E3: Multimodal Representations

Group: 94
Colab notebook: https://colab.research.google.com/drive/1wVj_6mJIM2XPBFn2cybCWMI2LdgRGyUh?usp=sharing

Viktor Maximilian Lehnhausen, Mikael Alves Brito, Georg Matthes

TECHNISCHE UNIVERSITÄT DARMSTADT

1. ✍

   CLIP (CONTRASTIVE LANGUAGE– IMAGE PRE-TRAINING) CLIP can classify images based on a description. CLIP uses two separate encoders, a text encoder that processes the descriptions and an image encoder that processes the input image (ViT / ResNET). Both are projected into a shared embedding space, where the model measures the similarity between the image and each text description. The description with the highest similarity score is chosen as the predicted label.
   It can handle a zero-shot prediction. This is a type of classification where the model is given textual descriptions of possible image contents (e.g., “brown dog”, “black cat”) along with an input image to classify. The key idea is that the model can recognize images based solely on their descriptions, even if it has never seen those specific classes or images during training.

2. ✍

   Each GPU needs to calculate the full contrastive loss (both image-to-text and text-to-image) for the samples it holds. The image embeddings stay on their original GPU and are not moved around. Because of this, each GPU can only calculate similarities between its own images and the text embeddings it currently has. However, to fully compute the loss for its local text embeddings, each GPU also needs the similarity scores (logits) between those texts and the images on the other GPU. Therefore these similarity scores must be shared between the GPUs.

3. ✍

   The encoder architecture of SigLIT is nearly identical to CLIP/SigLip, using a vision encoder for images and a text encoder for language. The key difference lies in the loss function and that the vision encoder is frozen during fine-tuning.
   (1) The sigmoid loss is computationally more efficient than the Softmax loss. We do not need a similarity matrices across all embeddings.
   (2) SigLiT freezes the Image model during fine-tuning, thus we only fine-tune the text encoder. The CLIP model does adopt both encoders.

4. ✍

   The SigLiT model maintains stable at 32k batches, while SigLIP worsens around this threshold. This difference likely comes from SigLiT frozen vision model, whereas SigLIP’s unfrozen vision model may learn poorer image representations in large batches. For both models, Sigmoid outperforms Softmax at this scale. Softmax’s global normalization pushes apart all non-positive pairs (even related ones i.e. ”school/university”), while Sigmoid preserves meaningful similarities between semantically close examples while separating truly unrelated ones.

Experimental Section (17 Pts.)

1. Zero-shot Image Classification with SigLIP. (5 Pts.)

   a) ✍️

   The model was trained to store images and text in a shared embedding space. Matching pairs (e.g. image of a cat and the text ”cat) are placed close together in the embedding space. To classify an image the model checks how similar the image is to all the classes. The calss with the highest similiarty to the image is selected as the predicition. No extra training is needed for the classification task (”zero-shot classification”).

   b) ✍️

   In the original train function the text_embeds are build only once before the evaluation and training part and are not updated. But since we fine-tune using both, text and images we update both, the image embeddings and the text embeddings. Thus we need to build the updated text embeddings in every epoch of the evaluation of the SigLIP model.

   c) ✍️💻

```
--- cell 3
+++ cell 4
@@ -19,7 +19,6 @@
     processor = AutoProcessor.from_pretrained(model_name)

     class_names, wnid2idx = download_imagenet_label_lists()
-    text_embeds = build_text_embeddings(processor, model, class_names, device)

     optimizer = torch.optim.AdamW(model.parameters(), lr=lr, weight_decay=0)
     total_steps = epochs * len(train_loader)
@@ -39,6 +38,7 @@
         base_model = model.module if use_multi_gpu else model

         with torch.no_grad():
+            text_embeds = build_text_embeddings(processor, base_model, class_names, device
             for batch in val_loader:
                 images: List["PIL.Image.Image"] = batch["image"]
                 wnids = [Path(p).parent.name for p in batch["image_path"]]
```

2. Finetuning SigLIP (7 Pts.)

   a) 💻✍️

Final validation accuracy using captions: 69.67%; using templates: 87.27%. Captions contain more information, but are unstructured and might include irrelevant details. This makes it harder for the model to find the important information. Templates are structured and only contain necessary information. This makes it easier for the model to understand what matters and learn the important information. The results show that templates perform better than captions. This suggests that the structure of the templates is more important than the extra information in the captions.

b) 💻✍️

batch_size	n_train_classes	val acc
128	10	86.95%
64	10	89.82%
32	10	89.19%
16	10	91.05%
8	10	93.44%
4	10	95.16%

batch_size	n_train_classes	val acc
64	9	85.99%
64	8	84.73%
64	7	83.22%
64	6	77.91%
64	5	86.51%
64	4	85.33%
64	3	72.43%
64	2	80.60%
64	1	1.82%

c) ✍️

A smaller batch size correlates with a higher accuracy. Since we pefrom a backward pass for each batch, the samller the batch itself is, the more backward passes we make. This means we can learn the data better, but with the drawback, that we need longer for the computation, because we with a lower back size make more backward passes. Lower number of classes does not correlate with the accuracies...?

3. Finetuning SigLIT (5 Pts.)

a) 💻

```
# ---- YOUR CODE STARTS HERE ----
#... same as train
    print("\nLoading␣model␣&␣processor␣…")
    model = model or AutoModel.from_pretrained(model_name).to(device)
    processor = AutoProcessor.from_pretrained(model_name)

    # freeze the vision tower
    for parameter in model.vision_model.parameters():
        parameter.requires_grad = False

    optimizer = torch.optim.AdamW(model.parameters(), lr=lr, weight_decay=0)

#... same as in train
# ---- YOUR CODE ENDS HERE ----
```

b) 💻✍️

batch_size	num_classes	optimazation
64	10	80.30%
64	9	80.74%
64	8	80.36%
64	7	75.78%
64	6	72.57%
64	5	79.53%
64	4	77.11%
64	3	68.29%
64	2	79.82%
64	1	2.34%

batch_size	num_classes	optimazation
128	10	81.54%
64	10	80.74%
32	10	83.02%
16	10	82.70%
8	10	85.56%
4	10	81.80%

c) 💻✍️

We can see, that if we use just 1 class for fine tuning the SigLiT model, the validation accuracy drops to 2.34%. This may be to the fact, that we have no negative pairs during fine-tuning. Something similar could be seen in the SigLIP model. The batch size on the other hand does not follow a trend like in SigLIP. Here the validation accuracy does not increase with a smaller batch size. This was also visable in the paper presenting SigLiT.