Week 11 Report - NFHM

Romouald Dombrovski

November 1, 2024

1 Time Log

What progress did you make in the last week?

- Met up with Biocosmos collaborators
- Examined UNICOM code base
- Generated notebook to perform training on generated clusters
- Ran training loop, reported findings to team

2 Abstract

Alexey Dosovitskiy, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In ICLR, 2021.

AbstractWhile the Transformer architecture has become the de-facto standard for natural language processing tasks, its applications to computer vision remain limited. In vision, attention is either applied in conjunction with convolutional networks, orused to replace certain components of convolutional networks while keeping their overall structure in place. We show that this reliance on CNNs is not necessary and a pure transformer applied directly to sequences of image patches can perform very well on image classification tasks. When pre-trained on large amounts of data and transferred to multiple mid-sized or small image recognition benchmarks (ImageNet, CIFAR-100, VTAB, etc.), Vision Transformer (ViT) attains excellent results compared to state-of-the-art convolutional networks while requiring substantially fewer computational resources to train.

Summary (GPT-4o) The paper "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale" proposes the Vision Transformer (ViT), an innovative approach

to image recognition that abandons convolutional neural networks (CNNs) in favor of pure Transformers. By dividing images into fixed-size patches (like 16x16 pixels) and treating these patches as tokens, ViT applies a standard Transformer directly to images, leveraging the same architecture used in natural language processing. Through large-scale pre-training on datasets like ImageNet-21k and JFT-300M, ViT achieves competitive or superior performance compared to CNNs on major benchmarks, including ImageNet, CIFAR-100, and VTAB tasks, while also being more computationally efficient. Unlike CNNs, which have inductive biases like translation invariance and locality, ViT relies on extensive data to learn these features, making it highly effective with ample data but less so with smaller datasets. The study also explores the potential of self-supervised pre-training in ViT, showing promising improvements. The authors conclude that with further scaling and exploration of self-supervision, ViT holds significant promise for expanding the applicability of Transformers to broader computer vision tasks.

3 Scripts and Code Blocks

Last week we generated the embeddings for our image dataset (VLM4Bio) and then were able to cluster them using the faiss library from Meta. The VLM4Bio dataset contained 30,000 images which were then organized to 3000 clusters as an initial test (the correct ratio of clusters to samples will be a future experiment).
We used the coding samples from the UNICOM repository to help write the training loop which was used to train the model. From the UNICOM paper and the scripts themselves, it is determined that the training loop actually trains both the partial fc layer as well as the base openclip model itself. The embeddings that were initially generated (and saved) are only to be used for the sake of generating initial clusters. Each image is given a "pseudolabel" which is the cluster id which it belongs to. The training loop is used to train the model to recognize each image as belonging to its specific cluster group.

The code for generating the embedding dataset, taking in the dataframe and the transformation function for the images:

```
class VLM4BioEmbeddingDataset(torch.utils.data.Dataset):
    def __init__(self, df, transform=None):
        self.img_dir = 'downloaded_images/'
        self.image_files = df['image_filename'].tolist()
        if 'cluster_ids' in df.columns:
            self.cluster_ids = df['cluster_ids'].tolist()
        else:
            self.cluster_ids = [-1] * len(self.image_files)
        self.transform = transform
```

```
    def __len__(self):
        return len(self.image_files)

    def __getitem__(self, idx):
        # Load image and apply transformations
        image_path = os.path.join(self.img_dir, self.image_files[idx]).strip()
        image = Image.open(image_path).convert("RGB")
        if self.transform:
            image = self.transform(image)
        label = self.cluster_ids[idx]
        return image, label
```

The code for the partial fc module layer was taken straight from the unicom $partial_fc.py$ code.
The code to generate the base model being trained was taken from the unicom model.py code.
The code for the training loop came from unicom's retrieval.py code. It was updated to work on a single GPU run, as the code from UNICOM was mean to be run on a multi-GPU architecture.

```
from torch import optim
from tqdm import tqdm
import torch.distributed as dist
import os

# setting s, m1, m2, m3, sample_rate, num_feat based on
    https://github.com/deepglint/unicom/blob/main/retrieval.py#L34
s = 32
m1 = 1.0
m2 = 0.3
m3 = 0.0
sample_rate = 1.0
num_feat = 256 # random number, not based on defaults

# initializing CombinedMarginLoss and PartialFC
margin_loss = CombinedMarginLoss(s, m1, m2, m3)

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

partial_fc = PartialFC_V2(
```

```
    margin_loss, embedding_size=EMBEDDING_SIZE,
    num_classes=NUM_CLUSTERS, sample_rate=sample_rate,
    sample_num_feat=num_feat)

BATCH_SIZE = 128

# defaults from retrieval.py
LR = 0.0001
LR_PFC = 5.0

num_epochs = 10

partial_fc.train().cuda()

backbone_model, transform = load("ViT-L/14@336px", device="cuda")
train_transform = get_transform(336)

train_dataset = VLM4BioEmbeddingDataset(train_df, transform=train_transform)
test_dataset = VLM4BioEmbeddingDataset(test_df, transform=train_transform)

steps_per_epoch = len(train_dataset) // BATCH_SIZE

dataloader = torch.utils.data.DataLoader(train_dataset, batch_size=BATCH_SIZE,
    shuffle=True, drop_last=True)
optimizer = torch.optim.AdamW(
    [
        {"params": backbone_model.parameters(), "lr": LR},
        {"params": partial_fc.parameters(), "lr": LR * LR_PFC}
    ]
)
lr_scheduler = optim.lr_scheduler.OneCycleLR(
    optimizer,
    max_lr=[LR, LR*LR_PFC],
    steps_per_epoch=len(dataloader),
    epochs=num_epochs,
    pct_start=0.1
)

backbone_model.train()
backbone_model.float().cuda()
partial_fc.train()

for epoch in range(num_epochs):
    epoch_loss = 0
    with tqdm(dataloader, unit="batch") as batch_epoch:
        batch_epoch.set_description(f"Epoch [{epoch+1}/{num_epochs}]")
```

```
        for images, labels in batch_epoch:
          images = images.cuda()
          labels = labels.long().cuda()

          optimizer.zero_grad()

          with torch.cuda.amp.autocast(True):
              batch_embeddings = backbone_model(images).float().cuda()
              # batch_embeddings = normalize(batch_embeddings)
              loss = partial_fc(batch_embeddings, labels)

          loss.backward()
          optimizer.step()
          lr_scheduler.step()

          epoch_loss += loss.item()
          batch_epoch.set_postfix(loss=loss.item())

    avg_loss = epoch_loss / len(dataloader)
    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {avg_loss:.4f}")

# Final centroids after training
updated_centroids = partial_fc.weight.detach().cpu().numpy()

# Save model and centroids
torch.save(backbone_model.state_dict(), 'trained_model.pth')
np.save('final_centroids.npy', updated_centroids)

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

I ran the training loop and was unable to get a lowered loss function. I discussed this with Thomas, and he was able to get a decreased loss with his implementation, so I will compare the two.

4 Visualization

No visualization from this week.

5 Next Week Proposal

- Compare training loop with Thomas Deatherage
- Run the training loop on the clustered dataset
- Create an evaluation script using UNICOM code

- Test the newly trained model in comparison to the other models in the model_eval script
- Test different cluster sizes to find the most optimal
- Meet with Dr. Porto et al. to plan further