Week 12 Report - NFHM

Romouald Dombrovski

November 8, 2024

1 Time Log

What progress did you make in the last week?

- Set up training loop for visual encodings
- Trained ViT-B/32 models from UNICOM and CLIP
- Evaluated results using eval metrics script
- Reported findings to team, discussed following steps

2 Abstract

Steinke D., et al. Towards a Taxonomy Machine - A Training Set of 5.6 Million Anthropod Images. doi: https://doi.org/10.1101/2024.07.15.600863.

AbstractThe taxonomic identification of organisms from images is an active research area within the machine learning community. Current algorithms are very effective for object recognition and discrimination, but they require extensive training datasets to generate reliable assignments. This study releases 5.6 million images with representatives from 10 arthropod classes and 26 insect orders. All images were taken using a Keyence VHX-7000 Digital Microscope system with an automatic stage to permit high-resolution (4K) microphotography. Providing phenotypic data for 324,000 species derived from 48 countries, this release represents, by far, the largest dataset of standardized arthropod images. As such, this dataset is well suited for testing the efficacy of machine learning algorithms for identifying specimens to higher taxonomic categories.
Summary (GPT-4o) The paper "Towards a Taxonomy Machine – A Training Set of 5.6 Million Arthropod Images" introduces a groundbreaking dataset of 5.6 million high-resolution arthropod images, representing over 320,000 species collected from 48 countries.

Captured through the Keyence VHX-7000 Digital Microscope system, these images feature specimens from 10 arthropod classes and 26 insect orders, presented in various orientations, which enhances the dataset's utility for training machine learning models on complex visual recognition tasks. This collection, derived from the global DNA barcoding project, supports the development of machine learning algorithms for accurate taxonomic classification, enabling image-based identification of arthropods and providing crucial data for biodiversity research. Each image links to a DNA Barcode Index Number (BIN) on the Barcode of Life Datasystems (BOLD), offering taxonomic insights and facilitating biomass and abundance estimation. This vast dataset, the largest standardized collection of its kind, sets a new foundation for automated species identification, supporting scalability with millions of images added yearly, which can further optimize AI-driven identification tools.

3 Scripts and Code Blocks

Last week we used the UNICOM repo code samples to create the training loop which was to be used to generate our own UNICOM models based on the ecological data found on the VLM4Bio dataset. If successful, we would continue to train larger datasets, such as Tree of Life.
Combining from my training loop and Thomas' training loop, which showed better results (as per Thomas' pull request)

```
import clip
from torch.utils.data import DataLoader

config = None


if config is None:
    config = Config()

# Make config globally available as args
global args
args = config

# Initialize distributed processing with a single process
os.environ['MASTER_ADDR'] = 'localhost'
os.environ['MASTER_PORT'] = '12355'
distributed.init_process_group(backend='nccl', world_size=1, rank=0)


# Load model and initialize

# model, transform_clip = clip.load("ViT-B/32") # if loading the model from the
```

```
    CLIP library
model, transform_clip = load("ViT-B/32", device="cuda") # if loading the model
    from UNICOM archive


# model = model.visual # only used if using the CLIP model
model = model.float() # Convert to full precision because that's what the rest
    of the unicom seems to want

model = WarpModule(model)
model.train()
model.cuda()

train_dataset = VLM4BioEmbeddingDataset(train_df, transform=transform_clip)

loader_train = DataLoader(
    train_dataset,
    batch_size=args.batch_size,
    num_workers=args.num_workers,
    pin_memory=True,
    drop_last=True,
    shuffle=True
)

backbone = model

margin_loss = CombinedMarginLoss(
    args.margin_loss_s,
    args.margin_loss_m1,
    args.margin_loss_m2,
    args.margin_loss_m3,
    args.margin_loss_filter
)

module_partial_fc = PartialFC_V2(
    margin_loss,
    args.output_dim,
    train_dataset.num_classes,
    args.sample_rate,
    False,
    sample_num_feat=args.num_feat,
)
module_partial_fc.train().cuda()

opt = torch.optim.AdamW(
    params=[
```

```
        {"params": backbone.parameters()},
        {"params": module_partial_fc.parameters(), "lr": args.lr *
            args.lr_pfc_weight}
    ],
    lr=args.lr,
    weight_decay=args.weight_decay
)

steps_per_epoch = len(train_dataset) // args.batch_size + 1
lr_scheduler = optim.lr_scheduler.OneCycleLR(
    optimizer=opt,
    max_lr=[args.lr, args.lr * args.lr_pfc_weight],
    steps_per_epoch=steps_per_epoch,
    epochs=args.epochs,
    pct_start=0.1,
)

callback_func = SpeedCallBack(10, args.epochs * steps_per_epoch, args.batch_size)
auto_scaler = torch.cuda.amp.grad_scaler.GradScaler(growth_interval=200)
global_step = 0

# Main training loop - removed train_sampler check
for epoch in range(args.epochs):
    for _, (img, local_labels) in enumerate(loader_train):
        img = img.cuda()
        local_labels = local_labels.long().cuda()

        with torch.cuda.amp.autocast(False):
            local_embeddings = backbone(img)
        local_embeddings.float()

        loss = module_partial_fc(local_embeddings, local_labels)
        auto_scaler.scale(loss).backward()

        if global_step % args.gradient_acc == 0:
            auto_scaler.step(opt)
            auto_scaler.update()
            opt.zero_grad()

        lr_scheduler.step()
        global_step += 1

        with torch.no_grad():
            callback_func(
                lr_scheduler,
                float(loss),
```

```
                global_step,
                auto_scaler.get_scale()
            )

    print(f"Completed epoch {epoch}")

print("Training completed")

# Saving the centroids and the model state dict after training loop completes
updated_centroids = module_partial_fc.weight.detach().cpu().numpy()

torch.save(backbone.state_dict(), 'trained_model_uniclip.pth')
np.save('final_centroids_uniclip.npy', updated_centroids)

print("Model and centroids saved.")
```

To load the model after upon next sessions. The model must have the same number and type of parameters as the backbone model which was trained:

```
import clip

state_dict = torch.load('trained_model_uniclip.pth')

model, transform = clip.load("ViT-B/32")
# model = model.visual # only used if using the CLIP model
model = model.float()
mod = WarpModule(mod) # Warp module used on top of ViT layers
mod.cuda()
mod.load_state_dict(state_dict)
```

The evaluation was done using the model eval script's code when comparing the different out-of-the-box pretrained models:

```
def vlm_species_eval(full_species_df, model, transform):
  full_species_df['taxa'] = full_species_df['taxa'].apply(lambda x: x.strip())

  eval_df = pd.DataFrame(columns=['model', 'taxa', 'column',
      'accuracy_individual', 'accuracy_avg', 'distance_individual',
      'distance_avg'])
  rows = []
  model = model.float()
  analysis_taxa = ['Fish', 'Bird', 'Butterfly', 'All']

  full_species_df =
      full_species_df[full_species_df['scientificName'].str.strip() != '']
  full_species_df['genus'] =
```

```
    full_species_df['scientificName'].apply(extract_genus)

# note the amount of time necessary to generate all embeddings
start = time.time()

full_species_df['image_embeddings'] =
    full_species_df['image_filename'].apply(lambda x: get_embeds(x, model,
    transform))
full_species_df.dropna(subset=['scientificName'], inplace=True)

time_length = time.time() - start
print(f'Took {time_length} seconds to generate embeddings')

# we're gonna compare species by species within each taxa, and against each
for taxa in analysis_taxa: # cleaned_dfs can have full species appended to it
    too
  print(f"WORKING ON TAXA: {taxa}")

  df = full_species_df if taxa == 'All' else
      full_species_df[full_species_df['taxa'] == taxa]
  train_df, test_df = get_test_train_split(df, test_size=0.2,
      stratify_by_scientific_name=False)

  start = time.time()
  test_df, accuracy_individual, accuracy_avg, distance_individual, distance_avg
      = apply_best_match_vectorized(test_df, train_df, 'scientificName',
      'trained_vlm_unicom')
  time_length = time.time() - start
  print(f"Time for analysis for model trained_vlm_unicom: {time_length}")
  rows.append({'model': 'trained_vlm_unicom', 'taxa': taxa, 'column':
      'species', 'accuracy_individual': f"{accuracy_individual * 100:.2f}%",
      'accuracy_avg': f"{accuracy_avg * 100:.2f}%", 'distance_individual':
      f"{distance_individual * 100:.2f}%", 'distance_avg': f"{distance_avg *
      100:.2f}%"})

  test_df, accuracy_individual, accuracy_avg, distance_individual, distance_avg
      = apply_best_match_vectorized(test_df, train_df, 'genus',
      'trained_vlm_unicom')
  rows.append({'model': 'trained_vlm_unicom', 'taxa': taxa, 'column': 'genus',
      'accuracy_individual': f"{accuracy_individual * 100:.2f}%",
      'accuracy_avg': f"{accuracy_avg * 100:.2f}%", 'distance_individual':
      f"{distance_individual * 100:.2f}%", 'distance_avg': f"{distance_avg *
      100:.2f}%"})
  if taxa == 'All': # only relevant when comparing against other taxa
    test_df, accuracy_individual, accuracy_avg, distance_individual,
        distance_avg = apply_best_match_vectorized(test_df, train_df, 'taxa',
```

```
        'trained_vlm_unicom')
    rows.append({'model': 'trained_vlm_unicom', 'taxa': taxa, 'column': 'taxa',
        'accuracy_individual': f"{accuracy_individual * 100:.2f}%",
        'accuracy_avg': f"{accuracy_avg * 100:.2f}%", 'distance_individual':
        f"{distance_individual * 100:.2f}%", 'distance_avg': f"{distance_avg *
        100:.2f}%"})

eval_df = pd.concat([eval_df, pd.DataFrame(rows)], ignore_index=True)
return eval_df
```

4 Visualization

I am not posting the loss functions this week as I have not generated any, but as I retraining the models, it will be something that will be present on next week's log.

Here are the results of the evaluation script for the two trained models (both models are ViT-B/32, but obtained from different sources). As a summary, accuracy is the cosine similarity across a test set of the images in comparison to the training set. Distance is the measure of euclidean distance. The individual comparison is measuring weather the topmost matched result belongs to the correct classification (whether that's species-level, genus-level or taxa-level), and avg comparison indicates if the match exists across all of the matches on average.

eval_clip_train_32epoch_ViT-B:32

	model	taxa	column	accuracy_individual	accuracy_avg	distance_individual	distance_avg
0	trained_vlm_unicom	Fish	species	46.76%	3.19%	46.76%	2.17%
1	trained_vlm_unicom	Fish	genus	86.04%	4.59%	86.04%	2.08%
2	trained_vlm_unicom	Bird	species	57.41%	62.78%	57.41%	62.05%
3	trained_vlm_unicom	Bird	genus	68.05%	56.29%	68.05%	55.61%
4	trained_vlm_unicom	Butterfly	species	85.67%	27.81%	85.67%	29.56%
5	trained_vlm_unicom	Butterfly	genus	98.70%	17.42%	98.70%	18.77%
6	trained_vlm_unicom	All	species	62.69%	27.20%	62.69%	19.81%
7	trained_vlm_unicom	All	genus	83.17%	23.91%	83.17%	14.88%
8	trained_vlm_unicom	All	taxa	99.98%	69.11%	99.98%	34.97%

Figure 1: Model Eval results from CLIP ViT-B/32 model trained on UNICOM cluster method

eval_unicom_load_train_50epoch_ViT-B:32

	model	taxa	column	accuracy_individual	accuracy_avg	distance_individual	distance_avg
0	trained_vlm_unicom	Fish	species	48.60%	3.86%	48.60%	2.22%
1	trained_vlm_unicom	Fish	genus	87.10%	4.35%	87.10%	1.93%
2	trained_vlm_unicom	Bird	species	64.22%	72.51%	64.22%	64.49%
3	trained_vlm_unicom	Bird	genus	73.68%	63.41%	73.68%	54.39%
4	trained_vlm_unicom	Butterfly	species	86.77%	28.16%	86.77%	26.36%
5	trained_vlm_unicom	Butterfly	genus	98.95%	16.18%	98.95%	16.82%
6	trained_vlm_unicom	All	species	66.67%	32.28%	66.67%	24.64%
7	trained_vlm_unicom	All	genus	85.30%	27.44%	85.30%	18.80%
8	trained_vlm_unicom	All	taxa	99.97%	54.12%	99.97%	31.84%

Figure 2: Model Eval results from UNICOM archived ViT-B/32 model trained on UNICOM cluster method

5 Next Week Proposal

- Generate captions using Thomas' code on ICE cluster
- Create two separate sets of data, using avg and concatenated text + visual embeddings
- Train ViT-L/14@336px model on VLM4Bio dataset
  - Determine whether its better to use pretrained model after all
- Refactor evaluation code for re usability
- Evaluate newly trained models
- Train model at different cluster sizes (e.g. 10% of classes, 1%, etc.)
- Meet with Dr. Porto to show results