Hochschule Düsseldorf
Fachbereich Medien
Data Science, AI und Intelligente Systeme

Dokumentation zum Projekt:

Lyric to Video Generator

Von: Steffen Beginn
Matrikelnummer: 884917

Kurs: Advances in AI
Abgabedatum: 19.02.2024
Professor: Dr. Dennis Müller

GitHub Repository: https://github.com/sBeginn/Lyrics-to-Video-Generator

1 Inhaltsverzeichnis

2 Abbildungsverzeichnis

3 Einleitung

Das Projekt ist eine Prüfungsleistung aus dem Kurs „Advances in AI“, welches Teil des Studienganges „Data Science, AI und Intelligente Systeme“ an der Hochschule Düsseldorf ist.

Bei meinem Projekt handelt es sich um einen sogenannten Lyrics-to-Video Generator. Das Ziel meines Projektes ist es gewesen, durch einen beliebigen Songtext, ein passendes Musikvideo zu generieren. Dazu ist nur die Eingabe eines Künstlers mit dem jeweiligen Song und einem YouTube Link zu dem Song notwendig. Zunächst werden die wichtigsten Wörter aus dem Songtext gefiltert, damit die Bilder so genau wie möglich beschreiben, was in dem Song inhaltlich passiert. Anschließend werden anhand des Songtextes, durch ein KI-Modell Bilder generiert, welche in ein dynamisches Musikvideo umgewandelt werden. Damit das Musikvideo „aufregender“ wirkt, wurde das Ganze mit Übergängen und zufälligen Farben erweitert.
Das komplette Projekt wurde in der Programmiersprache Python umgesetzt.

Am Ende erhält man also ein Musikvideo, welches durch die erzeugten Bilder genau das abbildet, was in dem Songtext gesungen wird und bei jedem Neuen ausführen immer einzigartig ist. Um die Vorlieben jedes Nutzers zu erfüllen, kann man zudem die Variablen nach Belieben anpassen, um so ein Musikvideo nach seinen Wünschen zu erhalten.

In den folgenden Abschnitten erhalten Sie Einblicke über den bisherigen State-of-the-Art was die KI-Modelle auf die Erzeugung von Bildern angeht. Des Weiteren gehen wir detailliert auf den Aufbau des Codes meines Projektes ein, bei dem wir jede Funktion und deren Aufgabe erklären. Unter Technik/Methodik werden noch die anderen Bibliotheken und Modelle, welche für dieses Projekt nötig waren, ausführlich beschrieben. Zudem gibt es eine Anleitung für die optimale Implementierung und Nutzung meines Projektes. Weiter geht es mit Einblicken in den Entwicklungsprozess, wo die zeitliche Abfolge beschrieben wird, um genau zu erkennen, wann welche Fortschritte gemacht wurden. Außerdem gibt es Einblicke in die Probleme und Herausforderung die während des Entwicklungsprozesses entstanden sind und wie diese gelöst wurden. Dann kommen wir zu den Auswertungen vom Projekt, wo anhand von Beispiele beschrieben wird, welche Ergebnisse man mit dem Projekt erreichen kann. Am Schluss wird noch auf Verbesserungen eingegangen und die möglichen nächsten Schritte oder Anhaltspunkte erklärt, falls man das Projekt im nächsten Semester weiterführen möchte.

4 State-of-the-Art

Aktuell sind KI-Modelle wie DALL-E oder Midjourney unteranderem die Vorreiter, was den Bereich der Bild Generatoren angeht und somit State-of-the-Art.
DALL-E ist ein Modell, welches von OpenAI im Jahr 2021 entwickelt und veröffentlich wurde[1]. Midjourney wurde ebenfalls im Jahre 2021 veröffentlicht[2]. Beide Modelle sind kostenpflichtig. Dadurch das mir nur begrenzte Mittel zur Verfügung stehen, konnte ich die kostenpflichtigen Dienste dieser Modelle leider nicht nutzen, weshalb ich dementsprechend auch die API nicht in mein Projekt mit einbauen konnte.
Was den aktuellen State-of-the-Art angeht im Bereich kostenloser Bild Generatoren, ist Stable Diffusion von Hugging Face sehr weit vorne. Dieses KI-Modell habe ich selbst in diesem Projekt implementiert. Stable Diffusion ist nicht ganz auf dem Stand wie die kostenpflichtigen Modelle, liefert dennoch erstaunlich gute Ergebnisse. Dennoch muss man mit einer beachtlich hohen Wartezeit rechnen, wenn man das ganze Lokal auf seinem Computer ausführt, wenn man nicht über einen sehr Leistungsstarken Computer verfügt.

Was das aktuelle State-of-the-Art von Video-Generator aus einem Text angeht, sind die Fortschritte noch nicht so weit wie bei Text2Image Generatoren. Dennoch gibt es Möglichkeiten dieses umzusetzen, wie z.B. von „Runaway ML“ welche diese Funktion anbietet.[3] Dort wird durch die Eingabe von einem Text ein kurzes Video generiert, welches ebenfalls gute Ergebnisse liefert. Da das Generieren von Bildern schon Lokal eine hohe Rechenleistung beansprucht, habe ich davon abgesehen, solche möglichen Modelle wie „Runaway ML“ oder kostenlose Alternativen in meinem Projekt zu verwenden. Stattdessen wird in meinem Projekt, wie in der Einleitung erwähnt, das Video aus mehreren Bildern erzeugt. Dieser Ansatz erfordert zwar mehr Arbeitsschritt, hat jedoch den Vorteil, dass ich eine höhere Flexibilität habe und mehr Kontrolle über das Endprodukt. Zudem kann durch das Verwenden der Bilder ein viel besserer individueller Übergang im Video entstehen.

Kurz vor der Fertigstellung meines Projektes wurde zudem von OpenAI ein neues KI-Modell vorgestellt mit dem Namen „Sora“[4]. Damit lassen sich realistische und hochauflösende bis zur 1 Minute lange Video durch Eingabe von einem Text generieren. Während große Sprachmodelle über Text-Tokens verfügen, wird Sora mit sogenannten „visuellen Patches“ zur Video Generierung trainiert. Dabei wird das Video in ein niedrigdimensionalen Raum durch einen Video Encoder komprimiert und anschließend in sogenannte „Raum-Zeit Patches“ zerlegt, welche die zeitliche Abfolge mit repräsentieren.

[1] https://openai.com/dall-e-3
[2] https://www.midjourney.com/home
[3] https://runwayml.com/
[4] https://openai.com/sora

5 Codeanalyse

5.1 Hauptfunktion

Bei der Hauptfunktion handelt es sich um die Datei **main.py.** Diese wird benötigt, um das Programm vollständig auszuführen. Abgesehen von der Bibliothek „os“ werden in dieser Datei nur die anderen Funktionen aufgerufen.

```
15    current_path = os.path.dirname(__file__)
```

Abbildung 1: Pfad zur aktuellen Datei

Dabei startet der Code in Zeile 15, welche dazu dient, dass der aktuelle Pfad zur Datei in einer Variable gespeichert wird.

```
18    folder_generated_images = current_path + "/generated_images"
19    folder_generated_video = current_path + "/generated_video"
```

Abbildung 2: Pfade zu den Ordnern

In der Zeile 18 & 19 erhält man die jeweiligen Variablen zu dem Ordner, wo die generierten Bilder gespeichert werden und zum Ordner wo man am Ende das Video findet. Dabei wird der aktuelle Pfad (siehe Abbildung 1) einfach nur um den jeweiligen Ordner erweitert.

```
22    if not os.path.exists(folder_generated_images):
23        os.makedirs(folder_generated_images)
24
25    if not os.path.exists(folder_generated_video):
26        os.makedirs(folder_generated_video)
```

Abbildung 3: Erstellt Ordner (Bilder & Video)

In der Zeile 22&23 als auch 25&26 wird mit einer If-Abfrage überprüft, ob die beiden Ordner bereits existieren und falls nicht, werden diese angelegt.

```
29    def main():
30        input_songname = input("Songname:")
31        input_artist = input("Artist:")
32        input_url = input("url:")
33
34        samples_warp = song_duration(input_songname, input_artist)
35
36        read_lyrics(input_songname, input_artist)
37        spacy_words(input_songname, input_artist)
38        generate_image(input_songname, input_artist)
39        generate_rotated_images(samples_warp)
40        generate_video()
41        download_audio(input_url)
42        audio_with_video_connection()
```

Abbildung 4: main-Funktion

Die main-Funktion beläuft sich dabei von der Zeile 29-42 und der Aufruf der Funktion in Zeile 44. Da dies die main-Funktion ist, welche ausgeführt werden muss, befinden sich dort innerhalb alle anderen Funktionen aus den anderen Dateien. In den Zeilen 30-32 werden die Daten zum Song (Songname, Artist & URL) gesammelt. Wenn der Code ausgeführt wird, werden diese drei Abfragen im Terminal aufgerufen. Die Variable in Zeile 34 berechnet die Songlänge, welche einen wichtigen Faktor hat. In den restlichen Zeilen 36-42 werden die Funktionen aufgerufen. Jeder der Funktionen ist voneinander unabhängig und kann separat ausgeführt werden in dem die nicht benötigten Funktionen einfach ausgeklammert werden.

5.2 Songtexte einlesen

Bei der Funktion handelt es sich um die Datei **function_read_lyrics.py,** welche dafür genutzt wird, um die Songtexte einzulesen. Die benutzten Bibliotheken dabei sind spacy, scikit-learn, os, lyricsgenius, re und random.

```
 9   token = "SJT0EPoDLSVfMvvdwbQ41GTyplVGUIjG_N8UxVSCpNtyY6mbtSEq4FrwfxEKwRuD"
10   genius = Genius(token)
```

Abbildung 5: Genius Token

In der Zeile 9 wird in der Variable der Token eingetragen. Wie dieser zu erhalten ist, wird im Punkt 6.2 Implementierung detailliert erklärt. In Zeile 10 wird anschließend dieser Token durch die Bibliothek „lyricsgenius“ aufgerufen. In den nachfolgenden Zeile 13 wird wieder der aktuelle Pfad angegeben (siehe Abbildung 1).

```
16   def remove_bracketed_text(text):
17       return re.sub(r'\[.*?\]', '', text)
```

Abbildung 6: Funktion remove_bracketed_text

Bei der Funktion remove_bracketed_text in Zeile 16&17 handelt es sich um eine Funktion, die dazu dient, überflüssige Zeichen aus dem Songtext die in der Textdatei enthalten ist zu entfernen. Diese Funktion wird in der Funktion read_lyrics aufgerufen.

```
20 def read_lyrics(input_songname, input_artist):
21
22     song = genius.search_song(input_songname, input_artist)
23
24     file_lyrics = current_path + f"//lyrics_{input_songname}_{input_artist}.txt"
25
26     if song:
27
28         original_lyrics = song.lyrics
29
30         filtered_lyrics = remove_bracketed_text(original_lyrics)
31
32         filtered_lyrics = '\n'.join(filtered_lyrics.splitlines()[1:])
33
34         myfile = open(file_lyrics, "w", encoding="utf-8")
35         myfile.write(filtered_lyrics)
36         myfile.close()
37
38     else:
39         print("Not found")
```

Abbildung 7: read_lyrics Funktion

In der folgenden Funktion read_lyrics welche über die Zeilen 20-39 läuft, wird durch die Parameter Songname und Artist, welche man in der main Funktion eingetragen hatte (siehe Abbildung 4) aufgerufen. Dabei wird in Zeile 22 der Song über die Funktion von genius gesucht. In der Zeile 24 wird die dazu gehörige Textdatei, wo der Songtext später gespeichert werden soll, erstellt. Als nächstes folgt eine Abfrage, die dazu dient, um herauszufinden, dass der Songtext gefunden werden konnte. Innerhalb der Abfrage wird, die vorher erklärte remove_bracketed_text Funktion (siehe Abbildung 6) aufgerufen und noch die erste Zeile aus der Textdatei gelöscht. Über den Zeilen 34-36 wird die Textdatei geöffnet und der Songtext reingeschrieben und geschlossen.

```
49  def spacy_words(input_songname, input_artist):
50
51      nlp = spacy.load("en_core_web_sm")
52
53      file_lyrics = current_path + f"//lyrics_{input_songname}_{input_artist}.txt"
54
55      with open(file_lyrics, "r", encoding="utf-8") as file:
56          lyrics = file.read()
57
58      total_length = len(lyrics)
59      part_length = total_length // 3
60
61      part1 = lyrics[:part_length]
62      part2 = lyrics[part_length:2 * part_length]
63      part3 = lyrics[2 * part_length:]
```

Abbildung 8: spacy_words Funktion (1)

Die nächste Funktion spacy_words beläuft sich über die Zeilen 49-88. In der Zeile 51 wird dabei aus der Bibliothek spacy das nötige Wörterbuch (Englisch) geladen. Es ist möglich dieses anzupassen, falls man eine andere Sprache nutzen möchte. In den nachfolgenden Zeilen wird die eben erzeugte Textdatei mit dem Songtext geöffnet und eingelesen. Anschließend wird in Zeile 58 die Länge des Songtextes berechnet und in der Zeile darunter in drei Parts aufgeteilt.

```
65  for i, part in enumerate([part1, part2, part3]):
66
67      about_doc = nlp(part)
68
69      unique_nouns = set()
70      #unique_adjectives = set()
71
72      for token in about_doc:
73          if token.pos_ == "NOUN" and token.lemma_ in nlp.vocab:
74              noun_lemma = token.lemma_.lower()
75
76              unique_nouns.add(noun_lemma)
77
78          #if token.pos_ == "ADJ" and token.lemma_ in nlp.vocab:
79              #adj_lemma = token.lemma_.lower()
80              #unique_adjectives.add(adj_lemma)
81
82      selected_nouns = random.sample(list(unique_nouns), min(2, len(unique_nouns)))
83      #selected_adjectives = random.sample(list(unique_adjectives), min(1, len(unique_adjectives)))
84
85      summary_file = current_path + f"//lyrics_{input_songname}_{input_artist}_part{i+1}.txt"
86      with open(summary_file, "w", encoding="utf-8") as file:
87          file.write(", ".join(selected_nouns) + ", ")
88          #file.write(", ".join(selected_adjectives))
```

Abbildung 9: spacy_words Funktion (2)

Im zweiten Abschnitt der Funktion wird eine for-Schleife genutzt, die durch alle drei Parts iteriert. Dabei wird zunächst in Zeile 69 ein Set erstellt, um die Wörter, welche am Ende übrig bleiben in die Textdatei zu schreiben. Hier wurde ein Set ausgewählt, damit wird am Ende keine Duplikate in der Textdatei stehen haben. Ab der Zeile 72 geht es darum, dass der vorherige eingelesene Songtext aus der Funktion read_lyrics (siehe Abbildung 7) genommen wird und alle Nomen, die in diesem Text enthalten sind, herausgefiltert und in das Set hinzugefügt werden. Danach wird in Zeile 82 noch festgelegt, dass es 2 Nomen sein sollen. In der Zeile 85 wird dann für die jeweiligen Parts eine Textdatei erstellt und in den nachfolgenden Zeilen die vorher herausgefilterten Nomen durch Zufall in die Textdateien geschrieben wird. Wie man in der Abbildung sieht, wurde sich nur auf die Nomen konzentriert und der andere Teil ausgeklammert.

5.3 Bilder generieren

Bei dieser Funktion handelt es sich um die Datei function_generate_images.py, welche genutzt wird, um die nötigen Bilder zu generieren. Die dafür genutzten Bibliotheken sind diffusers, torch matplotlib, os und random.
Zunächst wird wieder mal in Zeile 8 der Pfad zur aktuellen Datei angegeben.

```
11    pretrained_model = "runwayml/stable-diffusion-v1-5"
```

Abbildung 10: Vortrainiertes Modell

In der nächsten Zeile 11 wird dann das benutzte vortrainierte Modell angegeben. In diesem Fall ist es Stable Diffusion 1.5, aber dies kann mit anderen Modellen ausgetauscht werden.

```
14    steps = 100
15    samples = 1
```

Abbildung 11: Steps & Samples

In den Zeilen 14 & 15 werden die nötigen Steps und Samples festgelegt. Unter Steps versteht man die nötigen Schritte, welche benötigt werden, um die Bilder zu generieren. Als Empfehlung sollten es für einfache Bilder mindestens 25 Steps eingestellt werden. Im besten Fall jedoch 50-100. Man muss dennoch beachten, dass so höher die Steps Anzahl ist, so detaillierter werden die Bilder aber dementsprechend verlängert sich auch der Prozess zum Generieren der Bilder. Bei Samples handelt es sich um die Anzahl der Bilder, die erzeugt werden sollen. Wenn die Anzahl der Samples auf 1 gestellt ist, bedeutet das, dass drei Bilder erzeugt werden, weil es drei Parts sind.

```
18  def generate_image(input_songname, input_artist):
19
20      output_folder = current_path + "/generated_images/"
21
22      text_files = [
23          current_path + f"/lyrics_{input_songname}_{input_artist}_part1.txt",
24          current_path + f"/lyrics_{input_songname}_{input_artist}_part2.txt",
25          current_path + f"/lyrics_{input_songname}_{input_artist}_part3.txt"
26      ]
27
28      image_counter = 1
```

Abbildung 12: generate_image Funktion (1)

In den Zeilen 18-49 wird die Funktion generate_image aufgerufen. Diese benötigt als Parameter ebenfalls die Parameter Songname und Artist.
In Zeile 20 wird der Ordner angegeben wo die später generierten Bilder gespeichert werden. In den Zeilen 22-26 werden dann die drei Textdateien in einer Liste aufgelistet durch die später iteriert wird. Zudem wird ein image_counter erstellt, welcher die bisherigen generierten Bilder zählt und bei jedem neuen generierten Bilder um eins erweitert wird, bis die in samples festgelegte Zahl erreicht ist.

```
30      for text_file in text_files:
31
32          pipeline = AutoPipelineForText2Image.from_pretrained(
33              pretrained_model,
34              torch_dtype=torch.float16,
35              safety_checker=None,
36              requires_safety_checker=False
37
38          ).to("cuda")
39
40          with open(text_file, "r", encoding="utf-8") as file:
41              prompt = file.read()
42
43          for sample in range(samples):
44              image_path = output_folder + f"output_{image_counter:03d}.png"
45              image_counter += 1
46
47              generated_image = pipeline(prompt, num_inference_steps=steps).images[0]
48
49              generated_image.save(image_path)
```

Abbildung 13: generate_image Funktion (2)

Im nächsten Abschnitt der Funktion wird mit einer for-Schleife gearbeitet, die durch alle drei Textdateien der jeweiligen Parts läuft. Dabei wird das vortrainierte Modell (siehe Abbildung 10) aufgerufen und mit einigen weiteren Informationen erweitert. Bei einem Leistungsstärkeren Computer kann die Zahl der Bits in Zeile 34 auch auf 32 oder 64 angepasst werden. Der safety_checker in Zeile 35&36 wurde deaktiviert, da dieser zu Fehlermeldungen geführt hat. Da mein Computer eine Grafikkarte besitzt, konnte ich in Zeile 38 festlegen, dass diese durch cuda genutzt wird anstatt nur meine CPU. Anschließend wird in Zeile 40&41 die Textdatei geöffnet und gelesen. In der nächsten for-Schleife wird dann das Bild generiert. Der prompt ist dabei, die zuvor zwei festgelegten Nomen, welche wir in die Textdatei geschrieben haben. Zum Schluss werden die Bilder in den angegebenen Ordner gespeichert.

5.4 Übergangsbilder generieren

In der nächsten Datei function_warp_affine.py werden die Übergangsbilder generiert, welche dazu dienen sollen das Video dynamischer und aufregender zu gestalten in dem im Hintergrund die Farbe wechselt und sich das Bild dreht. Die hierfür genutzten Bibliotheken sind cv2, numpy, os, spotipy und random.

```
11  SPOTIPY_CLIENT_ID = 'e1fd87ab69f3494c8955baf382855de8'
12  SPOTIPY_CLIENT_SECRET = 'a29340abbe9d495aa77f88ae96da3e12'
13  SPOTIPY_REDIRECT_URI = 'http://localhost:3000'
```

Abbildung 14: Spotify Verbindung

Als erstes werden in Zeile 11-13 die Informationen von den beiden Token eingetragen. Wie man die beiden Token erhält, wird unter 6.2 Implementierung erklärt.

```
16  sp = spotipy.Spotify(auth_manager=SpotifyOAuth(
17      client_id=SPOTIPY_CLIENT_ID,
18      client_secret=SPOTIPY_CLIENT_SECRET,
19      redirect_uri=SPOTIPY_REDIRECT_URI,
20      scope='user-library-read'))
```

Abbildung 15: Spotipy Bibliothek Verbindung

In den Zeilen 16-20 wird über die vorher beiden eingetragenen Token eine Verbindung mit der spotipy Bibliothek hergestellt.

```
29  width_image = 3840
30  height_image = 2160
31  resize_factor = 3
```

Abbildung 16: Bildinformationen

In den Zeilen 29 & 30 wird die Höhe und die Breite für das gesamte Bild festgelegt, welche zur Auflösung dient. Der resize_factor in Zeile 31 hilft dabei das Eingabebild zu vergrößern.

```
34  def generate_rotated_images(samples_warp):
35
36      for file_name in os.listdir(images_path):
37
38          if file_name.endswith(('.jpg', '.jpeg', '.png')):
39
40              img_path = os.path.join(images_path, file_name)
41              img = cv.imread(img_path)
42
43              img = cv.resize(img, (int(img.shape[1] * resize_factor), int(img.shape[0] * resize_factor)))
44
45              canvas = np.zeros((height_image, width_image, 3), dtype=np.uint8)
46              random_background_color = (random.randint(0, 255), random.randint(0, 255), random.randint(0, 255))
47              canvas[:, :] = random_background_color
48
49              x_offset = (width_image - img.shape[1]) // 2
50              y_offset = (height_image - img.shape[0]) // 2
51
52              canvas[y_offset:y_offset + img.shape[0], x_offset:x_offset + img.shape[1]] = img
53
54              original_file = os.path.join(current_path, "generated_images", f'{file_name}_original.png')
55              cv.imwrite(original_file, canvas)
```

Abbildung 17: generate_rotated_images Funktion (1)

Die Funktion generate_rotated_images läuft über die Zeilen 34 bis 72. Als Parameter bekommt diese Funktion den Wert samples_warp, welcher in der nächsten Funktion song_duration festgelegt wird. Samples_warp ist dabei die Anzahl der Bilder, die generiert werden, muss. Zunächst wird in Zeile 36 durch eine for-Schleife durch alle Dateien in dem generated_images Ordner durchgegangen und wie in Zeile 38 angegeben, mit allen generierten Bildern weitergearbeitet. Die in der Funktion generate_image generierten Bilder werden zunächst eingelesen und auf eine einheitliche Größe angepasst (siehe Zeile 43). Über die Zeilen 45-47 wird dann ein Hintergrund erzeugt, welcher eine zufällige Farbe enthält, die bei jedem Bild wechselt. Die Zeilen 49&50 sind dafür da, dass die Mitte vom gesamten Bild festgelegt werden kann, damit anschließend die vorher generierten Eingabebilder genau in die Mitte platziert werden.

```
57              for i in range(samples_warp):
58
59                  angle = i * (360 / samples_warp)
60
61                  rot_mat = cv.getRotationMatrix2D((width_image // 2, height_image // 2), angle, 1.0)
62
63                  rotated_img = cv.warpAffine(canvas, rot_mat, (width_image, height_image))
64
65                  alpha = i / samples_warp
66
67                  blended_img = cv.addWeighted(rotated_img, alpha, canvas, 1 - alpha, 0)
68
69                  output_file = os.path.join(current_path, "generated_images", f'{file_name}_rotation_{i+1:04d}.png')
70                  cv.imwrite(output_file, blended_img)
71
72              os.remove(img_path)
```

Abbildung 18: generate_rotated_images Funktion (2)

Die nächste for-Schleife dient dazu, dass das aktuelle Eingabebild, welches sich in der Mitte befindet, noch einmal im Hintergrund angezeigt wird und sich rotiert. Dazu wird das sogenannte warpAffine von der Bibliothek cv2 genutzt, was man in Zeile 63 erkennen kann. Zum Schluss werden die Bilder gespeichert und die drei Ausgangsbilder werden gelöscht.

```
76  def song_duration(input_songname, input_artist):
77
78      song_name = input_songname
79      artist_name = input_artist
80
81      results = sp.search(q=f'track:{song_name} artist:{artist_name}', type='track', limit=1)
82
83      if results['tracks']['items']:
84          track_info = results['tracks']['items'][0]
85          track_id = track_info['id']
86
87          duration_ms = track_info['duration_ms']
88
89          duration_sec = duration_ms // 1000
90
91          samples_warp = int((duration_sec * function_images_to_video.fps) / (function_generate_images.samples * 3))
92
93          return samples_warp
94
95      else:
96          print(f'No Song found')
```

Abbildung 19: song_duration Funktion

Die Funktion song_duration dient dazu, herauszufinden wie lang ein Song dauert, um die Anzahl der nötigen rotierenden Bilder zu ermitteln. In Zeile 81 wird der Song gesucht und die Songdauer in Sekunden in Zeile 89 in die Variable geschrieben. In der Zeile 91 wird dann die nötige Anzahl der rotierenden Bilder ermitteln, da das generierte Video am Schluss genau die gleiche Dauer haben soll wie der Song. Um das umzusetzen, wird die Songdauer in Sekunden genommen und mit der FPS multipliziert. Dieser Wert wird dann durch die Anzahl der Samples(multiplizieren mit drei, weil samples ja nur eins ist, aber es drei Bilder sind) geteilt. Jetzt haben wir als integer die Anzahl der Übergangsbilder von jedem generierten Bild.

5.5 Bilder in Video umwandeln

Die Datei function_images_to_video ist dafür da, um die bisher generierten Bilder nun in ein Video umzuwandeln. Dafür wird die Bibliothek cv2 und os verwendet.

```
18    fps = 20
```

Abbildung 20: FPS

Als erstes legen wir die Anzahl der FPS (Frames pro Seconds) fest. Als Empfehlung sollte hier mindestens 10 angegeben werden, da dadurch die Rotation deutlich flüssiger ist.

```
21  def generate_video():
22
23      images_array = []
24
25      for image_file in sorted(os.listdir(path_images)):
26          image_path = os.path.join(path_images, image_file)
27          images_array.append(image_path) # Add the images to the list
28
29      if not images_array:
30          print("No generated images")
31          return
32
33      created_video = cv2.VideoWriter(path_video, cv2.VideoWriter_fourcc(*'mp4v'), fps, (width_image, height_image))
34
35      for image in images_array:
36          img_read = cv2.imread(image)
37          img_resize = cv2.resize(img_read, (width_image, height_image))
38          created_video.write(img_resize)
39
40      created_video.release()
41
42      for image in images_array:
43          os.remove(image)
```

Abbildung 21: generate_video Funktion

In der Funktion generate_video welche in der Zeile 21 bis 43 läuft, wird das Video generiert. Zunächst wird eine Liste in der Zeile 23 für die Bilder erstellt. Als nächstes läuft eine for-Schleife durch den ganzen Ordner, in dem sich die generierten Bilder befinden und fügt diese der Liste hinzu (Zeile 25-27). In Zeile 33 werden dann die Einstellungen festgelegt (Video im mp4 Format) und die Funktion für das Video aufgerufen. Anschließend durchläuft eine for-Schleife die ganze Liste mit den Bildern und schreibt diese in das Video. Am Ende werden dann noch die Bilder, welche sich noch im Ordner befinden, gelöscht.

5.6 Audio zum Video hinzufügen

In der Datei function_audio_connection wird zu dem zuvor generierten Video noch der Song als Audio hinzugefügt. Dazu werden die Bibliotheken pytube und moviepy verwendet.

```
15    audio_path = current_path + "/generated_video/output_audio.mp4"
```

Abbildung 22: Pfad zur Audio

In den Zeilen 6, 9 und 12 werden wie schon in den anderen Dateien die Pfade festgelegt. In der Zeile 15 ist dabei ein neuer Pfad, der dazu kommt, um den Ort der Audiodatei festzulegen.

```
18    finished_video_path = current_path + "/generated_video/output_finished_video.mp4"
```

Abbildung 23: Pfad zum fertigen Video

In der Zeile 18 kommt noch der Pfad für das fertige Video am Ende dazu.

```
21    def download_audio(input_url):
22        
23        yt = YouTube(input_url)
24        stream = yt.streams.get_by_itag(140)
25        stream.download(output_path=folder_video_path,filename="output_audio.mp4")
```

Abbildung 24: download_audio Funktion

Die Funktion download_audio dient dazu das die entsprechende Audiodatei gedownloadet wird. Das geschieht mit der Bibliothek pytube, mit der wir das Video von Youtube runterladen können. Als benötigten Parameter wird dafür die URL zu dem Song auf Youtube hinzugefügt.

```
28    def audio_with_video_connection():
29
30        video = VideoFileClip(video_path) # Video
31        audio = AudioFileClip(audio_path) # Audio
32
33        finished_video = video.set_audio(audio)
34        finished_video.write_videofile(finished_video_path, codec='libx264', audio_codec='aac')
35        
36        os.remove(video_path)
37        os.remove(audio_path)
```

Abbildung 25: audio_with_video_connection Funktion

Die Funktion audio_with_video_connection läuft über die Zeilen 28-37. Dort wird zunächst der Pfad für die Audiodatei und das Video in Zeile 30 und 31 angegeben. Als nächstes wird in Zeile 33 die Audiospur auf das Video gesetzt und dieses gespeichert. Zum Schluss werden noch die beiden einzelnen Dateien gelöscht, sodass nur das fertige Video übrig bleibt.

Technik/Methodik

5.7 Bibliotheken

Bibliothek	Beschreibung
os	Interaktion mit dem Betriebssystem und um Dateien und Pfade abzurufen
re	Verarbeitung von regulären Ausdrücken
random	Zufallsbasierte Wiedergabe von Zahlen
scikit-learn	Maschinelles Lernen
lyricsgenius	Enthalten alle Songtexte
spacy	Verarbeitung von Texten (NLP)
diffusers	Zum Generieren von Bildern
torch	Maschinelles Lernen
matplotlib	Visualisierung
cv2	Bild- und Videobearbeitung
numpy	Mathematische Berechnungen
spotipy	Verbindung mit Spotify für Informationen über Songs (wie Songdauer etc.)
pytube	Herunterladen von YouTube Videos
moviepy	Bearbeitung von Videos

5.8 Implementierung

Das vollständige Projekt finden man in GitHub unter:

https://github.com/sBeginn/Lyrics-to-Video-Generator

Die Anleitung für die Implementierung findet man ebenfalls ausführlich in der enthaltenen Readme Textdatei, wo jeder Schritt einzeln in Form einer Checkliste genau erklärt wird, um das Programm auszuführen.

Als ersten Schritt ist es wichtig das Repository auf GitHub auf deine Lokale Festplatte zu kopieren:
git clone https://github.com/sBeginn/Lyrics-to-Video-Generator.git

Als nächstes muss die Eingabeaufforderung (bei Windows „cmd“) geöffnet werden und der Pfad zum gerade eben kopierten Repository angegeben werden:
cd path\to\folder

Nun wird die requierements.txt Datei ausgeführt. Diese enthält alle nötigen Bibliotheken, welche installiert werden. Das hat den Vorteil, dass man nicht alle Bibliotheken einzeln installieren muss:
pip install -r requirements.txt

Als nächstes muss die Verbindung zu https://genius.com/ hergestellt werden, da wir deren Python Bibliothek „lyricsgenius“ nutzen, um die Songtexte zu erhalten.

Dafür rufen wir folgende Seite auf und erstellen einen Account oder loggen uns ein, falls bereits ein Account vorhanden ist:
https://genius.com/api-clients

Als nächstes klicken wir auf „New API Client“ und tragen die nötigen Informationen. Es ist nicht relevant was dort eingetragen wird. Danach klicken wir auf „Generate Access Token“ und kopieren den dort angezeigten Token. Dieser wird nun in der folgenden Datei:
function_read_lyrics.py

eingetragen unter der Variable:

- token = (Dein Token)

Der letzte Abschnitt ist das wir ebenfalls für Spotify einen Account anlegen oder mit einem bereits angelegten Account einloggen unter:
https://developer.spotify.com/

Als nächstes öffnen wir das Dashboard:
https://developer.spotify.com/dashboard

Dort klicken wir auf „create app“ und geben unter Redirect URL folgendes ein:
http://localhost:3000

Falls man einen anderen Port als „3000“ verwenden möchte kann man das natürlich anpassen, jedoch muss man beachten, dass dies unter dem späteren Punkt im Code auch dementsprechend angepasst werden muss. Was man bei den anderen Eingabefeldern einträgt, ist unrelevant. Klick man auf „Settings“ erhalten wir nun unseren „Client ID“ und „Client Secret“, welche wir kopieren und in die Datei:
function_warp_affine.py

eintragen und folgende Zeilen anpassen:

- SPOTIPY_CLIENT_ID = (Dein Client ID)
- SPOTIPY_CLIENT_SECRET = (Dein Client Secret)
- SPOTIPY_REDIRECT_URI = http://localhost:3000 (Anpassen falls anderer Port)

Nun ist die Installation erfolgreich abgeschlossen und wir können das Programm nutzen. Dafür öffnen wir die **main.py** Datei und führen diese aus. Als nächstes müssen nur noch die Informationen zu dem beliebigen Song, zu dem ein Musikvideo erstellt werden soll, eingetragen werden:

- Songname (Der komplette Songname, wobei die Groß- und Kleinschreibung keine Rolle spielt)
- Artist (Dabei reicht der Hauptinterpret und mögliche Features sind nicht nötig)
- URL (Ein YouTube Link zu dem Song, damit dieser im Video enthalten ist)

Das fertige Video ist anschließend im Ordner **„generated_video“** zu finden.

5.9 Entwicklungsprozess/Zeitplan

02.10.2023:
- ❖ Ideenfindung

23.10.2023:
- ❖ Recherche für passende Bibliotheken & APIs

30.10.2023:
- ❖ Installation & testen von benötigten Bibliotheken & APIs:
  - API für Songtexte in Python -> Bibliothek: lyricgenius
    - Auf Genius registriert, um Token zu erhalten
    - Bibliothek in Python installiert
  - API für Songsuche -> Bibliothek: spotipy
    - Auf Spotify registriert, um Token zu erhalten
    - Bibliothek in Python installiert
- ❖ Möglichkeiten ausprobiert, wie man mehrere Bilder aneinander zum Video umwandeln kann -> Mit „opencv“ möglich

06.11.2023:
- ❖ Bibliothek für Text2Image Generator gefunden -> Python Diffusers
  - Installiert und verschiedene vortrainierte Modelle ausprobiert (Stable-Diffusion verschiedene Versionen, karlo, kandinstky)

Problem:
Fehlermeldung erhalten: "Potential NSFW content was detected in one or more images. A black image will be returned instead. Try again with a different prompt and/or seed."
Bei jedem Bild, welchem ich generieren wollte, erhielt ich den Fehler.
Lösung: Durch Recherche im Internet bin ich darauf gestoßen, dass ich den Safety Checker deaktivieren muss

- ❖ Für Stable-Diffusers v1-5 entschieden -> lieferte mir die besten Bilder

18.11.2023:
- ❖ Funktion geschrieben, um die Bilder mit einem Prompt zu generieren
- ❖ Funktion geschrieben, um Bilder in ein Video umzuwandeln[5]

Problem:
Video wurde nirgendswo gespeichert, aber auch keine Fehlermeldung.
Mit „hardcoded Pfad“ hat es funktioniert, aber nicht, wenn ich versucht habe mit: *path_video = os.path.join(os.getcwd(), "Lyrics-to-Video", "generated_video", "output_video.mp4")*
den Pfad zu entnehmen, damit wenn jemand anderes den Code ausführt, nicht erst den Pfad anpassen muss ->

[5] https://theailearner.com/2018/10/15/creating-video-from-images-using-opencv-python/

Lösung: ChatGPT hat Lösung gegeben ->
Erst aktuellen Pfad:
current_path = os.path.dirname(__file__)
und dann zum Video:
path_video = current_path + r"\generated_video\output_video.mp4"

- Man muss jetzt nur einen beliebigen Prompt in der main.py angeben und es wird ein Bild generiert und anschließend in ein Video umgewandelt

20.11.2023:

- Funktion eingefügt, dass sobald ein Video erstellt wurde, die generierten Bilder aus dem Ordner gelöscht werden, sodass nur das Video übrig bleibt
- Funktion geschrieben, dass Nutzer einen Songnamen und Artist eingeben können und der Songtext wird in einer extra Textdatei gespeichert
- Hinzugefügt, dass gewisse Wörter, die nichts mit dem Songtext zu tun haben, aus der Datei gelöscht werden (sowas wie [Hook] oder [Part 1] usw.)
    - Bibliothek Bert Summarizer installiert und ausprobiert, damit aus den Songtexten die wichtigsten Lines gefiltert werden[6]

26.11.2023:

- Neuen Text Summarizer ausprobiert(Hat meiner Meinung nach besser funktioniert).[7]
  Das ist ein abstraktiver Summarizer, welcher eine neue Darstellung des Textinhaltes wiedergibt ohne die genauen Sätze aus dem Originaltext.
- Kompletter Ablauf funktioniert jetzt, indem man nur noch einen Songnamen und Artist eingeben muss -> Der Songtext wird in einer Textdatei gespeichert und mit dem Summarizer wird der Songtext mit eigenen Worten vom Summarizer in einem Satz zusammengefasst. Danach wird der Satz genommen und daraus werden drei Bilder generiert, welche in ein Video umgewandelt wird

28.11.2023:

- Mit den Übergängen für die Bilder beschäftigt. Hinweis von unserem Professor Dennis Müller bekommen, dass man dafür WarpAffine von OpenCV benutzen kann. Mit der Funktion lassen sich Bilder drehen[8]
- Weitere Funktion von OpenCV ist lineare Blending, mit der ich die Bilder übereinanderlegen kann, um einen Übergang zu erzeugen.
  Angefangen Funktion zu schreiben, damit mit den Übergängen ein Video entsteht[9]

01.12.2023:

- Die Funktion damit die Übergänge zwischen den generierten Bildern funktioniert, fertig gestellt.

[6] https://pypi.org/project/bert-extractive-summarizer/
https://huggingface.co/docs/transformers/tasks/summarization
[7] https://towardsdatascience.com/abstractive-summarization-using-pytorch-f5063e67510
[8] https://docs.opencv.org/4.x/d4/d61/tutorial_warp_affine.html
[9] https://docs.opencv.org/4.x/d5/dc4/tutorial_adding_images.html

Problem:
Fehlermeldung erhalten:
cv2.error:OpenCV(4.8.1)D:\a\opencv-python\opencv-python\opencv\modules\core\src\arithm.cpp:650: error: (-209:Sizes of input arguments do not match) The operation is neither 'array op array' (where arrays have the same size and the same number of channels), nor 'array op scalar', nor 'scalar op array' in function 'cv::arithm_op'
Lösung: Keine Lösung im Internet gefunden -> ChatGPT gefragt:
Es musste Zeile:
src1 = cv.imread(images_array[i])
und Zeile:
src2 = cv.imread(images_array[i+1])
noch die *width_image* und *height_image* hinzufügt werden.
Alle Bilder müssen dieselbe Größe vorweisen

08.12.2023:

- Anfangen die Webanwendung zu programmieren mit der Bibliothek Streamlit. Eingabefeld für Songname und Artist mit einem Button zum „Generieren“ erstellt und noch ein Feld, wo das Video abgespielt wird.

Problem:
Wenn ich ein Video generiere und das dann auf der Webanwendung abspielen will, funktioniert das nicht, sondern nur schwarzer Bildschirm. Wenn ich ein YouTube Link oder ein anderes Video als Test in den Code einsetze, spielt er das Video ab.
Lösung: Zum derzeitigen Standpunkt noch keine Lösung dafür gefunden

- Text Summarizer weiter angepasst und besser optimiert, damit der Songtext in drei Teile unterteilt wird, damit drei unterschiedliche Bilder generiert werden

Problem:
Schwierigkeiten damit, wie ich den Text am besten aufteilen soll, damit ich am Ende aus dem Songtext, drei verschiedene Sätze generieren kann und diese auch in der richtigen Reihenfolge hintereinander ablaufen.
Lösung: ChatGPT genutzt, um eine mögliche Lösung zu finden:

```
total_length = len(text)
part_length = total_length // 3

part1 = text[:part_length]
part2 = text[part_length:2 * part_length]
part3 = text[2 * part_length:]
```

11.12.2023:

- Neue Funktion geschrieben damit der Songtext verarbeitet wird mit Bibliothek „SpaCy“ (ohne einen Summarizer diesmal)[10]:
  - Dabei habe ich mir überlegt, dass der Songtext immer noch in drei Parts aufgeteilt wird, aber durch die Bibliothek „SpaCy“, werden nur die Nomen und Verben aus jedem Part rausgeschrieben und in die Text-Datei geschrieben und nicht mehr ein ganzer Satz. So stehen nämlich nur noch die „wichtigsten“ Wörter in der Datei und der Generator hat mehr Spielraum. Durch Ausprobieren ist mir bei der Variante mit dem Summarizer aufgefallen, dass oft zum einen die Sätze, die er aus den Parts generiert nicht so gut sind und die generierten Bilder ebenfalls nicht. Funktion so erweitert, dass nur jedes Wort einmal in die Text-Datei geschrieben wird und zudem alle Buchstaben klein sind und die Groß- und Kleinschreibung ignoriert wird.

    Problem:
    Wörter, die sehr ähnlich sind, in meinem Beispiel das Wort „thing“ und „things“, beide Wörter am Ende in der Text-Datei standen. Wollte das nur immer eins davon drin steht.
    Lösung: ChatGPT gefragt ->
    Hat mir Lemmatisierung vorgeschlagen. Dafür gibt es in der Bibliothek „SpaCy“ den Befehl: *.lemma_*, welches dafür sorgt, dass ähnliche Wörter, mit der gleiche Bedeutung, auf ihre Grundform gesetzt werden.

29.12.2023:

- Die Datei „function_transitions“ nochmal überarbeiten und in der Funktion „Transitions“ noch „sorted“ hinzugefügt, damit die Bilder in der richtigen Reihenfolge sind, wenn das Video generiert wird.
- Die Datei „function_read_lyrics“ überarbeitet -> bisher mit Nomen und Adjektiven zu viele Wörter enthalten -> Keine guten Bilder werden generiert
  - In der Funktion „spacy_words“ hinzugefügt, dass zufällig jeweils ein Nomen und ein Adjektiv nur für jeden der drei Parts in die Textdatei geschrieben wird:

Problem:
TypeError: Population must be a sequence. For dicts or sets, use sorted(d).
Lösung: ChatGPT gefragt ->
Das Nomen und Adjektive mussten noch in eine Liste umgewandelt werden:
selected_nouns = random.sample(list(unique_nouns), min(1, len(unique_nouns)))
selected_adjectives=random.sample(list(unique_adjectives),min(1,len(unique_adjectives)))

[10] https://realpython.com/natural-language-processing-spacy-python/

31.12.2023:

- Die Funktion „warp_affine“ hinzugefügt. Durch Professor Dennis Müller auf die Variante aufmerksam geworden -> damit die Übergänge zwischen den Bildern besser aussehen.
  Dabei werden die Eingangsbilder immer in kleinen Schritten in eine Richtung im Kreis rotiert, wodurch eine einfache Animation erstellt wird.
  Problem:
  Wenn ich das Video aus den Bildern generieren wollte, waren die Bilder im Video nicht mehr in der gleichen Reihenfolge wie sie eigentlich im Ordner gespeichert sind. In dieses Problem habe ich sehr viel Zeit investiert, da ich nicht wusste, wo der Fehler lag, bis ich das Problem selbst herausgefunden:
  -> lag daran, dass die rotierenden Bilder, welche im Ordner gespeichert werden, am Ende des Bildnamens durchnummeriert werden und mit „_1, _2, usw.“ enden. Wenn zweistellige Anzahl an rotierenden Bildern erstellt werden, dann ist logischerweise die Endung „_11“. Wenn ich das Video generiere, sind zwar die Bilder im Ordner in der richtigen Reihenfolge, aber die Funktion, die ich gebaut habe, um die Bilder in ein Video umzuwandeln, verwendet nach dem ersten Bild „_1“ das Bild „_11“ anstatt das eigentlich zuerst Bild „_2“ kommen müsste.
  Lösung: ChatGPT gefragt:
  Habe die Bildnamen mit „:03d“ erweitert. Das führt dazu, dass nun die Bildnamen mit der Endung „_001, _002 usw.“ gespeichert werden

02.01.2024:

- Funktion „song_duration“ erstellt. Die ist dafür da, dass man mit der Bibliothek „Spotipy“ die Songlänge erhält. Ich habe das jetzt so angepasst, dass automatisch das generierte Video am Schluss so lange läuft wie der Song.

06.01.2024:

- Funktion „download_audio“ und „audio_with_video_connection“ erstellt. Eine Möglichkeit gefunden, wie ich auch die Audio-Datei von einem Song in dem Video hinterlegen kann. Meine ursprüngliche Idee, dass ich mit der „Spotipy“ Bibliothek den Song im Hintergrund laufen lassen kann, indem ich mich mit meinem privaten Spotify Account verbinden, hat nicht funktioniert. Mit der Bibliothek „pytube“ kann man von YouTube Videos legal herunterladen. So habe ich die Funktion „download_audio“ gebaut, wo man nur den Link zu dem gewünschten Song eintragen muss, leider habe ich es nicht hinbekommen, dass es automatisiert läuft, deshalb muss dieser Schritt getan werden. Anschließend wird die Audio-Datei gedownloadet und mit in dem Ordner „generated_video“ gespeichert. Danach sorgt die Funktion „Audio_with_video_connection“ mit der Bibliothek „moviepy“ dafür, dass unter dem vorherigen generierten Video die gedownloadete Audiodatei hinterlegt wird und als neue Datei gespeichert wird.

14.01.2024:

- Ausführliche Anleitung in der Readme-Datei auf GitHub für die Implementierung meines Projektes hinzugefügt. Zudem eine requirements.txt Datei erstellt, welche alle Bibliotheken enthält, welche ich in diesem Projekt genutzt habe.

02.02.2024:

- Der Übergang zwischen den Bildern war noch nicht gut genug. Das ganze Video war noch nicht so dynamisch, weil bisher sich nur das Eingabebild gedreht hat. Verschiedene Funktionen ausprobiert und rum experimentiert ->
  Die Variante, dass ein Übergang zwischen dem ersten Bild und dem zweiten Bild entsteht, indem das zweite Bild sich langsam von rechts nach links über das erste Bild legt, hat bei mir aber nicht ganz so gut funktioniert. Das erste Bild hat sich immer mit nach links verschoben, bis zu dem Zeitpunkt das es aus dem Bildschirm verschwand. Ich habe dann eine Art Übergang hinzugefügt, bei dem das Eingabebild weiterhin am Anfang statisch dargestellt und im Hintergrund wird langsam das Eingabebild nochmal aber rotierend im Kreis immer mehr sichtbar eingeblendet. Was dazu führt, dass das statische Eingabebild immer mehr an Transparenz verliert, bis zu dem Zeitpunkt, dass sich das rotierende Eingabebild komplett im Vordergrund stellt.

03.02.2024:

- Weiter an der Funktion für die Übergangsbilder gearbeitet. Bisher war der Hintergrund vom Eingabebild schwarz, was ich geändert habe. Angepasst, dass im Hintergrund immer eine zufällige Farbe ausgewählt wird, welche sich zusätzlich auch noch bei jedem nächsten Bild ändern. Zudem bleibt die Farbe nicht statisch, sondern wird während des Videos immer dunkler.

04.02.2024:

- Weitere Überarbeitungen vorgenommen, was das generelle Code Design angeht und Kommentare hinzugefügt. Funktionen weiter ausgetestet und experimentiert, um die optimalen Einstellungen (z.B. Anzahl der Steps, FPS etc.) zu finden

6 Auswertung/Bewertung

Als ich mein Modell fertiggestellt habe, habe ich versuch die bestmöglichen Bilder zu generieren, indem ich die Steps Anzahl anpasse und auch durch den Prompt nochmal versucht das Ganze zu verbessern.

Abbildung 27: 50 Steps

Abbildung 26: 100 Steps

Die gezeigten Abbildungen 26-30 sind generierte Bilder, welche mit dem Song Umbrella von Rihanna entstanden sind. Für den Test habe ich mich für den Song entschieden, da dieser einen sehr bildlichen Text besitzt, welcher gut in Bilder widergespiegelt werden kann. Der Prompt dabei war bei den Abbildungen 27 & 26 die Wörter Umbrella“ & „Heart“. Bei meinen Tests konnte ich herausfinden, dass wenn ich die Anzahl auf 100 Steps lege, vermehrt Bilder generiert werden die im Stil „gezeichnet“ herauskommen (siehe Abbildung 26), stattdessen bei 50 Steps dies nicht der Fall war, sondern es dort mehr um realistische Bilder handelt (Siehe Abbildung 27), die jedoch keine gute Qualität besaßen. In der Abbildung 27 habe ich ein gutes ausgewählt, jedoch ist das nicht Normalität und im Großteil kommen komische Bilder heraus. Der Schluss, den ich daraus ziehen konnte, war das es bei einer höheren Anzahl an Steps, das Modell versucht, die Wörter, welche im Prompt angegeben wurden, viel besser verarbeiten kann und die ins Bild einbaut, wie man erkennen ,dass das auch das Herz, welches im Prompt enthalten war, auch einigermaßen sinnvoll im Regenschirm eingebaut wurde.

Abbildung 30: 100 Steps (High Quality) (1)

Abbildung 29: 100 Steps (High Quality) (2)

Abbildung 28: 100 Steps (High Quality) (3)

Des Weiteren habe ich ausprobiert den Prompt durch die Wörter im Song, dahinter noch durch „High Quality“ zu erweitern, was tatsächlich noch zu deutlich besseren Ergebnissen geführt hat. Zumindest wenn man sich die Abbildung 30 anschaut im Vergleich zu Abbildung 26. Dies aber jedoch auch nur, wenn die Steps Anzahl bei 100 lang. Bei 50 Steps konnte ich keine Verbesserungen feststellen, eher im Gegenteil, noch mehr Wörter haben zu schlechteren Bilder geführt.

6.1 Verbesserungen / Nächste Schritte

Das Projekt in der jetzigen Version ist vollständig und funktionsfähig. Dennoch ist das ganze Modell ein Projekt welches beliebig erweitert werden kann durch neue Funktionen. Einige Ideen davon werden hier nun beschrieben.

Zum einen gibt es ein Problem, welches schwer zu lösen ist, aber das Modell dadurch deutlich bessere Ergebnisse liefern würde. Da jeder Song unterschiedlich lang ist und zudem auch der Songtext unterschiedliche längen aufweist, ist es schwer dafür immer passende Zusammenfassungen zu erstellen, woraus die Bilder generiert werden. Wenn der Song nämlich sehr viel Text hat, ist es logischerweise so, dass viel mehr Wörter benötigt werden, um den Inhalt des Textes wiederzugeben. Mein Modell hingegen, generiert für alle Songs, egal welche Länge, die gleiche Anzahl an Bilder und Wörter. Weshalb es sinnvoll wäre, die Anzahl der Parts, in denen der Song aufgeteilt wird, abhängig von der Länge des Textes ist. In meinem Programm ist dies standardisiert für jeden Song auf drei Parts. Möglichkeiten um das Umsetzen wäre, dass man die Anzahl der Wörter oder Sätze im Song zählt und anhand dieser Information eine bestimmte Anzahl an Parts generiert.

Weitere Anhaltspunkte, um das Projekt weiter zu definieren wäre, dass man sich das vortrainierte Modell, welches ich nutze (Stable Diffusion) genauer anschaut. Dort gibt es noch unzählige Änderungen, die man vornehmen kann, um die Bilder anzupassen. Man könnte einen vordefinierten Stil hinzufügen, sodass die Bilder, die generiert werden, zum Beispiel nur in der Art „Comic“ wären. Das hätte den Vorteil ,dass das ganze Video einheitlicher aussieht und die Bilder sich ähneln. Die andere Variante wäre, dass man ein ganz neues Modell benutzt, wie zum Beispiel die API von DALL-E oder Midjourney. Diese sind zwar kostenpflichtig, aber vielleicht gibt es die Möglichkeit, dass dies über die Hochschule läuft und ein Zugang angelegt werden kann. Das würde auch nochmal zu besseren Bildern führen, da dieses kostenpflichtige Modell deutlich mehr Steps durchlaufen. Etwas komplizierter dahingehen wäre die Variante, dass man ein eigenes Modell trainiert, welches genau auf Songtexte spezialisiert ist.
Um das Modell noch weiter zu verbessern, könnte man Funktionen hinzufügen wie das der Songtext auch noch als Untertitel im Video angezeigt wird oder eine Art Audio-to-Text, wo nicht nur anhand des Songtextes die Bilder generiert werden, sondern auch anhand der Audio welche dann als Text umgewandelt wird.
Eine weitere Idee wäre, dass man sich den Songtext genauer anschaut. Das Filtern der Wörter, woraus die Bilder generiert werden, könnte man auch noch weiter verfeinern. So könnte man die Wörtern im Songtext mit Gewichtungen einfließen lassen, so dass als Beispiel, oft im Text vorkommenden Wörter eine höhere Gewichtung haben, was dazu führt, dass mit diesen Wörtern mehr generiert wird. Zudem könnte man auch den Songtitel noch zum Teil mit einfließen lassen, da dieser oftmals auch das beschreibt, was der Inhalt vom Song ist.

7 Literaturverzeichnis

https://pypi.org/project/lyricsgenius/
https://lyricsgenius.readthedocs.io/en/master/
https://github.com/johnwmillr/LyricsGenius
https://spotipy.readthedocs.io/en/2.22.1/
https://github.com/spotipy-dev/spotipy
https://pypi.org/project/spotify/
https://docs.streamlit.io/
https://github.com/streamlit/streamlit
https://pypi.org/project/diffusers/
https://huggingface.co/docs/diffusers/index
https://huggingface.co/docs/transformers/index
https://pypi.org/project/transformers/
https://docs.opencv.org/4.x/d4/d61/tutorial_warp_affine.html
https://zulko.github.io/moviepy/
https://pypi.org/project/moviepy/
https://pytube.io/en/latest/
https://github.com/pytube/pytube
https://www.youtube.com/watch?v=P5TfA5Y9jbo
https://www.youtube.com/watch?v=gf6egTeP6EI
https://docs.python.org/3/library/random.html
https://numpy.org/doc/stable/reference/generated/numpy.zeros.html
https://ipycanvas.readthedocs.io/en/latest/drawing_images.html
https://theailearner.com/2018/10/15/creating-video-from-images-using-opencv-python/
https://pypi.org/project/bert-extractive-summarizer/
https://huggingface.co/docs/transformers/tasks/summarization
https://towardsdatascience.com/abstractive-summarization-using-pytorch-f5063e67510
https://docs.opencv.org/4.x/d4/d61/tutorial_warp_affine.html
https://docs.opencv.org/4.x/d5/dc4/tutorial_adding_images.html
https://realpython.com/natural-language-processing-spacy-python/