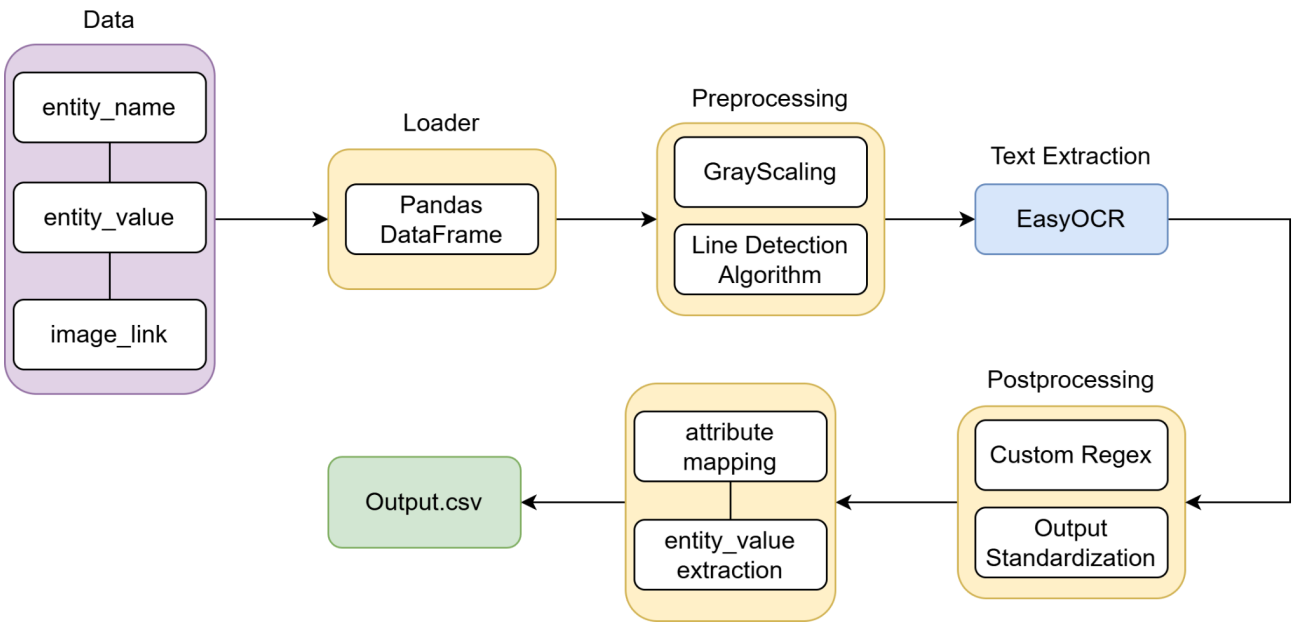
Amazon ML Challenge 2024

Team Aloo Parathaaa

1. Problem Overview

The task is to extract entity values (e.g., weight, dimensions) from product images using machine learning, targeting applications in fields such as e-commerce and healthcare. The extracted entities must match specific units and follow a strict format. The evaluation is based on the F1 score, considering precision and recall.

2. Workflow Overview



3. Detailed Approach

3.1 Preprocessing

Before performing OCR, the image is preprocessed to enhance the quality of the text and ensure better OCR results.

3.1.1 Grayscale Conversion

The input image is first converted to grayscale using OpenCV's `cvtColor()` function. Grayscale images simplify the complexity of image processing by reducing the color channels and focusing purely on text and edge detection.

This step is crucial as OCR libraries perform better on high-contrast, noise-reduced grayscale images.

3.1.2 Line Detection Algorithm

For images with entity_name: height, width, and depth OCR is used to extract all text and corresponding bounding boxes in the image. Bounding boxes containing numeric values and units are filtered out.
OpenCV (**cv2.Canny** and **cv2.HoughLinesP**) is used to extract all lines in the image.
Algorithm Logic:

- For each of the filtered bounding boxes, the line closest to their center is obtained.
- The angle of the line is obtained using standard mathematical equations:
    - If the angle lies between 75° to 105°, the corresponding bounding box's numeric value along with its corresponding unit is added to a list (of tuples) for potential candidates for height or depth.
    - If the angle lies between 0° to 60°, we have a potential candidate for width.
- Among all candidates for each category (height, width, depth), the candidate with the maximum numeric value of dimension is found and returned as the answer along with the relevant extracted unit.

Using this process, each image, end-to-end, now takes ~0.2 seconds allowing us to experiment and prepare the test.csv file within a reasonable time frame.

3.2 OCR for text extraction

We conducted experiments with various OCR (Optical Character Recognition) technologies:

- Tesseract
- EasyOCR
- PaddleOCR

After testing on a small sample of files, we found that EasyOCR performed the best. We then proceeded to run OCR on all files in the test set.

3.3 Postprocessing

3.3.1 Text parsing using Custom Regex

OCR output is a collection of characters and numbers. Regular expressions (Regex) is used to filter out relevant numeric values along with their units.

We develop a robust Regex, accounting for decimal points and commas as decimal markers (European). Different Units and multiple of the same units in a sentence. Other functionality mentioned below:

1. *Number Formatting*: The `*format_number*` function removes trailing zeros and unnecessary decimal points when formatting numbers to two decimal places.
2. *Regex for Extraction*:
    a. *Weight*: The `*extract_weight*` function extracts the largest value associated with units like milligrams, grams, kilograms, ounces, pounds, and tons.
    b. *Wattage*: The `*extract_wattage*` function retrieves the largest number related to wattage, either in watts or kilowatts.
    c. *Voltage*: The `*extract_voltage*` function extracts values in millivolts, volts, or kilovolts.
    d. *Volume*: The `*extract_volume*` function handles units like milliliters, liters, gallons, fluid ounces, and cubic feet.
3. *OCR Integration*: The `*compute_dimension*` function performs OCR on an image to extract text and then calls the specific extraction functions to obtain key values, such as weight or wattage.
4. *Error Handling*: It includes basic error handling to ensure the process doesn't break if the image is not found or OCR fails to extract text.
5. *Center Marking*: The image is processed to mark its center using OpenCV before extracting the text using OCR.
6. *Timing and Debugging*: The script calculates the total time taken for the OCR process and displays it depending on the `*time_mode*` flag.

3.3.2 Output Standardization

The extracted values are often in different units (centimeters, millimeters, inches, etc.). We standardize the output by converting them to a common unit, based on the context:- Unit Conversion: Depending on the detected unit (e.g., cm, mm, inches), we convert the values to a standardized format. For example, 1 cm = 10 mm, 1 inch = 2.54 cm.

```python
    if unit == "cm":
        return value, "centimetre" ```
```

4. Output & Results

The processed data is stored in a CSV file where each row contains:- The image index.- The predicted height value along with the unit. The final prediction file is saved using pandas' `to_csv()` method: ```python output_df.to_csv(output_file, index=False) ``` This approach ensures that the height information is extracted in a structured manner for further analysis.

5. Challenges and Improvements

Our team faced difficulties in handling large datasets i.e., 1.3 lakh test samples and 3.1 lakh train samples which was around 53 GBs in total. Downloading and then preprocessing took us 11+ hours of computation using EasyOCR. We experimented with PaddleOCR parallely as well which turned out to be 27+ hours of computation so we had to discard it.

We faced significant challenges with OCR tools, balancing efficiency and accuracy under strict time constraints. **EasyOCR**, though fast and time-saving, resulted in approximately 55,000 blank rows (~40% of the dataset) where it failed to extract meaningful text. In contrast, **PaddleOCR** delivered superior accuracy and handled challenging cases better, but its processing time was significantly longer, making it less practical for large-scale datasets in the given time frame. Due to the urgency of the task, we opted for EasyOCR despite its limitations, prioritizing speed over accuracy to meet the deadlines.

Open-source pre-trained models like LLaVA, Qwen2-VL-for-OCR-VQA, and Eagle-V5-13B could be leveraged for faster and more effective problem-solving.

We started a bit late and a significant part of the time went into experimenting on processing data and training models. This forced us to make trade-offs with fine-tuning. Large Vision-Language Models (VLMs) like Florence-2 and PaliGemma Vision could be used for fine-tuning to extract entity values. Using techniques like LoRA (Low-Rank Adaptation) and PEFT (Parameter-Efficient Fine-Tuning) by fine-tuning only a subset of the model’s parameters could achieve significant performance gain and reduce compute demands. Experimenting with hyperparameters like learning rate, batch size, and epochs to optimize model performance.

6. Conclusion

To conclude, by utilizing EasyOCR for text extraction and implementing Custom Regex and Line Detection Algorithms, we efficiently addressed resource limitations while extracting key dimensions and values. This streamlined approach, with minimal GPU dependency, reduced processing time to ~0.2 seconds per image, enabling large-scale data generation in a practical and resource-efficient manner without relying on heavy vision models.

The four-day hackathon, the longest we've participated in so far, posed significant challenges, particularly in processing the test.csv file, which required downloading over 52 GB of images and running them through an OCR model, a task that collectively took more than 20 hours. Despite these obstacles, this was an invaluable experience to work on a real-world problem faced by Amazon.

We are immensely grateful to Amazon for hosting such an engaging hackathon and eagerly look forward to competing again next year!