Week 5 Report

Thomas Deatherage
NFHM/BioCosmos
20 September 2024 (Late)

Time Slot

1) What progress did you make in the last week?
    - Met with GaTech + UoF collaborators.
    - Successfully benchmarked florence-2 trait grounding against the fish-vista dataset
2) What are you planning on working on next?
    - Trait referral perf evaluation.
3) Is anything blocking you from getting work done?
    - Nope

Abstracts & Summaries

1) Attention Is All You Need

Conference/Journal: Not stated
Abstract: The dominant sequence transduction models are based on complex recurrent or convolutional neural networks in an encoder-decoder configuration. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.
Summary: The paper introduces a new model called the Transformer, which uses only attention mechanisms instead of complex neural networks with recurrence or convolutions. This model performs better and trains faster on machine translation tasks compared to existing methods,

achieving state-of-the-art results.
Link: https://arxiv.org/abs/1706.03762
Code: N/A
Relevance: Transformers were critical in the development of modern LLMs. I'm curious how they work, as we'll be incorporating a LLM interface into BioCosmos.

Scripts and Code Blocks

Code contributions can be found at this PR: https://github.com/BioCosmos-AI/BioCosmos/pull/3

This week was spent on exploration of the florence-2 vision-language model. Specifically, I created a script that does trait grounding performance evaluation on the fish-vista benchmark dataset, which can be found at hugging face https://huggingface.co/datasets/imageomics/fish-vista. Overall, there's a grounding.py file, which contains the primary code for doing performance evaluations, and a Jupyter .ipynb notebook for calling the script and setting up the environment.

Trigger the benchmark against a particular trait (in this case "Eye"):

Fish Eye Benchmark

```
:
args = Args(trait_option='Eye', num_queries=100)

grounding.main(args)
```

Pull in segmentation mask and test images from fish-vista dataset and generate florence-2's grounding prediction:

```
            try:
                # Load image
                image = Image.open(image_path)

                # Load segmentation mask
                seg_mask = np.array(Image.open(seg_mask_path))
                debug_print(f"Segmentation mask shape: {seg_mask.shape}")
            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)
            target_polygon = find_polygon_from_segmap(seg_mask,
target_class)

            question = f"Identify and segment the {args.trait_option} in
the image."

            florence_output = run_florence2(model, processor, image,
'<REFERRING_EXPRESSION_SEGMENTATION>', text_input=question)

            result = {
                "question": question,
                "target-output": target_polygon,
                "image-path": image_path,
                "florence-output": florence_output,
            }

            predicted_polygon = process_florence_output(florence_output)

            # Add debug information
```

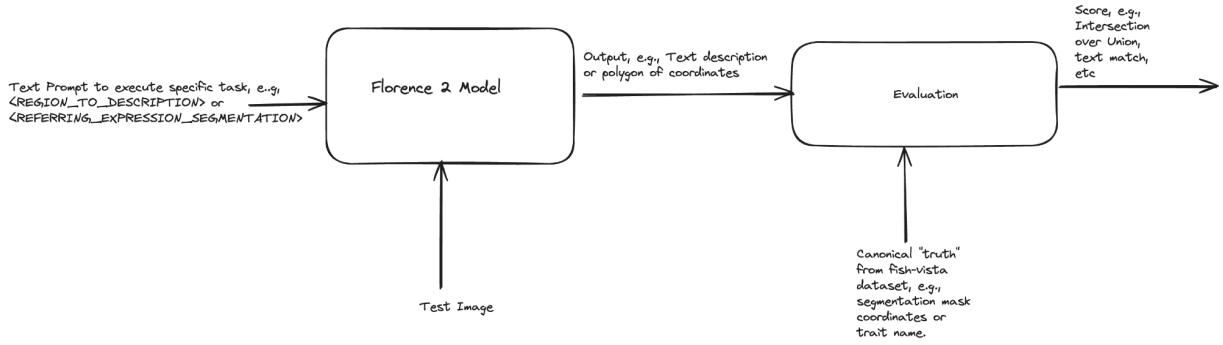
Calculate the “match” between the target and the predicted by dividing the intersection between the two shapes and dividing the by their union:

```
+             iou = calculate_iou_polygons(reshaped_target,
  reshaped_predicted)
+             result["iou"] = iou
+             total_iou += iou
+             total_count += 1
+
+             json.dump(result, writer)
+             writer.write('\n')
+
+             processed_rows += 1
+
+             # If visual comparison is enabled, create and save the
  comparison image
+             if args.visual_compare:
+                 output_image_path = os.path.join(images_dir,
  f"comparison_{idx}.png")
+                 if reshaped_target and reshaped_predicted:  # Check if
  both polygons are non-empty
+                     draw_comparison(image_path, reshaped_target,
  reshaped_predicted, output_image_path)
+                 else:
+                     debug_print(f"Skipping visual comparison for image
  {idx} due to empty polygon(s)")
+
+             processed_rows += 1
+
```

And the actual function to calculate the intersection/union of the target and predicted:

```
+
+ def calculate_iou_polygons(poly1, poly2):
+     poly1 = reshape_polygon(poly1)
+     poly2 = reshape_polygon(poly2)
+
+     if len(poly1) < 3 or len(poly2) < 3:
+         print(f"Warning: Invalid polygon. Poly1 length: {len(poly1)},
  Poly2 length: {len(poly2)}")
+         return 0.0
+
+     try:
+         polygon1 = Polygon(poly1)
+         polygon2 = Polygon(poly2)
+     except ValueError as e:
+         print(f"Error creating polygon: {e}")
+         print(f"Poly1: {poly1[:5]}...")
+         print(f"Poly2: {poly2[:5]}...")
+         return 0.0
+
+     if not polygon1.is_valid or not polygon2.is_valid:
+         print(f"Warning: Invalid polygon. Poly1 valid:
  {polygon1.is_valid}, Poly2 valid: {polygon2.is_valid}")
+         return 0.0
+
+     intersection = polygon1.intersection(polygon2).area
+     union = polygon1.union(polygon2).area
+
+     if union == 0:
+         return 0.0
+
+     return intersection / union
```

Flow Charts/Diagrams



Documentation

To run the script:

```
$ python grounding.py  –trait_option=Eye
```

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

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

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

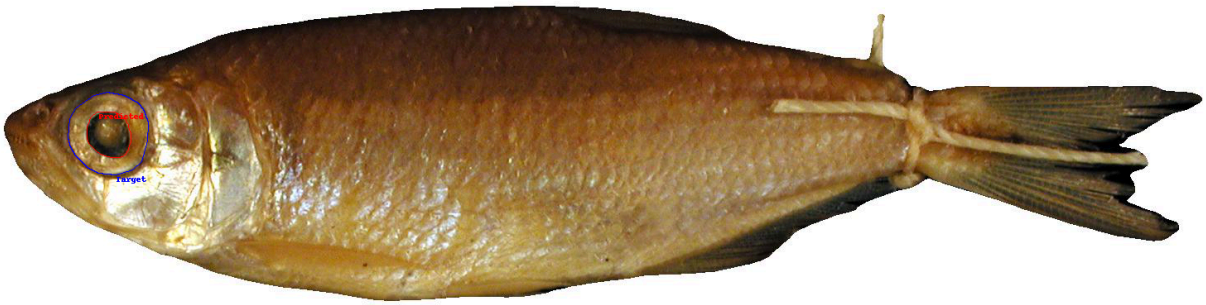
Results Visualization + Proof of Work

Against 100 test rows, I got the following intersection/union (IoU) metrics.

Average IoU for Eye: 0.5148637984469006
Average IoU for Head: 0.08609843586352085
Average IoU for Dorsal fin: 0.02202529608433633
Average IoU for Pectoral fin: 0.07570385185511015
Average IoU for Pelvic fin: 0.06050116803835856
Average IoU for Anal fin: 0.0003406790886847885
Average IoU for Caudal fin: 0.16787344625161896
Average IoU for Adipose fin: 0.0056045198627131245
Average IoU for Barbel: 0.00018929093307924375

So, as the results indicate, the best-recognized trait was “Eye”. However, I believe there are a few minor tweaks with prompting that might improve performance, even without fine-tuning. Additionally, I discovered a bug in my code where a small subset of images were generating polygons with a data structure that my code didn’t recognize. I want to dig into that and fix it. I have a feeling this will improve performance, too.

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



The blue outline is from the test dataset. The red outline is what was predicted by florence-2. As a quick visual inspection can confirm, some of the predicted results are very accurate like the example fish eye above.

Next Week’s Proposal

- Merge this week’s work

- Baseline benchmarks on Florence-2 for trait referral.