המחלקה להנדסת תעשייה
הפקולטה להנדסה ע"ש איבי ואלדר פליישמן
אוניברסיטת תל אביב

פרויקט סיום

Your Movie

מרצה המעבדה:

פרופ' עירד בן גל

אחראי המעבדה:

מר. גד הלוי

מגישים :

טל אילון | עמיחי כלב | אביהו מנחם

המחלקה להנדסת תעשייה
הפקולטה להנדסה ע"ש איבי ואלדר פליישמן
אוניברסיטת תל אביב

תוכן עניינים

המחלקה להנדסת תעשייה
הפקולטה להנדסה ע"ש איבי ואלדר פליישמן
אוניברסיטת תל אביב

תיאור הפרויקט

המערכת שנבנתה במסגרת פרויקט זה הינה מערכת אינטראקטיבית ידידותית למשתמש ליצירת סרטונים. במסגרת המערכת, המשתמש יוכל לבחור האם להרכיב את הסרטון מתמונות אותם הוא צייר במסגרתה, או מתמונות קיימות אותם הוא מעוניין לאחד לכדי סרט המציג אותם ברציפות.

את הציורים שצוירו במסגרת המערכת יוכל המשתמש לעבד ברשת GauGAN, רשת גנרטיבית עמוקה על שם הצייר פול גאוגן שהוקמה ע"י חברת nVIDIA בשנת 2019, הממירה את הציור לתמונת נוף. למשתמש יש גישה אל הרשת העמוקה שכבר נלמדה מראש, כאשר הצבעים האפשריים בהם המשתמש יכול לבחור במסגרת הציור מותאמים לאובייקטים שונים כגון הרים, עננים ועוד. כך יוכל המשתמש ליצור בעזרת ציורים אלו סרטוני טבע.

לאחר שהמשתמש בחר את התמונות שישולבו בסרטון אותו הוא מעוניין ליצור, המערכת מאפשרת לייבא לסרטון כתוביות הכתובות בקובץ, או לכתוב אותם במסגרת המערכת, ולהציגם בסרטון. בנוסף, המערכת מאפשרת לייבא לסרטון מוזיקה מקובץ mp3, או ממוזיקה מסרטון Youtube שבו יבחר המשתמש.

בסיום ההליך יוכל המשתמש לצפות בסרטון שיצר.

מטרת הפרויקט

מטרת הפרויקט הינה יצירת אפליקציה אינטרנטית ידידותית למשתמש על תשתית streamlit שמאפשרת ליצור סרטונים מרצף של תמונות או ציורים, כאשר את הציורים יוכל המשתמש לבחור לעבד ברשת GauGAN של חברת nVIDIA במטרה ליצור מהם סרטון טבע. האפליקציה תאפשר לצרף לסרטון גם כתוביות ושמע.

ציוד נדרש

- מחשב עם חיבור לאינטרנט, ודפדפן Chrome מותקן על המחשב.
- התוכנה ImageMagick, גרסה 7, הורדה כאן. במהלך ההתקנה יש לבחור באופציה "Install legacy utilities (e.g. convert)".

המעבדות שלוקחות חלק בפרויקט

שימוש בstreamlit – מעבדה 7 Deep Learning ומעבדה 9 Web Scraping

- במסגרת הפרויקט ממשק המשתמש נוצר על גבי תשתית החבילה streamlit בפייתון. במסגרת הממשק יכול המשתמש לנווט בין הדפים השונים באפליקציה.

שימוש בmoviepy ובעיבוד תמונה ברשת GauGAN – מעבדה 10 מעבדת הכנה לפרויקט

- במסגרת הפרויקט האופן בו הציורים עובדו ברשת GauGAN התבסס על הדרך שבה העיבוד מומש במעבדת ההכנה לפרויקט.
- במסגרת הפרויקט יצירת הסרט הסופי שכולל גם כתוביות ושמע התבססה על האופן בו בוצע שימוש בחבילת moviepy במעבדת ההכנה לפרויקט.

המחלקה להנדסת תעשייה
הפקולטה להנדסה ע"ש איבי ואלדר פליישמן
אוניברסיטת תל אביב

תיאור תהליך השימוש במערכת

המערכת כוללת את 5 המסכים הבאים:

1. מסך הבית
2. מסך ציור/העלאת קבצים
3. מסך עיבוד ב GauGAN
4. מסך יצירת סרטון
5. מסך צפייה בסרטון המוכן

תהליך השימוש באפליקציה הינו טורי וכולל את השלבים הבאים, בסדר הבא:

1. **דף הבית** – המסך הראשון אותו המשתמש רואה כאשר ניגש לראשונה למערכת. במסך זה מוצג למשתמש תקציר על מטרת המערכת, פירוט קצר על המסכים הקיימים במערכת ודרך המעבר המומלצת אותה יש לבצע כדי ליצור את הסרטון הרצוי.
2. **מסך ציור/מסך העלאת קבצים** – במסך זה ניתנות למשתמש 2 אפשרויות, לבחירתו. האפשרות הראשונה היא מסך קנבס (Canvas) עליו המשתמש יכול לצייר בעזרת סרגל כלים, בהתאם לאובייקטים הנבחרים המתאימים לעיבוד ברשת ה GauGAN. המשתמש יכול לבחור האם לשמור את הציורים באופן ידני, או באופן אוטומטי בעת סיום הלחיצה על המקש השמאלי בתום האיור עליו. האפשרות השנייה היא להעלות קובצי תמונות מוכנות. התמונות יכולות להיות תמונות המתאימות לעיבוד בGauGAN או תמונות אחרות לבחירת המשתמש.
3. **מסך עיבוד בGauGAN** – במסך זה המשתמש בוחר האם לעבד או לא את הציורים אותם צייר, או את התמונות אותם העלה. המשתמש יוכל לבחור ב10 סגנונות לכל היותר המתארים מצבים שונים במהלך היום כגון שקיעה, ולעבד את התמונה בהתאם לסגנונות אלו. למשתמש יינתן חיווי באשר להצלחת עיבוד התמונות, וניתן לו חיווי באשר להתקדמות תהליך עיבודן.
4. **מסך יצירת הסרט** – במסך זה המשתמש יוכל ליצור את הסרט המתבסס על הציורים שעובדו ב GauGAN או התמונות שבחר ליצור עמן סרט. אם ירצה בכך, יוכל המשתמש להוסיף כתוביות ושמע לסרט. המשתמש יוכל גם לשלוט בקצב המעבר בין התמונות במהלך הסרטון (frame rate). אם בחר להוסיף כתוביות, הוא יוכל להעלות את הכתוביות כקובץ טקסט, או לכתוב אותם במסגרת המערכת. אם בחר להוסיף שמע, הוא יוכל לייבא את השמע הרצוי באמצעות קובץ mp3 או באמצעות סרטון Youtube.
5. **מסך צפייה בסרט** – במסך הזה המשתמש יוכל לצפות בתוצרים של הסרטון שיצר בשלב הקודם.

המחלקה להנדסת תעשייה
הפקולטה להנדסה ע"ש איבי ואלדר פליישמן
אוניברסיטת תל אביב

קשיים שעלו במסגרת הפרויקט

1. **יצירת קנבס בפלטפורמה של streamlit**. במסגרת מעבדה 10, מעבדת ההכנה לפרויקט, בוצע שימוש בחבילת opencv שבמסגרתו נפתח חלון ייעודי בו יכול היה המשתמש לצייר את הציור, כאשר אותו ציור בהמשך יעובד ברשת ה GauGAN. מאחר שלא היתה אפשרות לשלב חלון זה במסגרת הממשק האינטרנטי שמציע streamlit, היה צורך באיתור פתרון ייחודי לעניין זה. לשמחתנו מצאנו כי קיימת חבילה ייעודית שנבנתה ממש בעת האחרונה, העונה לשם streamlit_drawable_canvas שנותנת מענה לבעיה זו באמצעות Fabric.js[1].
2. **המרת הצבעים מ RGBA ל RGB**. כחלק מהמענה שניתן לבעיה לעיל, הקנבס החזיר את הצבעים בפורמט RGBA ולא בפורמט RGB. בהתאם לכך היה צורך לבצע התאמה לקלט המתקבל מציור המשתמש, ולהמיר את מערך הצבעים למימדים 512X512X3 במקום 512X512X4.
3. **השלמת הרקע**. בהמשך לבעיות והפתרונות לעיל, במסגרת ציור הקנבס, מאחורי הקלעים מתקבל רק המיקומים בהם המשתמש מצייר, ללא הרקע המתאים ל GauGAN. זו מגבלה מובנית בחבילה streamlit_drawable_canvas. הפתרון לכך היה להוסיף לולאה שעוברת על המימדים ומשלימה, היכן שנדרש, את צבעי הרקע הסטנדרטיים (של שמיים וים) בתמונה. התהליך גרר עלייה משמעותית בזמן שמירת התמונה שמגיע עד לכ3 שניות.
4. **פורמט התמונות**. פורמט התמונות האפשרי לקבל כקלט במסגרת הרשת GauGAN הינו פורמט PNG ברזולוציה 512X512. לצורך התאמת התמונות ומתן האפשרות למשתמש להעלות תמונות כרצונו, הוגבלו הפורמטים המורשים – הפורמטים שנבחרו הם JPG,PNG (כאשר הפורמט JPG הומר לפורמט PNG) תוך צמצום התמונה לרזולוציה 512X512, וזאת במסגרת החבילה המובנית בפייתון Pillow לעריכת תמונות.
5. **שילוב מילון הצבעים של GauGAN**. בstreamlit הצבעים נתמכים כקלט בפורמט הקסדצימלי בלבד, ואילו מילון הצבעים שנבנה במסגרת מעבדה 10 היה נתון בפורמט RGB בלבד. לצורך פתרון הבעיה, בוצעה המרה של הצבעים מפורמט (R,G,B) לפורמט #XXXXXX, כאשר לכל צבע (אדום, ירוק וכחול) יש 2 ספרות ייעודיות בפורמט הקסדצימלי. ההתאמה נשמרה כקובץ pickle אותו יש לטעון במסגרת השימוש במערכת.
6. **התאמת צבעי המערכת לצבעים המתקבלים בGauGAN**. בחלק מההמרות שמתבצעות לעיל, הולך לאיבוד מידע של חלק מהצבעים. כתוצאה מכך ייתכן כי ציור לא יתקבל ברשת ה GauGAN ותוחזר שגיאה. הבעיה נובעת מהcanvas עמו נעשה שימוש – התאמתו מראש למערך תלת מימדי המתאים לערכי RGB ובנוסף התאמה לסוג uint8 היה נותן פתרון מיידי לסוגיה, אך במסגרת החבילה לא ניתן לכך מענה. לכן, במידה שבה עיבוד התמונה הצליח או נכשל ברשת ה GauGAN, המשתמש יקבל על כך חיווי במסגרת שלב עיבוד התמונות.

שימוש בכלים/טכנולוגיות שלא נלמדו במסגרת הקורס

במסגרת הפרויקט נעשה שימוש במספר פונקציות שניתנו במסגרת חבילות שלא נלמדו במהלך הסמסטר. לצורך הורדת הסרטון מהיוטיוב נדרש להשתמש בחבילה pytube שמאפשרת את ההורדה.

כמו כן, שילוב הקנבס במערכת התאפשר באמצעות החבילה sreamlit_drawable_canvas המתואר לעיל. בנוסף, שולבה במסגרת המערכת חבילת stqdm שמאפשרת להראות בשלב עיבוד התמונות את קצב ההתקדמות.

במסגרת הפרויקט נדרשה גם כתיבה ומחיקה של קבצים ותיקיות, אלו נעשו באמצעות החבילה המובנית בפייתון os.

[1] Fabric.js - http://fabricjs.com/

המחלקה להנדסת תעשייה
הפקולטה להנדסה ע"ש איבי ואלדר פליישמן
אוניברסיטת תל אביב

החבילות הנדרשות להפעלת המערכת

להלן רשימת החבילות הנדרשות במסגרת הפרויקט בצירוף הגרסאות בהן נעשה שימוש.

דרישות אלו מתוארות גם בקובץ requirements.txt.

- **פייתון גרסה 3.8**
- moviepy>=1.0.3
- pytube>=10.8.4
- streamlit>=0.82.0
- numpy>=1.19.2
- gaugan>=1.1
- opencv-python>=4.5.2.52
- pillow>=8.0.1
- stqdm>=0.0.3

קבצים נדרשים להפעלת המערכת

לצורך הפעלת המערכת, נדרשים הקבצים הבאים, המגיעים יחד עם דוח זה:

1. colors.p – קובץ pickle המכיל את מילון הצבעים המתאימים לGauGAN כאשר המפתח הוא סוג האובייקט, והערך הוא הצבע בפורמט (R,G,B).
2. colors_hex.p – קובץ pickle המכיל את מילון הצבעים המתאימים לGauGAN בפורמט הקסדצימלי.
3. error_detection.png – קובץ תמונה המתקבל מ GauGAN כאשר עיבוד התמונה נכשל. תמונה זו מצורפת למערכת כדי לאפשר זיהוי של הבעיה במקרה שבו אכן העיבוד ברשת GauGAN נכשל.

הוראות הפעלה

1. יש תחילה לשמור את הפרויקט המצורף, על כל קבציו, במחשב האישי.
2. יש לפתוח cmd של אנקונדה, או לפתוח את הקובץ בתוכנה PyCharm ולהפעיל את ה Terminal.
3. יש לוודא כי כל החבילות הרלוונטיות, המתוארות בקובץ requirements.txt, מותקנות.
4. יש לוודא כי סביבת הריצה הינה פייתון 3.8.
5. במסגרת ה Terminal, לאחר וידוא כי הניתוב תואם למיקום הקובץ, יש להריץ את הפקודה:
   streamlit run main_project_group05.py
6. יש לפעול לפי ההוראות באפליקציה שתעלה בדפדפן באופן אוטומטי.

המחלקה להנדסת תעשייה
הפקולטה להנדסה ע"ש איבי ואלדר פליישמן
אוניברסיטת תל אביב

קוד הפרויקט

להלן קוד הפרויקט.

```
######################### PROJECT CIM
######################### Created by Tal Eylon, Amihai Kalev and Avihoo Menahem
######################### June 2021


##################################################################################
###########
######################### IMPORTS
##################################################################################
###########

import os.path
import numpy as np, pickle
import cv2
import gaugan
from PIL import Image
import streamlit as st
from streamlit_drawable_canvas import st_canvas
import pytube
import stqdm
from moviepy.editor import *
import moviepy.video.io.ffmpeg_tools as ffmpg

##################################################################################
###########
######################### BACKEND FUNCTIONS
##################################################################################
###########

@st.cache
def load_colors():
    """
    This functions imports the GauGAN colors dictionary from predefined pickle files
    :return: colors dictionary with RGB values, and colors_hex dictionary with HEX
colors values
    """
    with open("colors.p", "rb") as f:
        colors = pickle.load(f)
    with open("colors_hex.p", "rb") as f:
        colors_hex = pickle.load(f)
    return colors,colors_hex

colors, colors_hex = load_colors()


def convert_to_hex(rgb):
    """
    This function converts RGB color value into HEX value
    :param rgb: a list/tuple with R, G and B values
    :return: The RGB color in HEX format
    """
    r,g,b = rgb
    return '#{:02x}{:02x}{:02x}'.format(r, g, b)

@st.cache(allow_output_mutation=True)
def countering():
```

```
    """
    This function along with enabling the cache allows to save state in streamlit
    :return: empty list that will be recalled from the cache
    """
    return []


counter = countering()


def make_canvas() -> np.array:
    '''
    Make initial canvas sea and sky that fits the requirements for GauGAN
    :return: Initial default draw - a numpy array with the dimensions 512x512x3
    '''
    # Create canvas of zeros 512*512*3
    img = np.zeros((512,512,3), np.uint8)
    # Set 300 upper pixels to sky color
    img[0:300, 0:512] = colors['Sky']
    # Set the rest pixels to sea color
    img[300:512, 0:512] = colors['Sea']
    # Return canvas
    return img


def sort_files(path: str) -> list:

    '''
    Sort files in ascending order of creation time.
    Use for sorting drawings to make nature photos out of them,
    and for sorting drawings/pictures to make a movie out of them.
    :param path: directory path.
    :return: list of files
    '''
    # Make list of file names given the path
    path = path+'/'
    files = os.listdir(path)
    # Join path with each file
    files = [path+f for f in files]
    # Sort the files inplace using key=os.path.getmtime
    files.sort(key=os.path.getmtime)
    # Return sorted files
    return files


def make_nature(styles_dict:
dict,foldername='tmp',processedfoldername='Processed_imgs',styles=[1]):
    """
    Process drawings in GauGAN
    :param styles_dict: dictionary of styles where keys are the style's name and the
values are the style's number
    :param foldername: The directory that the drawings will be read from
    :param processedfoldername: The directory where the processed drawings will be
saved
    :param styles: the styles the user has chosen to process in GauGAN
    :return:
    """
    # initialize counters for success/failure of process
    success,failure = 0,0

    # Sort files by creation time in drawings directory.
```

```
    files = sort_files(foldername)
    # Loop files
    for i in stqdm.stqdm(range(len(files))):
        p = files[i]
        for j in stqdm.stqdm(range(len(styles))):
            # Read each file
            with open(p, "rb") as f:
                # Get processed image back from GauGAN server.
                image = gaugan.processImage(f.read(), style=styles_dict[styles[j]])
            # Open the processed file for writing add style num to img name
            saving_path = processedfoldername+"/"+os.path.split(p[:-4])[-
1]+'%s.jpg'%j

            # Load the error image that can be received as result in case of colors
issue
            error_detector = cv2.imread('error_detector.png')

            if np.array_equal(image,error_detector): # If the process in GauGAN
server failed
                failure += 1
            else:
                if not os.path.exists(processedfoldername):
                    os.makedirs(processedfoldername) # create the processed in
GauGAN folder if it doesn't exist

                with open(saving_path, "wb") as f: # Save the processed drawing
                    # Write the processed file.
                    f.write(image)
                success += 1
    if success>0:
        st.success("%s Files were processed successfully!"%success)
    if failure>0:
        st.warning("%s Files failed to be processed due to an error!")



def make_seret(processed_files_directory='files/',fps=5):
    """
    This function creates the initial movie in AVI format using opencv.
    The movie frame rate will be dfined by the fps variable.
    :param processed_files_directory: The drawings/pictures that will be composited
    :param fps: frames per second
    :return:
    """
    # Sort files in processed images directory
    files = sort_files(processed_files_directory)
    # Create list as container for the movie.
    img_array = []
    # For each file
    for file in files:
        file_format = file.split(".")
        if file_format[-1] == 'jpg': # verify that we will include jpg files only in
the movie
            # Read the file
            img = cv2.imread(file)
            # Extract height, width, channels from image
            height, width, layers = img.shape
            # size = (width, height)
            size = (width, height)
            # Append image to movie container
            img_array.append(img)
```

```
    # Create a video writer for the movie
    out = cv2.VideoWriter(processed_files_directory+'initial.avi',
cv2.VideoWriter_fourcc(*'DIVX'), fps, size)
    # For each image in container
    for image in img_array:
        # Write image by video writer
        out.write(image)
    # Release video writer.
    out.release()


def make_movie(processed_files_directory='files/', WITH_SUBTITLES=False,
WITH_AUDIO=False):
    """
    Based on the product of the make_seret function, this function will create the
final movie,
    with audio/subtitles/both.
    :param processed_files_directory: The directory in which the final movie will be
saved
    :param WITH_SUBTITLES: boolean
    :param WITH_AUDIO: boolean
    :return:
    """
    # Declare the text for sub-titles

    if WITH_SUBTITLES: # if the user is willing to have subtitles in the movie
        with open(processed_files_directory+'subtitles.txt', 'r', encoding='utf8')
as f:
            txt = f.read() # read the subtitles file
        # Split text to lines.
        subtitles = txt.split('\n')
        # Declare  VideoFileClip from the movie that I already have.
        clip = VideoFileClip(processed_files_directory + "initial.avi")
        # Declare duration of one sub-title as total duration of the video divided
by number of lines.
        duration = clip.duration/len(subtitles)
        # Set start to zero.
        start=0
        # Set container for the clips.
        videos=[]
        # Loop all sub-titles
        for line in subtitles:
            # Make text clip from the reversed Hebrew text
            txt_clip = TextClip(line[::-1], fontsize=30, color='yellow',
font='Calibri')
            # Set position to the bottom of screen.
            txt_clip = txt_clip.set_position('bottom').set_duration(duration)
            # Make sub clip of the movie with same duration as text clip.
            sub_clip = clip.subclip(start,start+duration)
            # Set CompositeVideoClip from the text clip and sub clip.
            video = CompositeVideoClip([sub_clip, txt_clip])
            # Insert the video to the clips container
            videos.append(video)
            # Set start time for next sub-title.
            start+=duration
        # Concatenate all clips of the container.
        res = concatenate_videoclips(videos)
        clip = res # now the clip is res
    else:
        clip = VideoFileClip(processed_files_directory+ "initial.avi") # the clip
won't have subtitles
```

```
    # Set audio clip from mp3 file.
    if WITH_AUDIO: # if the user has chosen to include soundtrack in the movie
        f = 'audio.mp3' # change to mp3 soundtrack file of the movie
        # set the duration of the audioclip to max(duration of clip), even if the
audioclip is longer
        audioclip = AudioFileClip(processed_files_directory+f)

        # check if the clip length is bigger than the
        if clip.duration > audioclip.duration:
            number_of_duplicated = int(np.ceil(clip.duration/audioclip.duration))
            # duplicate the audioclip in order to later fit the movie's duration
            audioclip =
concatenate_audioclips([AudioFileClip(processed_files_directory+f) for i in
range(number_of_duplicated)])

        # Now fit the audioclip duration to the movie's
        audioclip = audioclip.set_duration(clip.duration)

        # Set audio for the container.
        if not WITH_SUBTITLES: # if the user wanted to have audio included without
subtitles
            videoclip = clip.set_audio(audioclip)
        else: # if the user wanted to have both audio and subtitles
            videoclip = res.set_audio(audioclip)
    else:
        videoclip = clip # if the user didn't want audio in the movie

    # Write the video file.
    f = 'final_movie.mp4' # change to the desired movie filename
    videoclip.write_videofile(processed_files_directory+f)

def download_audio_from_youtube(youtube_link: str):
    """
    This function will extract the audio from youtube video
    :param youtube_link: The youtube video link
    :return:
    """
    with st.spinner("Extracting audio from Youtube..."):
        try:
            a =
pytube.YouTube(youtube_link).streams.first().download('files/','video_for_audio') #
Download video from youtube
            b =
ffmpg.ffmpeg_extract_audio('files/video_for_audio.mp4','files/audio.mp3')         #
extract sound and save as mp3
            os.remove('files/video_for_audio.mp4')
# remove unecessary video
            # Release the process from the downloaded files
            del a, b
            st.success("Sound was extracted successfully from the youtube video!")
        except:
            st.error("Unexpected error has occured, please try again!")


##############################################################################
###########
#################### FRONTEND FUNCTIONS - STREAMLIT PAGES
##############################################################################
```

```
###########



def draw(canvas_result,automatic_save,manual_save):
    """
    This function will save the user's drawing into a PNG file.
    :param canvas_result: The user's drawing
    :param automatic_save: True if the user has chosen to have automatic saving
after releasing the mouse from drawing
    :param manual_save: True if the user has chosen to manually save the drawings
    :return: Success message if all went correctly
    """
    if canvas_result is not None and canvas_result.image_data is not None and
(automatic_save or manual_save):
        # Receive the user's drawing with the dimensions: 512X512X4
        img_data = canvas_result.image_data
        # the user's drawing is in RGBA mode with floats instead of integers -
convert to uint8 type and to RGB format
        im = Image.fromarray(img_data.astype(np.uint8)[:,:,:3]).convert('RGB') #
convert to dimensions 512X512X3
        # initialize a copy of the user's drawing.
        add_bg = np.array(im, dtype='uint8') # initalize a copy
        # allow the user to know that the saving is in progress.
        with st.spinner("Saving image..."):
            # the drawing is lack of the GauGAN background because
streamlit_drawable_canvas library doesn't allow it yet.
            # Because of that the background will be added manually - o(n^3) at the
moment.
            for i in range(add_bg.shape[0]):
                for j in range(add_bg.shape[1]):
                    if list(add_bg[i,j]) != [0,0,0]: # if the current RGB value is
not (0,0,0) (black) -
                        for k in range(add_bg.shape[2]): # then make sure we don't
have white values (255)
                            if add_bg[i,j][k] == 255: # we will fill them with the
relevant background color position
                                add_bg[i,j][k] = colors['Sky'][k] if i<300 else
colors['Sea'][k]
                    else: # else, we do indeed have RGB value of (0,0,0), then
replace it by its entirety to the relevant
                        # background color.
                        add_bg[i,j] = colors['Sky'] if i<300 else colors['Sea']

            # Create PIL object of the manually added background with drawing on the
canvas
            add_bg = Image.fromarray(add_bg)
            # Assign the path where the file will be saved
            if not os.path.exists("tmp/"):
                os.makedirs("tmp/")
            file_path = f"tmp/pic%s.png"%(len(counter))
            # Increase the counter by adding dummy element into the counter list
            counter.append(0)
            # Save the drawing in PNG format

            add_bg.save(file_path, "PNG")
            st.success("Image saved successfully. Keep drawing!!")

def save_uploadedfiles(uploadedfiles: list, foldername: str, process=True):
    """
    If the user has chosen to upload files instead of creating drawings, this
```

```
function will
    receive the list of files and save them into the given folder name
    :param uploadedfiles: A list of the uploaded files
    :param foldername: The directory where the files will be saved
    :param process: Whether to process prepare the files for processing or not
    :return: Success message if all went correctly
    """
    # make sure the foldername exists
    if not os.path.exists(foldername+"/"):
        os.makedirs(foldername+"/")
    # Go over each file
    for i, file in enumerate(uploadedfiles):
        if process:
            # define the picture filename according to the counter i, png file for
processing with GauGAN
            joined_path = os.path.join(foldername,"%s.png"%i)
        else:
            # define the picture filename according to the counter i, jpg file if
processing is skipped
            joined_path = os.path.join(foldername,"%s.jpg"%i)
        # open the file
        resized = Image.open(file)
        # make sure to have the picture in 512X512 resolution
        resized = resized.resize((512,512))
        # save the file
        if process:
            resized.save(joined_path)
        else:
            resized.save(joined_path, "JPG")
    return st.success("Files were saved successfully")


def edit():
    """
    The Edit page.
    :return:
    """
    st.title("Draw/Upload Your Pictures")
    st.write("------------")
    select_action = st.radio("Choose one of the following options:",["Draw","Upload
Files"])
    st.write("------------")
    if select_action=="Draw":
        st.sidebar.header("Drawing Toolbox")
        # Specify canvas parameters in application
        st.write("In the below canvas you can paint your drawings.")
        st.write("Each color represents an object, which you can choose in the
sidebar.")
        st.write("The paintings can be saved either manually or automatically.")
        st.write("")

        ## Sidebar toolbox options
        # Stroke Width - the brushe's thickness
        st.sidebar.subheader("Stroke Width")
        stroke_width = st.sidebar.slider("", 2, 15, 5)

        # Allow the user to choose the desired object according to GauGAN dictionary
        st.sidebar.subheader("Object to Draw")
        stroke_color = colors_hex[st.sidebar.selectbox('',list(colors_hex.keys()))]

        # Allow the user to choose either freedraw, line drawing and replace the
```

```
objects on the canvas
        st.sidebar.subheader("Drawing Tool")
        drawing_mode = st.sidebar.selectbox("", ("freedraw", "line", "transform"))

        # Apply the initial GauGAN background into the canvas
        bg_image = Image.fromarray(make_canvas())

        # Define the canvas properties - will be updated live
        canvas_result = st_canvas(
            fill_color="#000000",
            stroke_color=stroke_color,
            stroke_width=stroke_width,
            background_image=bg_image,
            height=512,
            width=512,
            drawing_mode=drawing_mode,
            key="canvas"
        )

        # Allow automatic/manual saving
        manual_save = st.button("Save manually")
        automatic_save = st.checkbox("Save automatically")

        # Save the drawing according to the user's choice
        draw(canvas_result,automatic_save,manual_save)

    # if the user has chosen to upload files instead of drawing drawings:
    if select_action == "Upload Files":
        # Accept multiple files from the user - with jpg/png format only.
        lst =
st.file_uploader("Upload",type=['jpg','png'],accept_multiple_files=True)

        # If the user has uploaded GauGAN pictures and he wants to process them in
GauGAN
        if st.button("Save For GauGAN Processing"):
            if lst: # then save the files in the folder from which the pictures will
be processed
                save_uploadedfiles(lst,"tmp")
            else: # if the user has chosen this option but didn't upload any file
                st.error("Please select files")
        # If the user wants to avoid the GauGAN processing:
        elif st.button("Save Without GauGAN Processing"):
            if lst: # then take the pictures as if they were processed in GauGAN
                save_uploadedfiles(lst,"files", process=False)
            else: # if the user has chosen this option but didn't upload any file
                st.error("Please select files")

def process():
    """
    The process in GauGAN page.
    :return:
    """
    st.title("Process in GauGAN")
    st.subheader("Now choose the styles you wish to process with the paintings.")

    # Styles dictionary
    styles_dict = {"Afternoon 1": 1, "Afternoon 2": 2, "Sunset 1": 3,
                   "Sunset 2 Red Sun": 4, "Afternoon 3": 5, "Afternoon 4": 6,
"Sunset 3": 7,
                   "Sunset 4": 8, "Sunset 5": 9, "Sunset 6": 10}
```

```
    # Allow the user to choose the keys from the styles dictionary
    styles = st.multiselect("Styles: ",list(styles_dict.keys()),"Afternoon 1")

    # set the directory where the pictures will be imported from
    DIR = 'tmp/'
    # Calculate the number of files that are going to be processed
    number_of_files = len([name for name in os.listdir(DIR) if
os.path.isfile(os.path.join(DIR, name))])
    # Show it in the sidebar
    st.sidebar.subheader("Total pictures to process: %s"%number_of_files)
    # If the user has chosen to process them:
    if st.button("Start processing with GauGAN"):
        if number_of_files > 0:
            # Then process it: take the directory where the going-to-be-imported
pictures exist,
            # Process them, and save them in 'files' directory.
            make_nature(styles_dict,DIR[:-1],'files',styles)
        else: # the number of files is zero
            st.warning("There are no files to process.")


def create_movie():
    """
    The create movie page.
    :return:
    """
    st.title("Create The Movie")
    files = sort_files('files/')
    total_frames = np.sum([True if file.split(".")[-1]=='jpg' else False for file in
files])
    st.sidebar.write("Total frames detected: %s"%total_frames)

    with_subtitles = st.checkbox("Enable Subtitles")
    subtitles_selected = False

    # If the user wants to include subtitles in the movie
    if with_subtitles:
        st.write("---------")
        # Allow him to choose to create subtitles or import them
        option = st.radio("",["Write your subtitles","Upload subtitles"])
        if option == "Upload subtitles":
            # Upload txt file
            txt = st.file_uploader("Upload", type=["txt"])
            if txt:
                subtitles = txt.read()
                if subtitles: # if it was read successfully
                    subtitles_selected = True # subtitles loaded successfully
                    st.success("Subtitles were loaded successfully")
        elif option == "Write your subtitles":
            st.write("Please write your wanted subtitles in עברית only.")
            st.write("Separate each line by pressing Enter.")
            subtitles = st.text_area("Write the subtitles here:")
            if subtitles:
                # Write the subtitles hard-coded to the file subtitles.txt
                with open('files/subtitles.txt','w', encoding='utf-8') as f: #
                    f.write(str(subtitles))
                subtitles_selected = True # subtitles loaded successfully
                st.success("Subtitles were loaded successfully")
        st.write("---------")

    # If the user has chosen to add audio to the movie
```

```
    with_audio = st.checkbox("Enable Audio")
    audio_selected = False

    if with_audio:
        st.write("---------")
        # Allow him to choose to extract audio from youtube video or to upload a mp3
file
        select_action = st.radio("Choose one of the following options:",["Upload
mp3","Extract from Youtube video"])
        if select_action == "Upload mp3":
            # Receive the mp3 file from the user
            audio_file = st.file_uploader("Upload", type=["mp3"])
            if audio_file:
                with open('files/audio.mp3','wb') as f:
                    f.write(audio_file.getbuffer()) # write it in the processed
files directory
                audio_selected = True # Audio imported successfully
                st.success("Audio was imported successfully")
        elif select_action == "Extract from Youtube video":
            # Receive youtube link from the user
            youtube_link = st.text_input("Youtube link:")
            process_button = st.button("Process")
            if process_button or youtube_link:
                audio_selected = True
            if process_button:
                # check if the youtube link is correct
                if youtube_link[:32] == 'https://www.youtube.com/watch?v=' and
len(youtube_link)==43:
                    download_audio_from_youtube(youtube_link)
                    audio_selected = True
                else: # if it's not correct, let the user know
                    st.error("Invalid youtube link, please try again")
        st.write("---------")

    # Allow the user to choose the frame rate
    fps = st.slider("Frames per second:",0.5,20.0,3.0,0.5)

    if st.button("Start!"):
        if total_frames > 0:
            # if subtitles or audio are selected, ensure we have the info for
processing them
            if with_audio and not audio_selected:
                st.error("You have chosen to include audio, but haven't included
any. Please try again")
                st.stop()
            if with_subtitles and not subtitles_selected:
                st.error("You have chosen to include subtitles, but haven't included
any. Please try again.")
                st.stop()
            with st.spinner("Creating raw movie..."):
                make_seret(fps=fps)
                st.success("Raw movie created successfully!")
            with st.spinner("Creating final movie..."):
                make_movie(WITH_SUBTITLES=with_subtitles,WITH_AUDIO=with_audio)
                st.success("The movie has been created successfully!")
        else: # no frames were detected!
            st.warning("0 Frames were detected. Please process some pictures before
using this screen!")

def watch_movie():
    """
```

המחלקה להנדסת תעשייה
הפקולטה להנדסה ע"ש איבי ואלדר פליישמן
אוניברסיטת תל אביב

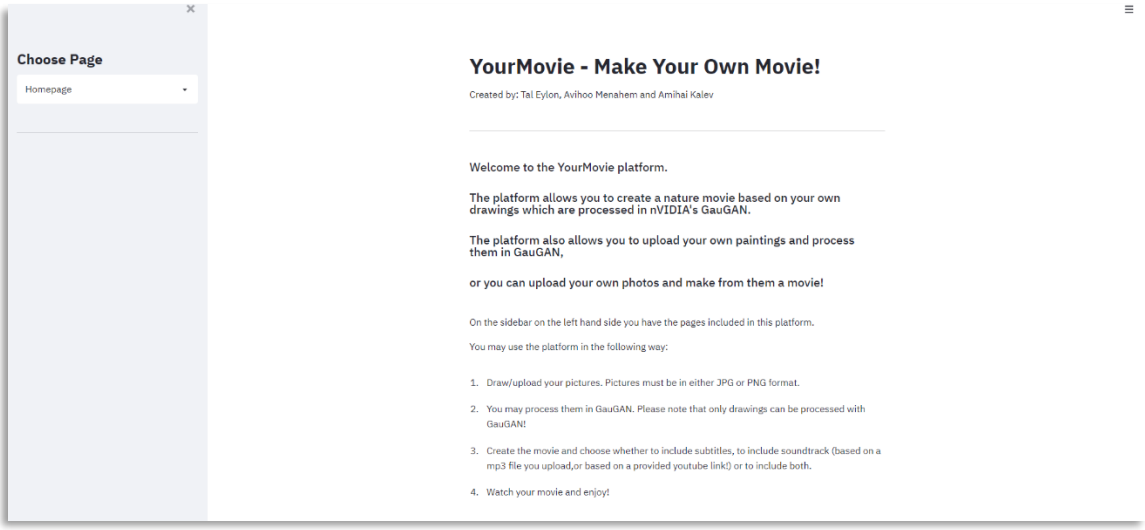
```
    The watch movie page. Allows the user to watch the just created movie.
    :return:
    """
    if os.path.isfile('files/final_movie.mp4'): # if the file exists
        with open('files/final_movie.mp4', 'rb') as f:
            video_data = f.read()
        st.video(video_data)
    else: # if the file doesn't exist, let the user know
        st.header("You haven't created a movie yet!")

def homepage():
    """
    The homepage.
    :return:
    """
    st.title("YourMovie - Make Your Own Movie!")
    st.write("Created by: Tal Eylon, Avihoo Menahem and Amihai Kalev")
    st.write("------------")
    st.subheader("Welcome to the YourMovie platform.")
    st.subheader("The platform allows you to create a nature movie based on your own
drawings which are processed in nVIDIA's GauGAN. ")
    st.subheader("The platform also allows you to upload your own paintings and
process them in GauGAN, ")
    st.subheader("or you can upload your own photos and make from them a movie!")
    st.subheader("")
    st.write("On the sidebar on the left hand side you have the pages included in
this platform.")
    st.write("You may use the platform in the following way:")
    st.write("")
    st.write("1. Draw/upload your pictures. Pictures must be in either JPG or PNG
format.")
    st.write("2. You may process them in GauGAN. Please note that only drawings can
be processed with GauGAN!")
    st.write("3. Create the movie and choose whether to include subtitles, to
include soundtrack (based on a mp3 file you upload,"
             "or based on a provided youtube link!) or to include both.")
    st.write("4. Watch your movie and enjoy!")

#################### MAIN PROGRAM
##############################################################################
def main():
    # The app's menu
    menu = ["Homepage","Edit","Process in GauGAN","Create Movie","Watch Your Movie"]
    st.sidebar.title("Choose Page")
    option = st.sidebar.selectbox("",menu)
    st.sidebar.write("------------------------------")

    # According to each chosen page, redirect the user accordingly.
    if option=="Homepage":
        homepage()
    if option=="Edit":
        edit()
    elif option=="Process in GauGAN":
        process()
    elif option=="Create Movie":
        create_movie()
    elif option=="Watch Your Movie":
        watch_movie()

# Start the app!
main()
```

המחלקה להנדסת תעשייה
הפקולטה להנדסה ע"ש איבי ואלדר פליישמן
אוניברסיטת תל אביב

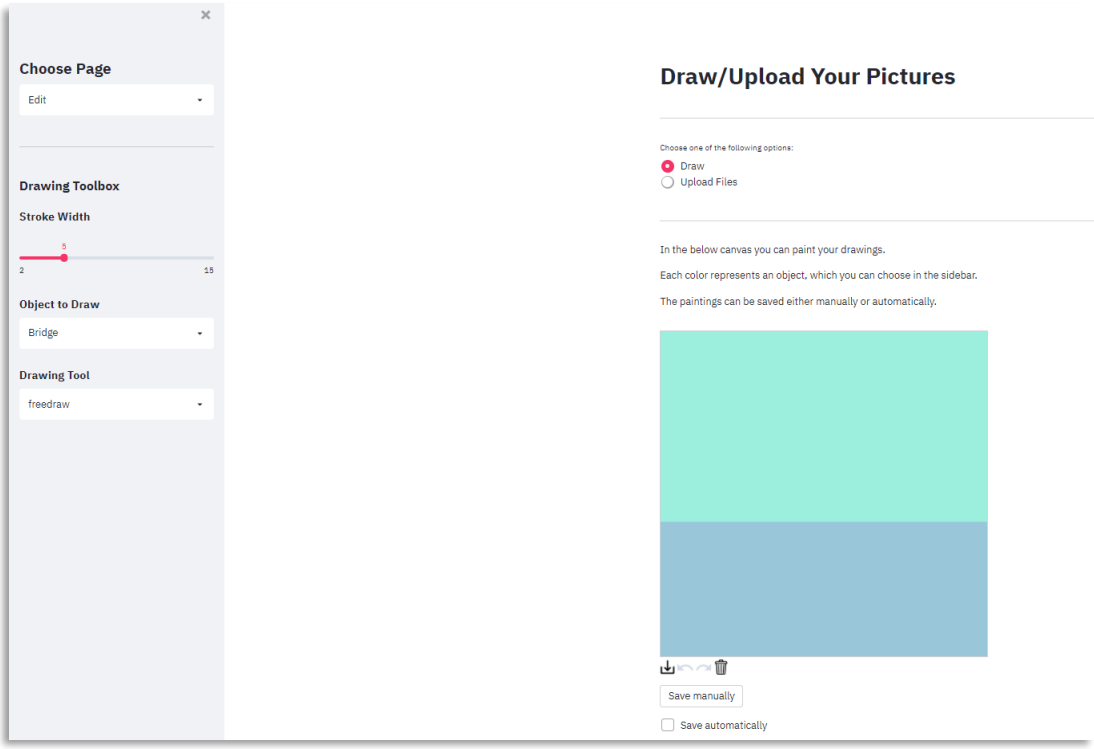
נספחים

נספח 1 – מסכי המערכת

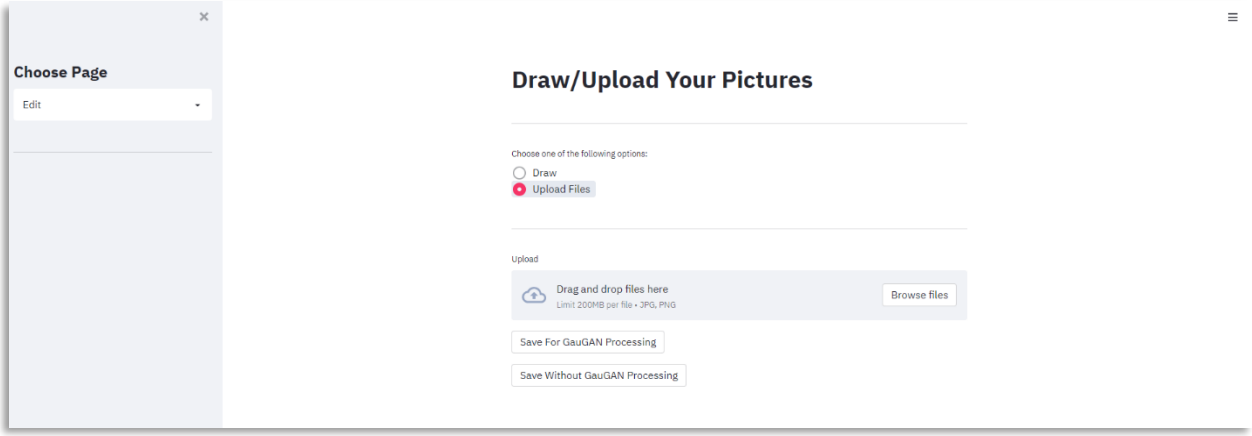
מסך הבית -



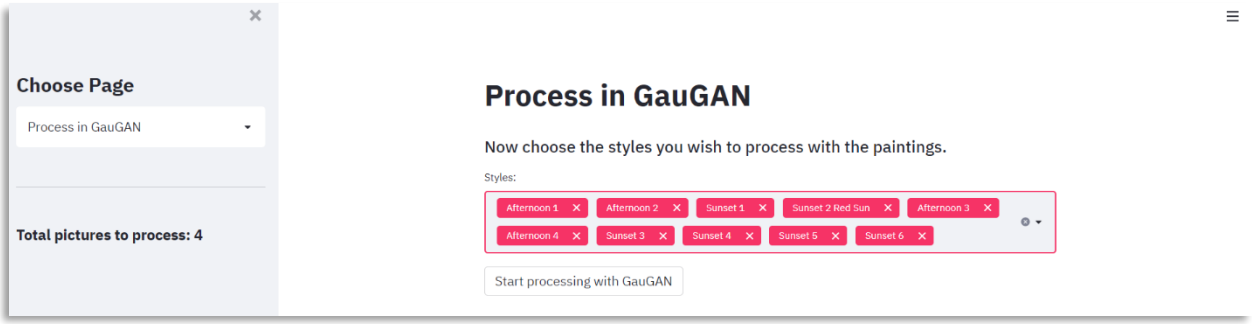
מסך הציור בקנבס -



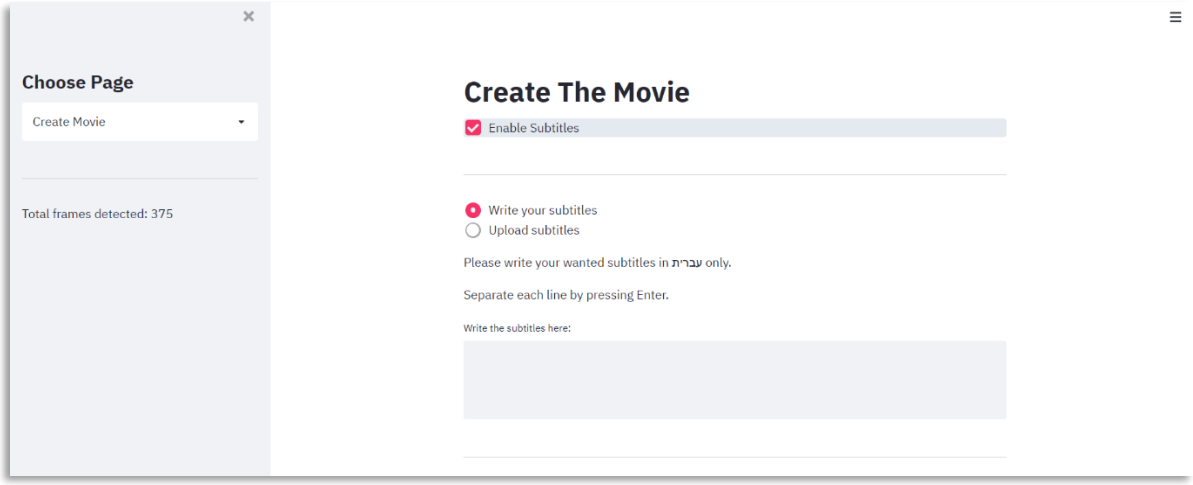
מסך העלאת התמונות -



מסך עיבוד התמונות ב GauGAN -

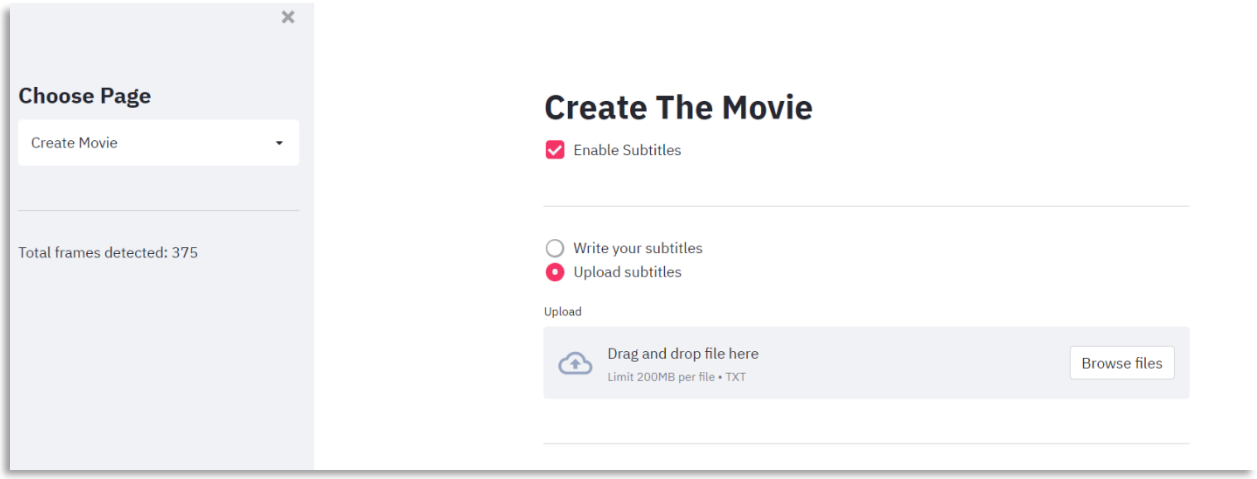


מסך יצירת הסרט עם כתוביות – כתיבת הכתוביות במסגרת המערכת -

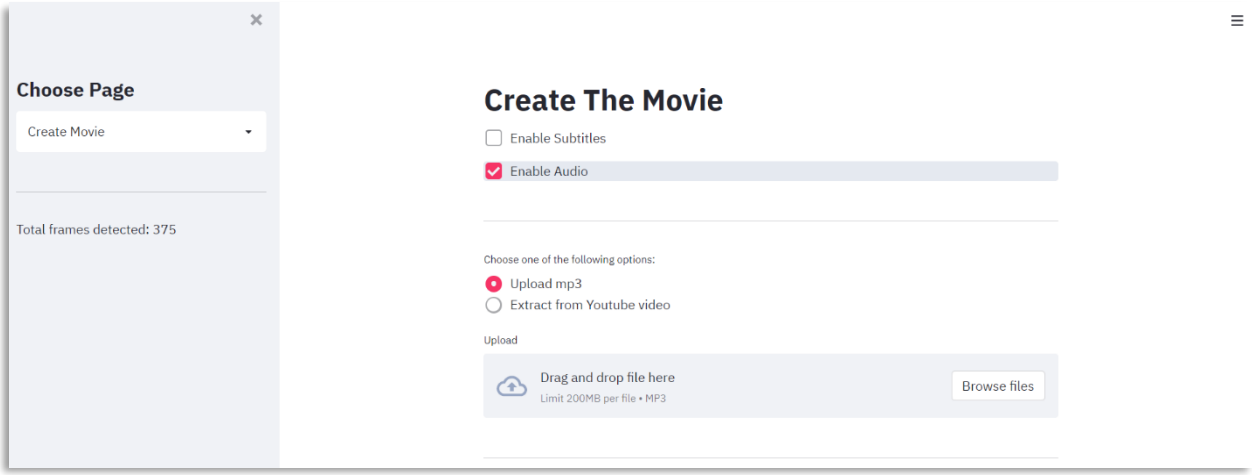


המחלקה להנדסת תעשייה
הפקולטה להנדסה ע"ש איבי ואלדר פליישמן
אוניברסיטת תל אביב

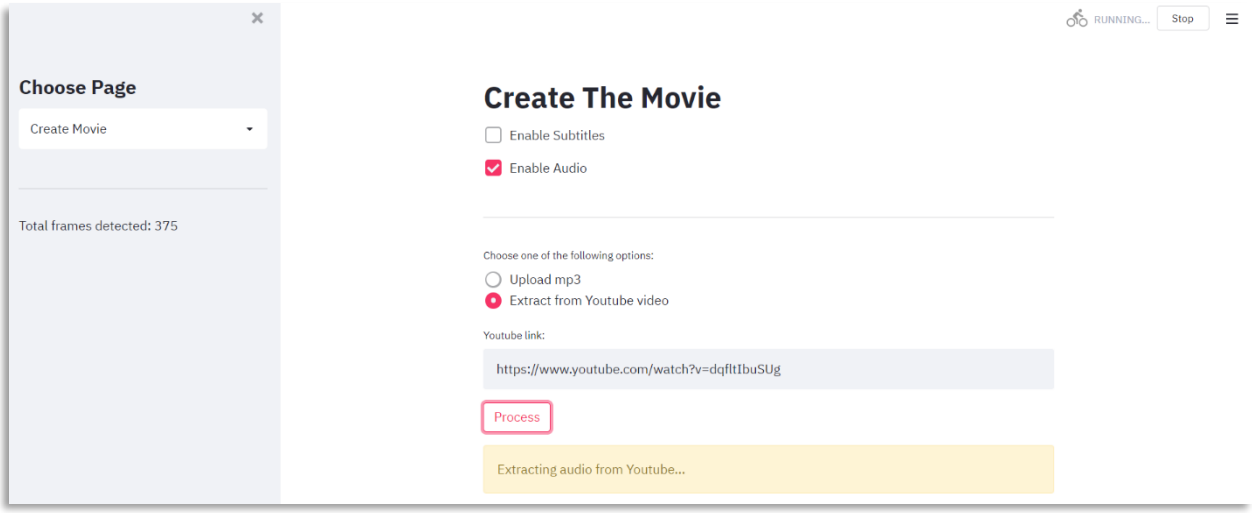
מסך יצירת הסרט עם כתוביות – ייבוא הכתוביות מקובץ טקסט –



מסך יצירת הסרט עם שמע – העלאת הקובץ כ mp3 -



מסך יצירת הסרט עם שמע – טעינת השמע מתוך סרטון Youtube –



מסך נגינת הסרט –