Computer Vision Project: Image Description using Embedding Models and LLMs

Sohum Kothavade, Cole Marco, Kenta Terasaki

November 7, 2023

1 Introduction

In recent years, advancements in the fields of computer vision and natural language processing have led to innovative approaches in understanding and describing visual content. In this project, we explore a fascinating intersection of these two domains by leveraging the capabilities of Embedding Models and Large Language Models (LLMs) to convert images into a semantically meaningful vector space and subsequently generate natural language descriptions for those images.

1.1 Project Overview

At the heart of our project lies the transformation of images into a continuous, low-dimensional vector space using embedding models. The purpose of this step is to represent complex visual data in a form that can be processed and understood by neural networks. Each image is mapped to a unique vector in this space, allowing for the extraction of essential features, patterns, and relationships that are otherwise challenging to capture in their raw form.

Subsequently, we pass these image embeddings to Large Language Models (LLMs) for the generation of natural language descriptions. This fusion of image embeddings with LLMs results in the automatic generation of textual descriptions for a wide range of visual content.

1.2 Applications

The application of this novel approach extends across several domains, making it an exciting and valuable undertaking. Some notable applications include:

1. **Assistive Technologies**: This technology can greatly assist individuals with visual impairments by providing detailed textual descriptions of images they encounter. It can enhance their understanding of the world around them, making everyday life more accessible.

2. **Content Tagging and Search**: In the context of content management, the automatic generation of image descriptions can significantly improve the efficiency of content indexing and retrieval, making it easier to search for specific images in large datasets.

3. **Visual Storytelling**: By generating natural language descriptions for images, this technology can be used in creative applications like generating captions for photo albums or creating visual narratives in art and design.

4. **Image Accessibility**: The automatic description of images can enhance web accessibility. Websites and applications can provide textual descriptions for images, making online content more inclusive.

1.3 Rationale

The decision to employ embedding models and LLMs in our project is rooted in the potential of these technologies to bridge the gap between visual and textual information. The use of embedding models enables the conversion of images into a structured format, while LLMs are adept at generating coherent and contextually relevant textual descriptions. This approach not only holds the promise of improving accessibility and searchability but also opens the door to creative applications.

In the following sections, we delve into the technical details of our methodology, experiments, results, and discuss the implications of our findings.

2 What is an Embedding Model?

At the core of our methodology lies the transformation of images into a continuous, low-dimensional vector space using embedding models. This essential step is designed to represent complex visual data in a format that can be processed and understood by neural networks. Each image in our dataset is mapped to a unique vector or point within this space, enabling us to extract crucial features, patterns, and relationships that are challenging to capture in their raw form. Since we now have a way to turn each image into a point in this space, previously complex analysis now becomes as simple as computing the distance between coordinates. This makes it easier for subsequent neural networks, such as Large Language Models (LLMs), to analyze and generate meaningful insights from the data. Let's break down the key components and steps involved in the functioning of our embedding models:

1. **Input Data (Images):** The embedding model begins with a set of images as input. Each image is represented as a high-dimensional array, with each element corresponding to the color or intensity of a pixel, as well as other features depending on the complexity of the image.

2. **Feature Extraction:** Prior to mapping images to a vector space, the embedding model conducts feature extraction. This step is critical because it identifies and extracts relevant visual features or patterns from the input images. Features can range from basic edges and textures to complex shapes and object parts. The primary objective is to reduce the dimensionality of the data while preserving important information.

3. **Mapping to Vector Space:** The central operation of the embedding model involves mapping these extracted features into a continuous, low-dimensional vector space. This mapping is a crucial step, as it transforms the high-dimensional, pixel-level data into a structured, numerical representation. To achieve this transformation, neural networks are often employed, with convolutional neural networks (CNNs) being a popular choice. CNNs excel in spatial hierarchies and patterns, capturing abstract visual information and encoding it into lower-dimensional vectors.

4. **Embeddings:** The outcome of this mapping process is a vector referred to as an "embedding." Each image corresponds to a unique point within this vector space. The position of

each point encodes valuable information about the visual characteristics of the image. Images with similar visual features are located closer together in the vector space, while dissimilar images are farther apart. These embeddings serve as a compact and semantically rich representation of the original images, enabling various downstream tasks, including image retrieval, similarity measurement, and even natural language descriptions.

2.1 Example: Embedding Natural Language

To understand the concept of embedding in natural language, let's consider a simplified example. Imagine we have a dictionary of words, and our goal is to represent these words in a way that captures their semantic meaning. We aim to create a vector space where words with similar meanings are closer to each other.

Let's examine how the embedding process works:

1. **Input Words:** We begin with a set of words: "cat," "dog," and "sat."
2. **Semantic Analysis:** The embedding model analyzes the semantic relationships between these words. It recognizes that "cat" and "dog" have a closer semantic connection because they both represent animals, whereas "sat" is a different concept altogether.
3. **Mapping to Vector Space:** The model maps each word to a vector in a continuous vector space. The position of each word in this space encodes its semantic meaning. In this vector space, "cat" and "dog" end up closer to each other, reflecting their shared semantic similarity, while "sat" is located farther away.
4. **Similarity Measurement:** Using the embeddings, we can now measure the similarity between words quantitatively. For instance, the cosine similarity between the "cat" and "dog" vectors is higher compared to the cosine similarity between "cat" and "sat."

This example illustrates how embedding natural language words can capture their semantic meaning in a continuous vector space. It highlights that words with shared semantic content end up closer to each other in the vector space, despite their visual or syntactic differences. These embeddings are crucial for various natural language processing tasks, such as text classification, machine translation, and sentiment analysis.

2.2 Mathematical Foundations of Embedding Models

We can envision embedding models as mathematical functions that map images to embedding vectors. Let's denote the embedding function as $E(image)$, where $image$ represents the input image, and the resulting embedding vector is denoted as $v = E(image)$.

The process involves a series of mathematical operations designed to distill essential visual features and reduce the dimensionality of the image data, ultimately creating a meaningful representation:

2.2.1 Convolutional Layers

Convolutional Neural Networks (CNNs) form the backbone of many embedding models. These networks utilize convolutional layers to extract visual features from input images. The key mathematical operation is convolution, which is applied to the input image using learnable filters. At each

step, the convolution operation computes a weighted sum of pixel values in the local neighborhood, producing a feature map. Mathematically, it can be expressed as:

$$F_i(x, y) = (I * K_i)(x, y) = \sum_m \sum_n (I(m, n) \cdot K_i(x - m, y - n))$$

Here, $F_i(x, y)$ represents the output feature map at position (x, y) for the ith filter, I is the input image, K_i is the ith filter, and m and n are indices for the rows and columns of the filter.

2.2.2 Activation Functions

Following convolution, activation functions introduce non-linearity to the model. A common choice is the Rectified Linear Unit (ReLU), defined as:

$$\text{ReLU}(x) = \max(0, x)$$

This function retains positive values and sets negative values to zero, allowing the model to capture complex relationships in the data.

2.2.3 Pooling Layers

Pooling layers are employed to reduce the spatial dimensions of the feature maps while preserving essential information. Max pooling, for instance, selects the maximum value from a local region, effectively downsampling the data. Mathematically, max pooling can be expressed as:

$$P(x, y) = \max(F(x, y))$$

In this equation, $P(x, y)$ represents the output after max pooling, and $F(x, y)$ is the feature map.

2.2.4 Fully Connected Layers

Fully connected layers are the final step in the embedding process. They flatten the extracted features from the previous layers and reduce their dimensionality to generate the ultimate embedding vector. This is typically accomplished through matrix multiplications and the application of activation functions, resulting in the production of the embedding vector.

2.3 Application to Images

By embedding images, we create a bridge between the visual and textual domains, enabling LLMs to generate natural language descriptions. The image embeddings serve as a compact representation of the visual content, making it easier for LLMs to understand and generate meaningful textual content based on the images.

In the next section, we will explore how our Large Language Models leverage these image embeddings to automatically generate descriptive text for a wide range of visual content, completing the fusion of visual and textual information in our project.

3 Our MVP Implementation

Our implementation process involved a combination of existing resources, external APIs, and programming tools to bring our vision to life. Here, we outline the key components and steps taken during our project's implementation:

1. **Getting Started:** We initiated our project by delving into a Medium article that offered insights into context injection with Large Language Models (LLMs). This initial exploration helped us understand how to leverage LLMs and get acquainted with the APIs required for the task. It also provided a foundation for working with embedding models to convert text into vectors that LLMs could process.

2. **Embedding Model Selection:** Instead of creating our own embedding model, we opted to use OpenAI's Ada-002 architecture. This choice allowed us to harness the capabilities of a powerful pre-trained model for the embedding process.

3. **Programming Stack:** Our implementation was carried out in Python. We made use of libraries such as langchain for text chunking, OpenAI's API library, and basic machine learning libraries to facilitate the integration of the embedding model and LLMs into our project.

4. **Image Preprocessing:** While we didn't perform image preprocessing ourselves, Ada-002 handled the preprocessing of images as part of its internal workflow. The model's embedded capabilities encompass the entire image-to-vector process.

5. **Semantic Matching:** The goal of our project was to assess the semantic similarity between images and their textual descriptions. For this, we relied on Ada-002's inherent capabilities and didn't implement additional mechanisms.

4 Challenges

The most difficult part of the pipeline to establish was getting the Torch-based computer vision deep learning models to interface with the OpenAI Large Language Models (LLMs), such as GPT-3.5 Turbo and GPT-4. Utilizing Salesforce's BLIP (Bootstrapping Language-Image Pre-training for Unified Vision-Language Understanding and Generation) Deep Learning library, we built 'tools' the LLM could utilize at will to aid it in specialized Image-Text Retrieval and Analysis-based tasks. Currently, the challenge we are facing is API call latency and model run time. The most powerful use case of the application is when it is capable of running locally on an edge-computing device in real time. To get to this steady state of being able to work over a constant stream of image + sensor data, a large amount of optimization, complexity reduction, and hardware-specific acceleration will be required.

5 Results

Currently, we have a working system that can be fed an image and results in bounding boxes generated over identified objects. In this sample case, the fed image is a green stop light with a slightly blurred background of cars, bikes, and pedestrians. It first generates a simple caption describing what it observes in the image. The app then identifies two points that create a bounding box with weighted confidence on the object classification and positioning.

6 Future Work

For the future, the goal is to use our Computer Vision Project as a building block for navigation software that can take in real-time video and sensor data and have a Large Multimodal Model (LMM) be able to give instructions to a visually impaired person in a human-like manner. For example, the software will state, "Walk straight with caution of a car approaching 20 ft to the left and pedestrians 5 ft to the right". A navigation system using simpler algorithms will have more unnatural instructions such as, "Move forward 5 ft and then turn 30 degrees to the right. Beware of approaching object". We would also like to have the software capable of localization and storing the environments traversed, recognizing when revisiting the same environment based on globalized mapping. If we are able to implement this with high accuracy and low latency, this software will become a great way for visually impaired people to navigate the world.