**Documentation**

**Project overview**

In this hackathon, the goal is to create a machine learning model that extracts entity values from images. This capability is crucial in fields like healthcare, e-commerce, and content moderation, where precise product information is vital. As digital marketplaces expand, many products lack detailed textual descriptions, making it essential to obtain key details directly from images. These images provide important information such as weight, volume, voltage, wattage, dimensions, and many more, which are critical for digital stores.

**Image Analysis and Text Extraction using MiniCPM-Llama3 Model**

**Overview**

This code performs image analysis and text extraction using the MiniCPM-Llama3 model for various entities such as width, height, weight, and more. It processes images to extract specific information, such as dimensions or weight, and converts the extracted units into standard formats.

**Getting Started**

**Prerequisites**

Before running the code, make sure you have the necessary libraries installed. You can install them using the following commands:

|  |
| --- |
| !pip install -q Pillow  !pip install -q torch  !pip install -q torchvision  !pip install -q transformers  !pip install -q sentencepiece  !pip install -q accelerate  !pip install -q bitsandbytes  !pip install -q tqdm |

**Required Libraries**

* Pandas: For data manipulation.
* NumPy: For numerical operations.
* torch: For handling models in PyTorch.
* Pillow: For image processing.
* transformers: To load pre-trained models and tokenizers.
* tqdm: For progress bars during processing.
* requests: For downloading images from URLs.

**Model Setup**

You need to download and load the pre-trained MiniCPM-Llama3 model and tokenizer:

|  |
| --- |
| from transformers import AutoModel, AutoTokenizer  model\_id = 'openbmb/MiniCPM-Llama3-V-2\_5-int4'  model = AutoModel.from\_pretrained(model\_id, trust\_remote\_code=True)  tokenizer = AutoTokenizer.from\_pretrained(model\_id, trust\_remote\_code=True)  model.eval() |

**Downloading the Image**

This function downloads an image from a given URL and saves it locally:

|  |
| --- |
| def download\_image(url, image\_path):  try:  response = requests.get(url)  if response.status\_code == 200:  with open(image\_path, 'wb') as file:  file.write(response.content)  print(f"Downloaded: {image\_path}")  else:  print(f"Failed to download: {url}")  except Exception as e:  print(f"Error downloading {url}: {e}") |

* url: URL of the image to download.
* image\_path: Local path to save the downloaded image.

**Text Preprocessing**

This function takes the model's response and extracts quantities and units using regex:

|  |
| --- |
| def text\_preprocess(res):  match = re.search(r'(\d+\.\d+|\d+)\s\*(\w+)', res)  if match:  quantity = match.group(1)  unit = match.group(2).lower()  prediction.append(f"{quantity} {unit}")  print(f"{quantity} {unit}")  else:  prediction.append("")  print("No match found") |

* It identifies numerical values and their units (e.g., "20 cm" or "45 grams").
* Extracts both the quantity and unit for further processing.

**Extracting Entities from Images**

This function sends the image and query to the model for entity extraction:

|  |
| --- |
| def llm\_text(entity\_name):  image = Image.open('/kaggle/working/img.jpg').convert('RGB')    question = f"Analyse the given image and give me the {entity\_name} from the image and give the answer in this form The {entity\_name} value from the image is"  msgs = [{'role': 'user', 'content': question}]    res = model.chat(  image=image,  msgs=msgs,  tokenizer=tokenizer,  sampling=True,  temperature=0.7,  system\_prompt='You are an AI that extracts and transcribes text from images. Please read the text in the image and transcribe it accurately.'  )    return res |

* **Inputs**:
  + entity\_name: The name of the entity you want to extract (e.g., width, height).
  + The function processes the image and asks the model to extract specific information.
* **Outputs**:
  + A response containing the extracted value from the image.

**Unit Conversion**

The code handles unit conversions to standard formats. Here’s how the units are mapped:

|  |
| --- |
| entity\_unit\_map = {  'width': {'centimetre', 'foot', 'inch', 'metre', 'millimetre', 'yard'},  'depth': {'centimetre', 'foot', 'inch', 'metre', 'millimetre', 'yard'},  'height': {'centimetre', 'foot', 'inch', 'metre', 'millimetre', 'yard'},  'item\_weight': {'gram', 'kilogram', 'microgram', 'milligram', 'ounce', 'pound', 'ton'},  'maximum\_weight\_recommendation': {'gram', 'kilogram', 'microgram', 'milligram', 'ounce', 'pound', 'ton'},  'voltage': {'kilovolt', 'millivolt', 'volt'},  'wattage': {'kilowatt', 'watt'},  'item\_volume': {'centilitre', 'cubic foot', 'cubic inch', 'cup', 'decilitre', 'fluid ounce', 'gallon', 'litre', 'microlitre', 'millilitre', 'pint', 'quart'}  } |

**Conversion Logic**

The convert\_units function converts short units (e.g., cm) to their long form (centimetre):

|  |
| --- |
| def convert\_units(input\_list, entity\_unit\_map):  converted\_list = []  unit\_conversion = {  'cm': 'centimetre',  'g': 'gram',  'kg': 'kilogram',  'v': 'volt',  # Add other unit conversions here...  }  for item in input\_list:  if not item.strip():  converted\_list.append(item)  continue    value, unit = item.split()    if unit in unit\_conversion:  long\_unit = unit\_conversion[unit]  converted\_list.append(f"{value} {long\_unit}")  else:  converted\_list.append(item)  return converted\_list |

* **Inputs**: A list of extracted values with units (e.g., ['20 cm', '45 grams', '7 v']).
* **Outputs**: The values with converted units in standard form.

**Conclusion**

This script allows users to extract and process specific information from images using a pre-trained model (MiniCPM-Llama3). It handles downloading the image, extracting text from the image, preprocessing the extracted text, and converting units into a standardized format. This setup is ideal for tasks that involve analyzing product images or dimensions and extracting relevant information automatically.