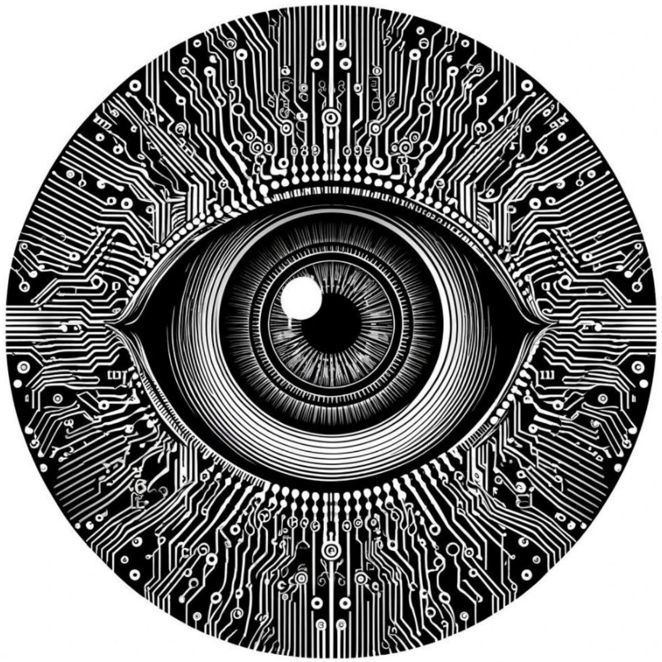
Image Embeddings



Antonio Rueda-Toicen

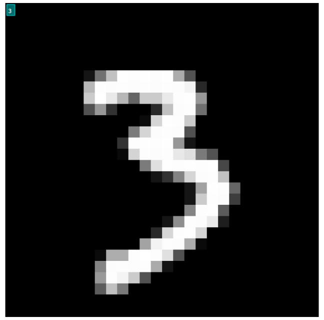
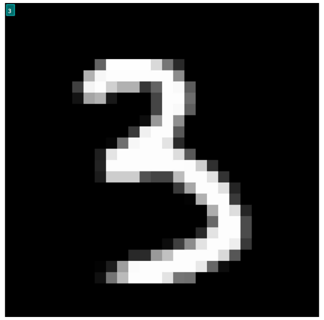
KI Service Zentrum by Hasso-Plattner-Institut HPI

SPONSORED BY THE
Federal Ministry of Education and Research

Learning goals

- Understand dense vector embeddings as learned representations
- Describe how image embeddings are produced in neural networks
- Extract image embeddings out of a pretrained Resnet50
- Inspect image neighborhoods using the cosine similarity metric

Sparse vs dense vector representations

$\mathbf{x}_1 =$  $\mathbf{x}_2 =$ 

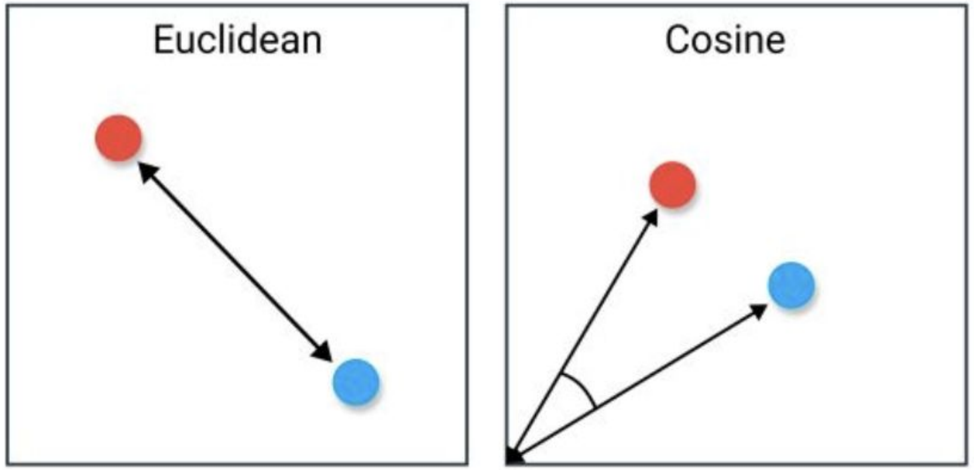
28 x 28 pixels = 784 values

$$\mathbf{x}_1\text{sparse} = \mathbf{x}_2\text{sparse} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^\top$$

$$\mathbf{x}_2\text{dense} = \begin{bmatrix} 0.10 & -0.75 & 0.35 & 0.41 & -0.20 & 0.89 & -0.60 & 0.03 & 0.59 & -0.50 \end{bmatrix}^\top$$

$$\mathbf{x}_1\text{dense} = \begin{bmatrix} 0.12 & -0.85 & 0.37 & 0.44 & -0.22 & 0.90 & -0.63 & 0.01 & 0.58 & -0.49 \end{bmatrix}^\top$$

Common metrics to compare embeddings



$$d_{\text{euclid}}(x_1, x_2) = \sqrt{\sum_{i=1}^{n} (x_{1i} - x_{2i})^2}$$

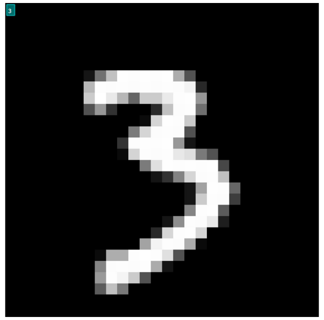
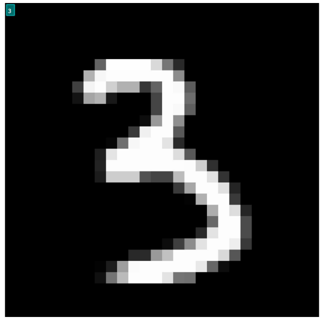
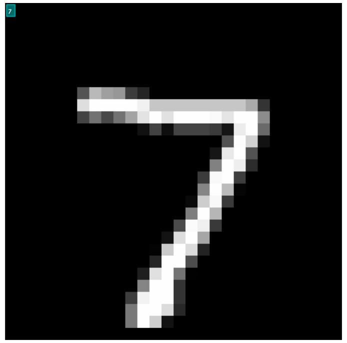
$$\text{cosine similarity}(x_1, x_2) = \frac{x_1 \cdot x_2}{\|x_1\| \, \|x_2\|}$$

Embeddings as representations learned during training

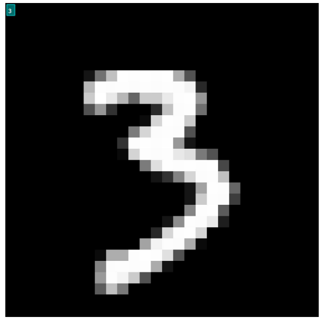
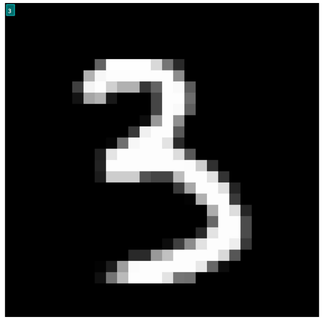
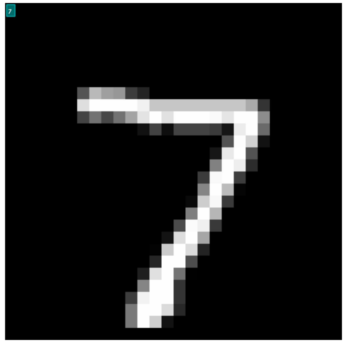
```python
import torch.nn as nn


conv_linear_embedder = nn.Sequential(
    nn.Conv2d(1, 32, kernel_size=3),  # 26x26x32
    nn.ReLU(),
    nn.Conv2d(32, 64, kernel_size=3),  # 24x24x64
    nn.ReLU(),
    nn.Flatten(),  # 24*24*64
    # This layer can be extracted as 64-dimensional
    # embedding
    nn.Linear(24*24*64, 64),
    nn.ReLU(),
    # This layer can be a 10-dimensional embedding
    nn.Linear(in_features=64, out_features=10)
)
```

$$H(p,q) = -\frac{1}{n}\sum_{i=1}^{n} p(i)\log q(i)$$

$\mathbf{x}_1 =$  $\mathbf{x}_2 =$  $\mathbf{x}_3 =$ 

Similar objects have similar embeddings

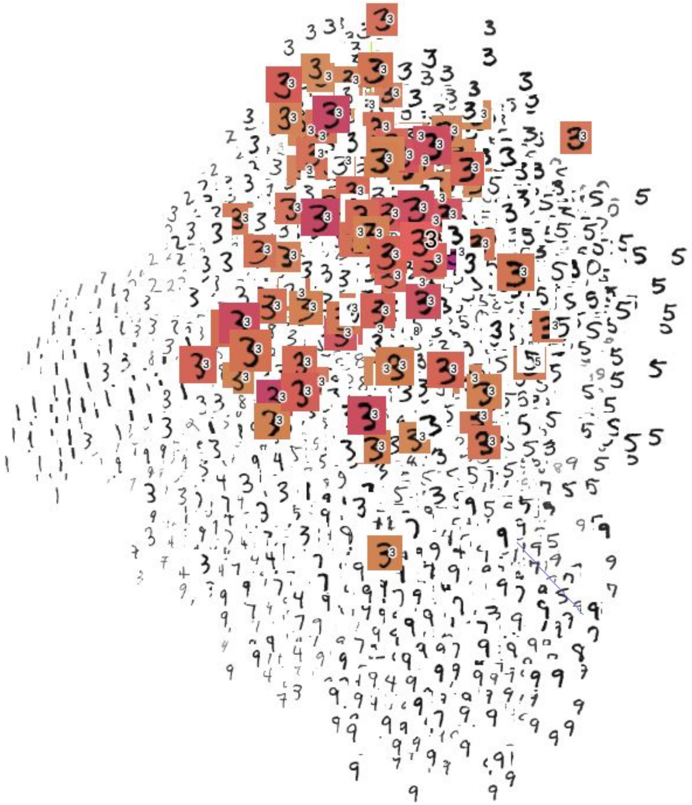
$\mathbf{x}_1 =$  $\mathbf{x}_2 =$  $\mathbf{x}_3 =$ 

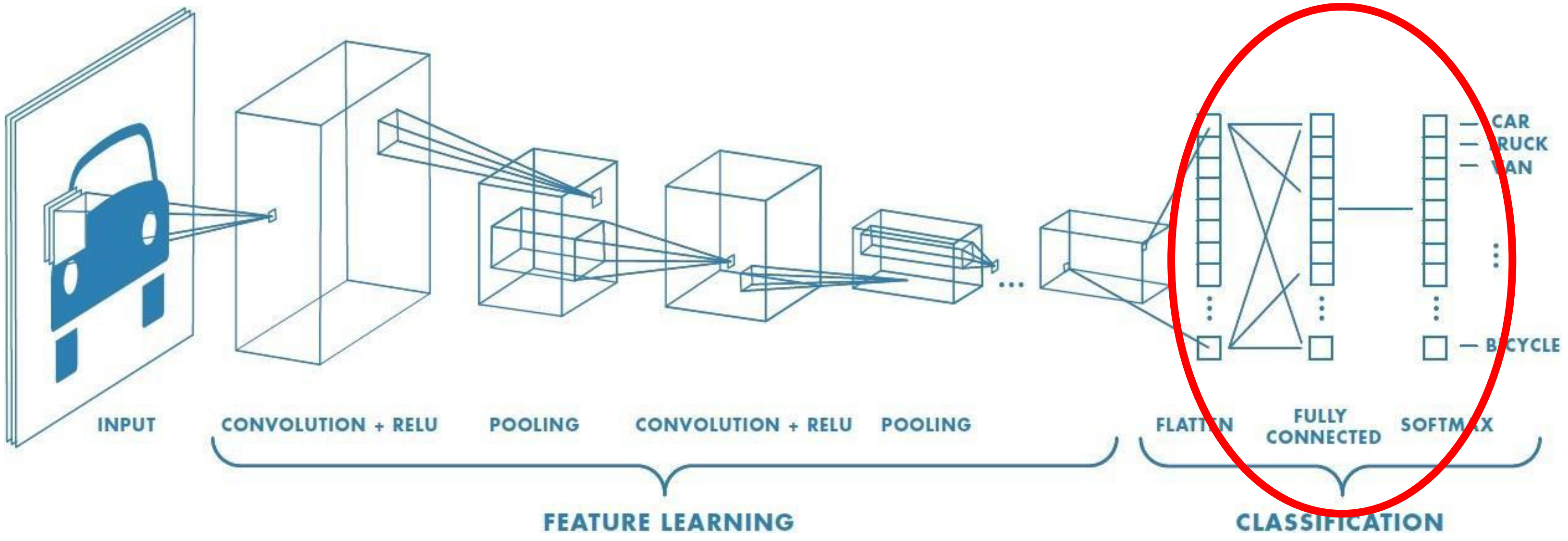
$$\text{Similar digits } (X_1, X_2): \quad \text{High cos similarity } (\approx 0.98), \ \text{Low Euclidean distance } (\approx 0.3)$$

$$\text{Different digits } (X_1, X_3): \quad \text{Low cos similarity } (\approx 0.4), \ \text{High Euclidean distance } (\approx 2.1)$$

Image embeddings as a byproduct of training

When we train a neural network, we make it **embed** inputs that are semantically similar, close to each other





Dense vector embeddings can be produced by flattening, pooling, or feeding (learned) features to linear layers

Source of image (try exploring interactively)

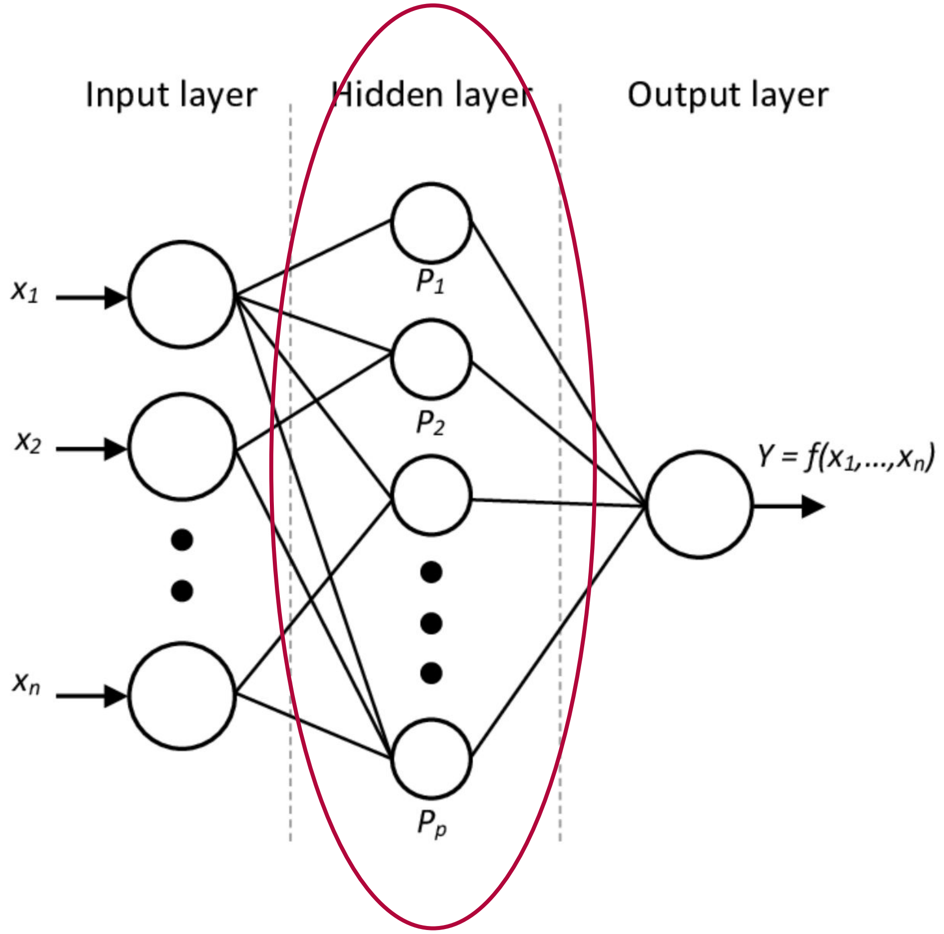
Approaches to produce embeddings with neural networks

```python
import torch.nn as nn

# Approach 1: Direct embedding
linear_embedder = nn.Sequential(
    nn.Flatten(),  # 784
    nn.Linear(784, 10)
)

# Approach 2: Convolution and then fully connected layer
conv_linear_embedder = nn.Sequential(
    nn.Conv2d(1, 32, kernel_size=3),  # 26x26x32
    nn.ReLU(),
    nn.Conv2d(32, 64, kernel_size=3),  # 24x24x64
    nn.ReLU(),
    nn.Flatten(),  # 24*24*64
    nn.Linear(24*24*64, 10)
)

# Approach 3: Global pooling after convolutions
conv_pool_embedder = nn.Sequential(
    nn.Conv2d(1, 32, kernel_size=3),  # 26x26x32
    nn.ReLU(),
    nn.Conv2d(32, 10, kernel_size=3),  # 24x24x10
    nn.ReLU(),
    nn.AdaptiveAvgPool2d((1, 1)),  # 1x1x10
    nn.Flatten()  # 10
)
```



$$L_{L1} = \frac{1}{N}\sum_{i=1}^{N} |y_i - \hat{y}_i|$$

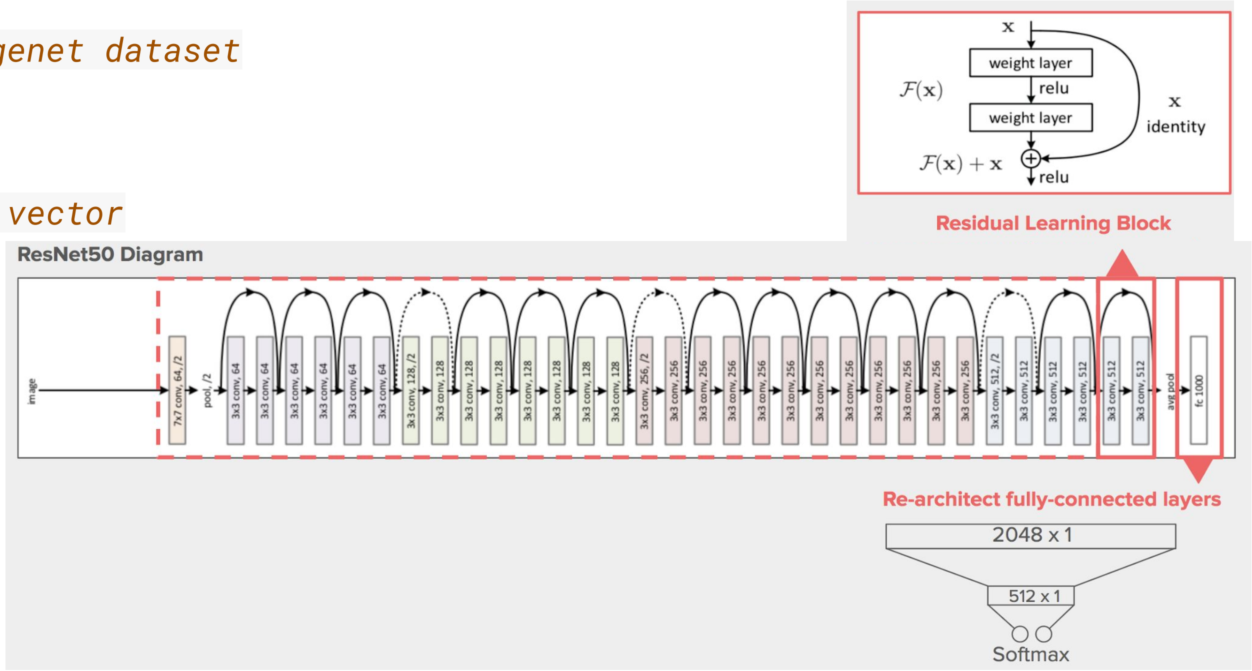
Extract embeddings from a pretrained Resnet50

```python
import torch
from torchvision import models

# Load pre-trained ResNet50 weights and transforms
weights = models.ResNet50_Weights.IMAGENET1K_V2
model = models.resnet50(weights=weights)
# Preprocess the image using the transformations from the imagenet dataset
preprocess = weights.transforms()

# Remove final Imagenet classification layer, keep 2048 dense vector
# Take time to unpack this syntax in the practical notebook
embedder = torch.nn.Sequential(*list(model.children())[:-1])

# Disable BatchNormalization
embedder.eval()
# Disable gradient computation
with torch.inference_mode():
    # unsqueeze(0) adds the batch size for inference
    # squeeze() removes the singleton dimensions from the output vector
    embedding = embedder(preprocess(image).unsqueeze(0)).squeeze()
```



README.md

Art Recommendation system

This is the repository of a portfolio project at DSR. This project aims to identify similar images using pre-trained computer vision networks. For an explanation of the technology see the technology section.

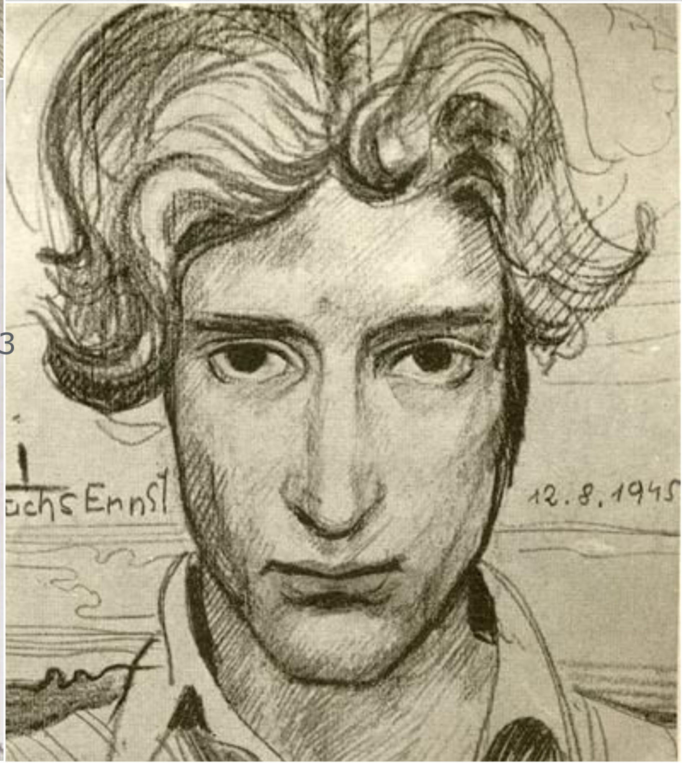
Contributors

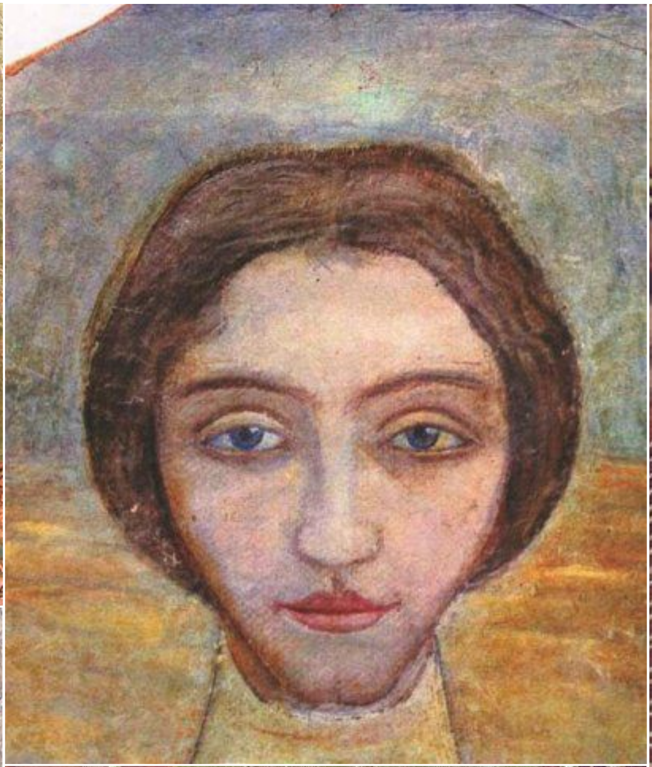
- Catarina Ferreira
- Gargi Maheshwari

WIKIART
VISUAL ART ENCYCLOPEDIA

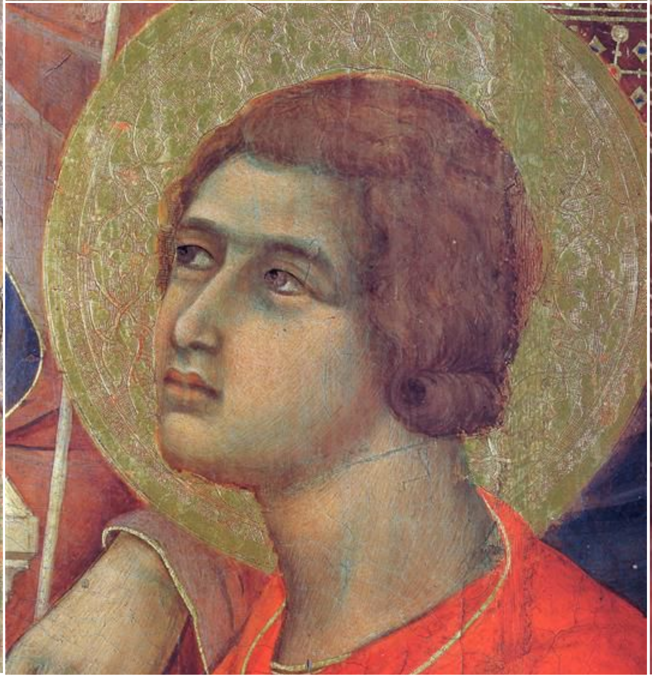
https://github.com/gargimaheshwari/Wikiart-similar-art

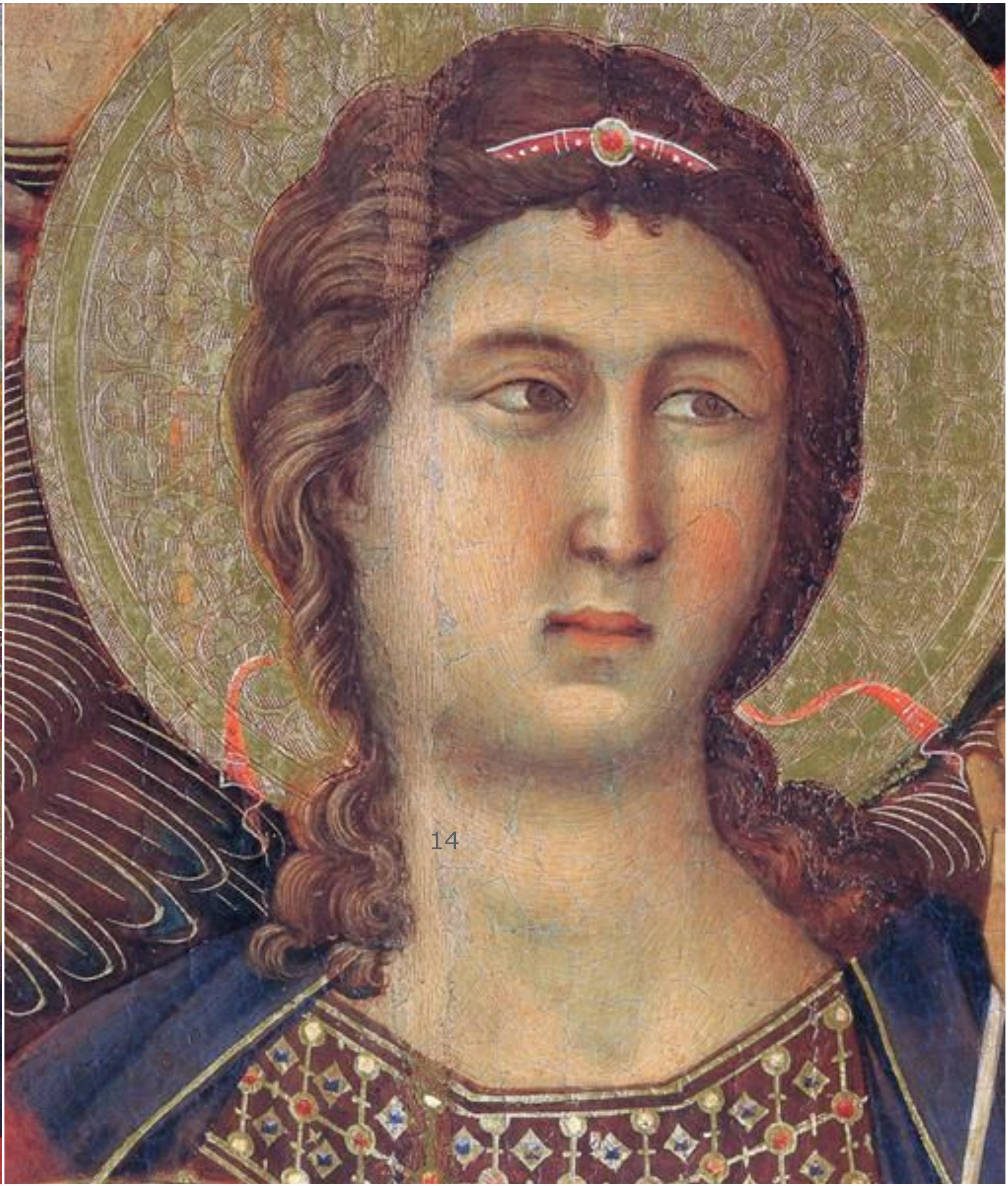




uchs Ernst
12.8.1945




S·BAR





Fine-grained embeddings with triplets

```
import torch import torch.nn.functional as F
def triplet_loss(query, positive,
                 negative, margin=1.0):
    pos_dist = F.pairwise_distance(query, positive)
    neg_dist = F.pairwise_distance(query, negative)
    loss = F.relu(pos_dist - neg_dist + margin)
    return loss.mean()
```

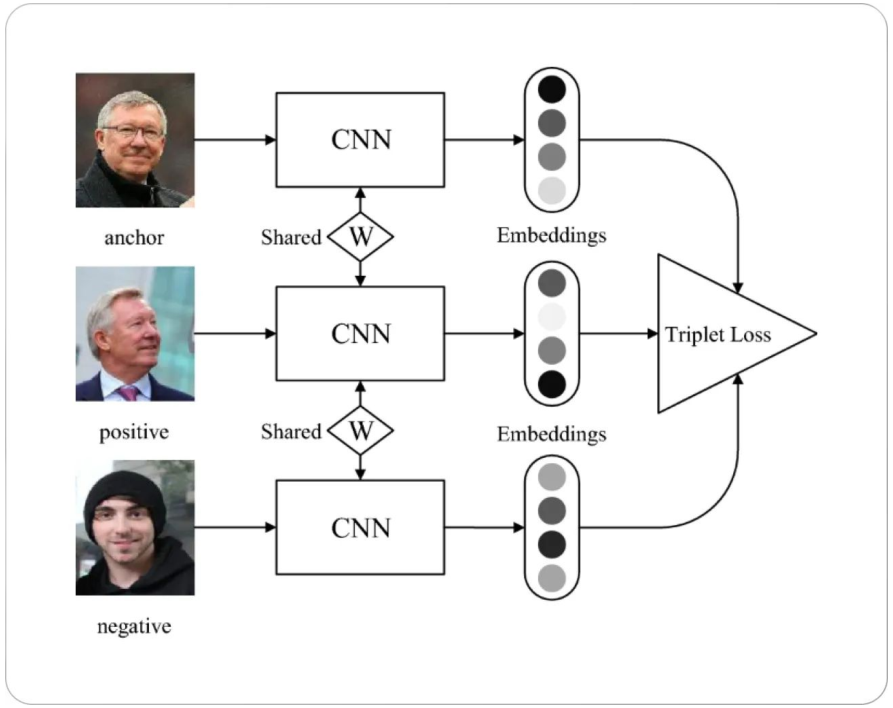


Image from [Triplet Loss: Intro, Implementation, Use Cases]

$$L = \max\left(\|f(a) - f(p)\|_2^2 - \|f(a) - f(n)\|_2^2 + \alpha, 0\right)$$

Application: differential diagnosis

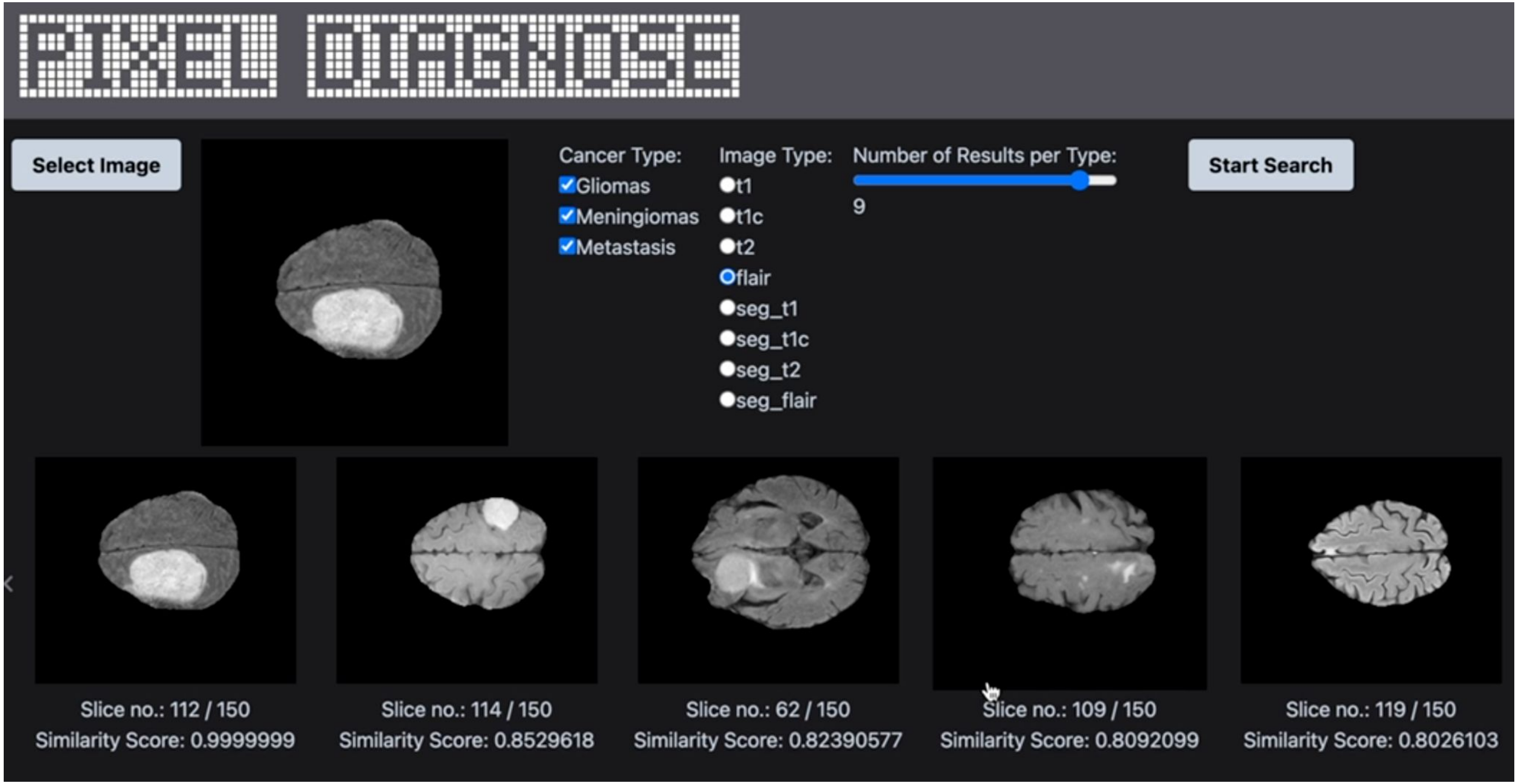
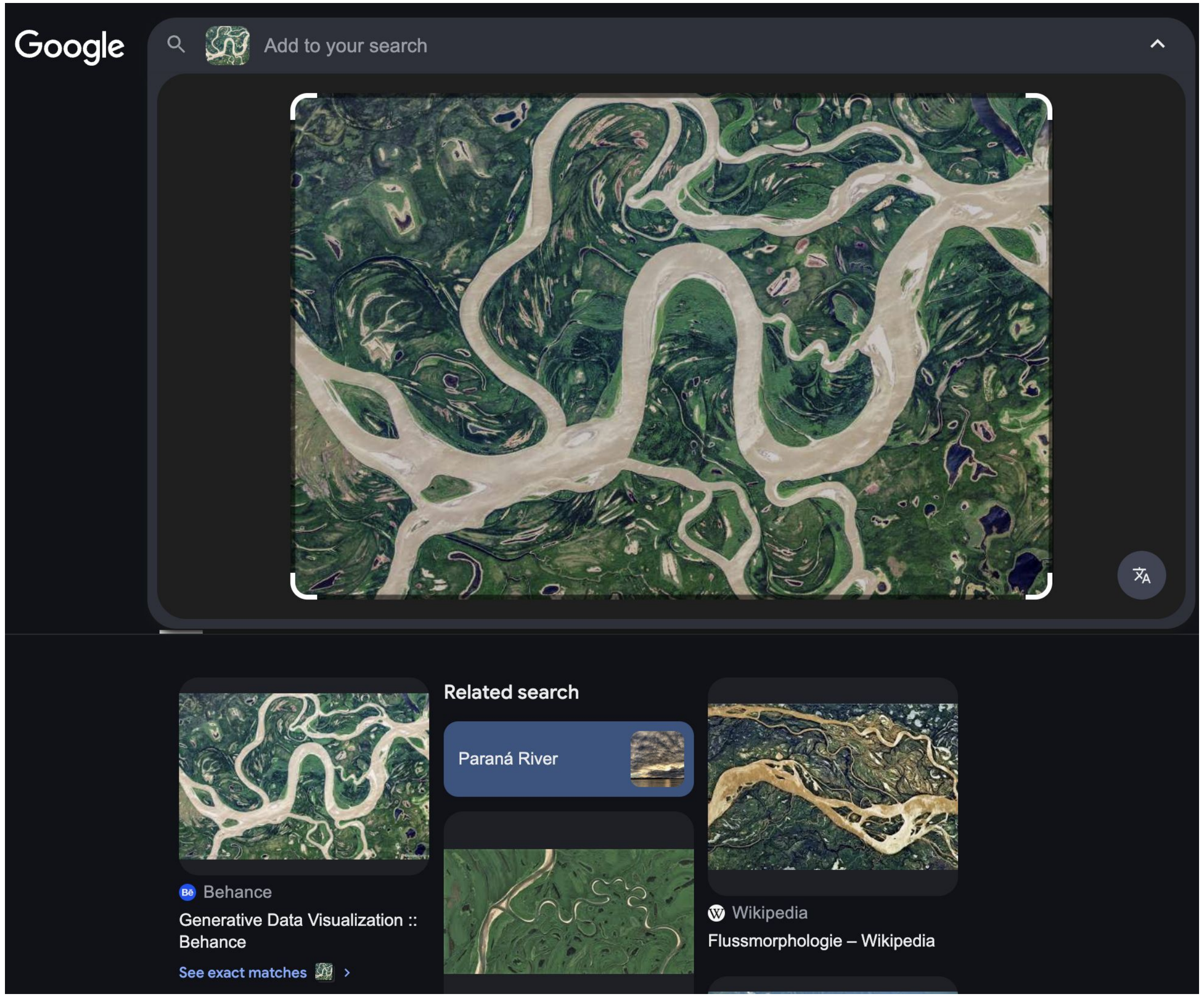


Image from https://pixel-diagnose.github.io/

Application: search by image



Search for similar images in images.google.com

Summary

Embeddings are learned vector representations

- Neural networks learn to map similar images to similar vectors through training
- We measure the similarity of embeddings using metrics like cosine similarity and Euclidean distance

There are different architectural approaches to create embeddings

- We can use linear layers, or flattened and pooled convolutions to produce embeddings
- Imagenet-pretrained models produce rich representations in their embeddings
- Embeddings can also be fine tuned through the triplet loss

Embeddings have multiple applications in semantic search and dataset curation

- They can be used for reverse image search, recommendation systems, and deduplication

SPONSORED BY THE

Federal Ministry of Education and Research