Punjab Engineering College (Deemed to be University), Chandigarh

Image to Text to Speech Conversion

Mentor: Dr. Sanjeev Sofat

Harshit Kumar (20103011)

Sanil Gupta (20103018)

Abhitash Singh (20103019)

Suyash Rajvanshi (20103053)

Acknowledgement

We would like to thank Dr. Sanjeev Sofat, (Professor Computer Science department) who guided and helped us throughout the project. It was because of Sofat sir that we made this project successfully. We would also like to thank Punjab Engineering College (PEC) for giving us the opportunity to work on this project.

We sincerely would like to thank all the people that believed in us and made possible this period of professional and personal growth. First, our parents who supported us during difficulties. Even if they do not have a clear idea of what we are doing, they were always excited to share with us satisfactions and great results

We would also want to thank Punjab Engineering College, and in particular, the Computer Science department, that offers an extremely high level of teaching and that brings willing people to a successful carrier.

We would also like to thank all the group members who did all the work through day and night to successfully complete the project. We hope that this wonderful friendship, in and outside the lab, will last forever.

Lastly, we want to thank everyone that appreciated or criticised our work. The process of research needs your support to go ahead and to get always new incitement.

DECLARATION

We hereby declare that the project work entitled "*Image to Text to Speech Conversion*" is an authentic record of our work carried out as a requirement Minor project in the 5th semester of our degree of B-Tech (Computer Science), Punjab Engineering College under the guidance of Dr. Sanjeev Sofat.

Harshit Kumar (20103011)

Sanil Gupta (20103018)

Abhitash Singh (20103019)

Suyash Rajvanshi (20103053)

Date: 19/12/2022

Certified that the above statement made by the students is correct to the best of our knowledge and belief.

Dr. Sanjeev Sofat
Computer Science

Punjab Engineering College

(Deemed to be university)

Abstract

Often, images of documents are shared across the internet without sharing the original source of the document itself. While there are a variety of speech synthesizers that can work with word documents etc., there are none that can work with images. This presents a problem to blind people who often rely on speech synthesis to work on a computer. Moreover, even for people with normal vision, reading from computer screens for prolonged periods can lead to eye strain, migraines, etc.

Therefore our project will not only convert the text in images to editable text, it will also directly output it as a voiceover similar to an audiobook. This has potential applications in cataloguing as well, as books can be scanned and then digitized with text that is searchable using RegEx, word matching etc.

The project that we plan to create is an Image to Speech using OCR. There are a variety of tools that can convert text to speech, however there aren't any tools that can convert an image into speech directly. This creates problems for blind people, who often have trouble navigating menus on a computer and thus there is a need for a tool that can work seamlessly. The field of our project primarily lies in OCR and image processing, since most OCR tools require images to be relatively noise free to work properly. At the same time, phone cameras cannot produce noise-free images, hence our project will deal primarily with cleaning up these images and working on them.

Index

Topic **Page number**

CHAPTER 1
INTRODUCTION

Overview

This chapter gives brief insight into various non-theoretical aspects of the project. This chapter will help to understand why we need this project and how will it serve the humanity or humanity for betterment. It will also give insights about what we are going to do in this project and what all things motivated us to do so.

1.1 Motivation

During any types of examinations, we often have to refer to many books to read about a single topic or subject. While the physical copies of some books may be available, most books are not available easily. The books that are available often don’t have a good quality, or may be otherwise damaged. Therefore, most of the times, digital copies of books are required, especially when only a chapter or two out of a particular book is required.

Digital copies present their own problems such as being created from photos of the original book, which makes it incredibly hard to search books by searching for keywords. Such copies are also sometimes hard to read off.

Prolonged exposure to computer screens also has harmful effects on the eyes. In the field of computer engineering, a lot of time is spent looking at computer screens, therefore any time that can be reduced can do wonders for the health of a person.

Hence, converting these images to speech can take the load of eyes of a person, and at the same help those whose vision is impaired.

1.2 Need for Project

Currently Optical Character Recognition (OCR) technology can convert text in an image to editable text, however most OCR suffer a drawback which is that they are very susceptible to noise. Grainy images, such as those taken by a smartphone can lead to completely wrong and unexpected results.

Therefore, images need to be processed before they can be input in the OCR. Further, no OCR comes with a speech generator in it, therefore there is need to make an integrated solution that can not only process images, but also convert them to text, and then further convert the text into speech.

Computer vision techniques can be used to process images and reduce the noise in these images. Further, 100% noise removal is not needed as OCR can tolerate small amounts of noise, and the human brain has an innate capacity to understand misspelt text by the context.

1.3 Objective

The main objective of this project will be to create a tool that can de-noise images, convert them to text, and finally convert text to speech. The focus will be to eliminate false positives, i.e., OCR recognising text where there is just noise.

Chapter 2
Software Used

Overview

This section provides all the technology that we have used to create this project along with rationale behind using them.

2.1 Python -

Fig 2.1 python logo

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

Python has been used to process images, and to create the backend server of our webapp.

The reason for choosing python was because python comes with a suite of libraries that are well suited to image processing and server side processing. Further, our OCR library also has python API, therefore all processing can be done in a single language.

2.2 OpenCV -

Fig 2.2 open cv logo

OpenCV is a great tool for **image processing and performing computer vision tasks**. It is an open-source library that can be used to perform tasks like face detection, objection tracking, landmark detection, and much more. It supports multiple languages including python, java C++.

OpenCV's application areas include:

1. 2D and 3D feature toolkits
2. Egomotion estimation
3. Facial recognition system
4. Gesture recognition
5. Human-computer interaction (HCI)
6. Image Processing
7. Motion understanding
8. Object identification

We have used OpenCV to run all our image processing. The reason for using OpenCV instead of writing our own processing code was that OpenCV can harness the GPU of a

machine if available, thus making image processing much faster. This allowed us to reduce our processing times. Moreover, images output by OpenCV can directly be fed into the OCR, making it easier to work with.

2.3 Tesseract –

Tesseract is an **optical character recognition engine** for various operating systems. It is open source, released under Apache License. Its development has been sponsored by Google since 2006, and it is one the best OCR that is open source and freely available. It has its API available in several languages such as C++, Java, Python.

Fig 2.3 Tesseract logo

Its features include:

1. OCR using LSTM neural network.
2. Several modes of operations.
3. Recognising text from blocks of text, sparse text etc.

We have used Tesseract as it is among the best OCR engines that is free and open source. Further, using neural networks, the accuracy on properly de-noised images is very good. It also has bindings available in many languages including python making it a good choice for our project.

2.4 gTTs –

gTTS (*Google Text-to-Speech*), is a Python library and CLI tool to **interface with Google Translate's text-to-speech API**. Writes spoken mp3 data to a file, a file-like object (bytestring) for further audio manipulation, or stdout. It features flexible pre-processing and tokenizing.

Fig 2.4 gTTs Logo

gTTS has several features:

1. It can write to a file, as a string of bytes, or directly to output.
2. It can be configured to set the speech.
3. It can have a male or female voice.
4. It can also provide localised accents.

We have used gTTS to generate speech from text. The reason for using gTTS is because of its feature rich speech generation. Moreover, gTTS does some preliminary text analysis to understand how to pronounce each word in the given text, making it far better than other text-to-speech engines.

Chapter 3

Working of The Project

Overview

The image received is first converted to grayscale since the OCR does not require colour to analyse text and works better on black & white images.
De-noising methods are then applied to image in order to remove any grains in the image. To denoise a combination of blurring, and edge detection is used to remove most of the noise.
Finally, dynamic thresholding is done on the image which removes any remaining noise and converts it into a pure black and white image.

The processed image is then fed into the OCR and the text is fed into Text-to-Speech, all of which is sent back to the user.

3.1 Conversion to grayscale

Image that is received is first converted to grayscale, this is done to remove the effects of coloured text on the OCR. Further, gaussian noise, which is noise that is in the form of coloured dots is converted to salt-and-pepper noise which is far easier to deal with. Moreover, denoising techniques are far more effective on a single channel, rather than three channels.

OpenCV uses the formula

$$\mathrm{Y} = 0.299\,\mathrm{R} + 0.587\,\mathrm{G} + 0.114\,\mathrm{B}$$

to calculate the grayscale equivalent of a pixel. This formula is effective in retaining the luminosity as seen by the human eye, and

hence is more effective in preserving the image than simple averaging.

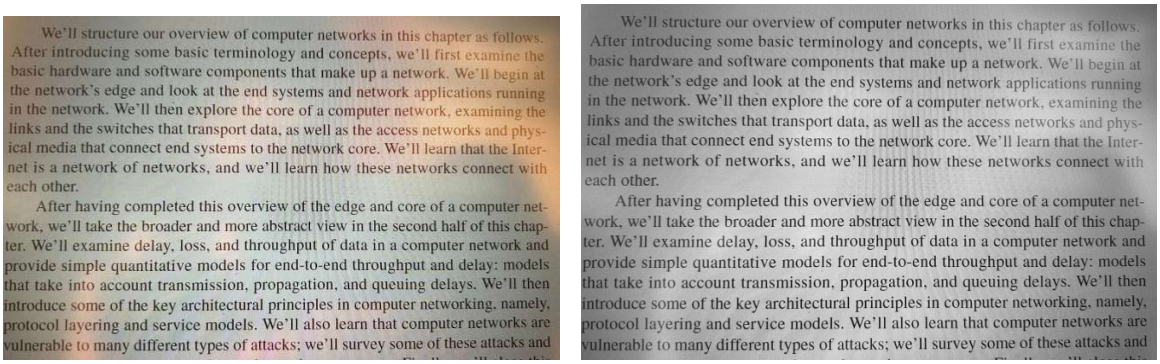


Fig 3.1 Result of changing color

3.2 Removing Noise from Image

Noise removal is the most important step in our entire process. It is essential to remove any noise in order to produce a good OCR output. We primarily use two steps to remove noise from our image.

First, the image is dilated and then eroded, this works as a sort of edge detection algorithm and the resultant image contains the outlines of edges of the image. We then divide this image from the original image, this helps to correct the color of our image. Since images often have non-uniform lighting, this steps helps to remove the effect of non-uniform lighting conditions on the image.

Next, the image is blurred using a Gaussian blur, which uses the kernel to blur.

$$\frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

The resultant blurred image is then added to added the original image using the formula:

$$Y = 0.7 * Original + 0.3 * Blurred$$

Where Y is the output.

The result is then median blurred which replaces the value of the current pixel by the median in a surrounding 3x3 kernel. This is a non-linear blurring method which is good for removing noise.
This image is then added to the original using the equation:

$$Y = 0.5 * Original + 0.5 * Blurred$$

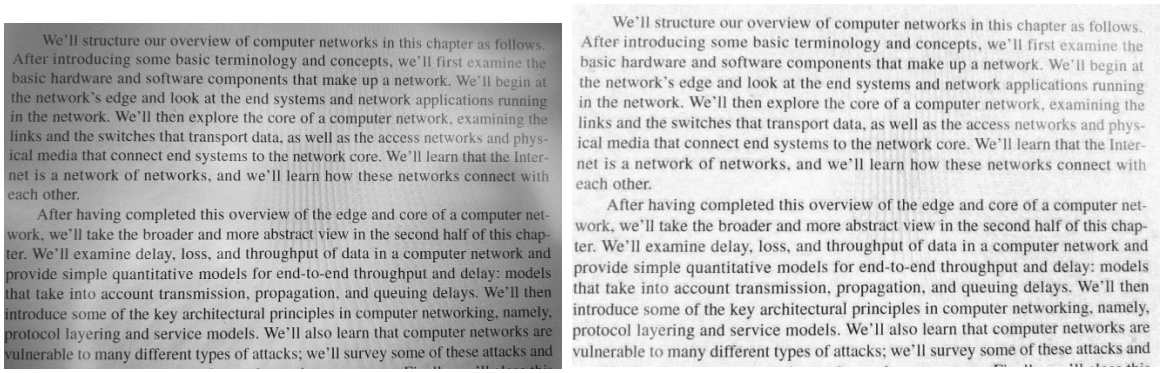


Fig 3.2 Result of de-noising

3.3 Thresholding the image

The image obtained after blurring twice is not very sharp, and lacks clarity. Moreover, it has a large number of grey colours which are not ideal for the OCR to work, therefore the image is thresholder to add some clarity back into the image, and remove the last remaining noise.

We have used Otsu thresholding which is a dynamic thresholding technique, meaning that it can automatically determine the threshold for an image, and threshold it. This allows us to automatically threshold an image based on its lighting conditions and the amount of noise present in it which can be quite different from one to the other image.

Otsu thresholding works by testing a bunch of values of threshold to minimise the intraclass variance among the foreground and background parts of the image, defined by the formula:

$$\sigma 2(t) = \omega bg(t)\sigma 2bg(t) + \omega fg(t)\sigma 2fg(t)$$

where $\omega_{bg}(t)$and $w_{fg}(t)$ represents the probability of number of pixels for each class at threshold t and σ2 represents the variance of colour values.

To understand what this probability means, Let,

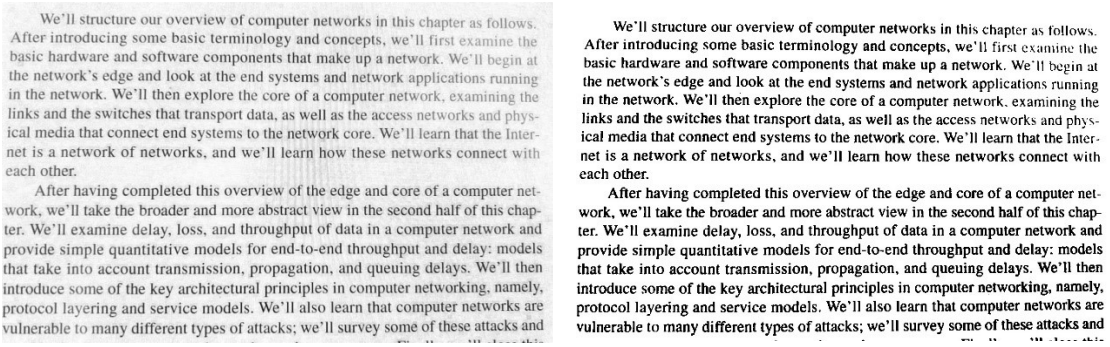
Pall be the total count of pixels in an image,
$P_{BG}(t)$ be the count of background pixels at threshold *t,*
$P_{FG}(t)$ be the count of foreground pixels at threshold *t*

So, the weights are given by,

$$\omega_{bg}(t) = P_{BG}(t)P_{all}$$

$$\omega_{fg}(t) = P_{FG}(t)P_{all}$$

Thresholding removes any residual noise in the image, and makes sure that all colour other than black and white is removed. This allows the OCR to perform a much better job with little chance of detecting false positives.


We'll structure our overview of computer networks in this chapter as follows. After introducing some basic terminology and concepts, we'll first examine the basic hardware and software components that make up a network. We'll begin at the network's edge and look at the end systems and network applications running in the network. We'll then explore the core of a computer network, examining the links and the switches that transport data, as well as the access networks and physical media that connect end systems to the network core. We'll learn that the Internet is a network of networks, and we'll learn how these networks connect with each other.

After having completed this overview of the edge and core of a computer network, we'll take the broader and more abstract view in the second half of this chapter. We'll examine delay, loss, and throughput of data in a computer network and provide simple quantitative models for end-to-end throughput and delay: models that take into account transmission, propagation, and queuing delays. We'll then introduce some of the key architectural principles in computer networking, namely, protocol layering and service models. We'll also learn that computer networks are vulnerable to many different types of attacks; we'll survey some of these attacks and

Fig 3.3 Final Processed Image

3.4 Input to OCR and gTTS

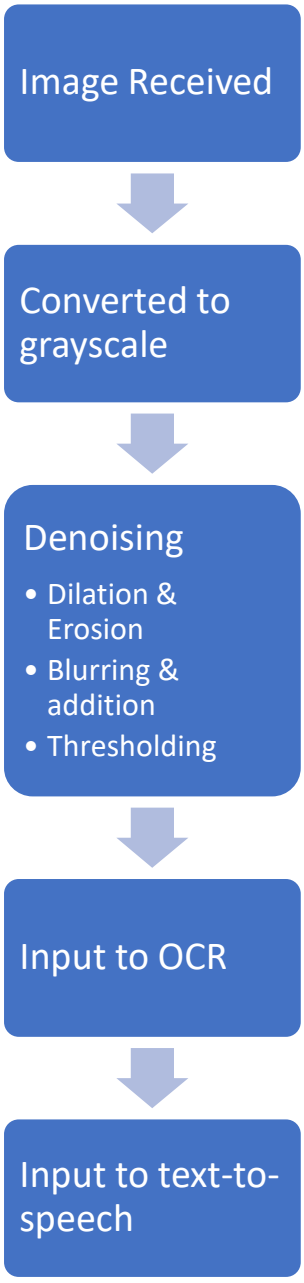
The image is then input to tesseract, which converts the image into text using a neural network. It uses the LSTM architecture to perform this conversion. The output for the given image is

We'll structure our overview of computer networks in this chapter as follows. After introducing some basic terminology and concepts, we'll first examine the basic hardware and software components that make up a network. We'll begin at the network's edge and look at the end

systems and network applications running in the network. We'll then explore the core of a computer network. examining the links and the switches that transport data…..

The speech converter, is input this data, which then outputs and Mp3 file which can be played on any computer.

3.5 Flow of algorithm



CHAPTER 4

Results And Observation

Overview

The following chapter concludes whatever is learned from this project and what were our observation in the project

4.1 Results

OCR tools are good at recognising most text when it comes to clear images, however de-noising images is a complex task that requires quite a lot of processing. The first thing that is important is to make the lighting across an image uniform. This can be done by dividing an image by its morphology, however in case the image is too dark or too bright, there is simply no way to correct the image while still preserving the text data. Hence this technique has its limits as it cannot pull data from nothing, there must be at least some data for it to work.

The actual process of de-noising itself is a careful balancing act, as it is not very difficult to over blur the image. While blurring is a good technique to remove noise, it also leads to the loss in information of the actual text data. Hence blurring must be done carefully, and the noised image should be applied to the original image to reduce the intensity of the noise in the image, while still preserving most of the text data.

Finally, thresholding is another important step in denoising the image, as blurring itself cannot completely remove noise from a picture. Blurring can only reduce the intensity of noise from an image, which is what we take advantage of when thresholding an image. Thresholding however should be completely dynamic, as

there is no chance that a single universal threshold can work for every single picture. In fact, some technique to locally threshold an image might work even better but has a chance to include some noise in areas with a lot of noise.

The actual OCR has an accuracy of about 70-90% depending on certain things such as text size, amount of noise, lighting conditions etc. These factors affect the quality of the OCR massively as the de-noising techniques produce different results for change in nearly every factor. Notably, varying lighting conditions seem to affect the least, while the size of text seems to affect it the highest.

4.2 Observations

The grayscale is the least sophisticated technique of all, hence has the least variance when it comes to the quality of the image produced.

Dividing the image to obtain a uniformly lit image also seems to work well, even for varying image conditions. However, it is quite sensitive to the size of the text in the image. For small text, it tends to darken the area around the text too much.

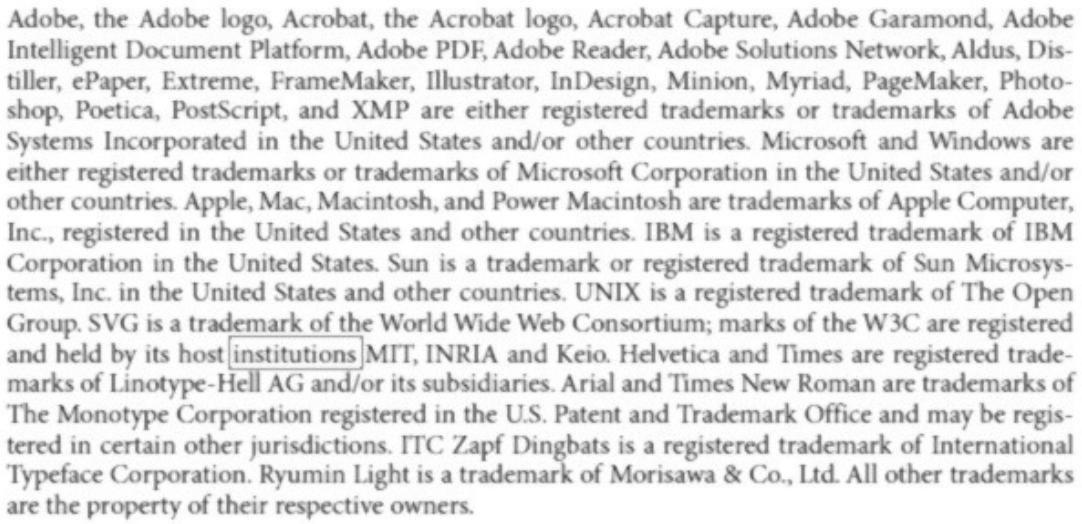

Adobe, the Adobe logo, Acrobat, the Acrobat logo, Acrobat Capture, Adobe Garamond, Adobe Intelligent Document Platform, Adobe PDF, Adobe Reader, Adobe Solutions Network, Aldus, Distiller, ePaper, Extreme, FrameMaker, Illustrator, InDesign, Minion, Myriad, PageMaker, Photoshop, Poetica, PostScript, and XMP are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Apple, Mac, Macintosh, and Power Macintosh are trademarks of Apple Computer, Inc., registered in the United States and other countries. IBM is a registered trademark of IBM Corporation in the United States. Sun is a trademark or registered trademark of Sun Microsystems, Inc. in the United States and other countries. UNIX is a registered trademark of The Open Group. SVG is a trademark of the World Wide Web Consortium; marks of the W3C are registered and held by its host institutions MIT, INRIA and Keio. Helvetica and Times are registered trademarks of Linotype-Hell AG and/or its subsidiaries. Arial and Times New Roman are trademarks of The Monotype Corporation registered in the U.S. Patent and Trademark Office and may be registered in certain other jurisdictions. ITC Zapf Dingbats is a registered trademark of International Typeface Corporation. Ryumin Light is a trademark of Morisawa & Co., Ltd. All other trademarks are the property of their respective owners.

Fig 4.1 Denoising image

Denoising is also a technique highly sensitive to the size of text. Both the median and gaussian blur use a kernel size of 3x3, therefore if the size of the text in image is smaller than 3 px, it can lead to blurry edges around the text which can affect the result of next steps. However, it is worth noting that most of the noise present in the image seems to go away by using this technique.

Thresholding is by far the most sophisticated technique used in the processing, and at the same time is highly susceptible to the size of the text.

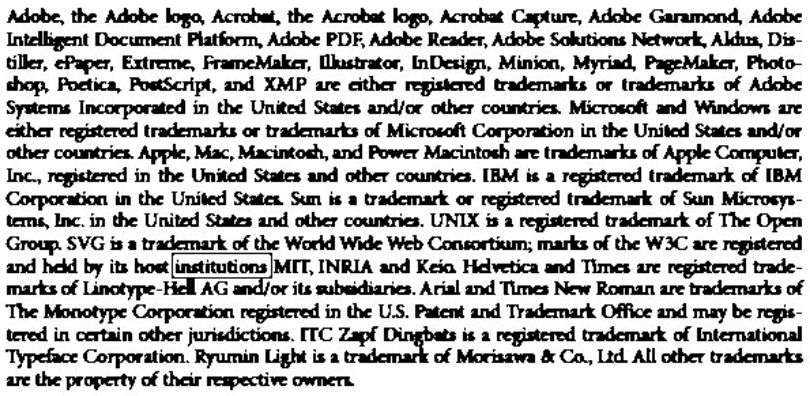

Adobe, the Adobe logo, Acrobat, the Acrobat logo, Acrobat Capture, Adobe Garamond, Adobe Intelligent Document Platform, Adobe PDF, Adobe Reader, Adobe Solutions Network, Aldus, Distiller, ePaper, Extreme, FrameMaker, Illustrator, InDesign, Minion, Myriad, PageMaker, Photoshop, Poetica, PostScript, and XMP are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States and/or other countries. Microsoft and Windows are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Apple, Mac, Macintosh, and Power Macintosh are trademarks of Apple Computer, Inc., registered in the United States and other countries. IBM is a registered trademark of IBM Corporation in the United States. Sun is a trademark or registered trademark of Sun Microsystems, Inc. in the United States and other countries. UNIX is a registered trademark of The Open Group. SVG is a trademark of the World Wide Web Consortium; marks of the W3C are registered and held by its host institutions MIT, INRIA and Keio. Helvetica and Times are registered trademarks of Linotype-Hell AG and/or its subsidiaries. Arial and Times New Roman are trademarks of The Monotype Corporation registered in the U.S. Patent and Trademark Office and may be registered in certain other jurisdictions. ITC Zapf Dingbats is a registered trademark of International Typeface Corporation. Ryumin Light is a trademark of Morisawa & Co., Ltd. All other trademarks are the property of their respective owners.

Fig 4.2 Effects of thresholding on small text

Thresholding is quite sensitive to the effects of blurring, hence has a tendency to over blacken images, especially in between letters like ‘B’, ‘D’ etc. as they contain loops in the letter which can get darkened due to the blurring. This can be remedied by reducing the histogram equalisation, but might lead to worse results for other pictures.

Chapter 5

APPLICATIONS & LIMITATIONS

5.1 Applications

1) Convenience to blind people – The biggest application of this project is that it can provide a way for blind people to easily read text from images.

2) Convenience to students – This project can also be helpful to students who often have to study from digital copies of books, which are very hard to search, and put a lot of strain on the eyes.

3) Helpful to people with long screen times – This project can also help anyone who has to sit in front of their computer screens everyday for prolonged periods of time. This project can help reduce the number of hours spent in front of a computer by offloading some of that time to audio.

5.2 Limitations

1. Image To Image Variance – The denoising algorithm is quite sensitive to per image variance, there are several factors that affect the quality of output of the algorithm. Two of the biggest factors are text size, noise amount.

2. Image Quality – The quality of the image, i.e., how clear the image also affects the performance of the processing algorithms, moreover it is impossible to extract text data where it is so blurred to the point where it is not even visible to a human.

3. Format of the text – The text in the picture should only be left aligned English text, against a contrasting background. In case there isn't sufficient contrast between the background and foreground, it is very difficult to extract text especially after the image has been grayscale.

4. Quality of the text-to-speech – The text to speech can show local variance in word pronunciation. Since the text-to-speech does a basic preliminary analysis on the text to figure out the correct pronunciation, it sometimes figures out the wrong word sense which in turn leads it to mispronounce a word. This mostly has to do with the fact that surrounding words are considered while making the choice for the correct word sense.

CHAPTER 7

Accuracy & Future Scope

Overview

This unit will tell us about all the future aspects where this project can be used and what all improvements can be made so that we can use this project in various new fields. This chapter will also talk about accuracy of the overall project and accuracy of individual steps.

7.1 Accuracy

The project was not made with a 100% accuracy in mind. Our main objective was to limit the instances of false positives, i.e., the OCR detects text where there is none. We were quite successful in that area, as in our testing, the OCR rarely detected false positives, and usually just either returned no text, or slightly erroneous version of the actual text.

The accuracy in correctly detecting text varies from about 70-90% depending upon the image, and more importantly the text size in the image. Our algorithms rely on the fact that the images will have text against a white background, and if this pre-requisite is not fulfilled then the accuracy suffers.

However, the accuracy of about 90% is usually enough since the Text-to-Speech engine can make some rudimentary corrections, and the human mind itself is also capable of understanding a misspelt word from the context.

7.2 Future Scope:

Our project can be used as a baseline to further research and develop this technology. The major problem that we faced was that nearly every image requires a slightly different algorithm to properly process it. Therefore, integrating machine learning models is a step that might help with the image pre-processing. Convolutional neural networks are usually very good at extracting the features from an image. They may be used to figure out the correct parameters such as blur strength, kernel size etc. for each image.

However, these steps come with the additional cost of computation, which might make the entire process slower than it currently is.

Further, improved versions of this project can be useful to create real-time image to speech converters than can be attached to people who have some sort of vision impairment. These can be useful to help them read certain things which are not available in braille, such as bus schedules, menus at restaurants etc.

More work can also be done to make the tool work on languages other than English. This can help anyone in a foreign country to simply point their camera at a block of text, and have it translated and read aloud.

Annexure

Codes-

Image Processing

```
import cv2
import pytesseract

def image_path_to_text(path):
    img = cv2.imread(path)
    return image_to_text(img)

def image_to_text(img):
    img = process_image(img)

    custom_config = r'--oem 3 --psm 6'
                    return pytesseract.image_to_string(img,
config=custom_config), img

def process_image(img):
    img = convert_to_grayscale(img)
    img = remove_noise(img)
    img = threshold(img)
    return img

def convert_to_grayscale(img):
```

```
    return cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

def remove_noise(img):
    se = cv2.getStructuringElement(cv2.MORPH_RECT , (10,10))
    bg = cv2.morphologyEx(img, cv2.MORPH_DILATE, se)
    img = cv2.divide(img, bg, scale=255)

    blurred = cv2.GaussianBlur(img, (3,3), sigmaX=5, sigmaY=5)
    img = cv2.addWeighted(img, 0.7, blurred, 0.3, 0)

    medianBlurred = cv2.medianBlur(img, 3)
    img = cv2.addWeighted(img, 0.5, medianBlurred, 0.5, 0)

    return img

def threshold(img):
    equalised = cv2.equalizeHist(img)
    img = cv2.addWeighted(img, 0.8, equalised, 0.2, 0)
                img = cv2.threshold(img, 0, 255,
cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]
    return img
```

Backend Response Code

```
from flask import Blueprint, request, render_template
import cv2
import numpy as np
from imageToSpeech.image_processing import image_to_text

import io
from PIL import Image
from base64 import b64encode

bp = Blueprint('images', __name__, url_prefix='/')

@bp.route("/", methods=["GET", "POST"])
def index():
    if request.method == "POST":
        file = request.files.get("image")
        if (file.filename == ""):
            return render_template("index.html")

        img = convert_to_cv_img(file)
        text, processed_image = image_to_text(img)

        file_object = io.BytesIO()
        img = Image.fromarray(processed_image.astype("uint8"))
```

```
        img.save(file_object, 'PNG')
                    base64img = "" +
b64encode(file_object.getvalue()).decode('ascii')

                return render_template("result.html", text=text,
image=base64img)

    return render_template("index.html")

def convert_to_cv_img(file):
    x = np.fromstring(file.stream.read(), dtype='uint8')
    img = cv2.imdecode(x, cv2.IMREAD_UNCHANGED)

    return img
```

Audio Generation Code

```
import os
from functools import wraps
import atexit as at_exit
from shutil import rmtree
from uuid import uuid4
from flask import url_for, jsonify, Markup
from gtts import gTTS

from flask_gtts.constants import PY2



class gtts(object):
    def __init__(self, app=None, temporary=True, tempdir='flask_gtts', route=False,
                 route_path='/gtts', route_decorator=None, failsafe=False, logging=True):
        '''Extension to help in generating Google Text-To-Speech files.

        Parameters
        ----------
        app : Flask Application, optional
            Flask application instance, by default None
        temporary : bool, optional
```

```
        Remove the audio files before existing, by default True
    tempdir : str, optional
        Name of the static directory to store audio files in, by default 'flask_gtts'
    route : bool, optional
        Enable endpoint to generate TTS file dynamically, by default False
    route_path : str, optional
        Endpoint route path, by default '/gtts'
    route_decorator : callable, optional
        Decorator to wrap route endpoint, by default None
    failsafe : bool, optional
        Failsafe or throw exceptions, by default False
    logging : bool, optional
        Enable or disable logging, by default True
    '''
    self.temporary = temporary
    self.tempdir = 'flask_gtts' if not tempdir or tempdir.startswith('/') else tempdir
    self.route = route
    self.route_path = route_path
    self.route_decorator = route_decorator
    self.failsafe = failsafe
    self.files = {}
    self.logging = logging
```

```
        app and self.init_app(app)

    def init_app(self, app):
        '''Lazy load Flask Application.

        Parameters
        ----------
        app : Flask Application
            Flask application instance.
        '''
        self.app = app
        self.tempdir = os.path.join(self.app.static_folder,
self.tempdir)
        self.inject()

        if not os.path.isdir(self.tempdir):
            if PY2:
                os.makedirs(self.tempdir)
            else:
                os.makedirs(self.tempdir, exist_ok=True)

        self.route and self.set_route()
        self.temporary and at_exit.register(self.teardown)

    def _handle_exception(self, exception):
        if not self.failsafe:
```

```
        raise exception

    if self.logging:
        self.app.logger.exception(exception)

    return ''

def teardown(self):
    '''Remove the cache directory and its content.'''
    self.files = {}

    try:
        os.path.isdir(self.tempdir) and rmtree(self.tempdir)
    except Exception as e:
        self._handle_exception(e)

def inject(self):
    '''Inject say and read into templates.'''
    @self.app.context_processor
    def inject_vars():
        return dict(sayit=self.say, read=self.read)

def say(self, lang='en-us', text='Flask says Hi!'):
    '''Generate a TTS audio file.
```

```
    Parameters
    ----------
    lang : str, optional
        Language to produce the TTS in, by default 'en-us'
    text : str, optional
        Text to convert into audio, by default 'Flask says Hi!'

    Returns
    -------
    str
        Relative url of the generated TTS audio file.
    '''
    if (text, lang) not in self.files:
        generator = gTTS(text=text) if lang == 'skip' else gTTS(lang=lang, text=text)
        file_name = None
        file_path = None

        while not file_name:
            temp_name = str(uuid4()).replace('-', '') + '.mp3'
            file_path = os.path.join(self.tempdir, temp_name)

            if not os.path.isfile(file_path):
                break

        self.files[(text, lang)] = file_path
```

```
            try:
                generator.save(file_path)
            except Exception as e:
                return self._handle_exception(e)

        file_name = os.path.basename(self.files.get((text, lang)))
        relative_dir = os.path.basename(self.tempdir)

        with self.app.app_context():
            try:
                print(os.path.join(relative_dir, file_name))
                return url_for('static',
                               filename=os.path.join(relative_dir,
file_name).replace("\\", "/"))
            except Exception as e:
                return self._handle_exception(e)

    def read(self, id='.toRead', mouseover=False):
        '''Read an HTML element inner text.

        Parameters
        ----------
        id : str, optional
            HTML element css selector, by default '.toRead'
        mouseover : bool, optional
```

```
        Read text on `mouseover` event instead of `click`, by
default False


    Returns

    -------

    str

        Safe JavaScript to read an HTML element content.

    '''

    if not self.route:

        try:

            self.set_route()

        except Exception:

            pass


    file_path = os.path.join(os.path.dirname(__file__),
'read.html')


    with open(file_path) as html_file:

        return Markup(html_file.read()

                      .replace('{id}', id)

                      .replace('{route}', self.route_path)

                      .replace('{event}', 'mouseover' if mouseover
else 'click'))


  def set_route(self):

    ''' Setup a route endpont on
`self.route_path/<language>/<text>` '''
```

```
def empty_decorator(function):
    @wraps(function)
    def wrapper(*args, **kwargs):
        return function(*args, **kwargs)
    return wrapper

decorator = self.route_decorator or empty_decorator

@self.app.route(self.route_path + '/<language>/<text>')
@decorator
def gtts_route(language, text):
    if PY2:
        language = language.encode('utf8')
        text = text.encode('utf8')

    mp3_link = self.say(language, text).replace('%5C', '/')
    return jsonify(mp3=mp3_link), 200 if mp3_link else 500
```