APC Mastery Path

Guide to Query Images using Llava

This document aims at explaining to the reader and the end-user different aspects regarding querying images, how to obtain structural useful information, building an image based chatbot, use cases within the construction industry and much more.

Mohamed Ashour
December 2023

About the author

The author, with a robust background spanning nearly seven years in the construction industry, brings a unique blend of expertise and academic achievement to the table. Holding a bachelor's degree in construction and engineering management, he has laid a solid foundation in the technical aspects of the industry. Further elevating his qualifications, the author pursued and attained a Master of Science degree in Commercial Management and Quantity Surveying, a discipline that marries the technicalities of construction with the nuances of business management.

Complementing their construction-centric education, the author also delved into the realm of data analytics, acquiring a degree that marks a significant pivot in their career trajectory. This educational journey is crowned by their chartered status from two prestigious institutions: the Royal Institution of Chartered Surveyor and the British Computer Society, reflecting a rare confluence of construction expertise and computational acumen.

In recent years, the author has shifted their focus towards the data world, dedicating the past three years to working intensively in this domain. Their interest particularly lies in the deployment of Artificial Intelligence (AI) and Machine Learning (ML) within the construction industry, a sector ripe for digital transformation. Recognizing the potential of AI and ML to revolutionize traditional practices, the author has been at the forefront of integrating these technologies into construction processes, aiming to enhance efficiency, accuracy, and overall project management.

One of the author's notable contributions is the development of a series of chatbots using open-source large language models. These chatbots represent a significant innovation, leveraging the power of AI to streamline communication, automate routine tasks, and provide intelligent assistance in various construction-related scenarios. The author's work in this area not only showcases their technical prowess but also their commitment to driving the construction industry forward through the adoption of cutting-edge technologies. Their unique blend of construction knowledge, data analytics expertise, and passion for AI and ML positions them as a visionary figure, poised to make a lasting impact on the industry.

You can reach out to me on my LinkedIn page: https://www.linkedin.com/in/mohamedashour-0727/ or via email on mohamed_ashour@apcmasterypath.co.uk.

Acknowledgement

I would like to extend my sincere acknowledgment and appreciation to Venelin Valkov for his invaluable contribution in preparing extensive educational material related to large language models on YouTube. His dedication to providing informative content has been a tremendous resource for those seeking to understand these complex models better.

Venelin Valkov's YouTube channel, accessible at (https://www.youtube.com/@venelin_valkov), hosts a wealth of videos that delve into the workings of large language models. The specific video I utilized from his channel, titled [Large Language Models Explained](https://www.youtube.com/watch?v=s0qXmwdssHc&ab_channel=Venelin Valkov), provided a concise and insightful explanation of how these models function and focused on their end results.

I want to acknowledge the effort and commitment I have invested in comprehending these models, dissecting their components, optimizing their performance, and conducting extensive research. My goal was to develop a comprehensive report that not only outlines various use cases but also highlights areas for further development, identifies limitations, and presents step-by-step implementation procedures. Venelin Valkov's educational content played a crucial role in facilitating my understanding and research in this field.

Once again, I express my gratitude to Venelin Valkov for his exceptional contribution to the educational resources available on large language models. His work has been instrumental in my journey of exploration and understanding.

Disclaimer

You can contact the author of this document via email on mohamed_ashour@apcmasterypath.co.uk & mo_ashour1@outlook.com.

Table of Contents

List of Figures

Executive summary

This guidebook is a comprehensive manual designed to facilitate the understanding and application of Microsoft's Llava (Large Language and Vision Assistant) Large Language Model (LLM) in the domain of image querying. It serves as a pivotal resource for professionals and enthusiasts in AI, machine learning, and computer vision.

- *Overview of Context Recognition from Images:* Begins with an introduction to the groundbreaking capabilities of Llava LLM in contextualizing and interpreting images, setting the stage for advanced image recognition.
- *Scope of This Guide Booklet:* Defines the target audience, the breadth of topics covered, and the objectives, ensuring readers have clear navigation and understanding of the guide's content.
- *Context recognition from images:* Context recognition from images, enabled by large language models, significantly enhances image processing in machine vision by identifying and classifying objects within their environmental context. Deep learning, especially through Convolutional Neural Networks, drives advancements in this field, aiding in diverse applications from healthcare to security. This technology offers comprehensive scene understanding and accurate object interpretation, revolutionizing image analysis.
- *Use Cases:* Illustrates various practical applications of Llava LLM, ranging from digital asset management to enhanced user interfaces and automated content generation, showcasing the model's versatility.
- *Limitations:* Candidly discusses the current boundaries and challenges of Llava LLM in image querying, such as accuracy in diverse conditions and computational requirements, providing a balanced view.
- *Areas for Development:* Highlights the potential future enhancements in Llava LLM, including expanding dataset diversity, increasing efficiency, and integrating emerging technologies.
- *Deployment Steps for Image Recognition:* Offers a detailed, step-by-step guide to implementing Llava LLM for image recognition, covering setup, configuration, and practical execution.
- *Deployment Steps for Optical Character Recognition:* Provides a comprehensive walkthrough for employing Llava LLM in OCR tasks, detailing pre-processing, model tuning, and operational deployment.

This guidebook aims to empower users to fully leverage the Llava Large Language Model's capabilities in image querying, driving innovation in the field of AI-driven image analysis.

Scope of this guide booklet

This guidebook is designed to provide comprehensive insights into the process of creating chatbots capable of querying images using Llava Large Language Models

(LLM). While this guidebook aim to offer a thorough understanding of the subject matter, it is important to note the following:

1. ***Focus on Image Queries:*** This guide primarily focuses on the integration of Llava LLM into chatbots to facilitate image-based queries. We will explore techniques, best practices, and use cases related to leveraging Llava LLM's capabilities for image understanding and retrieval.
2. ***Not a Comprehensive LLM Guide:*** This guidebook is not intended to serve as a comprehensive resource for understanding Llava LLM in its entirety. We will delve deeply into the specific aspect of image querying, and readers seeking a broader understanding of Llava LLM should consider additional resources.
3. ***Technical Prerequisites:*** To effectively use this guide, readers should have a basic understanding of chatbot development, image processing, and some familiarity with Llava LLM. This guidebook will assume a foundational knowledge of these topics.
4. ***Dynamic Field:*** The field of natural language processing and large language models is constantly evolving. While this guide provides insights based on current knowledge, advancements in Llava LLM or related technologies may lead to changes or refinements in the techniques discussed.
5. ***Use Cases May Vary:*** The application of chatbots for image queries can vary greatly across industries and scenarios. This guidebook will present general principles, but readers should adapt and refine these principles based on their specific use cases and requirements.

Readers are encouraged to use this guide as a valuable starting point and explore additional resources and updates to stay informed about the latest developments in the field of chatbots, image querying, and Llava LLM.

Context recognition from images

In order to digest this guidebook in a deeper manner, the reader has to have a brief background about context recognition from images. The following titles can give the reader a sense of how large language models arrive at getting structured data from unstructured data sources such as images.

What is Image Recognition?

Image recognition refers to the ability of software to identify objects, places, people, writing, and actions in digital images. This technology, pivotal in the field of machine vision, employs algorithms and models to process and interpret visual information from digital images [1].

Contextual Image Classification

Contextual image classification, a key topic in pattern recognition within computer vision, involves classifying images based on the context or surrounding information present in them. This approach enhances the accuracy and relevance of image classification by considering the environment and interactions within the image [2].

The Role of Context in Object Recognition

In object recognition, context plays a crucial role. It is not just about matching a target object with available image features but also involves rejecting irrelevant information and focusing on how objects relate to their surroundings. Context aids in more accurate and meaningful interpretation of objects within their environments [3].

Image Recognition with Deep Learning and Neural Networks

Deep learning and neural networks have revolutionized image recognition. These technologies enable the training of models that can recognize and categorize complex patterns in images. Use cases span across various sectors, including healthcare, automotive, and security [4].

Context-based Image Explanations for Deep Neural Networks

Deep neural networks use context-based explanations to enhance scene classification. This involves assigning labels to images based on their overall context, such as identifying settings like playgrounds or beaches, thereby providing a more comprehensive understanding of the scene [5].

Convolutional Neural Networks and Their Applications

Convolutional Neural Networks (CNNs) are a type of deep learning model particularly effective in image recognition tasks. They have found applications in numerous fields like radiology for tasks including classification and segmentation, significantly contributing to advancements in medical imaging [6].

Use cases

The integration of context recognition from images using large language models (LLMs) presents a transformative potential in the construction industry. This synergy leverages the capabilities of LLMs to understand and interpret contextual information from visual data, enhancing various applications in construction.

1. *Enhanced Project Planning and Design:* By analyzing construction site images, LLMs can provide contextual insights for project planning. This includes assessing site conditions, understanding geographical and environmental context, and aiding in design decisions [7].
2. *Improving Safety and Compliance:* Context recognition can be used to monitor construction sites for safety compliance. LLMs can process images to identify potential hazards, ensuring adherence to safety standards and regulations [8].
3. *Facilitating Maintenance and Inspection:* LLMs can analyze images from inspections, identifying structural issues or areas needing maintenance by understanding the context of the construction elements in the images [9].
4. *Material and Resource Management:* By recognizing and categorizing materials and equipment from site images, LLMs can assist in inventory management, optimizing resource allocation and usage [10].

5. *Documentation and Reporting:* LLMs can automatically generate detailed reports and documentation based on images captured on-site, providing contextual descriptions and analyses that aid in project tracking and stakeholder communication [11].
6. *Enhanced Collaboration and Decision Making:* Contextual image recognition allows for more informed decision-making by providing detailed visual insights, facilitating better collaboration among architects, engineers, and construction managers [12].

In summary, the use of context recognition from images via large language models holds substantial promise for the construction industry, offering advancements in safety, efficiency, planning, and overall project management.

Limitations

LLaVA is a large language model that combines text and image processing capabilities. While it demonstrates impressive abilities in understanding images, it also has its limitations in context recognition from images:

1. *Dataset's Size:* LLaVA was trained on a relatively small dataset. Limited training data can lead to reduced performance compared to models with access to larger and more diverse datasets [19].
2. *Contextual Understanding:* While LLaVA can process both text and images, its ability to understand context from images may not be as robust as specialized computer vision models. Large language models like LLaVA primarily excel in natural language understanding, and their image understanding capabilities might lag behind dedicated computer vision models [20].
3. *Adversarial Attacks:* LLaVA, like other large language models, can be vulnerable to adversarial attacks using images. Adversarial perturbations can trick the model into producing incorrect or biased results by subtly altering the input image [21].
4. *Complex Multi-Modal Prompts:* Handling complex multi-modal prompts, which involve both text and images, can be challenging for LLaVA. It may struggle to interpret and respond accurately to such prompts, especially when the context between the text and image is intricate [22].
5. *Context Awareness:* While LLaVA can be context-aware, its context recognition from images may not always capture nuanced details or context-specific information effectively. Specialized computer vision models may outperform LLaVA in this regard.

In summary, while LLaVA offers impressive capabilities in combining text and image processing, it has limitations related to dataset size, contextual understanding from images, vulnerability to adversarial attacks, and handling complex multi-modal prompts. These limitations emphasize the need for specialized models and additional training data to address these challenges effectively.

Areas for development

Improving Large Language Models (LLMs) for better context recognition from images involves several key development strategies:

1. *Enhanced Multimodal Training:* Integrating more robust multimodal training, combining textual and visual data, helps LLMs better understand the context of images. This training should include diverse datasets with comprehensive image-text pairings [13].
2. *Advanced Object and Scene Recognition:* Incorporating sophisticated object and scene recognition algorithms enables LLMs to more accurately identify and interpret visual elements within images [14].
3. *Context-Aware Algorithms:* Developing algorithms specifically designed to understand the context, such as the relationships between objects in an image and their surroundings, will significantly improve context recognition capabilities [15].
4. *Improved In-Context Learning Techniques:* Focusing on in-context learning (ICL) approaches allows LLMs to learn and adapt from the context within data samples, enhancing their ability to interpret images in varied scenarios [16].
5. *Data Protection and Privacy Considerations:* Ensuring ethical use and protection of data, particularly in sensitive fields like healthcare, is crucial in the development of LLMs for image context recognition [17].
6. *Interdisciplinary Collaboration:* Collaboration between experts in language processing, computer vision, and related fields can drive innovation and comprehensive solutions in LLM development for image context recognition [18].

Through these strategies, LLMs can be developed to more effectively recognize and interpret context in images, expanding their applicability across various domains.

Development environment

The main aspects of the development environment are as provided below:

- Integrated Development Environment (IDE): Visual Studio Cod.
- Python version : 3.11.5
- Operating system: Linux Pop OS
- Packages required:
  - glob
  - requests
  - transformers
  - torch (with CUDA enabled)
  - PIL
  - accelerate
  - transformers
  - bitsandbytes
  - llava-torch
- Recommended computer specifications:
  - CPU : Intel Core i7 12700
  - GPU : RTX 3080

- RAM: 16GB DDR5

Deployment Steps for Image recognition

This section is going to present the steps that need to be followed in order to be able to query your desired image(s) and get useful structured data from the image and deploy it according to the needs. You will find below the steps needed to deploy the system in place.

Step 1 - Importing Python Packages & Modules

Overview

This first step is for importing various Python packages and modules, primarily related to the 'torch' and 'llava' libraries, for tasks involving image processing and model loading:

1. textwrap: A utility for text wrapping and filling.
2. glob: This module is used to retrieve files/pathnames matching a specified pattern.
3. io.BytesIO: Supports an in-memory buffer for binary data. It's used for operations like reading and writing to a buffer, as if it were a file.
4. requests: A library for making HTTP requests in Python.
5. torch: The primary package for PyTorch, a popular machine learning library.
6. llava.constants: This imports specific constants from the 'llava' library, like DEFAULT_IMAGE_TOKEN and IMAGE_TOKEN_INDEX, which are probably used for image processing tasks.
7. llava.conversation: Imports tools related to conversation handling in the 'llava' framework, such as SeparatorStyle and conv_templates.
8. llava.mm_utils: This imports various utilities from 'llava' for multimodal (mm) tasks, like KeywordsStoppingCriteria, get_model_name_from_path, process_images, and tokenizer_image_token. These are likely used for handling images, processing them, and integrating them with text or other data types.
9. llava.model.builder: From here, a function to load pretrained models is imported, possibly for initializing machine learning models.
10. llava.utils: This includes utility functions from 'llava', including disable_torch_init, which might be used to modify or control the initialization behavior of PyTorch.
11. PIL.Image: The Python Imaging Library (PIL) is imported for image processing tasks.

Overall, the code is preparing the environment for tasks that involve image processing, model handling, and possibly multimodal (text and image) data processing, using the PyTorch and llava frameworks. You can check below the code related to this step.

Code

```
#Importing necessary packages including torch and llava. From Llava multiple
image processing and model loading packages are imported.

import textwrap
import glob
from io import BytesIO
import requests
import torch
from llava.constants import DEFAULT_IMAGE_TOKEN, IMAGE_TOKEN_INDEX
from llava.conversation import SeparatorStyle, conv_templates
from llava.mm_utils import (
        KeywordsStoppingCriteria,
        get_model_name_from_path,
        process_images,
        tokenizer_image_token
)
from llava.model.builder import load_pretrained_model
from llava.utils import disable_torch_init
from PIL import Image
```

Step 2 - Initializing PyTorch

Overview

In PyTorch, initialization usually refers to the setup processes that occur when PyTorch is loaded, often involving setting up default configurations, initializing random number generators, or preparing the environment for GPU acceleration. A function like disable_torch_init() might be used to alter or skip these default initialization steps, perhaps for reasons such as customizing the initialization process, reducing startup time, or avoiding conflicts with certain system configurations or other libraries.

Code:

```
#Necessary step from the llava documentation
disable_torch_init()
```

Step 3 - Importing the Llava model

Overview

This step pivots around the import of Llava model for image processing. The version required to be imported is version 1.5. Here's a breakdown of each line required for this step:

- *MODEL= "liuhaotian/llava-v1.5-13b":* This line declares a variable named MODEL and assigns it a string value "liuhaotian/llava-v1.5-13b". This string seems to be a path or identifier for a specific model version within the Llava library. The name suggests that this model is version 1.5 of a 13-billion parameter model created or maintained by a user or organization named 'liuhaotian'.
- *model_name=get_model_name_from_path(MODEL):* This line calls a function get_model_name_from_path, passing the previously defined MODEL variable as an argument. The purpose of this function is likely to extract or determine the model name from the provided path or identifier. The result of this function is then stored in a new variable model_name.
- *model_name:* This line is simply the variable model_name, which, when executed in a Python environment like a Jupyter notebook or an interactive Python shell, would display the value of model_name, i.e., the extracted model name.

Overall, this code is used to identify and potentially prepare for the downloading or loading of a specific machine learning model, "llava-v1.5-13b", from a repository or library, likely managed by 'liuhaotian'.

Code:

```
#Downloading and importing the 4 bit quantized model of Llava
MODEL= "liuhaotian/llava-v1.5-13b"
model_name=get_model_name_from_path(MODEL)
model_name
```

Step 4 - Importing images from a local saved directory

Overview

This step takes into account writing a script for importing JPEG images from a specified directory into two separate lists: one for the image objects and another for their filenames. The full explanation of the code lines required for this step is provided below.

- *Path Specification:* path = '/home/user/Pictures/*.jpg' - This line sets a variable path to a string that specifies the directory path where the images are stored. The *.jpg part means that it is looking for all files in that directory with a .jpg extension, which are typically JPEG images.
- *Initializing Lists:*
  - images = [] initializes an empty list named images to store the image objects.
  - filenames = [] initializes an empty list named filenames to store the names of the image files.
- *Reading Images:*

- The for loop for filename in glob.glob(path): iterates over all files that match the specified path pattern (all JPEG images in the given directory). glob.glob(path) returns a list of file paths matching the path pattern.
- Inside the loop, img = Image.open(filename) opens each image file using the Image.open() method from the PIL (Python Imaging Library) and stores the resulting image object in the variable img.
- images.append(img) adds the image object to the images list.
- filenames.append(filename) adds the file path (filename) of the image to the filenames list.

In summary, this script loads all JPEG images from the specified directory and stores them as image objects in one list (images), while also keeping track of their file paths in another list (filenames).

Code:

```
#Importing the images from a local saved directory

# Path to your Pictures directory
path = '/home/mohamedashour/Pictures/*.jpg'
```

```
# List to store images and their corresponding file names
images = []
filenames=[]
```

```
# Read each image file
for filename in glob.glob(path):
img = Image.open(filename)
images.append(img)
filenames.append(filename)
```

Step 5- Importing the tokenizer

Overview

In this step it is quite important to import a tokenizer for breaking down the text into chunks and facilitating the understanding by the model.Also a pretrained model is required to be imported, in addition to an image processor, and context length settings for a 'llava' model using a function load_pretrained_model. Here's a breakdown of its components:

- <u>*Importing Components:*</u>
    - tokenizer: A component for tokenizing input data (text or possibly other modalities) into a format suitable for processing by the model.
    - model: The actual pretrained machine learning model from the 'llava' library.

    - image_processor: A utility for processing images, likely converting them into a format suitable for input into the model.
    - context_len: This might refer to the length of the context (number of tokens) the model can handle or is configured to process.
- *Function Call load_pretrained_model():*
    - model_path=MODEL: Specifies the path or identifier of the model to be loaded. MODEL is a variable that likely contains this information.
    - model_base=None: This argument might be for specifying the base model architecture or version, but here it's set to None, indicating the default base model is used or the base is not applicable.
    - model_name=model_name: Specifies the name of the model to be loaded. model_name is a variable containing this name.
    - load_4bit=True: This indicates that the model to be loaded is quantized to 4-bit. Model quantization is a process that reduces the precision of the weights and activations of models to accelerate inference and decrease model size.

In summary, the code is used for setting up a 'llava' machine learning model environment by loading a specific pretrained model along with its tokenizer, image processor, and context length configuration, with the model being a 4-bit quantized version for efficient usage.

Code:

```
#Import the tokenizer and the pretrained model for llava
tokenizer,model,image_processor,context_len=load_pretrained_model(
model_path=MODEL,model_base=None,model_name=model_name,load_4bit=Tru
e
)
```

Step 6 - Loading the desired image

Overview

Having saved all the required images in Step 4 above, This step is all about summoning the required image from the saved array of images. Here's a breakdown of what the code does:

- *`required_image=3`:* This line assigns the value `3` to the variable `required_image`. This variable is used to specify which image from a list of images should be displayed. In this case, it's set to `3`, which means the fourth image in the list (Python uses zero-based indexing, so the first image is at index 0, the second at index 1, and so on).
- *`print(f"Image Name:* {filenames[required_image].split('/')[-1]}")`: This line prints the name of the image whose index is specified by `required_image`. Let's break it down further:

  - `filenames[required_image]` accesses the file name of the image at the index specified by `required_image`.
  - `.split('/')[-1]` splits the file path using the forward slash ('/') as a delimiter and then takes the last part of the resulting split, which is typically the file name. This is done to extract just the image's name from its full path.
- *`image=images[required_image]`:* This line assigns the image object at the specified index (`required_image`) from the `images` list to the variable `image`. This assumes that the `images` list contains a collection of image objects or references.
- *`image`:* This line, when executed, will display the image specified by the `image` variable. The code uses a library or tool that allows displaying images directly in the script, such as displaying the image in a Jupyter Notebook or within an interactive Python environment.

In summary, this code allows the user to select a specific image from a list of images (`filenames` contains the file paths to these images and `images` contains the actual image data) by setting the `required_image` variable and then displays the name of the selected image and the image itself. It's designed for testing purposes with a limited number of images. The image below represents the output of this step in Visual Studio Code.

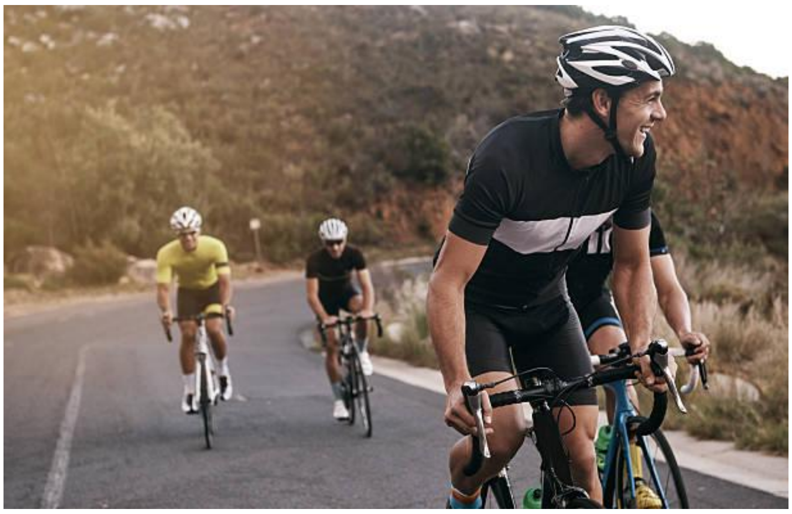


Figure 1: Choosing the image of a cyclist from an array of images.

Code:

```
# Gives the user the option to show the name and the content of the chosen image.
# The required_image index must be between 0 & 3.
#Only 4 images were downloaded as for the purpose of testing this model.
required_image=3
print(f"Image Name: {filenames[required_image].split('/')[-1]}") # Displaying the image name
image=images[required_image]
image
```

Step 7 - Converting images into tensors

Overview

This step is all about converting the image loaded into a tensor to be understood and processed by the large language model. To do that I created a function named process_image that is designed to perform two main tasks:

- *Convert an image into a tensor:* The input to this function is an image, and the function's goal is to convert that image into a tensor, which is a multi-dimensional array commonly used in deep learning and neural network operations.
- *Process the image for a specific model (referred to as Llava model):* After converting the image into a tensor, the function applies some processing to make it suitable for the Llava model. Model and image_processor are defined in Step 5 - Importing the tokenizer .

Having understood the above, here's a breakdown of the code:

- *def process_image(image):*: This line defines a function named process_image that takes one argument, image, which represents the input image to be processed.
- *args={"image_aspect_ratio":"pad"}:* This line creates a dictionary args with a single key-value pair. It sets the key "image_aspect_ratio" to the value "pad". These arguments are likely used to control the behavior of the image processing.
- *image_tensor=process_images(image,image_processor,args):* This lin calls a function process_images with three arguments:
  - image: The input image.
  - image_processor: An image processing function or object.
  - args: The dictionary of arguments, which includes the image aspect ratio.
  - The purpose of this line is to use the image_processor to process the input image with the specified arguments and convert it into a tensor. The resulting tensor is stored in the image_tensor variable.
- *return image_tensor.to(model.device,dtype=torch.float16):* This line performs two operations:
  - image_tensor.to(model.device, dtype=torch.float16): It converts the image_tensor to a new tensor with a specific data type (dtype) of torch.float16. This is typically done to change the precision of the tensor. torch.float16 is a 16-bit floating-point format, which can be used to reduce memory usage and accelerate computation in some cases.
  - return: Finally, the processed tensor is returned as the output of the process_image function.

The second part of the code is designed to make use of the created function to process the loaded image.Here's a breakdown of its components:

- *Image Processing:*

- processed_image = process_image(image): This line calls a function named process_image, passing image as an argument. The image variable is expected to be an image object or data. The process_image function is meant to perform some form of processing on the image, which could include operations like resizing, filtering, color correction, normalization, or format conversion. The result of this processing is stored in the variable processed_image.

- *Checking Type and Shape:*
    - type(processed_image), processed_image.shape: This line is evaluating two properties of the processed_image object.
        - type(processed_image): This part of the code returns the data type of processed_image. This tells you what type of object processed_image is, which is often a numpy array or a similar array-like structure in the context of image processing.
        - processed_image.shape: This part returns the dimensions or shape of the processed_image. For image data, this typically includes its height, width, and the number of color channels (like 3 for an RGB image).

In summary, this Step defines a function that takes an input image, converts it into a tensor, applies some image processing with specified arguments, changes the data type of the tensor to torch.float16, and then returns the processed tensor.

Code

```
#Building a function to convert the required image into a tensor and process it
for the Llava model
def process_image(image):
        args={"image_aspect_ratio":"pad"}
        image_tensor=process_images(image,image_processor,args)
return image_tensor.to(model.device,dtype=torch.float16)

#processing the chosen image above using the function developed in the
previous step
processed_image=process_image(image)
type(processed_image),processed_image.shape
```

Step 8 - Defining the chat model and creating Prompt Function

Overview

This step is quite crucial because it makes use of all the above loading and then paves the way to build a method to query the chosen/loaded image using the specified version of Llava defined in the revious step. Here's a breakdown of the step’s components:

Model Definition:

- *CONV_MODE="llava_v0":* This line sets a variable CONV_MODE to the string "llava_v0", which presumably specifies the version or mode of the 'llava' model to be used for the conversation. It implies that the model has different modes or versions for processing inputs.

Prompt Template Function create_prompt:

- *def create_prompt(prompt: str):* This line defines a function named create_prompt that takes a string argument prompt. This function is designed to format a prompt for input into the 'llava' model.
- *conv=conv_templates[CONV_MODE].copy():* This line gets a conversation template from conv_templates for the specified CONV_MODE and makes a copy of it. The conv_templates likely contains predefined templates for structuring conversation data.
- *roles=conv.roles:* Extracts roles from the conversation template. In a conversation model, roles typically define the participants in the conversation (e.g., user, system).
- *prompt=DEFAULT_IMAGE_TOKEN+"\n"+prompt:* The original prompt is modified by prepending DEFAULT_IMAGE_TOKEN and a newline. DEFAULT_IMAGE_TOKEN is probably a special token used to indicate the presence of an image in the conversation.
- *conv.append_message(roles[0],prompt):* Adds a message to the conversation with the first role (possibly the user or the system) and the modified prompt.
- *conv.append_message(roles[1],None):* Adds another message with the second role, but with no content (None). This could be for the model to generate a response.
- *return conv.get_prompt(),conv:* The function returns the formatted prompt and the updated conversation object.

In summary, the create_prompt function formats a prompt for a conversational model by incorporating a default image token and structuring the input according to predefined roles and conversation templates. This formatted prompt is then used for generating responses from the 'llava' model in a conversational setting involving images.

Code

```
#Defining the model that will be used for conversation with the image
CONV_MODE="llava_v0"
#Using the prompt template as per the llava torch documentation
def create_prompt(prompt:str):
conv=conv_templates[CONV_MODE].copy()
roles=conv.roles
```

```
prompt=DEFAULT_IMAGE_TOKEN+"\n"+prompt
conv.append_message(roles[0],prompt)
conv.append_message(roles[1],None)
return conv.get_prompt(),conv
```

Step 9 - Interacting with the AI model

Overview

This step pivots around defining a function to interact with an AI model for image interrogation, using a combination of text prompts and processed image data. Here's a breakdown of its components:

- *Default System Message Setup:*
  - prompt,_ = create_prompt("Describe the image"): This line uses a function create_prompt to create a prompt text for the AI model. The prompt is "Describe the image". The function likely formats this text for use with the model. The underscore _ is a conventional way to ignore the second returned value from the function.
  - print(prompt): This prints the formatted prompt to the console.
- *Function Definition ask_image:*
  - def ask_image(image: Image, prompt: str): This defines a function ask_image that takes an Image object and a string prompt as its parameters.
  - image_tensor = process_image(image): Processes the input image and converts it into a tensor (a multi-dimensional array used in deep learning models).
  - prompt, conv = create_prompt(prompt): Reformats the provided prompt using the create_prompt function.
  - input_ids: This is a complex expression that involves several steps:
  - Tokenizes the prompt text.
  - Adds a special token for the image using tokenizer_image_token.
  - Adjusts the tensor's dimensions with .unsqueeze(0).
  - Moves the tensor to the same device as the model (e.g., CPU or GPU).
  - stop_str = conv.sep if conv.sep_style != SeparatorStyle.TWO else conv.sep2: Determines the stopping condition for the model's response generation based on the separator style.
  - stopping_criteria = KeywordsStoppingCriteria(...): Sets criteria for when the model should stop generating text, based on the defined stop_str.
  - with torch.inference_mode(): A context manager that sets the PyTorch model to inference mode, optimizing performance for forward passes without backpropagation.

- output_ids = model.generate(...): Generates a response from the model using the input IDs, image tensor, and various generation parameters (like sampling temperature and maximum new tokens).
- return tokenizer.decode(...): Decodes the generated output into human-readable text, skipping special tokens, and returns the result.

In summary, this code defines a function ask_image for submitting images and text prompts to a machine learning model (presumably one that can process both textual and visual information, like Llava). The function processes the image, formats the prompt, generates a response from the model, and decodes this response into a readable format.

Code

```
#The following is the default system message used by the AI model
prompt,_=create_prompt("Describe the image")
print(prompt)

#Creating a function to interrogate the images using the model downloaded
def ask_image(image:Image,prompt:str):
        image_tensor=process_image(image)
        prompt,conv=create_prompt(prompt)
        input_ids=(
                tokenizer_image_token(prompt,tokenizer,IMAGE_TOKEN_INDEX,
                return_tensors="pt")
                .unsqueeze(0)
                .to(model.device)
        )

stop_str=conv.sep if conv.sep_style!=SeparatorStyle.TWO else conv.sep2
stopping_criteria=
KeywordsStoppingCriteria(keywords=[stop_str],tokenizer=tokenizer,input_ids=in
put_ids)
```

```
with torch.inference_mode():
        output_ids=model.generate(
                input_ids,
                images=image_tensor,
                do_sample=True,
                temperature=0.01,
                max_new_tokens=512,
                use_cache=True,
                stopping_criteria=[stopping_criteria]
        )
return tokenizer.decode(
                output_ids[0,input_ids.shape[1]:],skip_special_tokens=True
).strip()
```

Step 10 - Querying the image

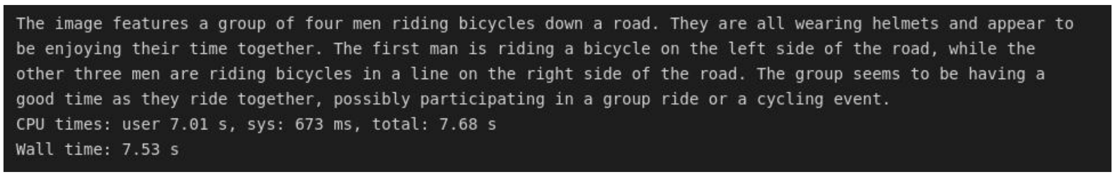
Overview

The code snippet is executing a function to analyze or interrogate an image using a predefined AI model and then prints the result in a formatted manner. Here's a detailed breakdown:

- *Timing the Execution: This is an optional step for monitoring purposes*
  - %%time: This is a Jupyter Notebook magic command that measures and displays the total execution time of the cell in which it is placed. It's used for performance analysis, showing how long it takes to run the code within the cell.
- *Image Interrogation:*
  - result = ask_image(image, "Your desired prompt"): This line calls the function ask_image, which was defined earlier in the code. This function is expected to take an image (image) and a text prompt ("Your desired prompt") as inputs. It processes the image, creates a prompt, and uses an AI model to generate a textual description or analysis of the image. The output of this function is stored in the variable result.
- *Printing the Result:*
  - print(textwrap.fill(result, width=110)): This line uses the textwrap.fill function from Python's textwrap module to format the string in result. It wraps the text so that each line is at most 110 characters wide. This makes long output more readable, especially if the result is a lengthy paragraph. The formatted result is then printed to the console.

In summary, the code interrogates an image using an AI model with the desired end-user prompt, measures the time taken to perform this operation, and then prints the result in a neatly formatted way with a maximum line width of 110 characters. This is typically used in scenarios where the image content needs to be described or analyzed using AI, and the result needs to be presented in a readable format.
The two images below show two results for different queries to the image on Visual Studio Code.

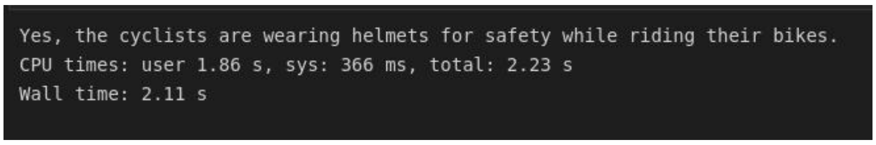
Query 1 - Describe the image



```
The image features a group of four men riding bicycles down a road. They are all wearing helmets and appear to
be enjoying their time together. The first man is riding a bicycle on the left side of the road, while the
other three men are riding bicycles in a line on the right side of the road. The group seems to be having a
good time as they ride together, possibly participating in a group ride or a cycling event.
CPU times: user 7.01 s, sys: 673 ms, total: 7.68 s
Wall time: 7.53 s
```

Figure 2: Query about Image Description

Query 2 - Do the cyclists wear helmets



```
Yes, the cyclists are wearing helmets for safety while riding their bikes.
CPU times: user 1.86 s, sys: 366 ms, total: 2.23 s
Wall time: 2.11 s
```

Figure 3: Query about the cyclists wearing a helmet

Code

```
%%time
#Using the above parameters for the prompting as well as the processing of the image, tokenizing of the prompt, encoding and decoding within the above function to interrogate the image.
result=ask_image(image,"Describe the image")
print(textwrap.fill(result,width=110))
```

```
%%time
#Using the above parameters for the prompting as well as the processing of the image, tokenizing of the prompt, encoding and decoding within the above function to interrogate the image.
result=ask_image(image,"do the cyclists wear helmets?")
print(textwrap.fill(result,width=110))
```

Deployment Steps for Optical Character Recognition

This section follows the previous section and makes use of all the steps deployed previously. The only change would be in steps 6 and 10. You will find below a summary of the updated steps and changes from the previous section.

The following steps will not change from the previous section:

- Step 1 - Importing Python Packages & Modules
- Step 2 - Initializing PyTorch
- Step 3 - Importing the Llava model
- Step 4 - Importing images from a local saved directory
- Step 5- Importing the tokenizer
- Step 7 - Converting images into tensors
- Step 8 - Defining the chat model and creating Prompt Function
- Step 9 - Interacting with the AI model

There are two main steps which will change from the previous section:

- Step 6 - Loading the desired image
- Step 10 - Querying the image

Step 6 - Loading the desired image

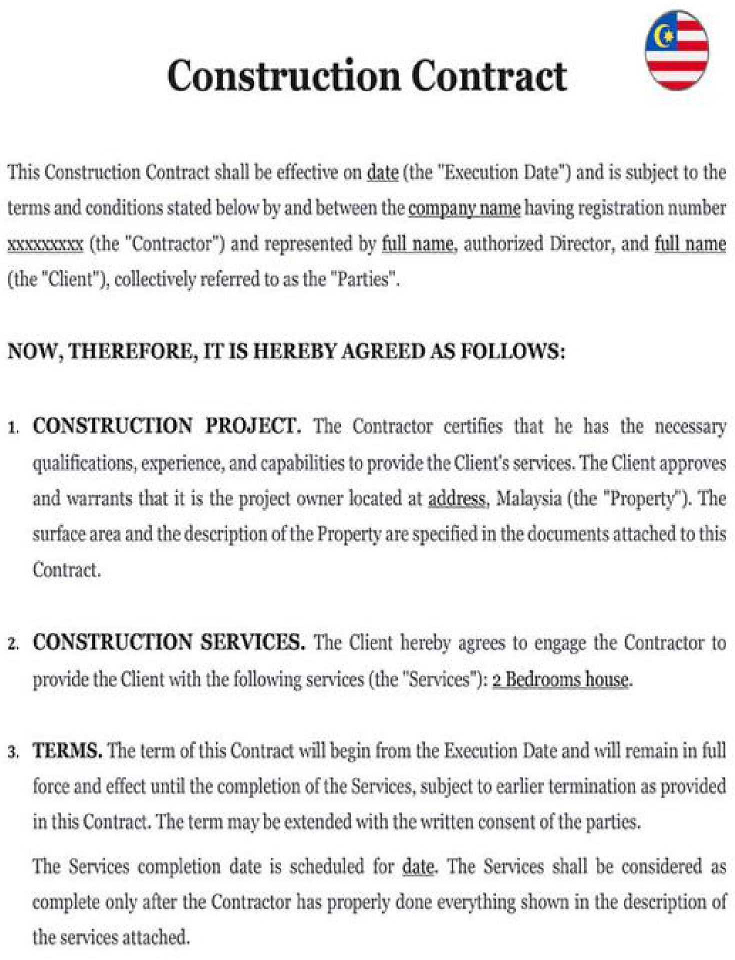
Overview

Having saved all the required images in Step 4 above, This step is all about summoning the required image from the saved array of images. Here's a breakdown of what the code does:

- *`required_image=1`:* This line assigns the value `1` to the variable `required_image`. This variable is used to specify which image from a list of images should be displayed. In this case, it's set to `3`, which means the fourth image in the list (Python uses zero-based indexing, so the first image is at index 0, the second at index 1, and so on).

- *`print(f"Image Name:* {filenames[required_image].split('/')[-1]}")`: This line prints the name of the image whose index is specified by `required_image`. Let's break it down further:
  - `filenames[required_image]` accesses the file name of the image at the index specified by `required_image`.
  - `.split('/')[-1]` splits the file path using the forward slash ('/') as a delimiter and then takes the last part of the resulting split, which is typically the file name. This is done to extract just the image's name from its full path.
- `image=images[required_image]`: This line assigns the image object at the specified index (`required_image`) from the `images` list to the variable `image`. This assumes that the `images` list contains a collection of image objects or references.
- `image.resize`: This line, when executed, will display the image specified by the `image` variable and it would be resized to present an image of 600x700 pixels. The code uses a library or tool that allows displaying images directly in the script, such as displaying the image in a Jupyter Notebook or within an interactive Python environment.

In summary, this code allows the user to select a specific image from a list of images (assuming that `filenames` contains the file paths to these images and `images` contains the actual image data) by setting the `required_image` variable and then displays the name of the selected image and the image itself. It's designed for testing purposes with a limited number of images. The image below represents the output of this step in Visual Studio Code.



Construction Contract

This Construction Contract shall be effective on date (the "Execution Date") and is subject to the terms and conditions stated below by and between the company name having registration number xxxxxxxxx (the "Contractor") and represented by full name, authorized Director, and full name (the "Client"), collectively referred to as the "Parties".

NOW, THEREFORE, IT IS HEREBY AGREED AS FOLLOWS:

1. CONSTRUCTION PROJECT. The Contractor certifies that he has the necessary qualifications, experience, and capabilities to provide the Client's services. The Client approves and warrants that it is the project owner located at address, Malaysia (the "Property"). The surface area and the description of the Property are specified in the documents attached to this Contract.

2. CONSTRUCTION SERVICES. The Client hereby agrees to engage the Contractor to provide the Client with the following services (the "Services"): 2 Bedrooms house.

3. TERMS. The term of this Contract will begin from the Execution Date and will remain in full force and effect until the completion of the Services, subject to earlier termination as provided in this Contract. The term may be extended with the written consent of the parties.

   The Services completion date is scheduled for date. The Services shall be considered as complete only after the Contractor has properly done everything shown in the description of the services attached.

Figure 4: Output of Step 6 for OCR on VScode

Code:

```
# Gives the user the option to show the name and the content of the chosen image.
# The required_image index must be between 0 & 3.
#Only 4 images were downloaded as for the purpose of testing this model.
required_image=1
print(f"Image Name: {filenames[required_image].split('/')[-1]}") # Displaying the image name
image=images[required_image]
image,resize((600,700))
```

Step 10 - Querying the image

Overview

The code snippet is executing a function to analyze or interrogate an image using a predefined AI model and then prints the result in a formatted manner. Here's a detailed breakdown:

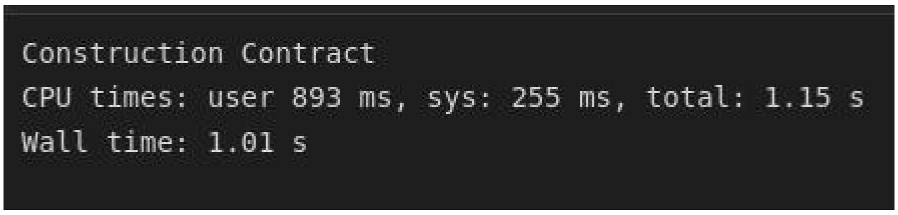
- *Timing the Execution: This is an optional step for monitoring purposes*

- %%time: This is a Jupyter Notebook magic command that measures and displays the total execution time of the cell in which it is placed. It's used for performance analysis, showing how long it takes to run the code within the cell.

- *Image Interrogation:*
  - result = ask_image(image, "Your desired prompt"): This line calls the function ask_image, which was defined earlier in the code. This function is expected to take an image (image) and a text prompt ("Your desired prompt") as inputs. It processes the image, creates a prompt, and uses an AI model to generate a textual description or analysis of the image. The output of this function is stored in the variable result.
- *Printing the Result:*
  - print(textwrap.fill(result, width=110)): This line uses the textwrap.fill function from Python's textwrap module to format the string in result. It wraps the text so that each line is at most 110 characters wide. This makes long output more readable, especially if the result is a lengthy paragraph. The formatted result is then printed to the console.

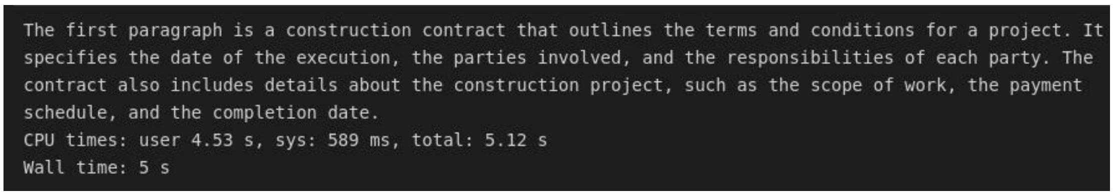
In summary, the code interrogates an image using an AI model with the desired end-user prompt, measures the time taken to perform this operation, and then prints the result in a neatly formatted way with a maximum line width of 110 characters. This is typically used in scenarios where the image content needs to be described or analyzed using AI, and the result needs to be presented in a readable format.

The two images below show two results for different queries to the image on Visual Studio Code.



```
Construction Contract
CPU times: user 893 ms, sys: 255 ms, total: 1.15 s
Wall time: 1.01 s
```

Figure 5: Output of Query 1 - What is the title of the paper?



```
The first paragraph is a construction contract that outlines the terms and conditions for a project. It
specifies the date of the execution, the parties involved, and the responsibilities of each party. The
contract also includes details about the construction project, such as the scope of work, the payment
schedule, and the completion date.
CPU times: user 4.53 s, sys: 589 ms, total: 5.12 s
Wall time: 5 s
```

Figure 6: Output of Query 2 - Summarize the first paragraph

Code

```
%%time
```

```
#Using the above parameters for the prompting as well as the processing of the
image, tokenizing of the prompt, encoding and decoding within the above
function to interrogate the image.
result=ask_image(image,"What is the title of the paper?")
print(textwrap.fill(result,width=110))
```

```
%%time
#Using the above parameters for the prompting as well as the processing of the
image, tokenizing of the prompt, encoding and decoding within the above
function to interrogate the image.
result=ask_image(image,"Summarize the first paragraph.")
print(textwrap.fill(result,width=110))
```

References

1. [TechTarget - What is Image Recognition?](https://www.techtarget.com/searchenterpriseai/definition/image-recognition)
2. [Wikipedia - Contextual Image Classification](https://en.wikipedia.org/wiki/Contextual_image_classification)
3. [ScienceDirect - The Role of Context in Object Recognition](https://www.sciencedirect.com/science/article/abs/pii/S1364661307002550)
4. [AltexSoft - Image Recognition with Deep Learning and Neural Networks](https://www.altexsoft.com/blog/image-recognition-neural-networks-use-cases/)
5. [ScienceDirect - Context-based Image Explanations for Deep Neural Networks](https://www.sciencedirect.com/science/article/pii/S0262885621002158)
6. [SpringerOpen - Convolutional Neural Networks: An Overview and Application in Radiology](https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9)
7. [LinkedIn - Combining Large Language Models and Knowledge Graphs](https://www.linkedin.com/pulse/combining-large-language-models-knowledge-graphs-wisecube)
8. [Medium - Giving Large Language Models Context](https://medium.com/@simon_attard/giving-large-language-models-context-2d1956a6a017)
9. [ScienceDirect - Artificial intelligence and smart vision for building and construction](https://www.sciencedirect.com/science/article/pii/S0926580522003132)
10. [MIT News - Solving a machine-learning mystery](https://news.mit.edu/2023/large-language-models-in-context-learning-0207)
11. [arXiv - Contextual Object Detection with Multimodal](https://arxiv.org/abs/2305.18279)
12. [AIMultiple - Large Language Models: Complete Guide in 2024](https://research.aimultiple.com/large-language-models/)
13. [Medium - Giving Large Language Models Context](https://medium.com/@simon_attard/giving-large-language-models-context-2d1956a6a017)
14. [Medium - How do LLMs work with Vision AI?](https://officegarageitpro.medium.com/how-do-llms-work-with-vision-ai-65a2310e5a07)
15. [Microsoft Tech Community - How do LLMs work with Vision AI?](https://techcommunity.microsoft.com/t5/microsoft-mechanics-blog/how-do-llms-work-with-vision-ai-ocr-image-video-analysis/ba-p/3835661)
16. [YouTube - How can LLMs improve Vision AI?](https://www.youtube.com/watch?v=4shB_qdU3Gs)

17. [arXiv - Understanding and Improving In-Context Learning on LLMs](https://arxiv.org/abs/2311.18021)
18. [EDPS - Large Language Models](https://edps.europa.eu/data-protection/technology-monitoring/techsonar/large-language-models-llm)
19. https://encord.com/blog/llava-large-language-vision-assistant/
20. https://medium.com/voxel51/understanding-llava-large-language-and-vision-assistant-8b7772f5eec4)
21. https://www.linkedin.com/pulse/how-images-audio-can-used-trick-multi-model-large-language-bhatia
22. https://openreview.net/forum?id=5KojubHBr8