Week 6 Report

Thomas Deatherage
NFHM/BioCosmos
27 September 2024

Time Slot

1) What progress did you make in the last week?

- Met with usual GaTech NFHM collaborators.
- Introductory meeting with Benjamin Yu.
- Successfully benchmarked florence-2 trait referring against the fish-vista dataset

2) What are you planning on working on next?

- Improvements to trait referring and trait grounding scripts to improve accuracy
- Explore LLM layer

3) Is anything blocking you from getting work done?

- Lack of power and internet after the hurricane.

Abstracts & Summaries

1) A Survey of Large Language Models

Conference/Journal: Not stated

Abstract: Language is essentially a complex, intricate system of human expressions governed by grammatical rules. It poses a significant challenge to develop capable AI algorithms for comprehending and grasping a language. As a major approach, language modeling has been widely studied for language understanding and generation in the past two decades, evolving from statistical language models to neural language models. Recently, pre-trained language models (PLMs) have been proposed by pre-training Transformer models over large-scale corpora, showing strong capabilities in solving various NLP tasks. Since researchers have found that model scaling can lead to performance improvement, they further study the scaling effect by increasing the model size to an even larger size. Interestingly, when the parameter scale exceeds a certain level, these enlarged language models not only achieve a significant performance improvement but also show some special abilities that are not present in small-scale language models. To discriminate the difference in parameter scale, the research community has coined the term large language models (LLM) for the PLMs of significant size. Recently, the research on LLMs has been largely advanced by both academia and industry, and a remarkable progress is the launch of ChatGPT, which has attracted widespread attention from society. The technical

evolution of LLMs has been making an important impact on the entire AI community, which would revolutionize the way how we develop and use AI algorithms. In this survey, we review the recent advances of LLMs by introducing the background, key findings, and mainstream techniques. In particular, we focus on four major aspects of LLMs, namely pre-training, adaptation tuning, utilization, and capacity evaluation. Besides, we also summarize the available resources for developing LLMs and discuss the remaining issues for future directions.
Summary: This paper presents an overview of the history of LLM technologies, their evolution, unique qualities and variations.
Link: https://arxiv.org/abs/2303.18223
Code: N/A
Relevance: Part of our long term vision is to have an LLM interface capable of delegating queries to appropriate, well-defined tasks. This paper is part of my research into their history and capabilities.

Scripts and Code Blocks

Code contributions can be found at this PR: https://github.com/BioCosmos-AI/BioCosmos/pull/4

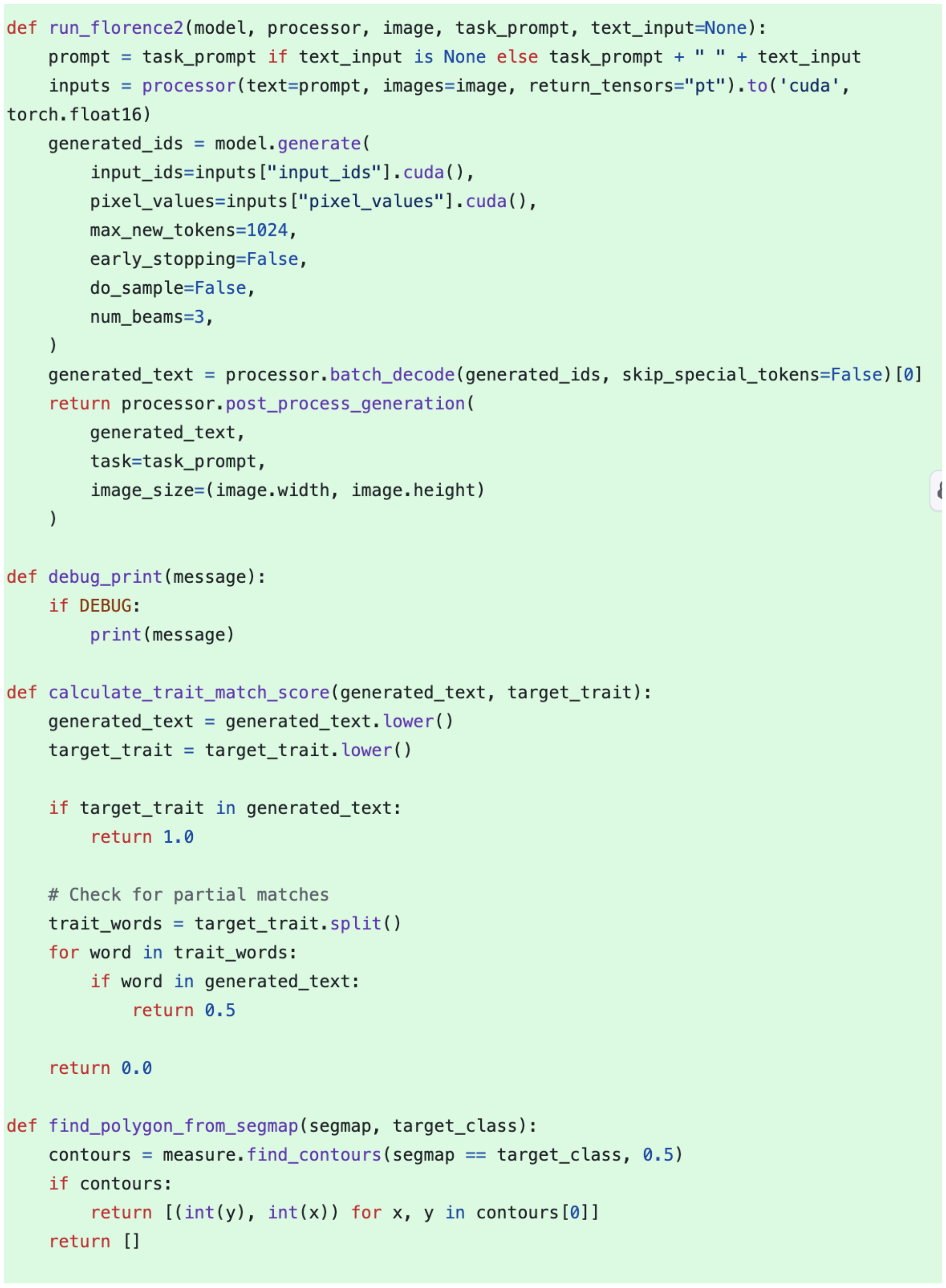
This week I continued my exploration of the florence-2 vision-language model. Specifically, I created a script that would do trait referring performance evaluation on the fish-vista benchmark dataset, which can be found at hugging face https://huggingface.co/datasets/imageomics/fish-vista. Trait referring is the complement of trait grounding, which I looked at in last week's report. Specifically, when provided a bounding box of some trait, I evaluate how well florence-2 recognizes the trait. I give complete credit when the model gets the full name of the trait within its category description (e.g., "This is a fish dorsal fin" for the trait "dorsal fin") and 50% credit when it gets at least one word (e.g., "this is a fish fin" for "pectoral fin").

Codewise, I've added a referring.py script that contains the primary code for doing performance evaluations. Additionally, like last week, there's a Jupyter .ipynb notebook for calling the script and setting up the environment. Here are the code highlights:

This block of code is executed *n* times for the desired number of samples to benchmark with. We use the <REGION_TO_DESCRIPTION> prompt that comes with florence-2 to generate a caption of a bounded portion of the image, which is pulled from the provided fish-vista segmentation masks. Note the \`polygon_to_bbox\` function. The provided segmentation masks are arbitrary shapes. It appears that florence-2's <REGION_TO_DESCRIPTION> functionality expects a rectangular shape of coordinates. This function creates the tightest rectangle that still contains the provided segmentation shape.

```python
                # Load image
                image = Image.open(image_path)

                # Load segmentation mask
                seg_mask = np.array(Image.open(seg_mask_path))
            except (IOError, SyntaxError) as e:
                print(f"Error loading image or mask for {img_filename}: {e}")
                skipped_rows += 1
                continue

            # Find present traits
            present_traits = [id_trait_map[str(trait_id)] for trait_id in
np.unique(seg_mask) if str(trait_id) in id_trait_map]

            if args.trait_option not in present_traits:
                skipped_rows += 1
                continue

            # Get polygon for the target trait
            target_class = next(int(k) for k, v in id_trait_map.items() if v ==
args.trait_option)
            polygon = find_polygon_from_segmap(seg_mask, target_class)

            if not polygon:
                skipped_rows += 1
                continue

            # Convert polygon to bounding box
            bbox = polygon_to_bbox(polygon)

            # Convert bbox to Florence-2 format
            florence2_bbox = bbox_to_florence2_format(bbox, image.width, image.height)

            task_prompt = '<REGION_TO_DESCRIPTION>'
            florence_output = run_florence2(model, processor, image, task_prompt,
text_input=florence2_bbox)

            result = {
                "image-path": image_path,
                "target-trait": args.trait_option,
                "florence-output": florence_output,
            }

            generated_text = florence_output['<REGION_TO_DESCRIPTION>']
            match_score = calculate_trait_match_score(generated_text, args.trait_option)

            result["match_score"] = match_score
            total_score += match_score
            total_count += 1
```

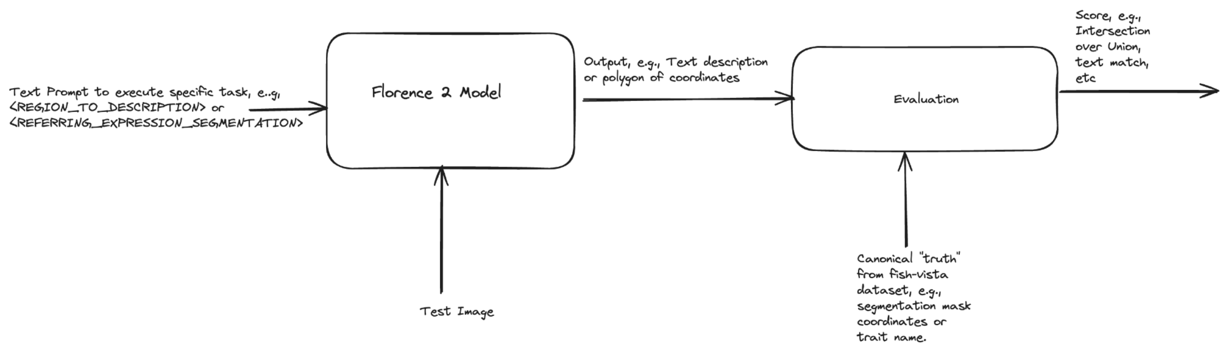
The other important functions are pretty self-explanatory: We prompt the model, we extract the polygon coordinates from the segmentation mask (i.e., the coordinates that we convert to a rectangle), and we calculate the trait match (this could definitely be improved; more on that in the results section).



```python
def run_florence2(model, processor, image, task_prompt, text_input=None):
    prompt = task_prompt if text_input is None else task_prompt + " " + text_input
    inputs = processor(text=prompt, images=image, return_tensors="pt").to('cuda',
torch.float16)
    generated_ids = model.generate(
        input_ids=inputs["input_ids"].cuda(),
        pixel_values=inputs["pixel_values"].cuda(),
        max_new_tokens=1024,
        early_stopping=False,
        do_sample=False,
        num_beams=3,
    )
    generated_text = processor.batch_decode(generated_ids, skip_special_tokens=False)[0]
    return processor.post_process_generation(
        generated_text,
        task=task_prompt,
        image_size=(image.width, image.height)
    )

def debug_print(message):
    if DEBUG:
        print(message)

def calculate_trait_match_score(generated_text, target_trait):
    generated_text = generated_text.lower()
    target_trait = target_trait.lower()

    if target_trait in generated_text:
        return 1.0

    # Check for partial matches
    trait_words = target_trait.split()
    for word in trait_words:
        if word in generated_text:
            return 0.5

    return 0.0

def find_polygon_from_segmap(segmap, target_class):
    contours = measure.find_contours(segmap == target_class, 0.5)
    if contours:
        return [(int(y), int(x)) for x, y in contours[0]]
    return []
```

Flow Charts/Diagrams

Same general chart of florence-2 performance evaluation architecture from last week.


Text Prompt to execute specific task, e.g,
<REGION_TO_DESCRIPTION> or
<REFERRING_EXPRESSION_SEGMENTATION>
Florence 2 Model
Test Image
Output, e.g., Text description or polygon of coordinates
Evaluation
Canonical "truth" from fish-vista dataset, e.g., segmentation mask coordinates or trait name.
Score, e.g., Intersection over Union, text match, etc

Documentation

To run the script:

```
$ python referring.py  —trait_option=Eye
```

Or from another python program (e.g., Jupyter notebook):

```
import referring
args = Args(trait_option='Eye', num_queries=100)
referring.main(args)
```

Available trait_options are: Eye, Head, Dorsal fin, Pectoral fin, Pelvic fin, Anal fin, Caudal fin, Adipose fin, and Barbel fin.

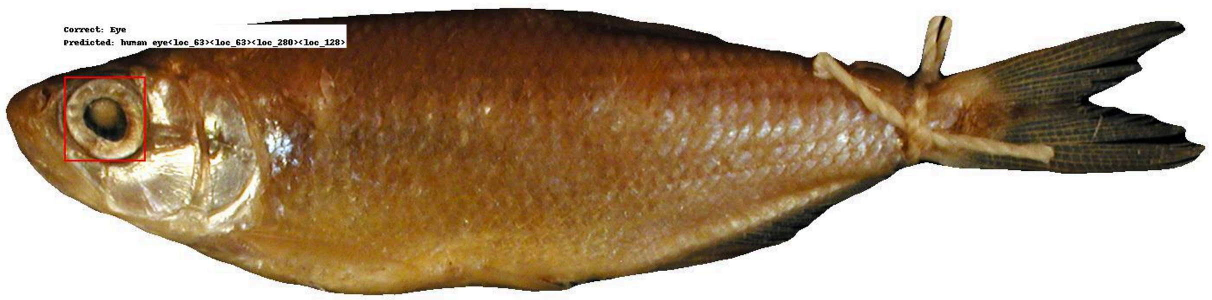
Results Visualization + Proof of Work

Against 50 test rows, I got the following metrics for each trait:

Average match score for Eye: 0.18
Average match score for Head: 0.04
Average match score for Dorsal fin: 0.0
Average match score for Pectoral fin: 0.0
Average match score for Pelvic fin: 0.0
Average match score for Anal fin: 0.0
Average match score for Caudal fin: 0.0
Average match score for Adipose fin: 0.0
Average match score for Barbel: 0.0

So, as the results indicate, the best-recognized trait was "Eye". The other traits really fell off a cliff. This is largely consistent with the trait grounding benchmark, which likewise showed eye being the most recognized trait.. No worries at any rate. Anatomical names like "Caudal fin" are pretty esoteric for a general VLM. Fine tuning should improve that.

Here's an example image I generated demonstrating the predicted (florence-2) vs the target (annotated groundings from the fish-vista dataset):



```
Correct: Eye
Predicted: human eye<loc_63><loc_63><loc_280><loc_128>
```

In this case, I gave 100% credit, since the correct output "eye" was contained within the predicted description "human eye". Of course, in actuality this isn't very good. It's a fish, not a human eye. My plan is to improve upon how "correctness" is measured. But for a first pass at trait referring performance evaluation, it helped me get an idea of florence-2's capabilities.

Next Week's Proposal

- Merge this week's work
- Improvements to both the trait referring and trait grounding evaluation scripts.
- Cursory exploration of LLMs for BioCosmos.