AI time machine based on stable diffusion

Alexander Gärtner, Felix Rader

June 26, 2023

Abstract

abstract

Contents

1 Aims and Context **3**

2 Project Details **4**

3 System Documentation **7**
3.1 Stable Diffusion Setup 7
3.2 The Frontend 8
3.2.1 Components 8
3.2.2 API Routes 9

4 Summary **11**
4.1 Possible Issues 11

References **12**

Chapter 1

Aims and Context

Imagine the potential of future AR devices for tourism: One could walk through a city like Vienna with AR glasses and get a fully modified view of any past reality. For example, one could use a "Time Machine Slider" to see the city 200 years back from now. This would mean that an AI system would take the live feed of the camera and adjust it accordingly - people would wear clothing from previous times, cars would be substituted with horse carriages, advertisements would hint to past events instead of upcoming concerts.

The purpose of this project was to do a first step towards this vision by investigating recent AI tools and their potential to modify images in a way that they plausibly depict the past. Instead of using video streams, single images were used and modified by stable diffusion to serve as a starting point.

Chapter 2

Project Details

Since no similar previous work exists, as a first step a general pipeline for transforming images to a "past" version of the image had to be developed. Stable diffusion allows for inputting prompts in addition to an image to further shape the outcome of the transformed image. The stable diffusion webUI [3], an open-source project which allows for locally hosting a web based UI to interact with a stable diffusion model, served as a basis for researching how to generate "past" images.

In addition to the whole image generation, certain limitation had to be taken into consideration as well. For example, it was important to limit how far back in time the images should be generated in. Since stable diffusion is trained on actual photographs, paintings and so on it naturally has less training data to work with depending on how far back in time one goes. Therefore, the year 1880 was deemed as a reasonable cut-off for the time machine as this was the year photography started to become more common. After a month of research a pipeline for the image generation was developed as can be seen in figure 2.1.

The pipeline goes as follows:

1. Upload Image
2. resize image to 512x512 which is the preferred format for stable diffusion
3. if available: read coordinates from image data and assign city and country
4. set desired past year between 1880 - 2013
5. interrogate image via stable diffusion to generate base prompt
6. combine year, geolocation and base prompt into a single prompt
7. if the year is below 1950 filter out all colors from the prompt and set colors to negative prompt to ensure a black and white image
8. send prompt to chatGPT and let it filter out objects from the prompt which did not exist in the specified year
9. add year specific information to filtered prompt like common cameras and techniques
10. send image and prompt to stable diffusion
11. generate canny edges via ControlNet plugin for stable diffusion
12. generate past image based on prompt and canny edges version of image
13. output generated image

Figure 2.1: pic of pipeline

Based on the pipeline, a simple frontend based on React and NextJS was developed as can be seen in figure 2.2. The frontend implements everything that is needed for uploading and resizing the image. Additionally, it also contains a repertoire of base prompts based on each year. These base prompts contain standard information for each year like which camera was typically used for the time span. Image interrogation and image generation are handled in the background via the stable diffusion webUI to which the frontend makes API calls. When both the stable diffusion webUI and the web based frontend are locally hosted on a machine it is possible to upload images to the frontend to generate a new image based on a selected year or a GIF based on the whole year range.

Figure 2.2: A screenshot of the developed frontend for the time machine. The picture selected is a photograph of a street in Salzburg, Austria taken on a mobile phone

Figure 2.3: A screenshot of the frontend after image generation. This shows as street in Salzburg, Austria in 1940 as generated by stable diffusion based on the previous input.

Chapter 3

System Documentation

In this chapter each part of the image generation is explained in detail.

3.1 Stable Diffusion Setup

The stable diffusion webUI [3] is locally hosted on a machine so that it can be used to transform images. A couple of changes were made the UI to make it possible to generate past version of images. The specific stable diffusion model used is realistic vision V2.0 [1], which is supposed to generate more realistic looking images compared to the base stable diffusion model included in the webUI.

Additionally, the ControlNet plugin [2] is used to enhance images further. With the help of ControlNet canny it is possible to trace the outlines of objects in images that are put into stable diffusion, thereby resulting in images which more closely resemble the original. Furthermore, the following settings were changed inside the webUI:

1. under ControlNet: Allow other script to control this extension
2. under Interrogate Options: Increase the minimum and maximum description length
3. under Interrogate Options: skip inquire categories for artists, flavors, mediums and movements

These settings ensure that ControlNet can be called from the frontend. Also, the changes to the interrogate options are vital, as otherwise the generated prompt for the past image could contain information about artists or non camera mediums which more often than not results in the generated image looking like a painting instead of a realistic photograph.

Finally, the command line section of the webui-user.bat file inside the folder for the stable diffusion webUI was changed to allow for API calls from the locally hosted front end as can be seen in the following code:

```
set COMMANDLINE_ARGS= --xformers --autolaunch --medvram --api
--cors-allow-origins=http://127.0.0.1:7860 --cors-allow-origins=http://localhost:300
```

3.2 The Frontend

The frontend was developed using React, Typescript and NextJS 13. Therefore the project has the standard nextJS structure. The “app” folder contain the base layout of a page as well as all API routes. The “components” folder contains all components used on the pages. For styling purposes TailwindCSS was used. Since the frontend only serves as a UI for easier interaction and access to stable diffusion it was developed as a single page application.

3.2.1 Components

A total of three components were developed.

- BottomBar
- TopNavigationBar
- FileUploader

The first two components are simply for cosmetic purposes while the FileUploader contains most of the logic.

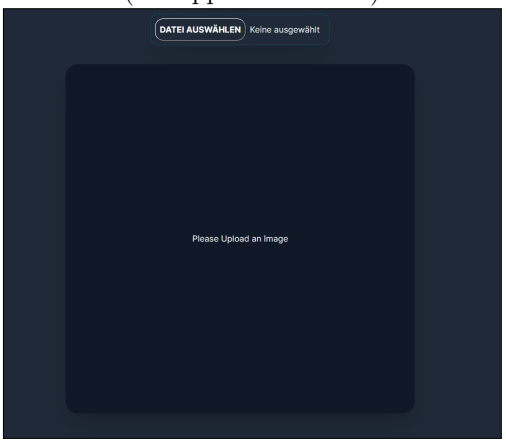
FileUploader

The FileUploader has multiple state variables as well as function for executing the image conversion. Depending on its current states the FileUploader displays different parts. For example the year slider will only display after an image has been uploaded. The different state of the FileUploader can be seen here 3.1.

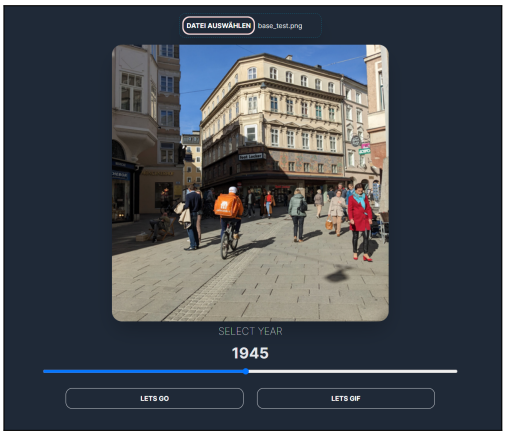
The various functions used in the component are as follows:

- **handleFileSelect:** executed when an image is uploaded. Checks the file size and type of the uploaded image and then resizes it to 512x512 as well as read the geolocation data from the image.
- **cropImage:** a helper function used in handleFileSelect to properly crop the uploaded image.
- **getCityAndCountry:** a helper function used in handleFileSelect that makes an axios call to openstreetmap based on the latitude and longitude contained in the image geo location data in order to determine the city and country the image is from.
- **handleBackButton:** executes when the back button is pressed. Resets the state of the FileUploader to its initial state.
- **handleShowOriginalButton:** executes when the show original button is pressed. Swaps the currently displayed image with the original uploaded image and vice versa.
- **handleClick:** executes when the “Let´s Go” button is pressed. Calls each API route, as described in the next section 3.2.2, after each other to generate a past version of the uploaded image.
- **handleClickGif:** executes when the “Let´s Gif” button is pressed. Functions the same way as handleClick but generates multiple images with the year starting at

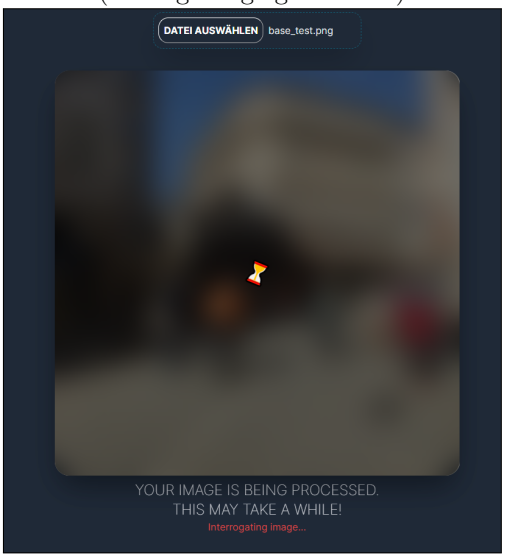
(On application start)



(After Image Upload)



(During image generation)



(Finished image conversion)

Figure 3.1: Screenshots of the FileUploader component during its different states.

1880 up to 2013 in five year increments instead. The generated images are then combined into a GIF. This function does not use the getGPT API route.

- **generateGif:** a helper function used in handleClickGif to combine the generated images into a GIF using gifjs.3
- **handleResponseImageClick:** executes when the generated image is clicked. Opens the image as a pop up.

3.2.2 API Routes

A total of five API routes were implemented in order to have a cleaner code structure as well as keep server side code out of the FileUploader component.

interrogateImage

The first API route is interrogateImage which makes an API call to the locally hosted stable diffusion implementation and returns the generated image description.

getPrompt

The getPrompt API route filters out all colors from the prompt and contains prewritten prompt parts to the base prompt depending on the year.

getGPT

The getGPT API route makes an API call to chatGPT containing the year and prompt and request chatGPT to generate a list of negative prompts filled with objects that are contained inside the prompt but did not exist in the selected year.

getNegativePrompt

The getNegativePrompt API rout add additional prewritten negative prompts based on the selected year.

convertImage

The convertImage API route makes an API call to the locally hosted stable diffusion implementation containing the generated prompt, negative prompts as well as the input image.

Chapter 4

Summary

Over the course of the semester a pipeline for generating a version of an image based on a specific past year was created. Based on this pipeline a front end application was developed which makes it possible to upload single images as well as select a specific year to generate a past version of the uploaded image based on the selected year. Additionally, uploaded images can also be turned into GIFs to have a time lapse over a whole year range going from 1880 to 2013

4.1 Possible Issues

In the current implementation the FileUploader component currently contains the bulk of the code in the front end, thereby making it a very bloated component. This can make it hard to expand the functionality of the whole application. The current solution always has to be locally hosted which requires users to have the stable diffusion webUI running as well. Since the stable diffusion webUI requires a decent GPU to run properly this can be an issue when working on a less powerful device. Steps were taken to try to host the stable diffusion webUI on one of the render nodes at the FH Hagenberg, but this still requires users to be inside the WiFi network of the universtiy as well as having the frontend application running locally.

References

[1] SG161222. *Realistic Vision Model*. 2023. URL: https://civitai.com/models/4201/realistic-vision-v20 (visited on 06/24/2023) (cit. on p. 7).

[2] various. *ControlNet*. 2023. URL: https://github.com/lllyasviel/ControlNet (visited on 06/24/2023) (cit. on p. 7).

[3] various. *Stable Diffusion Webui*. 2023. URL: https://github.com/AUTOMATIC1111/stable-diffusion-webui (visited on 06/24/2023) (cit. on pp. 4, 7).