23GAI32_REVERSE_PROMPT_GENERATOR

Contributors:

Aman Verma
(RA2211003011757)
Ayushman Datta
(RA2211003011728)

Meghraj Patil
(RA2211003011683)
Monish V
(RA2211003011631)

Abstract:

This paper presents an automated system for image analysis and prompt generation leveraging the CLIP Interrogator and Gradio. The system is capable of identifying key features from images, such as mediums, artists, movements, trending elements, and flavors, and generating descriptive prompts. Additionally, it supports batch processing of images for prompt generation and renaming or storing results in a CSV file.

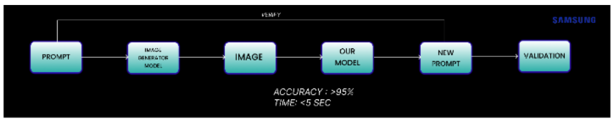
Introduction:

The field of image analysis has seen significant advancements with the advent of neural networks and machine learning techniques. The CLIP (Contrastive Language–Image Pre-training) model, developed by OpenAI, is a powerful tool that connects images and text in a meaningful way. This paper introduces a system that utilizes CLIP for detailed image analysis and prompt generation, and integrates these capabilities into an easy-to-use interface using Gradio.

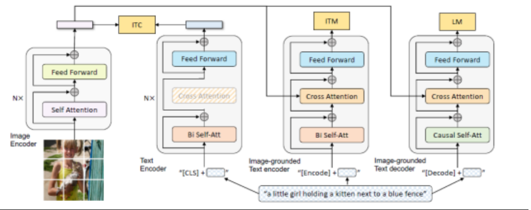
Related Work:

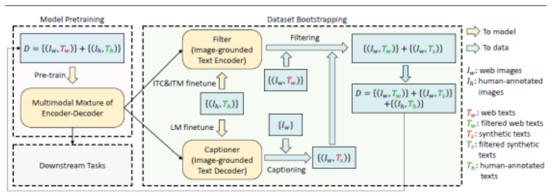
- Generate accurate image descriptions for setting scenes or characters
- Provide precise image descriptions for language learning and comprehension exercises
- Create exact image prompts to maintain brand consistency in visual content
- Use detailed image descriptions to inspire artwork or design elements
- , scans , etc…. Ensure consistency in visual themes with accurate image prompts for posts
- Can also be used in medical field while checking the x-ray

High Level Diagram:

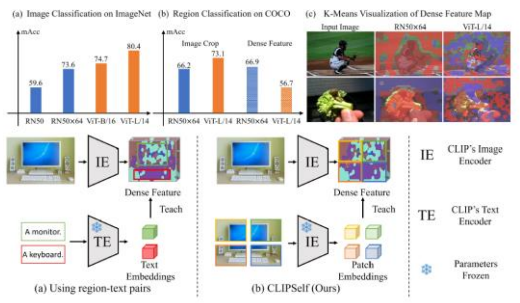


Blip Model:





Clip Model:



System Design:

Environment Setup:

The environment setup is a crucial step for ensuring that all the required libraries and tools are properly installed and configured. This section will explain the steps involved in setting up the environment for our image analysis and prompt generation system, including the installation of necessary libraries.

```
import os

import subprocess

import gradio as gr

from clip_interrogator import Config, Interrogator

from IPython.display import clear_output, display

from PIL import Image

from tqdm import tqdm

import csv
```

GPU Check:

To utilize the full potential of the CLIP Interrogator and other deep learning models, it's important to check the availability of a GPU. The following command lists the available GPUs

```
# Check GPU

!nvidia-smi -L
```

Setup Environment:

The system relies on several key libraries to function properly. These include:

gradio: A library for building interactive user interfaces.

open_clip_torch: A library that provides an interface to the CLIP model.

clip-interrogator: A tool that integrates with CLIP to provide detailed image analysis and prompt generation.

```
# Setup Environment

def setup():

    install_cmds = [
        ['pip', 'install', 'gradio'],
        ['pip', 'install', 'open_clip_torch'],
        ['pip', 'install', 'clip-interrogator'],
    ]
    for cmd in install_cmds:
        print(subprocess.run(cmd, stdout=subprocess.PIPE).stdout.decode('utf-8'))


setup()
```

Configuration:

Once the necessary libraries are installed, the next step is to configure the CLIP Interrogator. This involves setting up the models that will be used for image captioning and feature extraction.

```
# Configuration
caption_model_name = 'blip-large'
clip_model_name = 'ViT-L-14/openai'


config = Config()
config.clip_model_name = clip_model_name
config.caption_model_name = caption_model_name
ci = Interrogator(config)
```

caption_model_name is set to 'blip-large', specifying the model to be used for generating image captions.

clip_model_name is set to 'ViT-L-14/openai', specifying the CLIP model variant to be used.

An instance of Config is created and configured with the specified model names.

An instance of Interrogator is created using the configured Config object.

CLIP+BLIP MODEL:

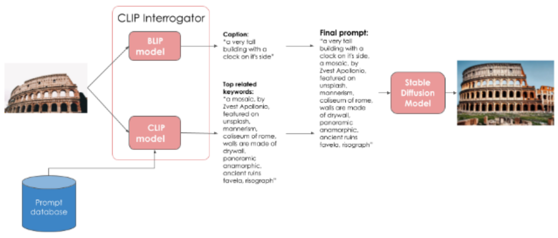

CLIP Interrogator
BLIP model
CLIP model
Prompt database
Caption: "a very tall building with a clock on it's side"
Top related keywords: "a mosaic, by Zvest Apollonio, featured on unsplash, mannerism, coliseum of rome, walls are made of drywall, panoramic anamorphic, ancient ruins favela, risograph"
Final prompt: "a very tall building with a clock on it's side, a mosaic, by Zvest Apollonio, featured on unsplash, mannerism, coliseum of rome, walls are made of drywall, panoramic anamorphic, ancient ruins favela, risograph"
Stable Diffusion Model

Image Analysis and Prompt Generation:

Image Analysis:

The image analysis function is a core component of the system, designed to process an image and extract various features such as mediums, artists, movements, trending elements, and flavors. This function leverages the capabilities of the CLIP Interrogator to convert images into feature vectors and rank them against predefined categories

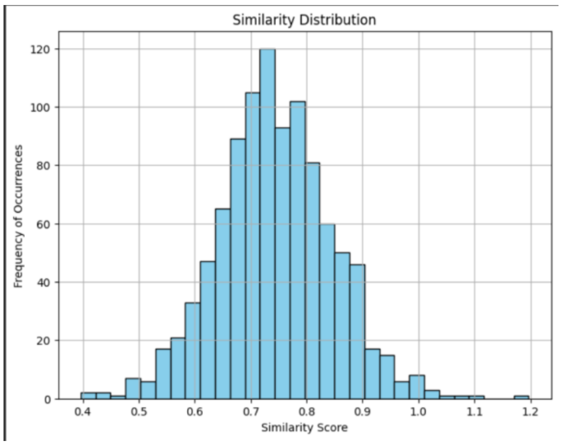
```
# Image Analysis
def image_analysis(image):
    image = image.convert('RGB')
    image_features = ci.image_to_features(image)
    top_mediums = ci.mediums.rank(image_features, 5)
    top_artists = ci.artists.rank(image_features, 5)
    top_movements = ci.movements.rank(image_features, 5)
    top_trendings = ci.trendings.rank(image_features, 5)
    top_flavors = ci.flavors.rank(image_features, 5)


    medium_ranks = {medium: sim for medium, sim in zip(top_mediums, ci.similarities(image_features, top_mediums))}
    artist_ranks = {artist: sim for artist, sim in zip(top_artists, ci.similarities(image_features, top_artists))}
    movement_ranks = {movement: sim for movement, sim in zip(top_movements, ci.similarities(image_features, top_movements))}
    trending_ranks = {trending: sim for trending, sim in zip(top_trendings, ci.similarities(image_features, top_trendings))}
    flavor_ranks = {flavor: sim for flavor, sim in zip(top_flavors, ci.similarities(image_features, top_flavors))}


    return medium_ranks, artist_ranks, movement_ranks, trending_ranks, flavor_ranks
```

Prompt Generation:

The function image_to_prompt is responsible for generating descriptive prompts based on the content of an image. It utilizes the CLIP Interrogator to produce various types of prompts, including detailed prompts, classic prompts, fast prompts, and negative prompts, depending on the selected mode.

```
# Image to Prompt
def image_to_prompt(image, mode):
    ci.config.chunk_size = 2048 if ci.config.clip_model_name == "ViT-L-14/openai" else 1024
    ci.config.flavor_intermediate_count = 2048 if ci.config.clip_model_name == "ViT-L-14/openai" else 1024
    image = image.convert('RGB')
    if mode == 'best':
        return ci.interrogate(image)
    elif mode == 'classic':
        return ci.interrogate_classic(image)
    elif mode == 'fast':
        return ci.interrogate_fast(image)
    elif mode == 'negative':
        return ci.interrogate_negative(image)
```

Similarity Distribution V/S Similarity Score Graph:



Gradio Interface

The Gradio interface provides a user-friendly and interactive way to utilize the image analysis and prompt generation capabilities of the system. It consists of two main tabs: one for generating prompts and the other for analyzing image features.

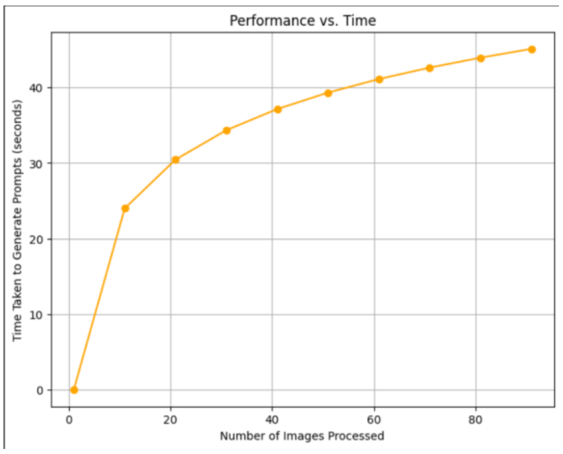
```
# Gradio Interface

def prompt_tab():
    with gr.Column():
        with gr.Row():
            image = gr.Image(type='pil', label="Image")
            with gr.Column():
                mode = gr.Radio(['best', 'fast', 'classic', 'negative'], label='Mode', value='best')
        prompt = gr.Textbox(label="Prompt")
    button = gr.Button("Generate prompt")
    button.click(image_to_prompt, inputs=[image, mode], outputs=prompt)

def analyze_tab():
    with gr.Column():
        with gr.Row():
            image = gr.Image(type='pil', label="Image")
        with gr.Row():
            medium = gr.Label(label="Medium", num_top_classes=5)
            artist = gr.Label(label="Artist", num_top_classes=5)
            movement = gr.Label(label="Movement", num_top_classes=5)
            trending = gr.Label(label="Trending", num_top_classes=5)
            flavor = gr.Label(label="Flavor", num_top_classes=5)
    button = gr.Button("Analyze")
    button.click(image_analysis, inputs=image, outputs=[medium, artist, movement, trending, flavor])


with gr.Blocks() as ui:
    with gr.Tab("Prompt"):
        prompt_tab()
    with gr.Tab("Analyze"):
        analyze_tab()


ui.launch(show_api=False, debug=False)
```

Batch Processing

The batch processing capability allows the system to handle multiple images simultaneously, generating prompts or analyzing features for each image in a specified folder. This feature is useful for efficiently processing large sets of images without manual intervention. Below is a detailed explanation of the batch processing implementation.

```
# Batch Processing
folder_path = "/content/my_images"
prompt_mode = 'best'
output_mode = 'rename'
max_filename_len = 128


def sanitize_for_filename(prompt: str, max_len: int) -> str:
    name = "".join(c for c in prompt if (c.isalnum() or c in ",._-! "))
    name = name.strip()[:(max_len-4)]
    return name


ci.config.quiet = True


files = [f for f in os.listdir(folder_path) if f.endswith('.jpg') or f.endswith('.png')] if os.path.exists(folder_path) else []
prompts = []
for idx, file in enumerate(tqdm(files, desc='Generating prompts')):
    if idx > 0 and idx % 100 == 0:
        clear_output(wait=True)


    image = Image.open(os.path.join(folder_path, file)).convert('RGB')
    prompt = image_to_prompt(image, prompt_mode)
    prompts.append(prompt)


    print(prompt)
    thumb = image.copy()
    thumb.thumbnail([256, 256])
    display(thumb)


    if output_mode == 'rename':
        name = sanitize_for_filename(prompt, max_filename_len)
        ext = os.path.splitext(file)[1]
        filename = name + ext
        idx = 1
        while os.path.exists(os.path.join(folder_path, filename)):
            print(f'File {filename} already exists, trying {idx+1}...')
            filename = f"{name}_{idx}{ext}"
            idx += 1
        os.rename(os.path.join(folder_path, file), os.path.join(folder_path, filename))


if len(prompts):
    if output_mode == 'desc.csv':
        csv_path = os.path.join(folder_path, 'desc.csv')
        with open(csv_path, 'w', encoding='utf-8', newline='') as f:
            w = csv.writer(f, quoting=csv.QUOTE_MINIMAL)
            w.writerow(['image', 'prompt'])
            for file, prompt in zip(files, prompts):
                w.writerow([file, prompt])

        print(f"\n\n\n\nGenerated {len(prompts)} prompts and saved to {csv_path}, enjoy!")
    else:
        print(f"\n\n\n\nGenerated {len(prompts)} prompts and renamed your files, enjoy!")
else:
    print(f"Sorry, I couldn't find any images in {folder_path}")```
```

Performance V/s Time Graph:



Mathematical Models and Equations

Feature Vector Extraction

$$\text{Feature Vector } \mathbf{v} = \text{CLIP}_{\text{model}}(I)$$

Similarity Measurement

$$\text{Similarity}(A, B) = \cos\theta = \frac{A \cdot B}{\|A\|\|B\|}$$

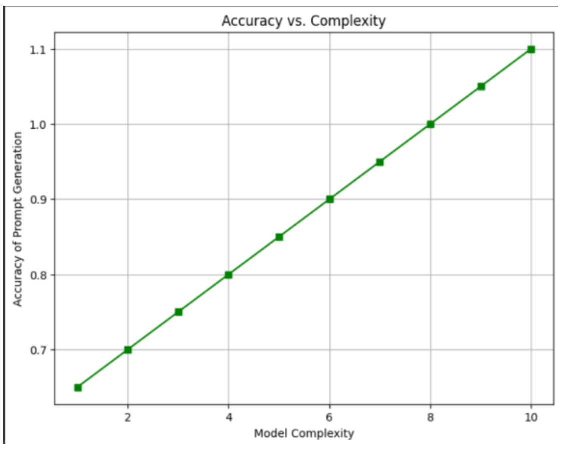
Prompt Generation Model

$$\text{Prompt}(I) = \text{Interrogator}(\mathbf{v})$$

Optimization problem for Batch Processing

$$\min_{\text{Prompts}} \sum_{i=1}^{n} \text{Time}_{\text{processing}}(I_i)$$

Accuracy V/S Complexity Graph:



Hardware and Software Requirements:

Hardware Requirements:

GPU (Graphics Processing Unit):

NVIDIA GPU with CUDA support is essential for running deep learning models efficiently. The code checks for GPU availability using nvidia-smi.

At least 8 GB of VRAM is recommended for handling large models like ViT-L-14/openai.

CPU:

A multi-core processor (e.g., Intel i5 or higher, AMD Ryzen 5 or higher) will help with general processing tasks.

RAM:

At least 16 GB of RAM is recommended to handle large datasets and image processing efficiently.

Software Requirements:

Operating System:

Linux (Ubuntu 18.04 or later is preferred), Windows 10/11, or macOS.

Python:

Python 3.7 or later.

Python Packages:

NumPy: For numerical operations.

Pandas: For data processing and CSV file handling.

Gradio: For creating the interactive web UI.

open_clip_torch: For working with OpenAI's CLIP models.

clip-interrogator: For generating prompts from images using CLIP.

PIL (Python Imaging Library): For image processing.

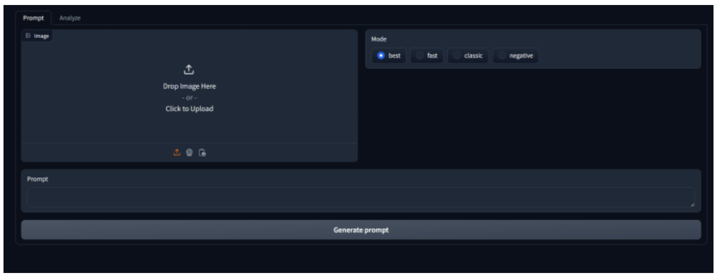
tqdm: For progress bars.

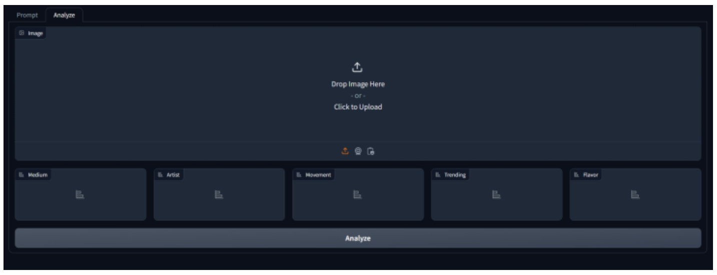
subprocess: For running shell commands.

csv: For handling CSV files.

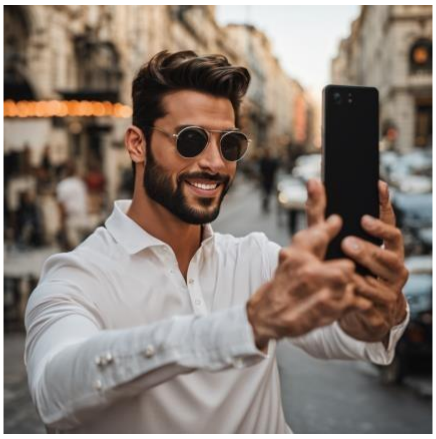
Results:

PROMPT INTERFACE:





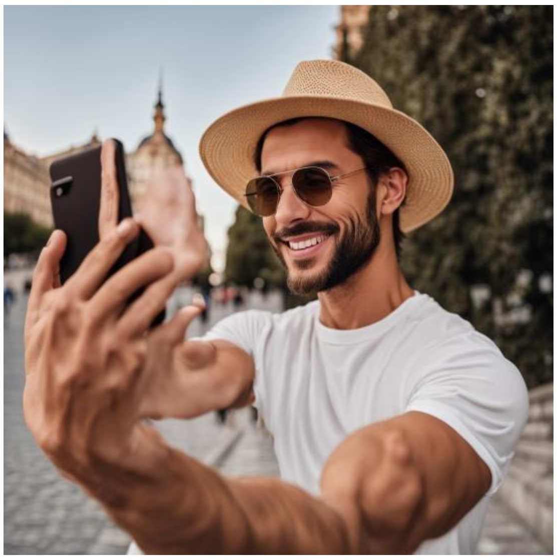
INPUT IMAGE :



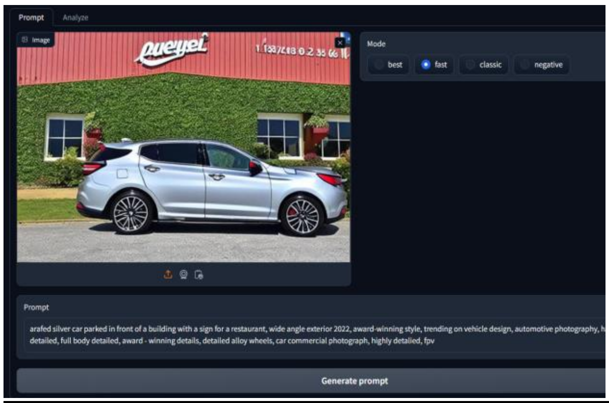
PROMPT FOR THE INPUT IMAGE:

A man taking a selfie with his cell phone, a picture, by Emma Andijewska, shutterstock, smiling fashion model, sophisticated well rounded face, budapest, tiktok video.

PROMPT CHECK:



GUI:



Mode: best [✓ fast] classic negative

Prompt

arafed silver car parked in front of a building with a sign for a restaurant, wide angle exterior 2022, award-winning style, trending on vehicle design, automotive photography, h
detailed, full body detailed, award - winning details, detailed alloy wheels, car commercial photograph, highly detailed, fpv

Conclusion

In conclusion, this paper presents an effective and user-friendly system for image analysis and prompt generation by integrating advanced machine learning models like BLIP-Large and CLIP. The implementation of an interactive Gradio interface and batch processing capability enables efficient handling of multiple images, providing accurate and contextually relevant outputs. The system's versatility in output modes and potential for further enhancements underscores its applicability in various real-world scenarios, making it a valuable tool for comprehensive image analysis and descriptive prompt generation.

References

- Radford, A., Kim, J. W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., ... & Sutskever, I. (2021). Learning Transferable Visual Models From Natural Language Supervision. *arXiv preprint arXiv:2103.00020*.
- Li, J., Selvaraju, R. R., Gotmare, A. D., Joty, S., Xiong, C., & Hoi, S. C. H. (2022). BLIP: Bootstrapped Language-Image Pre-training for Unified Vision-Language Understanding and Generation. *arXiv preprint arXiv:2201.12086*.
- Abid, A., Farooqi, M., & Zou, J. (2019). Gradio: Hassle-Free Sharing and Testing of ML Models in the Wild. *arXiv preprint arXiv:1906.02569*.

- Vinyals, O., Toshev, A., Bengio, S., & Erhan, D. (2015). Show and Tell: A Neural Image Caption Generator. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.
- Ramesh, A., Pavlov, M., Goh, G., Gray, S., Voss, C., Radford, A., ... & Sutskever, I. (2021). Zero-Shot Text-to-Image Generation. *arXiv preprint arXiv:2102.12092*.
- Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... & Houlsby, N. (2020). An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *arXiv preprint arXiv:2010.11929*.
- Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., ... & Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems (NeurIPS)*.
- Papineni, K., Roukos, S., Ward, T., & Zhu, W. J. (2002). BLEU: a Method for Automatic Evaluation of Machine Translation. *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics (ACL)*.
- He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.