Week 7 Report

Thomas Deatherage
NFHM/BioCosmos
4 October 2024

Time Slot

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

- Weekly meeting with GaTech NFHM collaborators.
- Weekly meeting for LLM/UI stuff with Ben and Bree
- Weekly GaTech<>UF meeting
- Worked on fine-tuning florence-2's model (not done).

2) What are you planning on working on next?

- Finish florence-2 fine tuning experiment
- Focus on LLMs

3) Is anything blocking you from getting work done?

- Nope.

Abstracts & Summaries

1) InternVL: Scaling up Vision Foundation Models and Aligning for Generic Visual-Linguistic Tasks

Conference/Journal: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2024

Abstract: The exponential growth of large language models (LLMs) has opened up numerous possibilities for multi-modal AGI systems. However the progress in vision and vision-language foundation models which are also critical elements of multi-modal AGI has not kept pace with LLMs. In this work we design a large-scale vision-language foundation model (InternVL) which scales up the vision foundation model to 6 billion parameters and progressively aligns it with the LLM using web-scale image-text data from various sources. This model can be broadly applied to and achieve state-of-the-art performance on 32 generic visual-linguistic benchmarks including visual perception tasks such as image-level or pixel-level recognition vision-language tasks such as zero-shot image/video classification zero-shot image/video-text retrieval and link with LLMs to create multi-modal dialogue systems. It has powerful visual capabilities and can be a good alternative to the ViT-22B. We hope that our research could contribute to the development of multi-modal large models.

Summary: The document presents "InternVL," a large-scale vision-language foundation model designed to align a vision encoder with a large language model (LLM) for improved performance in various visual and visual-linguistic tasks. Key features of InternVL include:

Model Architecture: InternVL comprises a vision encoder, scaled to 6 billion parameters, and an 8 billion parameter LLM middleware, which enhances integration and representation alignment between visual and linguistic inputs.

Training Approach: The model employs a progressive image-text alignment strategy, starting with contrastive learning on large-scale noisy data followed by generative learning on more refined datasets. This helps stabilize training and improve model performance effectively.

Applications and Performance: InternVL achieves state-of-the-art results across 32 generic visual-linguistic benchmarks encompassing tasks such as image classification, image-captioning, and multi-modal dialogue systems. It demonstrates strong representations in both visual and linguistic modalities, enhancing its versatility for various applications.

Overall, InternVL bridges the gap between vision and language models, contributing significantly to advancements in multi-modal artificial general intelligence (AGI)
Link:
https://openaccess.thecvf.com/content/CVPR2024/html/Chen_InternVL_Scaling_up_Vision_Foundation_Models_and_Aligning_for_Generic_CVPR_2024_paper.html
Code: https://github.com/OpenGVLab/InternVL
Relevance: Very promising LLM, especially with its emphasis on vision capabilities.

Scripts and Code Blocks

The past two weeks (reports 5 and 6) were spent working on performance evaluation of trait grounding and trait referral with the florence-2 VLM model. This week I tried to make that performance better with fine-tuning. However, fine-tuning is a very slow, resource intensive-process. It has also proven to not be trivial to implement. Consequently, I don't have that working yet. But I'm pretty close.

To reiterate – the goal in fine-tuning is to explore to what extent florence-2 might work as the model for certain BioCosmos sub tasks. For example, could BioCosmos delegate trait identification tasks to the model and expect quality results?

I've repeated the current script in its entirety below. The basic idea is to build on top of the florence-2 base model with the fish-vista training set. This is done by translating the training data into something the model can use (I'm using LoRA and PEFT to help with this since these tools can help improve training efficiency and speed, and reduce resource consumption). We do this via a series of iterative loops or "epochs".

```python
import os
import torch
from torch.utils.data import Dataset, DataLoader
from transformers import AutoProcessor, AutoModelForCausalLM, AdamW,
get_linear_schedule_with_warmup
from peft import LoraConfig, get_peft_model
from PIL import Image
import numpy as np
import pandas as pd
import json
from tqdm import tqdm
import logging
import psutil
import gc

logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(levelname)s
- %(message)s')

if torch.cuda.is_available():
    dtype = torch.float16
else:
    dtype = torch.float32

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model_id = 'microsoft/Florence-2-base-ft'
model = AutoModelForCausalLM.from_pretrained(model_id,
trust_remote_code=True, torch_dtype=dtype).to(device)
processor = AutoProcessor.from_pretrained(model_id, trust_remote_code=True)

# LoRA Configuration
lora_config = LoraConfig(
    r=8,
    lora_alpha=8,
    target_modules=["q_proj", "o_proj", "k_proj", "v_proj", "linear",
"Conv2d", "lm_head", "fc2"],
    task_type="CAUSAL_LM",
    lora_dropout=0.05,
    bias="none",
    inference_mode=False,
    use_rslora=True,
    init_lora_weights="gaussian",
)
```

```python
# Apply LoRA to the model
model = get_peft_model(model, lora_config)
model.print_trainable_parameters()


def log_memory_usage():
    process = psutil.Process(os.getpid())
    logging.info(f"CPU Memory: {process.memory_info().rss / 1e9:.2f} GB")
    if torch.cuda.is_available():
        logging.info(f"GPU Memory: {torch.cuda.memory_allocated() /
1e9:.2f} GB / {torch.cuda.memory_reserved() / 1e9:.2f} GB")


class FishDataset(Dataset):
    def __init__(self, csv_file, image_dir, seg_dir, trait_map_path,
processor):
        self.data = pd.read_csv(csv_file)
        self.image_dir = image_dir
        self.seg_dir = seg_dir
        self.processor = processor

        with open(trait_map_path, 'r') as f:
            self.trait_map = json.load(f)

        self.valid_indices = []
        self.skipped_count = 0

        for idx, row in self.data.iterrows():
            img_name = row['filename']
            img_path = os.path.join(self.image_dir, img_name)
            seg_name = os.path.splitext(img_name)[0] + '.png'
            seg_path = os.path.join(self.seg_dir, seg_name)

            if os.path.exists(img_path) and os.path.exists(seg_path):
                self.valid_indices.append(idx)
            else:
                self.skipped_count += 1
                logging.warning(f"Skipping image {img_name} due to missing
files.")

        logging.info(f"Skipped {self.skipped_count} images due to missing
files.")
        logging.info(f"Dataset contains {len(self.valid_indices)} valid
```

```
images.")

    def __len__(self):
        return len(self.valid_indices)

    def __getitem__(self, idx):
        true_idx = self.valid_indices[idx]
        img_name = self.data.iloc[true_idx]['filename']
        img_path = os.path.join(self.image_dir, img_name)
        seg_name = os.path.splitext(img_name)[0] + '.png'
        seg_path = os.path.join(self.seg_dir, seg_name)

        image = Image.open(img_path).convert('RGB')
        seg_mask = np.array(Image.open(seg_path))

        # Get all traits present in the image
        traits = [self.trait_map[str(i)] for i in np.unique(seg_mask) if
str(i) in self.trait_map and i != 0]

        # Create a prompt for the new tassk
        # This is a new task i made up.  I want to explore the difference
        # in fine-tuning existing tasks vs. fine-tuning a brand new task.
I really don't know what the outcome will look like
        prompt = f"<SPECIES_TRAIT_GROUNDING>{', '.join(traits)}"

        # Create polygons for each trait
        polygons = []
        for trait_id, trait_name in self.trait_map.items():
            if int(trait_id) != 0:  # Exclude background
                trait_mask = (seg_mask == int(trait_id)).astype(np.uint8)
                contours, _ = cv2.findContours(trait_mask,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
                if contours:
                    polygon = contours[0].reshape(-1).tolist()
                    polygons.append({"trait": trait_name, "polygon":
polygon})

        return prompt, polygons, image


def segmentation_loss(pred_polygons, target_polygons):
    # TODO -- Not Done yet!  Adapt the existing IoU function from the
grounding.py script . . .
```

```
    # Implement a custom loss function for polygon prediction
    loss = torch.tensor(0.0, device=device)
    for pred, target in zip(pred_polygons, target_polygons):
        # Calculate IoU or other relevant metric between predicted and
target polygons
        # Add to the loss
        pass
    return loss


def collate_fn(batch):
    prompts, polygons, images = zip(*batch)
    inputs = processor(text=list(prompts), images=list(images),
return_tensors="pt", padding=True)

    # Ensure all required inputs are present
    if 'input_ids' not in inputs:
        inputs['input_ids'] = processor.tokenizer(list(prompts),
return_tensors="pt", padding=True)['input_ids']

    # Convert specific tensors to the desired dtype and device, keeping
others as integers
    inputs = {k: v.to(device=device, dtype=dtype if k not in ['input_ids',
'attention_mask'] else torch.long)
              for k, v in inputs.items()}

    return inputs, polygons


def fine_tune_florence2(train_dataset, val_dataset, model, processor,
num_epochs=3, batch_size=1, learning_rate=5e-5,
gradient_accumulation_steps=4, checkpoint_dir="./checkpoints"):
    model.to(device)

    train_dataloader = DataLoader(train_dataset, batch_size=batch_size,
shuffle=True, collate_fn=collate_fn)
    val_dataloader = DataLoader(val_dataset, batch_size=batch_size,
collate_fn=collate_fn)

    optimizer = AdamW(model.parameters(), lr=learning_rate)
    total_steps = len(train_dataloader) * num_epochs //
gradient_accumulation_steps
    scheduler = get_linear_schedule_with_warmup(optimizer,
```

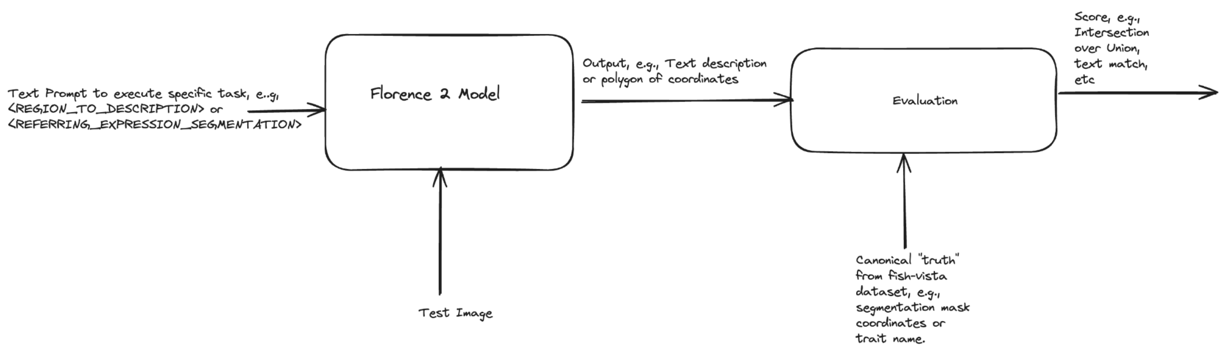
```
num_warmup_steps=100, num_training_steps=total_steps)

    for epoch in range(num_epochs):
        model.train()
        total_train_loss = 0
        optimizer.zero_grad()

        for step, (batch, target_polygons) in
enumerate(tqdm(train_dataloader, desc=f"Epoch {epoch+1}/{num_epochs}")):
            try:
                outputs = model(**batch)
                lm_loss = outputs.loss

                # Extract features for segmentation
                features = outputs.last_hidden_state[:, 0, :]  # Using the
first token's features

                # Segmentation prediction
                seg_logits = model.seg_head(features)
                seg_loss = segmentation_loss(seg_logits, target_polygons)

                loss = lm_loss + seg_loss
                loss = loss / gradient_accumulation_steps
                loss.backward()

                if (step + 1) % gradient_accumulation_steps == 0:
                    optimizer.step()
                    scheduler.step()
                    optimizer.zero_grad()

                total_train_loss += loss.item() *
gradient_accumulation_steps

                if step % 100 == 0:
                    log_memory_usage()
                    torch.cuda.empty_cache()
                    gc.collect()

            except Exception as e:
                logging.error(f"Error during training step: {e}")
                continue

        avg_train_loss = total_train_loss / len(train_dataloader)
```

```
        logging.info(f"Epoch {epoch+1}/{num_epochs} - Average training
loss: {avg_train_loss}")

        # Save checkpoint . . .kernel keeps crashing . . .
        checkpoint_path = os.path.join(checkpoint_dir,
f"checkpoint_epoch_{epoch+1}.pt")
        torch.save({
            'epoch': epoch,
            'model_state_dict': model.state_dict(),
            'optimizer_state_dict': optimizer.state_dict(),
            'loss': avg_train_loss,
        }, checkpoint_path)
        logging.info(f"Checkpoint saved: {checkpoint_path}")

        # Validation
        model.eval()
        total_val_loss = 0
        with torch.no_grad():
            for batch, target_polygons in tqdm(val_dataloader,
desc="Validation"):
                try:
                    outputs = model(**batch)
                    lm_loss = outputs.loss

                    features = outputs.last_hidden_state[:, 0, :]
                    seg_logits = model.seg_head(features)
                    seg_loss = segmentation_loss(seg_logits,
target_polygons)

                    loss = lm_loss + seg_loss
                    total_val_loss += loss.item()
                except Exception as e:
                    logging.error(f"Error during validation step: {e}")
                    continue

        avg_val_loss = total_val_loss / len(val_dataloader)
        logging.info(f"Epoch {epoch+1}/{num_epochs} - Average validation
loss: {avg_val_loss}")

        log_memory_usage()
        torch.cuda.empty_cache()
        gc.collect()
```

```python
    return model


def main():
    train_dataset = FishDataset(
        csv_file='./fish-vista/segmentation_train.csv',
        image_dir='./fish-vista/AllImages',
        seg_dir='./fish-vista/segmentation_masks/images',

trait_map_path='./fish-vista/segmentation_masks/seg_id_trait_map.json',
        processor=processor
    )

    val_dataset = FishDataset(
        csv_file='./fish-vista/segmentation_test.csv',
        image_dir='./fish-vista/AllImages',
        seg_dir='./fish-vista/segmentation_masks/images',

trait_map_path='./fish-vista/segmentation_masks/seg_id_trait_map.json',
        processor=processor
    )

    # Add a segmentation head to the model
    model.seg_head = torch.nn.Linear(model.config.hidden_size,
len(train_dataset.trait_map)).to(device)

    fine_tuned_model = fine_tune_florence2(train_dataset, val_dataset,
model, processor)

    # Save the fine-tuned model

fine_tuned_model.save_pretrained("./fine_tuned_florence2_species_trait")
    processor.save_pretrained("./fine_tuned_florence2_species_trait")
    logging.info("Fine-tuning completed and model saved.")

if __name__ == "__main__":
    main()
```

Flow Charts/Diagrams

Same general chart of florence-2 performance evaluation architecture from last week. I'm going to do this again, but against the fine-tuned florence-2 model once it's ready.



Documentation

Nothing new this week.

Results Visualization + Proof of Work

The fine tuning is still a WIP, so no results to show yet

Next Week's Proposal

- Finish fine tuning.
- Shift gears and focus on LLMs