

Week 1 Report

Thomas Deatherage
23 August 2024

Abstracts & Summaries

1) Vector search with small radiuses

Conference/Journal: Not stated

Abstract: We introduce Florence-2 a novel vision foundation model with a unified prompt-based representation for various computer vision and vision-language tasks. While existing large vision models excel in transfer learning they struggle to perform diverse tasks with simple instructions a capability that implies handling the complexity of various spatial hierarchy and semantic granularity. Florence-2 was designed to take text-prompt as task instructions and generate desirable results in text forms whether it be captioning object detection grounding or segmentation. This multi-task learning setup demands large-scale high-quality annotated data. To this end we co-developed FLD-5B that consists of 5.4 billion comprehensive visual annotations on 126 million images using an iterative strategy of automated image annotation and model refinement. We adopted a sequence-to-sequence structure to train Florence-2 to perform versatile and comprehensive vision tasks. Extensive evaluations on numerous tasks demonstrated Florence-2 to be a strong vision foundation model contender with unprecedented zero-shot and fine-tuning capabilities.

Summary: Florence-2 is a new advanced model for computer vision tasks that can understand text instructions and generate accurate results in various forms like captions, object detection, and segmentation. It has shown strong performance in different tasks without needing prior training.

Link:

https://openaccess.thecvf.com/content/CVPR2024/html/Xiao_Florence-2_Advancing_a_Unified_Representation_for_a_Variety_of_Vision_CVPR_2024_paper.html

Scripts and Code Blocks

We're pulling in a lot more collaborators this semester. And one of the first things we're probably (there's a meeting next week to discuss this) going to do is deploy an MVP version of the application to an internet-accessible host. In keeping with that general goal, this week I tried to prepare the codebase for deployment. I don't know the specifics of what our hosting provider (likely computing resources from UoF) looks like, nor what an CI/CD pipeline might look like. Consequently, I focused on broadly-applicable improvements to the code base that will 1) improved code quality and 2) make it easier to on board the newest collaborators.

Specifically, I:

- 1) Added and fixed a bunch of type hints in the ingestor scripts, specifically the vector_embedder, idigbio ingestor and postgres output.
- 2) We already have retry logic on the queue jobs themselves. I also added retry logic to external network calls to improve robustness and performance.
- 3) Improved error handling and visibility.
- 4) Calmed down the previously very noisy vector embedder logs.
- 5) Used black-formatter to format a few files.

These code changes can be found here:

<https://github.com/Human-Augment-Analytics/NFHM/pull/28/files>

There are too many code changes to repeat here, so I'll focus on the highlights. Here's one such example detailing most of everything from above (type hinting, less noisy logs, consistent formatting with

black-formatter, etc) in the vector embedder code:

<pre> 15 import open_clip 16 - from datetime import datetime 17 - from bson.objectid import ObjectId 18 19 - logger = getLogger('vector_embedder') 20 21 model = None 22 preprocess = None 23 tokenizer = None 24 - device = None 25 26 - if torch.backends.mps.is_available(): 27 - device = torch.device("mps:0") 28 - logger.info("Using MPS") 29 - else: 30 - device = torch.device("cpu") 31 32 async def load_model(): 33 - """ 34 - Loads the pre-trained model and tokenizer for vector embeddings. 35 - This is lazily evaluated once when the first call to vector_embedder is made. 36 - Loading the model is a slow operation that breaks ingestor.py when done at the top level 37 - upon import 38 - """ 39 - global model, preprocess, tokenizer, device 40 - # if model is not None: 41 - model, _, preprocess = open_clip.create_model_and_transforms('ViT-B-32', 42 - pretrained='laion2b_s34b_b79k', device=device) 43 - model.to(device) 44 - # model.eval() # model in train mode by default, impacts some models with BatchNorm or 45 - stochastic depth active 46 - tokenizer = open_clip.get_tokenizer('ViT-B-32') 47 - logger.info("Model loaded") 48 - # else: 49 - # logger.info("Model loaded already") 50 51 - def input_data_mapper(media: dict, line: dict) -> dict: 52 - tmp_d = { 53 - "specimen_uid": line.get("uid"), 54 - "scientific_name": line.get("data", {}).get("dwc:scientificName") or line.get("data", 55 - {}).get("dwc:genus"), 56 - "external_media_url": media.get("data", {}).get("ac:accessURI") or 57 - media.get("indexTerms", {}).get("accessuri"), 58 59 - "media_uid": media.get("uid"), 60 - "catalog_number": line.get("data", {}).get("dwc:catalogNumber"), 61 - "recorded_by": line.get("data", {}).get("dwc:recordedBy"), 62 63 - @ -61,30 +62,47 @@ def input_data_mapper(media: dict, line: dict): 64 - "tax_genus": line.get("data", {}).get("dwc:genus"), 65 - "common_name": line.get("indexTerms", {}).get("commonname"), 66 - "higher_taxon": line.get("indexTerms", {}).get("highertaxon"), 67 - "earliest_epoch_or_lowest_series": line.get("indexTerms", 68 - {}).get("earliestepochorlowestseries") or line.get("data", 69 - {}).get("dwc:earliestEpochOrLowestSeries"), 70 - "earliest_age_or_lowest_stage": line.get("indexTerms", 71 - {}).get("earliestageorloweststage") or line.get("data", 72 - {}).get("dwc:earliestAgeOrLowestStage"), 73 - "collection_date": line.get("indexTerms", {}).get("datecollected") 74 75 } 76 77 - if line.get("indexTerms", {}).get("geopoint", {}).get("lon") is not None: 78 - lat = str(line.get("indexTerms", {}).get("geopoint", {}).get("lat")) 79 - lon = str(line.get("indexTerms", {}).get("geopoint", {}).get("lon")) 80 - tmp_d["location"] = f"POINT({lon} {lat})" 81 82 83 - if tmp_d["collection_date"] is not None: 84 - tmp_d["collection_date"] = datetime.strptime('2012-01-12', '%Y-%m-%d'); </pre>	<pre> 8 import open_clip 9 + from dateutil import parser as date_parser 10 + from PIL import Image 11 + from io import BytesIO 12 + import logging 13 + from typing import Any, List, Dict, Optional 14 + from schema.processed_search_record import ProcessedSearchRecord 15 16 + logger = logging.getLogger("vector_embedder") 17 + logging.getLogger("PIL.TiffImagePlugin").setLevel(logging.INFO) 18 + logging.getLogger("pymongo.command").setLevel(logging.INFO) 19 + logging.getLogger("pymongo.serverSelection").setLevel(logging.INFO) 20 21 model = None 22 preprocess = None 23 tokenizer = None 24 + device = (25 + torch.device("mps:0") if torch.backends.mps.is_available() else torch.device("cpu") 26 +) 27 + 28 + logger.info(f"Using device: {device}") 29 30 31 async def load_model(): 32 + global model, preprocess, tokenizer 33 + if model is None: 34 + model, _, preprocess = open_clip.create_model_and_transforms(35 + "ViT-B-32", pretrained="laion2b_s34b_b79k", device=device 36 +) 37 + model.to(device) 38 + model.eval() 39 + tokenizer = open_clip.get_tokenizer("ViT-B-32") 40 + 41 + logger.info("Model loaded") 42 + else: 43 + logger.info("Model already loaded") 44 + 45 + def input_data_mapper(46 + media: Dict[str, Any], line: Dict[str, Any] 47 +) -> ProcessedSearchRecord: 48 + record: ProcessedSearchRecord = { 49 + "specimen_uid": line.get("uid", ""), 50 + "scientific_name": line.get("data", {}).get("dwc:scientificName") 51 + or line.get("data", {}).get("dwc:genus"), 52 + 53 + "external_media_url": media.get("data", {}).get("ac:accessURI") 54 + or media.get("indexTerms", {}).get("accessuri"), 55 + "media_uid": media.get("uid"), 56 + "catalog_number": line.get("data", {}).get("dwc:catalogNumber"), 57 + "recorded_by": line.get("data", {}).get("dwc:recordedBy"), 58 59 + "tax_genus": line.get("data", {}).get("dwc:genus"), 60 + "common_name": line.get("indexTerms", {}).get("commonname"), 61 + "higher_taxon": line.get("indexTerms", {}).get("highertaxon"), 62 + "earliest_epoch_or_lowest_series": line.get("indexTerms", {}).get(63 + 64 + "earliestepochorlowestseries" 65 + 66 +) 67 + or line.get("data", {}).get("dwc:earliestEpochOrLowestSeries"), 68 + "earliest_age_or_lowest_stage": line.get("indexTerms", {}).get(69 + 70 + "earliestageorloweststage" 71 + 72 +) 73 + or line.get("data", {}).get("dwc:earliestAgeOrLowestStage"), 74 + "collection_date": None, 75 + "location": None, 76 + "embedding": None, 77 + "tensor_embedding": None, 78 79 + "geopoint": line.get("indexTerms", {}).get("geopoint", {}) 80 + if geopoint.get("lon") is not None and geopoint.get("lat") is not None: 81 + record["location"] = f"POINT({geopoint['lon']} {geopoint['lat']})" 82 83 + "collection_date": line.get("indexTerms", {}).get("datecollected") 84 + if collection_date is not None: 85 + try: 86 + parsed_date = date_parser.parse(collection_date) 87 + record["collection_date"] = parsed_date.date() 88 + except ValueError: </pre>
---	--

Here's another code sample from the idigbio ingestor code:

```

137 Ingestor/inputs/Idigbio.py
16 - search_dict (dict): The rq query search params for the iDigBio API. A stringified dictionary. If
    - limiting to only records with media or images,
17 - include 'hasMedia': True and/or 'hasImages': True, respectively.

18
19 - import_all (bool): If true, function will fetch all pages of results until the end of the search results.
    - This is done by sequentially adding new jobs to idigbio redis queue.
20 - Results size could be millions of records, and hundreds of gigabytes of data. If false, will only
    - fetch the first page of results. Default is false. Default page size is 100 records.
21
22 - Example Usage:
23 - idigbio_search(json.dumps({ 'search_dict': {'genus': 'Puma', 'hasMedia': True}, 'import_all': True,
    - offset: 100 }), {'queue': queue })

24
25 Returns:
26 - List: Returns results of query as a List of dictionaries, with each dictionary populated with media
    - fetched from the iDigBio media API.
27
28
29 async with httpx.AsyncClient() as client:
30 - params = json.loads(args)
31 - search_dict = params.get('search_dict', {})
32 - import_all = params.get('import_all', False)
33 - pagesize = 100
34 - query_params = {
35 -     'rq': json.dumps(search_dict) # See https://github.com/iDigBio/iDigbio-search-api/wiki/Query-
    - Format#idigbio-query-format
36 - }
37 - if (import_all):
38 -     query_params['limit'] = pagesize
39 -     query_params['offset'] = params.get('offset', 0)
40 -
41 - r = await client.get(url, params=query_params)
42 - if r.status_code == 200:
43 -     try:
44 -         data = r.json()
45 -         if (import_all):
46 -             # Push job for next page of results onto queue
47 -             queue = opts['queue']
48 -             offset = params.get('offset', 0) + pagesize
49 -             next_job = { 'search_dict': search_dict, 'import_all': True, 'offset': offset }
50 -             if (data['itemCount'] > offset):
51 -                 logger.info(f'Enqueuing next job with offset {offset} out of {data["itemCount"]}
    - records.')
52 -                 await queue.enqueue('idigbio', json.dumps(next_job))
53 -             for record in data['items']:
54 -                 logger.info(f'Parsing {record["uuid"]}')
55 -                 record['media'] = []
56 -                 for uuid in record['indexTerms']['mediarecords']:
57 -                     mr = await client.get(media_url.format(uuid=uuid))
58 -                     if mr.status_code == 200:
59 -                         try:
60 -                             media_data = mr.json()
61 -                         except Exception:
62 -                             logger.exception(f'Failed to get the media with UUID {uuid}, belonging to
    - record {record["uuid"]}')
63 -                             logger.info(f'Found media record for {uuid}')
64 -                             record['media'].append(media_data)
65 -
66 -                 return data['items']
67 -             except Exception:
68 -                 logger.exception(f'Failed to parse the response to fetch')

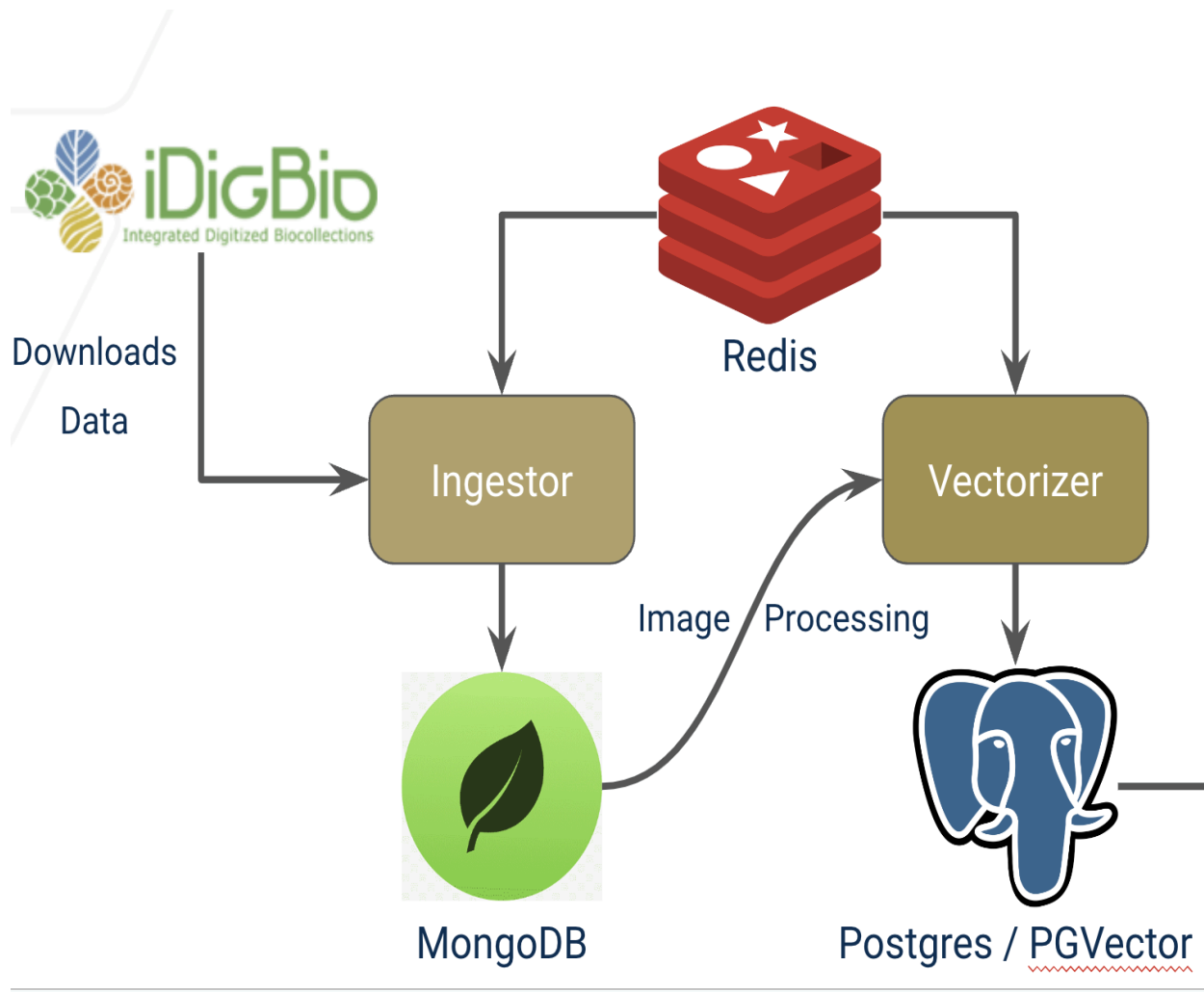
17 + async def fetch_with_retry(
18 +     client: httpx.AsyncClient, url: str, params: Optional[Dict[str, Any]] = None
19 + ) -> Dict[str, Any]:
20 +     response = await client.get(url, params=params)
21 +     response.raise_for_status()
22 +     return response.json()
23
24
25 + async def fetch_media_data(client: httpx.AsyncClient, uuid: str) -> Dict[str, Any]:
26 +     try:
27 +         return await fetch_with_retry(client, f"{MEDIA_URL}{uuid}")
28 +     except httpx.HTTPStatusError as e:
29 +         logger.error(
30 +             f"HTTP error occurred while fetching media data for UUID {uuid}: {e}"
31 +         )
32 +     except Exception as e:
33 +         logger.exception(
34 +             f"An error occurred while fetching media data for UUID {uuid}: {e}"
35 +         )
36 +     return {}
37
38
39 + async def idigbio_search(args: str, opts: Dict[str, Any]) -> List[Dict[str, Any]]:
40 +     """
41 +     Fetches data from the iDigBio API based on the search parameters provided, and populates the media field
    - with media data.
42 +     Supports pagination via looping with jobs.
43 +
44 +     Args:
45 +     args (str): A JSON string containing search parameters and options.
46 +     opts (Dict[str, Any]): A dictionary containing additional options, including the queue for pagination.
47
48 Returns:
49 +     List[Dict[str, Any]]: A List of records with populated media data.
50
51 +     """
52 +     params = json.loads(args)
53 +     search_dict = params.get('search_dict', {})
54 +     import_all = params.get('import_all', False)
55 +     offset = params.get('offset', 0)
56 +     page_size = 100
57 +     query_params = {'rq': json.dumps(search_dict), "limit": page_size, "offset": offset}
58
59     async with httpx.AsyncClient() as client:
60 +         try:
61 +             data = await fetch_with_retry(client, SEARCH_URL, query_params)
62 +
63 +             if import_all and data['itemCount'] > offset + page_size:
64 +                 queue = opts.get("queue")
65 +                 if queue:
66 +                     next_offset = offset + page_size
67 +                     next_job = {
68 +                         "search_dict": search_dict,
69 +                         "import_all": True,
70 +                         "offset": next_offset,
71 +                     }
72 +                     logger.info(
73 +                         f'Enqueuing next job with offset {next_offset} out of {data["itemCount"]} records.'
74 +                     )
75 +                     await queue.enqueue("idigbio", json.dumps(next_job))
76 +                 else:
77 +                     logger.warning("Queue not provided in opts, skipping pagination.")
78 +
79 +             media_tasks: List[Coroutine[Any, Any, Dict[str, Any]]] = []
80 +             for record in data["items"]:
81 +                 record["media"] = []
82 +
83 +                 for media_uuid in record["indexTerms"].get("mediarecords", []):
84 +                     media_tasks.append(fetch_media_data(client, media_uuid))
85 +
86 +             media_results = await asyncio.gather(*media_tasks)
87 +
88 +             for record, media_data in zip(data["items"], media_results):
89 +                 if media_data:
90 +                     record["media"].append(media_data)
91 +
92 +             return data["items"]
93 +
94 +         except httpx.HTTPStatusError as e:
95 +             logger.error(f"HTTP error occurred: {e}")
96 +         except Exception as e:
97 +             logger.exception(f"An unexpected error occurred: {e}")

```

Flow Charts/Diagrams

These scripts run on the same data they ran on before the refactor. No changes there. But for clarity's

sake, I'll reproduce the diagrams/flow charts below:



Documentation

Fortunately, this refactor doesn't change how to run anything. One thing that should be mentioned is that developers should add the Black formatter to their project directory (https://black.readthedocs.io/en/stable/getting_started.html). If they're using VSCode, you can just install the extension, which is probably the easiest way, especially if one configures black formatter to execute on save (<https://marketplace.visualstudio.com/items?itemName=ms-python.black-formatter>).

Results Visualization

No results to visualize this week.

Proof of Work

The refactor successfully did not alter the behavior of the ingestor scripts. Screenshots show as much:

```
> (ingestor_worker) vscode → /workspaces/NFHM (refactor/td-refactor-workers) $ python ingestor/ingestor.py
/opt/conda/lib/python3.11/site-packages/transformers/utils/generic.py:441: FutureWarning: `torch.utils._pytree._register_pytree_node` is deprecated. Please use `torch.utils._pytree.register_pytree_node` instead.
  _torch_pytree._register_pytree_node(
/opt/conda/lib/python3.11/site-packages/transformers/utils/generic.py:309: FutureWarning: `torch.utils._pytree._register_pytree_node` is deprecated. Please use `torch.utils._pytree.register_pytree_node` instead.
  _torch_pytree._register_pytree_node(
2024-08-23 18:12:47.429 - asyncio - DEBUG - Using selector: EpollSelector
2024-08-23 18:12:47.429 - __main__ - INFO - {"source_queue": "idigbio", "redis": {"host": "redis", "port": 6379, "database": 0, "username": null, "password": null}, "mongo": {"host": "mongo", "port": 27017, "database": "NFHM", "username": "root", "password": "example", "input_collection": "idigbio"}, "number_of_workers": 1, "postgres": {"host": "postgres", "port": 5432, "database": "nfhm", "table": "search_records", "user": "postgres", "password": "postgres"}, "queue": "ingest_queue", "redis_queue": "RedisQueue", "input": "inputs.idigbio.idigbio_search", "output": "outputs.mongo_output.dump_to_mongo"}
2024-08-23 18:12:47.432 - pymongo.serverSelection - INFO - {"message": "Waiting for suitable server to become available", "selector": "Primary()", "operation": "listDatabases", "topologyDescription": "<TopologyDescription id: 66c8d11f712c4f642a78b9aa, topology_type: Unknown, servers: [<ServerDescription ('mongo', 27017) server_type: Unknown, rtt: None]>}", "clientId": {"$oid": "66c8d11f712c4f642a78b9aa"}, "remainingTimeMS": 29}
```

Next Week's Proposal

- Meeting with Dr Porto and other collaborators at the University of Florida
- Both Roman and Vy snagged on getting Postgres to run their first time setting up the codebase. I'm going to try and fix this next week. Seems to be the only snafu they hit.
- Discuss and possibly start implementing an API accessible version of the ingestor jobs. (E.g., `POST /jobs/vector_embedder { model: "clip_ViT-B-32" }`) to make running jobs and experiments easier.