Text to Speech with Language Translation

University of North Texas

Team Members

Akansha Goel
Brijesh Gurung
Jagdish kumar Katariya
Mahima Sunmoriya
Raheyma Khan

Professor Ting Xiao
CSCE 5214 -Software Dev with AI
University of North Texas

February 5, 2021

Contents

Participants

Our group is a team of five. The below table shows our group members and their designated high-level roles.

Akansha Goel

Email: akanshagoel@my.unt.edu

Responsibilities:

i. Team Leader – Track progress and facilitate the development
ii. Identify possible roadblocks and set deadlines
iii. Research the service and API to use for the image to text translation.
iv. Help with front end development and design ideas
v. Code Integration

Brijesh Gurung

Email: Brijeshgurung@my.unt.edu

Responsibilities:

i. Research google service for language translation
ii. Implement code for language translation using “Cloud Translation” google API, backend development
iii. Front-end work to generate a table to display translated text

Jagdishkumar Katariya

Email: Jagdishkumarkatariya@my.unt.edu

Responsibilities:

i. Research google service for text to speech
ii. Implement code for the image to text generation using “Cloud Vision API.”
iii. Implement code for text to speech using “Cloud Text to Speech” google API.
iv. Front-end work for image capture

Raheyma Khan

Email: RaheymaKhan@my.unt.edu

Responsibilities:

i. Front end development with flask template
ii. Web page styling using HTML

Mahima Sunmoriya

Email: MahimaSunmoriya@my.unt.edu

Responsibilities:

i. Collect data sets for testing our software
ii. Help front end development.
iii. Perform Accuracy analysis through Testing and validation

Workflow

Each group member was a cooperative and contributing part of the group. Our communication means were a dedicated team's channel for instant updates and zoom meetings twice a week. We meet every Tuesday at 3 pm and Friday at 4 pm. We used Microsoft's one drive for project report and presentation writing. The technology used is python with flask, Google Cloud Platform, and GitHub for source control.

Akansha proposed the project idea to the group and set the project milestones and deadlines. She also researched the cloud API for the image to text detection. Google's "Cloud Vision API" was deemed a good service we could leverage for this project. Once we decided on the deadline, we started our work by creating a Google account to have the API key. Jagadish created a Google account for project development use and connected it to the Google cloud service platform. He also generated the API token file to use for authentication while accessing Google's cloud services. This account is subscribed to "Cloud Vision API" and "Cloud Text to Speech API" services for the image to text and text to speech conversion. Jagadish also implemented code to upload the image to cloud vision API and get detected text list from the image. For code sharing and version control, we created a Git repository.

Brijesh researched another Google API that provided the language translation services. The API was called "Cloud Translation API." Our Google account added a subscription to this API service as well. He also implemented code to access this "Cloud Translation API." This code's return value was a dictionary that is rendered in tabular form on a different HTML page. This API can detect and translate over 100 APIs.

Raheyma generated the front-end webpage for the user interface. The user interface used python with flask and HTML. It provided endpoints for the backend developers to build upon and interpret results in a presentable way.

All the pieces of code have been pushed into the GitHub repository. Akansha leads the code integration, so all the code pieces worked together as one cohesive build. Since Google already provides us, pre-trained models, we only had to collect test data. Mahima collected some test samples, i.e., images with different languages and orientation. Mahima performed an accuracy analysis of our software through rigorous testing and validation.

Abstract

The goal of the project is to provide seamless language translation from an image. This project product offers solutions to travelers who travel in different countries but do not necessarily speak the region's language. Although the significant users who benefitted from these would-be travelers, other users can take advantage of it. Some examples of such users would be a child having difficulty in recognizing the words and pronouncing them. Our solution detects texts in different languages, from images of signboards, pamphlets, hoardings, etc., and translates them to the language of the user’s choice. The user interface for this project will be served as a web application. Ideally, the user can upload images and get translated texts and speech as a result. The report below illustrates how we achieve this by leveraging the google cloud APIs and has details on our project's design.

Data Specification

To make our application benefit people. We must have excellent test results. For Testing the application data play an essential role. As our application focuses on providing benefits to many people with quick and straightforward ways to translate image data to speech and text, the data would be images. We are using a subset of the dataset which is publicly available by the Lionbridge-ai dataset. We also downloaded some images directly from google to check the efficiency of our project.

Image to text detection is a supervised learning and classification problem. However, any image detection model trains using convolutional neural networks on different feature layers. Like any deep learning problem, it isn't easy to understand what features the model learns to train from datasets. One important aspect of collecting data is, it should have a variety of images in it. For us, the data is an image that could be of different angles, positioning, sizes, lighting, etc., so the model properly learns.

Our application subscribes to Google’s API that uses already pre-trained, reliable, and highly accurate Machine Learning models. Thus, our model itself did not need any additional model training and did not require collecting any dataset. We were only required to collect the test dataset for validation. The test data set contains images with text of different languages, positioning, and sizes.

Project Design

Our project is a web application developed using python with flask. The project design can be divided into three sub-categories:

i. User Interface

For our application, we have tried to keep our user interface aesthetically pleasing and easy to navigate for the common user. It has been developed using Flask and HTML templates. The image below is a screenshot of the web page.

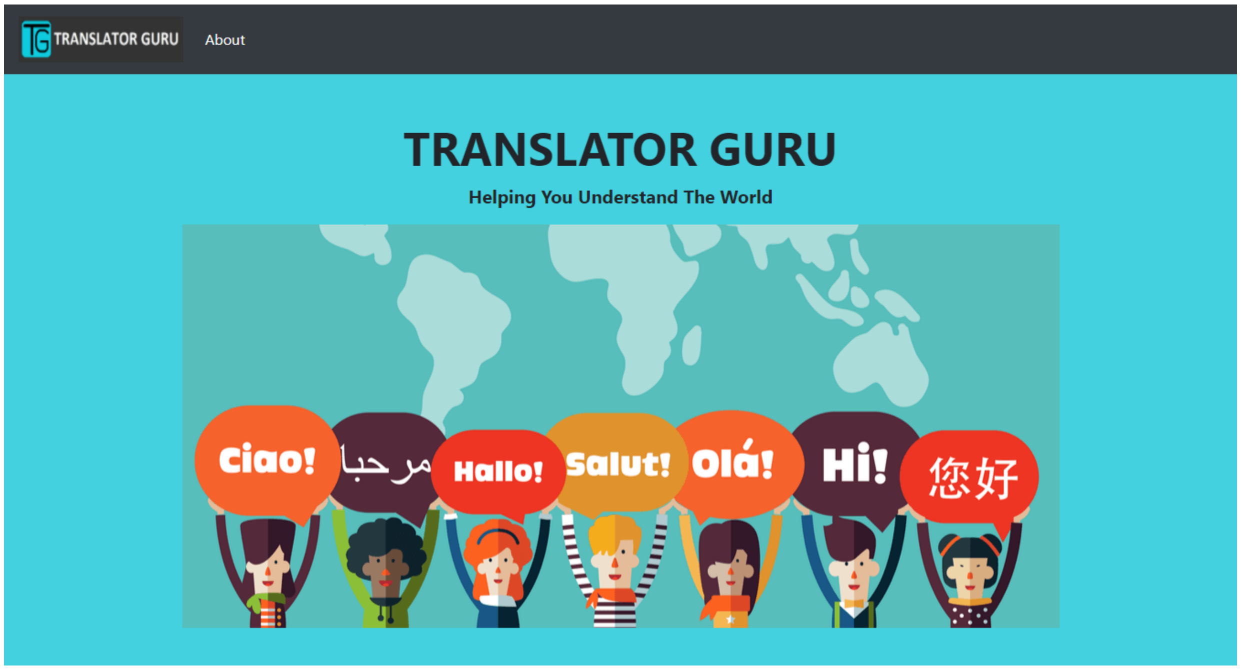
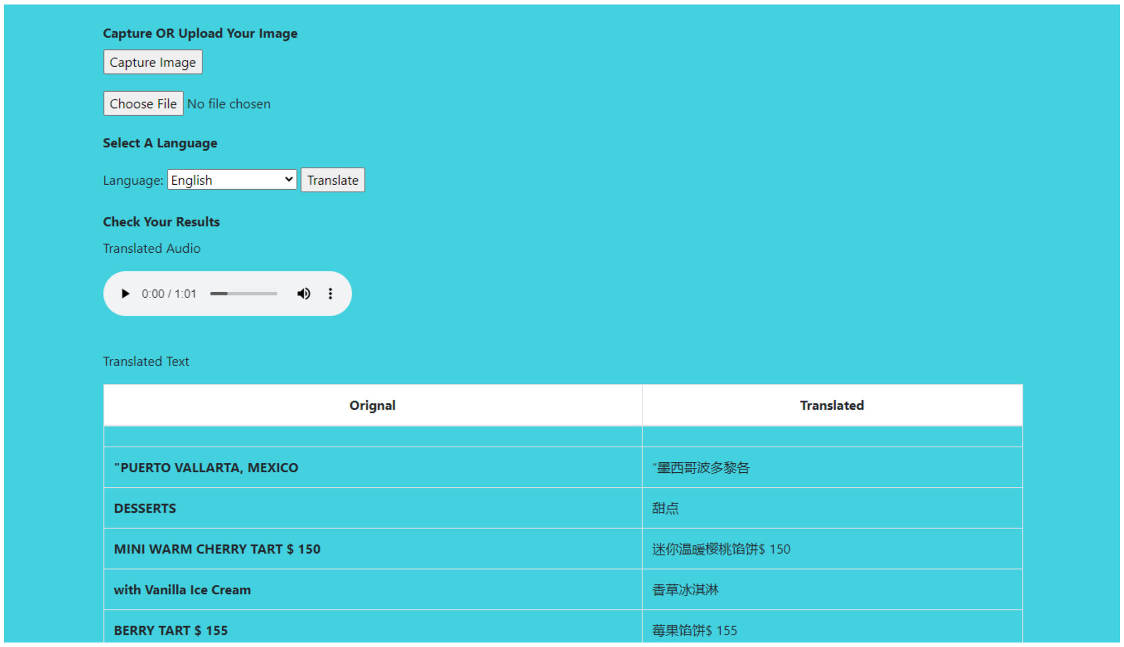


Figure 1 Web Interface 1



Capture OR Upload Your Image

Capture Image

Choose File No file chosen

Select A Language

Language: English Translate

Check Your Results

Translated Audio

0:00 / 1:01

Translated Text

Orignal	Translated
"PUERTO VALLARTA, MEXICO	"墨西哥波多黎各
DESSERTS	甜点
MINI WARM CHERRY TART $ 150	迷你温暖樱桃馅饼$ 150
with Vanilla Ice Cream	香草冰淇淋
BERRY TART $ 155	莓果馅饼$ 155

Figure 2 Web Interface 2

The navigation bar at the top has been made by bootstrapping CSS. Clicking on the logo redirects the user to the main page of the app. Using HTML template, we set up and positioned our application name, ‘TRANSLATOR GURU’; our slogan, ‘Helping You Understand the World’; and an image. Following that, we provided the user a step-by-step way to generate their image's translated audio and text.

First, the user either captures or uploads an image of the text they want to be translated. To capture the image, the user has to click on the ‘Capture Image’ button, redirecting them to the following webpage.

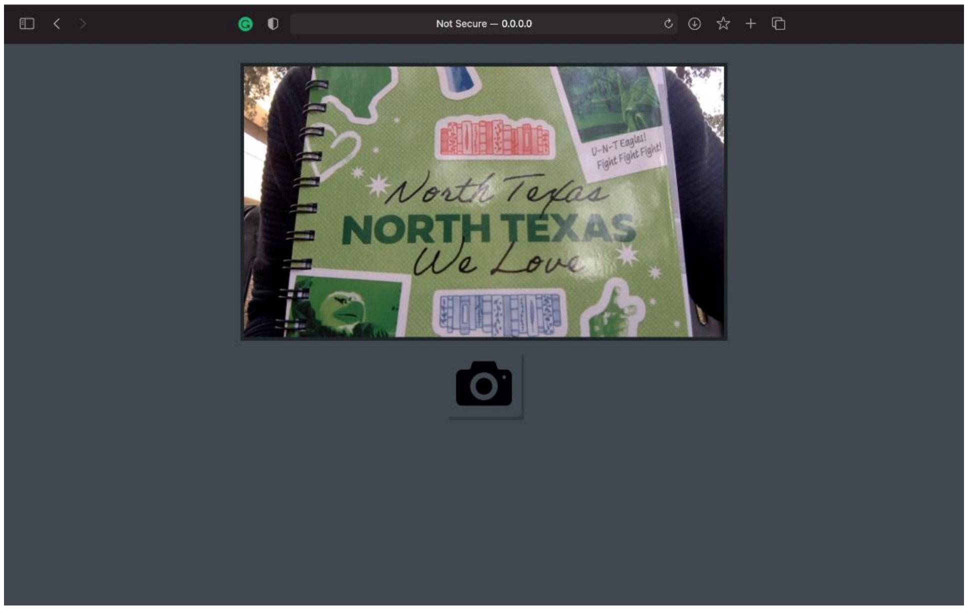


Figure 3 Image Capture with WebCam

The box shows the video from the front camera of the user’s computer. The user can capture an image by clicking on the camera icon. After the image has been captured, the user is redirected to the main page of the app. To upload an image instead, the user can click on the ‘Choose File’ button to browse and select an image from their computer. When the image is uploaded, its name appears on the right side of the button. Both these buttons have been set using HTML templates.

The image captured or uploaded must be stored in a folder on the computer to be accessed for translation and conversion to text and speech. For this, we used Flask to create an upload folder and specified it only to contain ‘png,’ ‘jpg,’ ‘jpeg’ or ‘gif’ files.

Then we ask the user to select the language they want to convert the text into using a drop-down menu. This menu was set up using HTML and has eight different language options. After selecting the desired language, the user clicks the ‘Translate’ button, sending the language choice to our flask code. In the flask, we take the captured/uploaded image saved in the upload folder, convert it to the desired language selected using the drop-down menu, and output it as translated audio and text.

For displaying our translated audio and translated text results, we again used HTML. The translated audio shows with audio controls to play, pause, etc. The translated text is displayed as a table with each row containing a text line from the image. The first column shows the original text, and corresponding rows in the second column show their translations.

ii. Image to Speech

For getting text from image, we have used “google Vision API text detection” (Optical Character Recognition). We are able to get the text from image. We are using google-cloud-vision library for text detection. After that, With the help of google API text to speech we are able to translate the text into speech. We are using google-cloud-texttospeech library for text to speech translation.

iii. Language Translation

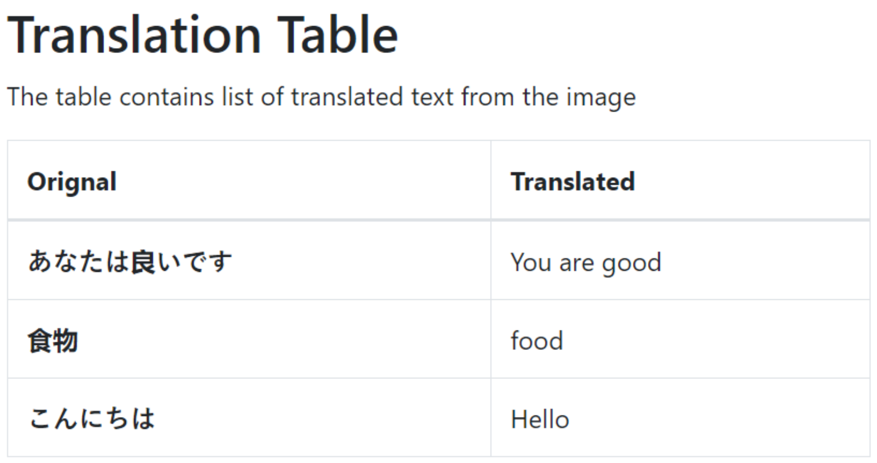
The application facilitates this using another Google API called “Cloud Translation.” After the image has been uploaded, the detected text is ready for translation. Users can initiate translation using the “Translate” option in the web interface. This will send a post request that will call the translate function. The application also provides an option of selecting the translation language for their uploaded image. If no option is selected, the text will be translated into English by default. Even though the API can support 109 language translations, we restrict the user to translate the detected text into only eight languages to limit this project's scope. The eight languages are:

1. English
2. Spanish
3. French
4. Russian
5. Hindi
6. Chinese
7. Korean
8. Japanese

Note that the application can still detect all the supported 109 languages. We only limit the language option that users can choose to translate the image text.

The translation function takes” target language” and “list of text” as input. A target language is received from user selection, and the list of text is generated from an image uploaded using “Cloud Vision API.” The translation function uses “Cloud Translation API” to translate the text into a specified target language. This API returns detected language and translate text for each request. The function separates the translated text and creates a dictionary where the original text is the key, and translated text is the value. This dictionary is returned to the caller as the translate function execution completes.

The dictionary is rendered in the HTML page in tabular form for better interpretation.



Translation Table

The table contains list of translated text from the image

Orignal	Translated
あなたは良いです	You are good
食物	food
こんにちは	Hello

Figure 4 Text Translation

iv. Testing and Validation

Our application uses Google's API modules that use already pre-trained, reliable, and highly accurate Machine Learning models. Thus, our model does not require any additional model training and so any training datasets. We were only required to collect the test dataset for validation. The test data set contains images with text of different languages, positioning, and sizes.

For validating our model, we have done our testing manually with 16 different images. Some of the images are extracted from lionbridge OCR datasets, containing images taken by pocket camera in different environments. The size and positioning of the test in these images are different. The text in the images is in Chinese and English. To collect images of other foreign languages, we have used google images and downloaded images with French, Italian, Arabic, Spanish, Hindi, and Thai text in it. Since our model can detect 109 languages and convert these texts in the image to eight different languages, we validated our application with these eight different languages.

As said earlier, we tested our application manually. The steps which we followed for testing are as follows:

- After collecting different text images, we first tested those images whose text we can understand, like English and Hindi.
- We uploaded an image to our application and selected the translation language like English, Spanish, French, etc.
- After the selection of the translation language, we hit the translate button, which translated our text.
- Then we review the results with our knowledge of the language, whether the text in the image translated correctly.
- Some images are not in a language we understand. To test those images, we translated the text in google translation and Google Lens to verify our results.

As we said, we have included images of different languages. We validate those images by checking the result of the same images in Google Lens mobile application for the images whose text we do not understand. We receive fascinating results. One example is in the below image; Google Lens was unable to detect whole Chinese text, while our application detects each word of image text and detects the translation.

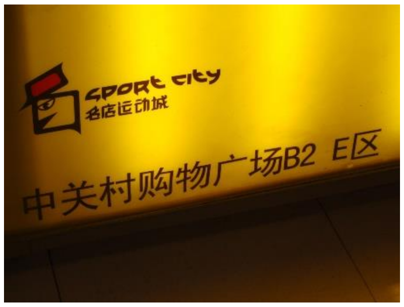
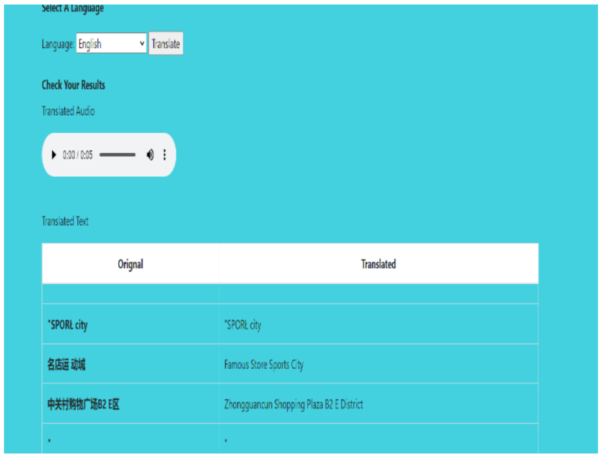
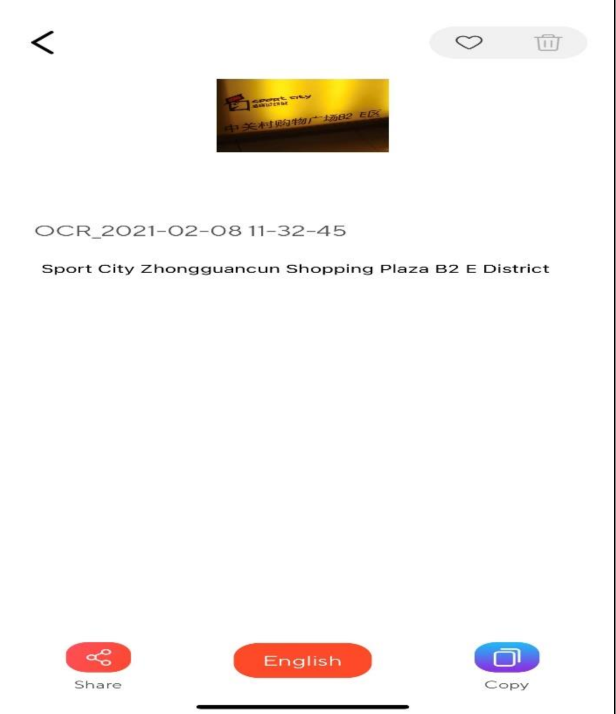


Figure 5 Test Image

Result with our Application	Result with google lens
	

Select A Language

Language: English Translate

Check Your Results

Translated Audio

0:00 / 0:05

Translated Text

Orignal	Translated
"SPORŁ city	"SPORŁ city
名店运 动城	Famous Store Sports City
中关村购物广场B2 E区	Zhongguancun Shopping Plaza B2 E District
.	.

OCR_2021-02-08 11-32-45

Sport City Zhongguancun Shopping Plaza B2 E District

There are some other exciting results we receive. One of the image texts is not translated by either of the application.

Since we did our testing manually, we have calculated accuracy manually considering how many images our application is successfully translating. We have considered the fact that if the text in the image is translated entirely or partially. Another calculation we took into consideration was how many out of 16 images we can translate successfully. We have achieved an accuracy of 93% for our application, which we claim as good accuracy.

The test document having resulted from our application can be found in below word document.

Flowchart

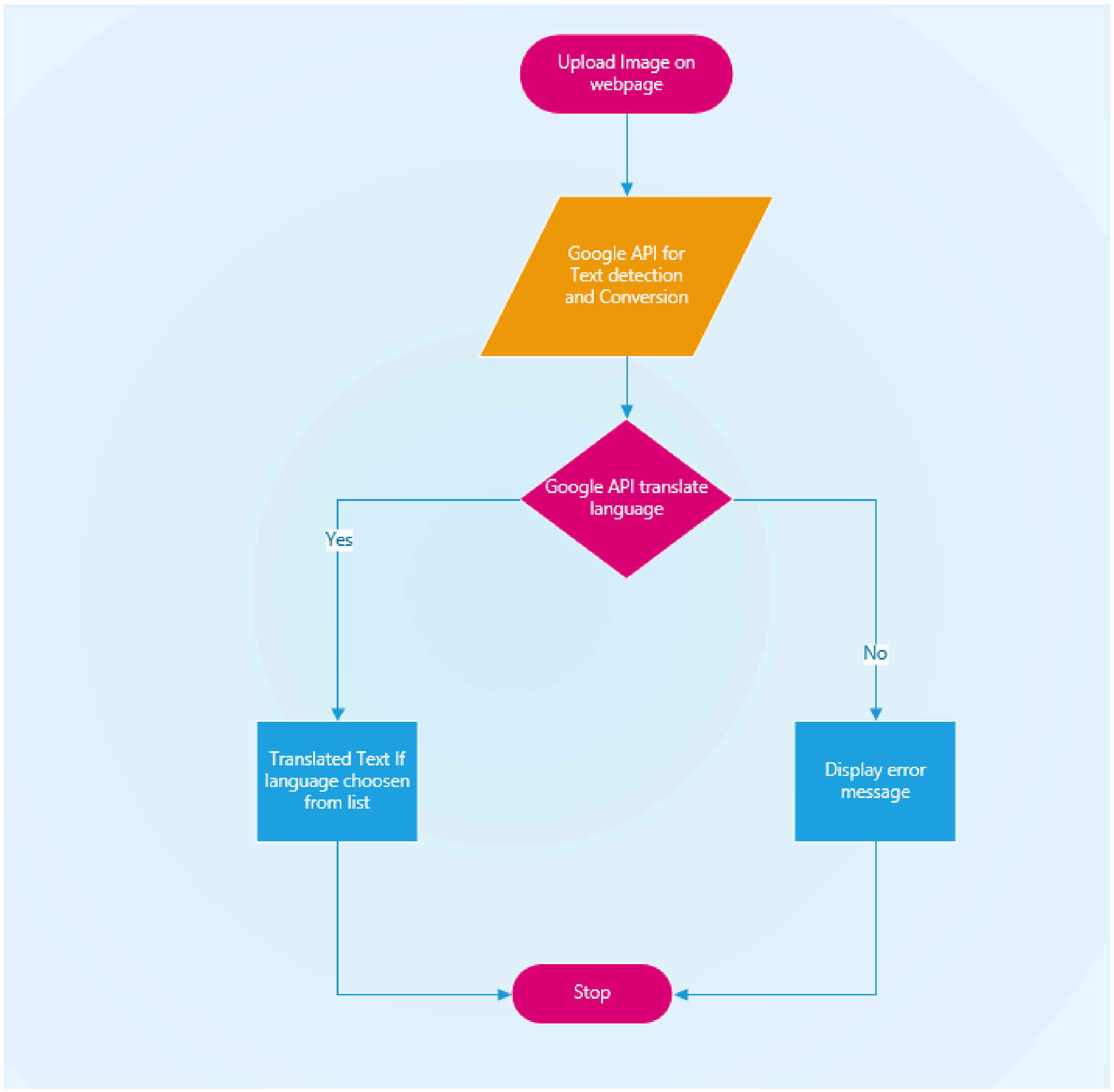

Upload Image on webpage
Google API for Text detection and Conversion
Google API translate language
Yes
No
Translated Text If language choosen from list
Display error message
Stop

Figure 6 Project Flowchart

Project Milestones

Dates	Task
1/24/2021	Set-up Git, Cloud Vision API on GCP, the base for Flask web application
1/31/2021	Build working web application to input images and print confirmation of upload
02/07/2021	Integrate Google API with web application
2/11/2021	Validate the output and improve performance

Repository

https://github.com/Jagdish05/ImageToSpeech

Code

Flask main python file (Application_code.py)

```
from flask import Flask, render_template,request, redirect, url_for, Response, flash
import imutils
import json
import matplotlib.pyplot as plt
import numpy as np
import os
import pandas as pd
import requests
import time
from base64 import b64encode
from IPython.display import Image
from pylab import rcParams
rcParams['figure.figsize'] = 10, 20
import sqlite3
from google.cloud import vision,translate_v2 as translate
import six
import io
from camera import Camera
import socket
camera = None
from cv2 import *

os.environ['GOOGLE_APPLICATION_CREDENTIALS'] = r"sixth-well-302300-cc8ce0454489.json"
app = Flask(__name__)
image_path = "static/captures/"

cam = VideoCapture(0)

##============================ Extract text from image==================
def convertor_image_to_text(filename):
	client = vision.ImageAnnotatorClient()
```

```
        file_name = os.path.abspath(image_path+filename)
        with io.open(file_name, 'rb') as image_file:
            content = image_file.read()

        image = vision.Image(content=content)

        response = client.text_detection(image=image)
        texts = response.text_annotations
        if response.error.message:
            raise Exception(
                '{}\nFor more info on error messages, check: '
                'https://cloud.google.com/apis/design/errors'.format(
                    response.error.message))
        if print(len(texts))==0:
            return redirect(url_for('index'))
        return '\n"{}"'.format(texts[0].description)

#-------------------------------------------------------------------------
def translate_text(target, Listoftext):
    import six
    from google.cloud import translate_v2 as translate
    d = dict()
    translate_client = translate.Client()
    for text in Listoftext:
        if isinstance(text, six.binary_type):
            text = text.decode("utf-8")
        result = translate_client.translate(text, target_language=target)
        temp = format(result["translatedText"])
        d[text]= temp
    return d

def convertor_text_to_speech(translated_text,language):
        from google.cloud import texttospeech
        client = texttospeech.TextToSpeechClient()
        synthesis_input = texttospeech.SynthesisInput(text=translated_text)
        voice = texttospeech.VoiceSelectionParams(
            language_code=language, ssml_gender=texttospeech.SsmlVoiceGender.NEUTRAL
        )
        audio_config = texttospeech.AudioConfig(
            audio_encoding=texttospeech.AudioEncoding.MP3
        )
        response = client.synthesize_speech(
            input=synthesis_input, voice=voice, audio_config=audio_config
        )
        try:
            os.remove("static/output.mp3")
        except OSError:
            pass
        with open("static/output.mp3", "wb") as out:
            out.write(response.audio_content)
            print('Audio content written to file "output.mp3"')

from werkzeug.utils import secure_filename

UPLOAD_FOLDER = image_path
ALLOWED_EXTENSIONS = {'png', 'jpg', 'jpeg', 'gif'}
```

```
app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

def allowed_file(filename):
    return '.' in filename and \
           filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

@app.route('/index', methods=('GET', 'POST'))
def index():
    try:
        os.remove("static/output.mp3")
    except OSError:
        pass
    result = "Translated_Text"
    translated_result  = dict()
    print("request",request)
    if request.method == 'POST':
        filename='Image.jpg'
        print("inside post request")
        file = request.files['file']
        if file and allowed_file(file.filename):
            filename = secure_filename(file.filename)
            file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))

        print(filename)
        print("checking for tanslation button",request.form)
                language = request.form["language"]
        print(language)
        result = convertor_image_to_text(filename)
        list_of_text = result.split('\n')
        print(result)
        translated_result = translate_text(language,list_of_text)
        translated_text = translated_result.values()
        translated_final_text = ''.join(translated_text)
        print("translated_final_text",translated_final_text)
        convertor_text_to_speech(translated_final_text,language)
    return render_template('index.html', content1 = translated_result, text = translated_result)


##===========================Capture image from camera ==================
@app.route('/video/')
def video():
    return render_template('video.html')

def gen(camera):
while True:
s, img = cam.read()
height, width, layers = img.shape
new_h = int(height / 2)
new_w = int(width / 2)
img = cv2.resize(img, (new_w, new_h))
ret, jpeg = cv2.imencode('.jpg', img)
frame=jpeg.tobytes()
yield (b'--frame\r\n'
b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n')
```

```
@app.route('/video_feed/')
def video_feed():
    capture()
    return Response(gen(camera),
        mimetype='multipart/x-mixed-replace; boundary=frame')

@app.route('/capture/')
def capture():
    s, img = cam.read()
    if s:imwrite("static/captures/Image.jpg",img)
    return redirect(url_for('index'))

@app.after_request
def add_header(r):
    """
    Add headers to both force latest IE rendering engine or Chrome Frame,
    and also to cache the rendered page for 10 minutes.
    """
    r.headers["Cache-Control"] = "no-cache, no-store, must-revalidate"
    r.headers["Pragma"] = "no-cache"
    r.headers["Expires"] = "0"
    r.headers['Cache-Control'] = 'public, max-age=0'
    return r

if __name__ == '__main__':
    app.secret_key = 'super secret key'
    app.config['SESSION_TYPE'] = 'filesystem'

    app.run(host="0.0.0.0",port="8000",debug=True)
```

Flask python file for video view (video.py)

```
import cv2 as cv
from time import localtime, strftime

class Camera(object):
    CAPTURES_DIR = "static/captures/"
    RESIZE_RATIO = 1.0

    def __init__(self):
        self.video = cv.VideoCapture(0)

    def __del__(self):
        self.video.release()

    def get_frame(self):
        success, frame = self.video.read()
        if not success:
            return None

        if (Camera.RESIZE_RATIO != 1):
            frame = cv.resize(frame, None, fx=Camera.RESIZE_RATIO, \
                fy=Camera.RESIZE_RATIO)
        return frame
    def get_feed(self):
        frame = self.get_frame()
        if frame is not None:
            ret, jpeg = cv.imencode('.jpg', frame)
            return jpeg.tobytes()

    def capture(self):
        frame = self.get_frame()
        timestamp = strftime("%d-%m-%Y-%Hh%Mm%Ss", localtime())
        filename = Camera.CAPTURES_DIR + timestamp +".jpg"
        if not cv.imwrite(filename, frame):
            raise RuntimeError("Unable to capture image "+timestamp)
        return timestamp
```

Flask HTML main file (index.html)

```
{% extends 'base.html' %}

{% block content %}
<h1 style="text-align:center; font-size:50px;"> <strong>TRANSLATOR GURU</strong> </h1>
<p style="text-align:center; font-size:20px;"> <strong>Helping You Understand The World</strong> </p>

<script>
function myFunction() {
  document.getElementById("adi").style.display = "block";
}
</script>

<center>
  <img style="padding-bottom: 50px;" src="static/images/translation.png" alt="Translation" />
</center>

<div class="form-group" method="GET">
    <label> <strong>Capture OR Upload Your Image</strong> </label>
    <form style="padding-bottom: 20px;" action="{{url_for ('video') }}" method="GET">
        <input type="submit" id=capture  value="Capture Image" action="{{url_for ('video') }}">
    </form>
    <form enctype=multipart/form-data style="padding-bottom: 20px;" action="{{url_for ('index') }}" method=post>
        <input type=file name=file>
        <!--input type=submit id=upload value=Upload-->
        <br>

<label style="padding-top: 20px;"> <strong>Select A Language</strong> </label>
<br>
    <label style="padding-bottom: 10px; padding-top: 10px" method=post>
        <label for="language">Language:</label>
        <select id="language" name="language">
            <option value="en">English</option>
            <option value="zh">Chinese(Mandarin)</option>
            <option value="es">Spanish</option>
            <option value="fr">French</option>
            <option value="ru">Russian</option>
            <option value="hi">Hindi</option>
            <option value="ko">Korean</option>
            <option value="ja">Japanese</option>
        </select>
    <input type="submit" id=translate name="Start_translate" value=Translate>
    </label>
<br>
<label> <strong>Check Your Results</strong> </label>
    <p> Translated Audio </p>
    <div>
<audio id = "adi" display = "none" controls>
    <source src="{{url_for('static', filename='output.mp3')}}" type = "audio/mpeg"></source>
</audio>
    </div>
</form>
</div>

<div class="form-group" method="GET">

</div>

<form value=Translate action="{{url_for ('index') }}" method=post>
    <p> Translated Text </p>
</form>


<table class="table table-bordered">
<thead>
  <tr>
    <th style="background-color:#FFFFFF; text-align:center;" scope="col">Orignal</th>
    <th style="background-color:#FFFFFF; text-align:center;" scope="col">Translated</th>
  </tr>
</thead>
<tbody>
  {% for key, value in text.items() %}
    <tr>
      <th> {{ key }} </th>
      <td> {{ value }} </td>
    </tr>
      {% endfor %}
</tbody>
</table>
{% endblock %}
```

Flask HTML file for web page frame (base.html)

```
<!doctype html>
<html lang="en">
  <head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

    <!-- Bootstrap CSS -->
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">

    <title>{% block title %} Translator Guru {% endblock %}</title>
  </head>
  <body style="background-image: url('static/images/background1.jpg');">
    <nav class="navbar navbar-expand-md navbar-dark bg-dark">
        <a class="navbar-brand" href="{{ url_for('index')}}">
    <img src="static/images/translator_logo.png" width="180" height="50" alt="">
</a>
        <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNav" aria-
controls="navbarNav" aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
        </button>
        <div class="collapse navbar-collapse" id="navbarNav">
            <ul class="navbar-nav">
            <li class="nav-item active">
                <a class="nav-link" href="#">About</a>
            </li>
            </ul>
        </div>
    </nav>
    <div class="container" style="position:relative; top:50px;">
        {% block content %} {% endblock %}
    </div>

    <!-- Optional JavaScript -->
    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-
q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo"
crossorigin="anonymous"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js" integrity="sha384-
UO2eT0CpHqdSJQ6hJty5KVphtPhzWj9WO1clHTMGa3JDZwrnQq4sF86dIHNDz0W1"
crossorigin="anonymous"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js" integrity="sha384-
JjSmVgyd0p3pXB1rRibZUAYoIIy6OrQ6VrjIEaFf/nJGzIxFDsf4x0xIM+B07jRM"
crossorigin="anonymous"></script>
  </body>
</html>
```

Flask HTML file for video view (video.html)

```
<html>
  <head>
            <link rel="stylesheet" type="text/css" href="{{ url_for('static',filename='style.css') }}">
    <title>Camera App</title>

    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css"
integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T"
crossorigin="anonymous">
  </head>

  <body>
    <center>
      <div id="camera">
        <img src="{{ url_for('video_feed') }}" alt="video feed">
      </div>
      <div id="btns-div">
```

```
        <a href="{{ url_for('capture') }}">
          <img src="{{ url_for('static',filename='picture.png') }}" alt="take picture" class="btn">
        </a>
      </div>
    </center>
  </body>
</html>
```

Resources and Related Projects

1. https://code.tutsplus.com/tutorials/creating-a-web-app-from-scratch-using-python-flask-and-mysql--cms-22972
2. https://www.digitalocean.com/community/tutorials/how-to-make-a-web-application-using-flask-in-python-3#step-5-%E2%80%94-displaying-all-posts
3. https://googleapis.dev/python/vision/latest/index.html
4. https://cloud.google.com/text-to-speech
5. https://lionbridge.ai/datasets/15-best-ocr-handwriting-datasets/
6. https://g.co/cloud/translate/v2/translate-reference#supported_languages
7. https://flask.palletsprojects.com/en/1.1.x/patterns/fileuploads/
8. Google Lens application for validation