 class="mb-16">A tool to analyze text from images for translation</h3>
    <v-btn fab x-large @click="redirectToUpload">
      <v-icon>mdi-arrow-right-bold</v-icon>
    </v-btn>
  </div>
</template>

<script>
export default {
  methods: {
    redirectToUpload() {
      this.$router.push({ path: "/upload" });
    },
  },
};
</script>

<style scoped>
h1 {
  font-size: 2.75rem;
  line-height: 2.75rem;
  letter-spacing: 0.0073529412em !important;
  word-spacing: 0.2em;
  font-weight: 300;
}

h2 {
  font-size: 3.75rem;
  line-height: 3.75rem;
  letter-spacing: -0.053333333em !important;
  word-spacing: 0.2em;
  font-weight: 900;
}

.welcome {
  position: absolute;
  left: 50%;
  top: 40%;
  -webkit-transform: translate(-50%, -50%);
  transform: translate(-50%, -50%);
  font-family: "Roboto";
}

@media (max-width: 500px) {
  h1 {
    font-size: 2rem !important;
    line-height: 2rem;
  }
}
</style>
```

Index.js:

```
const express = require("express");
const cors = require("cors");
const multer = require("multer");
const sharp = require("sharp");
const fs = require("fs").promises;
const path = require("path");

const app = express();

const fileFilter = function (req, file, cb) {
  const allowedTypes = ["image/jpeg", "image/png"];

  if (!allowedTypes.includes(file.mimetype)) {
    const error = new Error("Wrong file type");
    error.code = "LIMIT_FILE_TYPES";
    return cb(error, false);
  }

  cb(null, true);
};

const MAX_SIZE = 200000;
const upload = multer({
  dest: "./uploads/",
  fileFilter,
  limits: {
    fileSize: MAX_SIZE,
  },
});

app.use(express.json());
app.use(cors());
app.use("/static", express.static(path.join(__dirname, "static")));
```

```
// Vision API call
async function detectText(filePath) {
  // Imports the Google Cloud client library
  const vision = require("@google-cloud/vision");
  // Creates a client
  const client = new vision.ImageAnnotatorClient();
  const inputConfig = {
    content: await fs.readFile(`./static/${filePath}`),
  };

  const features = [{ type: "TEXT_DETECTION" }];

  const fileRequest = {
    image: inputConfig,
    features: features,
  };

  const request = {
    requests: [fileRequest],
  };

  const [result] = await client.batchAnnotateImages(request);
  //console.log(JSON.stringify(result, null, 2));
  return result.responses[0].textAnnotations[0].description;
}

//Translate API call
async function translateText(results, targetLang) {
  // Imports the Google Cloud client library
  const { Translate } = require("@google-cloud/translate").v2;

  // Creates a client
  const translate = new Translate();

  //Target language
  const target = targetLang;

  let [translations] = await translate.translate(results, target);
  translations = Array.isArray(translations) ? translations : [translations];
  return translations.toString();
}

app.post("/upload", upload.single("file"), async (req, res) => {
  let targetLang = req.body.dialog;
  console.log("ROUTE HIT");
  try {
    await sharp(req.file.path).toFile(`./static/${req.file.originalname}`);
    fs.unlink(req.file.path, () => {
      res.json({ file: `/static/${req.file.originalname}` });
    });

    //Calling Google endpoints
    let results = await detectText(req.file.originalname);
    let translatedResults = await translateText(results, targetLang);

    //Set return object for frontend
    let returnObject = {
      results,
      translatedResults,
      file: `http://192.168.0.13:5000/static/${req.file.originalname}`,
    };

    //Send response
    res.status(200).send(returnObject);
  } catch (error) {
    res.status(422).json({ err });
  }
});

async function quickstart() {
  const vision = require("@google-cloud/vision");

  // Creates a client
  const client = new vision.ImageAnnotatorClient();

  const fileName = "C:/Users/Shaun/Desktop/pictest.jpg";

  // Performs text detection on the local file
  const [result] = await client.textDetection(fileName);
  const detections = result;
  console.log(JSON.stringify(detections, null, 2));
  //console.log("Text:");
  //detections.forEach((text) => console.log(text));
}

app.use(function (err, req, res, next) {
  if (err.code === "LIMIT_FILE_TYPES") {
    res.status(422).json({ error: "Only jpg/png allowed" });
    return;
  }
  if (err.code === "LIMIT_FILE_SIZE") {
    res.status(422).json({ error: "File too large, max allowed size 200Kb!" });
    return;
  }
});

app.listen(5000, () => {
  console.log("Listening on port 5000");
});
```