🔍 Exploring Image Embeddings

Using DINOv2, UMAP, PCA & t-SNE

Chirayu Patel, Anish Kania, Aryan Jain and Manav Bhagat

⚙️ Setup & Model

- Using `timm` to load `ViT-small` pretrained with DINOv2
- Device: CUDA-enabled GPU (if available)
- Model scripted to TorchScript for deployment

About the Model:

- **DINOv2** is a self-supervised vision transformer (ViT) model trained to produce rich, general-purpose image representations without requiring labeled data.
- The **ViT-small** architecture splits each image into patches, processes them with transformer layers, and outputs a "CLS token" embedding summarizing the whole image.
- The **CLS token** is a special vector that captures global semantic information from the image, making it ideal for downstream tasks like clustering, visualization, and retrieval.
- Using a pretrained model like DINOv2 allows us to leverage powerful, transferable features even for images outside the original training set.

```
dino_model = timm.create_model('vit_small_patch16_224.dino', pretrained=True)
```

📦 Data Processing

- Images loaded via `OpenCV` and resized to 32x32 for uniformity
- Upsampled to 224x224 to match ViT input requirements
- Normalized using the mean and standard deviation expected by DINOv2
- Images are batched and passed through the model in groups for efficient GPU utilization
- Feature extraction from CLS token

Details:

- Each image is read from disk, converted to RGB, and resized to 32x32 pixels to ensure consistency across the dataset.
- Before passing to the model, images are upsampled to 224x224 pixels, as required by the ViT architecture.
- Images are normalized using the standard ImageNet mean and standard deviation, matching the DINOv2 training setup.
- Batching is used to process multiple images at once, maximizing GPU throughput and reducing inference time.

Feature Embeddings

- Total images: `{{ features_np.shape[0] }}`
- Feature vector shape: `(384,)`

What are Feature Embeddings?

- Each image is represented by a 384-dimensional feature vector (the CLS token from DINOv2 ViT-small).
- These embeddings capture high-level semantic and visual information, enabling comparison between images in a meaningful way.
- The feature vectors are used as input for dimensionality reduction techniques (UMAP, t-SNE, PCA) to visualize the structure of the dataset and discover patterns or clusters.
- Embeddings can also be used for tasks like image retrieval, clustering, anomaly detection, and more.

```
features = extract_features_dino(images_np)
```

👨‍🔬 DINOv2 ViT: Technical Overview

- Patch Embedding: The input image is split into fixed-size patches (e.g., 16x16), each linearly projected into a vector.
- Positional Encoding: Each patch embedding is added to a positional encoding to retain spatial information.
- Transformer Encoder: Multiple self-attention layers process the sequence of patch embeddings, allowing global context aggregation.
- CLS Token: A special learnable token is prepended to the sequence; after transformer processing, its output is used as the global image representation.
- DINOv2 Training: Uses self-distillation (teacher-student) to learn features without labels, making the embeddings highly transferable.

📉 Dimensionality Reduction: Technical Intuition

A summary of the main techniques used for visualizing and understanding high-dimensional feature embeddings:

- PCA: Projects data onto directions of maximum variance (linear, orthogonal axes).
- t-SNE: Minimizes divergence between pairwise similarities in high- and low-dimensional space; excels at preserving local structure.
- UMAP: Constructs a high-dimensional graph and optimizes a low-dimensional graph to be structurally similar; preserves both local and some global structure.
- Parametric UMAP: Learns a neural network mapping for dimensionality reduction, enabling fast inference on new data.
- t-SNE NN: Trains a neural network to approximate t-SNE, allowing scalable, parametric mapping.

UMAP Visualization

```
umap.UMAP(n_components=2).fit_transform(features)
```

PCA Visualization

- Fast linear projection
- Often used before t-SNE for speed

```
PCA(n_components=2).fit_transform(features)
```

🧠 t-SNE (Slow but Powerful)

- Perplexity: 30
- 1000 iterations

```
TSNE(n_components=2).fit_transform(features_pca_50)
```

Parametric UMAP (Keras)

- Feedforward MLP encoder
- GPU accelerated with TensorFlow

```
ParametricUMAP(encoder=keras_encoder).fit_transform(features_np)
```

🧠 t-SNE Approximation

- Learned mapping from features to 2D t-SNE space
- Train simple MLP on `(features, tsne_output)`

```
nn.Sequential(
  nn.Linear(384, 256), nn.ReLU(), nn.Linear(256, 2)
)
```

💾 Model Export

- Scripted feature extractor for deployment

```
torch.jit.script(DinoFeatureExtractor(dino_model))
```

Saved to: `dino_model/1/model.pt`

📚 Summary

- ✅ Extracted features using DINOv2
- ✅ Visualized using UMAP, PCA, and t-SNE
- ✅ Trained NN to approximate t-SNE
- ✅ Created deployable TorchScript model

🙌 Thank You!

Questions?

Let’s discuss further applications (e.g., clustering, search, anomaly detection)