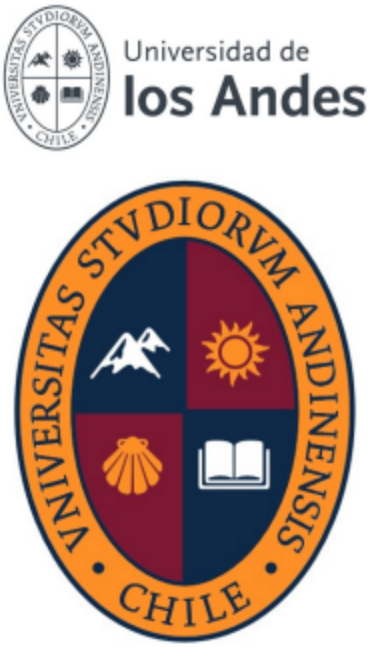


Universidad de los Andes

Faculty of Engineering and Applied Sciences.

Artificial Intelligence

Report 2:

Evaluation of Visual Encoders for Image Retrieval

Diego Eyzaguirre

Eduardo Saavedra

Santiago, Chile

May, 2025

Abstract

Neural networks enable the resolution of a wide range of problems, one of them being the retrieval of similar images by extracting visual features. This report evaluates how different pre-trained encoder models can extract meaningful image representations, which are then used to retrieve visually similar images.

To achieve this, four different encoders were tested: ResNet18, ResNet34, CLIP, and DINOv2. Scripts were implemented to process the images, extract their feature vectors, and compute similarity using cosine distance. Three distinct datasets were analyzed, and model performance was evaluated using metrics such as mean Average Precision (mAP) and precision–recall curves.

The results show that all encoders were capable of extracting relevant features, though with varying levels of accuracy. While the ResNet models performed reasonably well in simpler cases, CLIP and DINOv2 stood out in terms of both precision and consistency. DINOv2, in particular, achieved the best results, probably due to its Vision Transformer architecture and self-supervised training.

In terms of precision and recall of each encoder, if we take DINOv2 as the top performance (100%), CLIP would have 87.04% of DINOv2´s performance, ResNet34 would have 34.4% and ResNet18 would be at the bottom with a performance equivalent to 19.283% of DINO. It is important to note that the performance of each encoder was strongly influenced by its model´s architecture, meaning that with more complex architectures, like the ViT one of CLIP and DINOv2 the results obtained were better.

Finally, DINOv2 proved to be the most effective encoder for image retrieval tasks, significantly outperforming traditional convolutional architectures and showing a slightly better performance than CLIP. This suggests that transformer-based models offer a more robust and efficient solution for real-world visual similarity problems.

Index

1. Introduction

In recent years, the proliferation of AI models capable of processing image or text inputs to generate corresponding images has become increasingly common. While many users may assume that each of these models is entirely distinct, they often share a common encoder architecture and differ primarily in their decoders. The encoder is responsible for transforming the input into a meaningful latent representation, which the decoder then uses to generate the final output. This assignment aims to explore and evaluate various encoder architectures used for image retrieval, comparing their performance to determine which encoder yields the best results and by what margin.

1.1. Kernel Filter

A kernel filter, or simply kernel, is a sliding matrix which has different values alongside it and based on these values, it is capable of extracting specific features of the image by multiplying the image with the kernel and outputting a single value. In other words, it is a feature extractor and it relies on multiple hyperparameters, which are the kernel size (usually 3x3), the stride, which determines how the filter moves alongside the image, and the image padding, which optimally is calculated as (insert image padding formula here please). It is possible to combine the kernel with machine learning, making it so that the parameters are associated to each value inside of the kernel matrix, allowing it to extract even more specific features and even compress or expand the image using the hyperparameters.

A kernel filter, or simply kernel, is a small matrix used in image processing that slides over an image to extract specific features. This process involves element-wise multiplication between the kernel and the portion of the image it overlaps, followed by summation to produce a single output value. In essence, a kernel acts as a feature extractor, detecting patterns such as edges, textures, or gradients, depending on its values.

Several hyperparameters influence how a kernel operates. These include:

- Kernel size: typically a 3×3 matrix, though other dimensions like 5×5 or 7×7 are also used depending on the task.

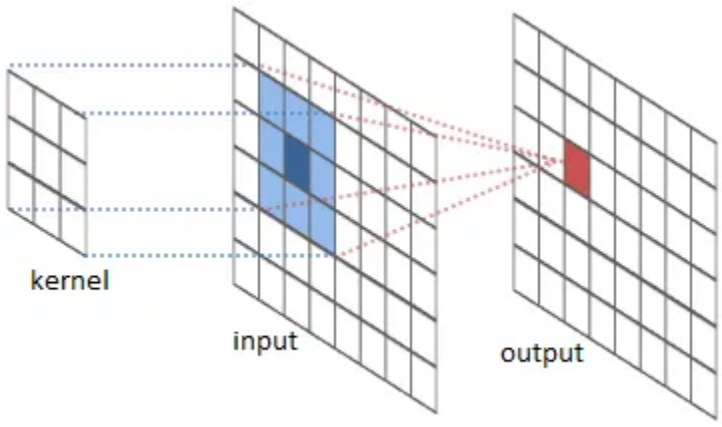
- Stride: the number of pixels the kernel moves at each step across the image.

- Padding: the process of adding borders to the input image to control the spatial dimensions of the output. Optimal zero-padding is often calculated to maintain the original image size using the formula:

$$Padding = floor(\frac{(Stride - 1) \cdot Input\ Size - Stride + Kernel\ Size}{2})$$

When combined with machine learning, particularly in convolutional neural networks (CNNs), the kernel's values become learnable parameters. This allows the model to automatically adapt and optimize the kernels during training to extract highly task-specific features. Furthermore, by adjusting the hyperparameters, kernels can be used not only for feature extraction but also for spatial transformation, such as compressing or expanding the image representation.

Figure 1.1: Kernel Filter

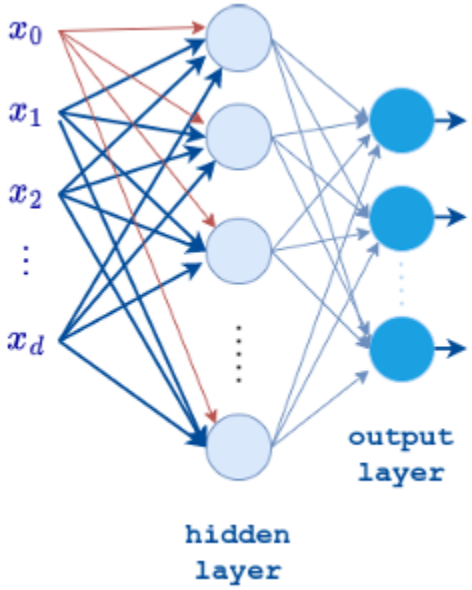


Source: River Trial (Github)

1.2. MLP

A Multilayer Perceptron (MLP) is an extension of the computational model of a biological neuron, known as a perceptron. The behavior of a single perceptron, or neuron, resembles that of a logistic regression model. By combining and connecting multiple perceptrons, MLPs form a type of neural network organized into layers, where the output of one neuron becomes the input to another. A network in which each neuron is connected to every other neuron in the adjacent layer is referred to as a fully connected network.

Figure 1.2: Multi Layer Perceptron



Source: Fundamentals of Machine Learning (José Manuel Saavedra)

The neurons in the inner layers, also known as hidden layers, use non-linear differentiable functions to transform the inputs layer by layer, making the final output easier to classify.

1.3. Convolutional Layer

A convolutional layer is a fundamental component of convolutional neural networks (CNNs), designed to automatically learn and extract spatial features from input data, typically images. It operates by applying a set of learnable kernels (also known as filters) that slide across the input,

performing element-wise multiplications followed by summation at each position. This process results in a feature map (or activation map), which highlights the presence and location of learned features such as edges, textures, or patterns.

Unlike traditional image processing where kernels are manually defined, in convolutional layers the kernel values are learned automatically through backpropagation. This allows the network to discover and optimize the most relevant features for the task at hand. Furthermore, convolutional layers contribute to parameter sharing and sparse connectivity, making them more computationally efficient and effective at recognizing local patterns in the input compared to fully connected layers.

1.4. Normalization Layer

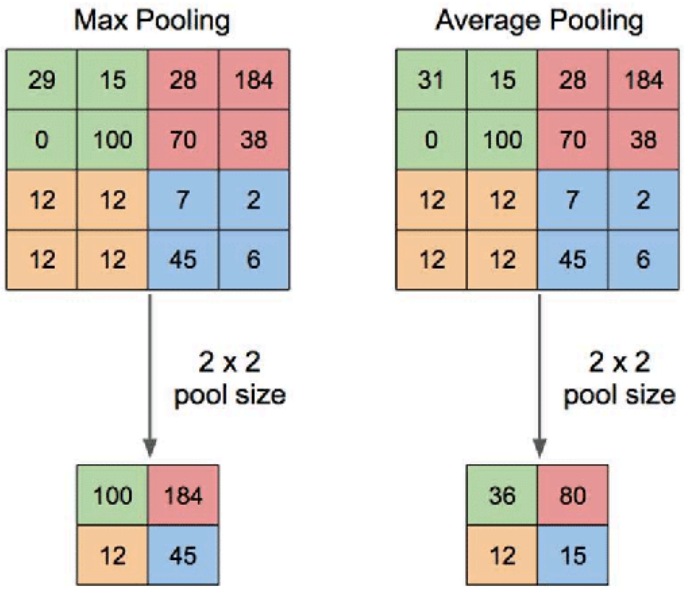
Normalization layers are used in neural networks to stabilize and accelerate training by ensuring that the inputs to each layer maintain consistent statistical properties. Two common techniques are batch normalization and layer normalization. Batch normalization operates across the batch dimension, normalizing inputs by subtracting the batch mean and dividing by the batch standard deviation, followed by learnable scaling and shifting. This helps reduce internal covariate shift and is especially effective in convolutional and feedforward networks. On the other hand, layer normalization normalizes across the features of each individual input (rather than across the batch), making it more suitable for models where batch sizes are small or variable, such as recurrent neural networks and transformers. Both methods help improve convergence speed and model generalization.

1.5. Pooling Layer

Pooling layers are used in CNNs to reduce the spatial dimensions of feature maps, helping to lower computational cost and control overfitting while preserving the most important information. The two most common types are max pooling and average pooling. Max pooling selects the highest value within a defined window (for example 2×2), effectively capturing the most prominent features such as edges or textures. In contrast, average pooling computes the average of all values in the window, providing a smoother and more generalized representation

of the region. Both methods help the model achieve spatial invariance, making it more robust to translations or distortions in the input.

Figure 1.3: Pooling Mechanism



Source: Research Gate (Casi Setianingsih)

1.6. Vanishing Gradient

As previously mentioned, a machine learning model, particularly a deep neural network, can consist of multiple layers, which progressively extract higher-level features from the input data. However, as the number of layers increases, the gradients used to update the model's parameters during backpropagation can become increasingly small in the earlier layers, a phenomenon known as the vanishing gradient problem. This occurs due to the repeated application of the chain rule when computing gradients across many layers. As a result, the early layers may receive gradients that are too small to significantly update their weights, rendering them ineffective at learning. This can ultimately reduce the model's overall efficiency and accuracy.

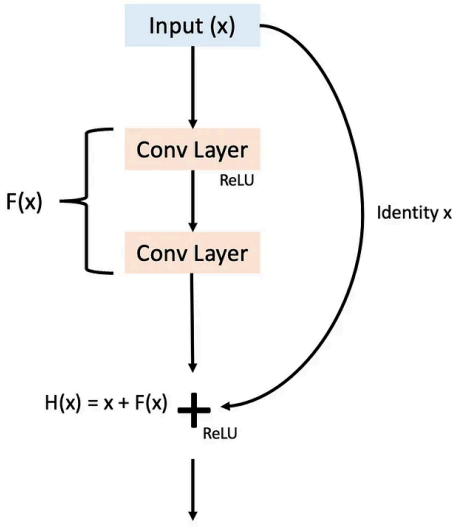
1.7. Residual Layer

As previously discussed, the vanishing gradient problem poses a significant challenge for training deep neural networks, as it can hinder the ability of earlier layers to learn meaningful representations. One widely adopted solution to mitigate this issue is the use of residual layers. Residual layers are designed to allow the network to bypass, or "skip over", certain layers during training by using shortcut (or skip) connections that directly pass the input to a deeper layer without modification.

This mechanism enables the model to retain and propagate useful information forward even when a particular layer fails to learn effectively due to vanishing gradients. In essence, if the learning in a layer stalls, the residual connection ensures that the information continues to flow, making the training process more stable and efficient.

For a residual connection to work correctly, the input and output dimensions of the layer must match, so that the skipped input can be directly added to the output of the deeper layer. If the dimensions differ, a linear transformation (such as a 1×1 convolution) is typically applied to the input to align the shapes before the addition operation. This architectural design allows the network to dynamically decide whether to use the transformation learned by the layer or to rely on the identity mapping, improving both convergence and accuracy in very deep models.

Figure 1.4: Residual Layer



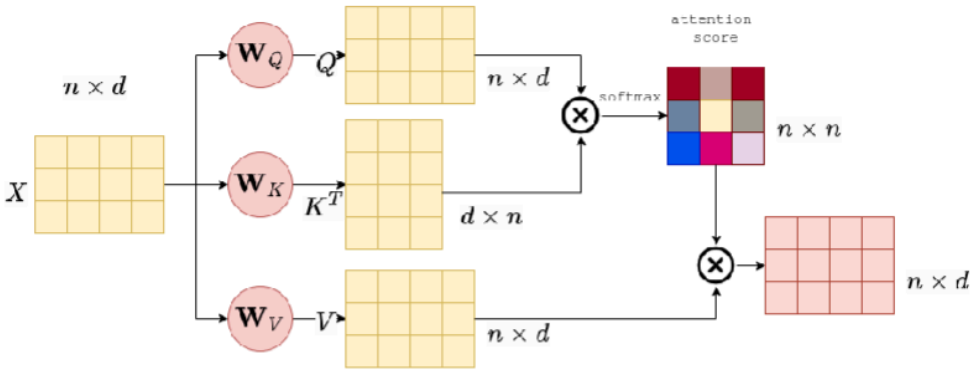
Source: Understanding Residual Connections in Neural Networks (Medium)

1.8. Attentional Layer

An attention layer is a neural network component designed to dynamically focus on the most relevant parts of the input when making predictions. Instead of treating all inputs equally, it assigns different weights to each element based on its importance to the task. This mechanism allows the model to better capture long-range dependencies and contextual relationships, especially in sequences such as text or time-series data.

This layer has two main steps, the first one being calculating the attention vector (for cross attention or self attention) and the second one is reducing the input values by calculating the dot product with the attention vector.

Figure 1.5: Attentional Layer Operation

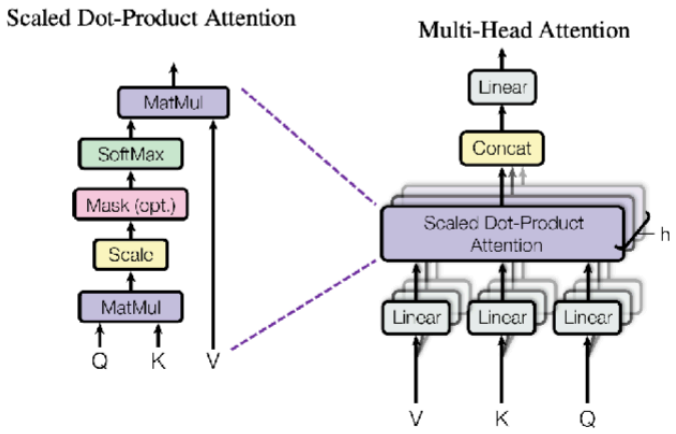


Source: Fundamentals of Machine Learning (José Manuel Saavedra)

1.9. Multi-Head Attention

Multi-head attention is an extension of the attention mechanism that allows the model to learn multiple types of relationships in parallel. It does this by running several attention layers (called “heads”) independently, each focusing on different parts of the input. The outputs from all heads are then combined and projected back into a single representation, enhancing the model’s ability to capture diverse features and patterns across the input.

Figure 1.6: Multi-Head Attention Architecture



Source: Fundamentals of Machine Learning (José Manuel Saavedra)

1.10. Spatial Distances

Most inputs to a machine learning model can be represented as points in a multi-dimensional space. In this space, the distance between two points serves as a measure of their similarity, closer points are more similar, while distant points are less alike. However, since the space is multi-dimensional, the method used to calculate this distance can vary, and each method captures a different notion of similarity.

a. Cosine Distance

Cosine distance measures the angle between two vectors originating from the origin and pointing toward the input points. It focuses on the orientation rather than the magnitude of the

vectors. A cosine similarity of 1 (or an angle of 0°) means the vectors point in the same direction (maximum similarity), while a similarity of -1 (or 180°) indicates they are pointing in opposite directions (maximum dissimilarity).

$$cos(\theta) = \frac{A \cdot B}{||A|| \cdot ||B|}$$

b. Euclidean Distance

Euclidean distance represents the diagonal between the two points to be compared, mathematically, it is the equivalent of tracing a diagonal from one point to another and the distance between them is how similar they are (the smaller the distance, the more similar they are).

$$distance(a, b) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}$$

1.11. Encoder

An encoder is a neural network component designed to transform input data (images, text, or audio) into a compact and informative representation, typically a fixed-size vector or a lower-dimensional feature map. This representation captures the most relevant features of the input, discarding redundant or irrelevant information. Encoders are commonly used as the first stage in many machine learning and deep learning architectures, and they are usually reused and only the decoder is modified to classify into different categories.

The encoder's primary role is to extract meaningful patterns or features from raw input data, enabling the rest of the model to perform tasks such as classification, translation, or generation. Depending on the input type, different architectures can be used as encoders.
By learning compact representations, encoders help reduce dimensionality, improve model efficiency, and facilitate generalization across different tasks.

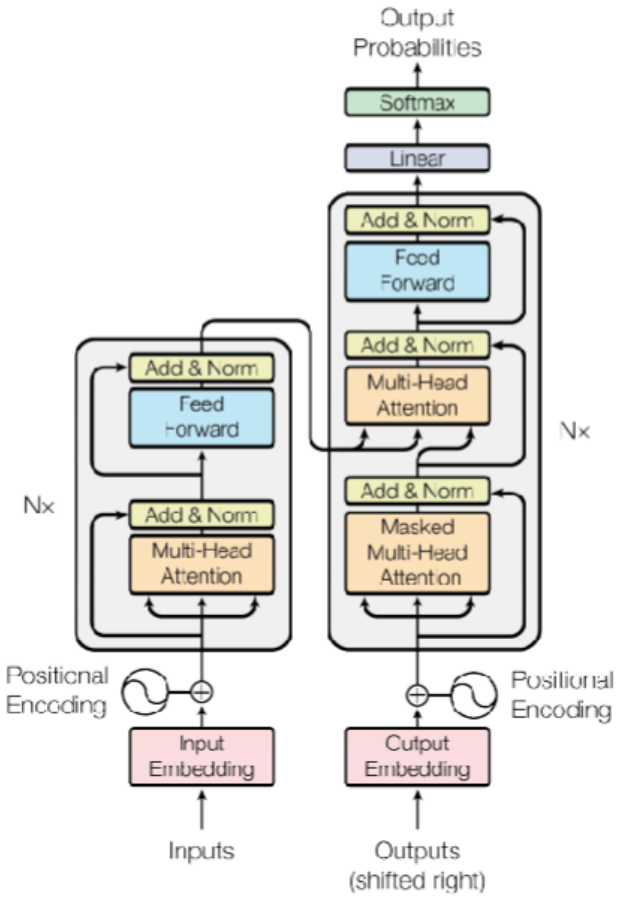
1.12. Vision Transformer

The Vision Transformer (ViT) is a model architecture that processes images using the same idea as transformers in language models. Instead of using filters like CNNs, ViT splits an image into small patches and treats them like a sequence of words.

Each image patch is turned into a vector (embedding), and a special token is added to represent the whole image. The model then adds information about the position of each patch and passes everything through transformer layers, which use attention to find relationships between patches.

At the end, the model uses the special token to understand the full image and make predictions. ViT works especially well when trained on large image datasets and is widely used as an encoder in many modern vision models.

Figure 1.7: ViT Architecture



Source: Fundamentals of Machine Learning (José Manuel Saavedra)

1.13. ResNet

ResNet, short for Residual Network, is an architecture that enables the construction of very deep neural networks while maintaining effective training. It achieves this through the use of residual blocks, which include skip connections that allow the model to bypass one or more layers. These shortcuts help mitigate the vanishing gradient problem, ensuring that deeper layers can still learn meaningful features without degrading the model's performance. There are different versions of ResNet based on the number of layers, the most common being ResNet18 and ResNet32.

By allowing information to flow directly through the network, even across many layers, ResNet enables the model to retain and refine features learned in earlier stages. This architecture is particularly well-suited for deep convolutional networks, where it consistently outperforms traditional models that lack residual connections, even without techniques like layer freezing or deactivation.

Figure 1.8: ResNet architecture for ImageNet

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
		3×3 max pool, stride 2				
conv2_x	56×56	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Source: Deep Residual Learning for Image Recognition

1.14. DINOv2

DINOv2 is built around a Vision Transformer (ViT) architecture, serving as a powerful image encoder in self-supervised learning. Unlike traditional convolutional encoders, DINOv2 relies on the transformer's ability to model long-range dependencies and global context through self-attention mechanisms.

The encoder processes input images by first dividing them into fixed-size patches, which are linearly embedded and combined with positional encodings. These patch embeddings are then passed through multiple transformer layers, each composed of multi-head self-attention and feedforward networks, allowing the model to capture hierarchical and semantic representations of the image.

A key characteristic of DINOv2's encoder is its use of a teacher-student framework, where the student encoder is trained to match the output of a slowly updated teacher encoder. This architecture helps the model learn stable and generalizable representations. The encoders share the same ViT backbone but operate under different augmentation views of the input, promoting consistency across varied perspectives.

1.15. CLIP

The CLIP (Contrastive Language–Image Pre-training) encoder is a neural network tool designed to transform images into rich, high-dimensional feature representations. It serves as the visual backbone in CLIP's dual-encoder architecture, which aligns visual and textual data in a shared embedding space.

CLIP's image encoder employs either a modified ResNet or a Vision Transformer (ViT) architecture, but in this case we will work with the ViT based one. The main difference between CLIP and the encoders mentioned before is that during training, it learns to project images into an embedding space where their representations are closely aligned with those of corresponding textual descriptions. This is achieved through a contrastive learning objective that maximizes the similarity between matching image-text pairs while minimizing it for non-matching pairs.

1.16. Precision

Precision is the metric describing how accurate the positive predictions of a model are, it will be used to evaluate the performance of the encoders for the three datasets, obtaining the mean average precision and building a precision-recall graph.

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives}$$

The graph shows the trade-off between the two metrics, precision and recall, this latter one represents how many positive predictions the model got right out of all positives.
The mean average precision tells how well the encoder ranks images, it is the average of the average precision for every class.

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AvgPrecision_i$$

2. Methodology

For this task, the libraries used were torch, torchvision, CLIP, scikit-image (skimage), NumPy, and Matplotlib.
Torch provides the core tools for building machine learning models, including the DINOv2 decoder used in this work. From torchvision, models like ResNet34 and ResNet18 are imported, along with various image transformations applied later during processing.
Scikit-image (skimage) is used for loading and transforming images, while the CLIP library provides the CLIP visual transformer model. NumPy handles data manipulation and facilitates integration with torch. Finally, Matplotlib is used to plot key metrics (such as the precision-recall curve) and to visualize image retrieval results for a given query.

The first step in image retrieval is to obtain feature vectors for every image in each dataset and for every encoder used. To extract these vectors, images are passed through the encoder models. This process is straightforward for DINOv2 and CLIP, as they directly output feature vectors.

However, ResNet models typically end with a dense classification layer. To repurpose them for feature extraction, this final layer is replaced with an identity layer, allowing the model's output to serve as the feature vector.

It's also important to note that when initializing the ResNet models, the pretrained parameter is set to True, ensuring that no additional training is required.

A script is then developed to extract feature vectors from a selected encoder for every image in a chosen dataset. The resulting vectors are stored as NumPy arrays, maintaining the same order as they were processed, and saved as files in NumPy's native format for later use.

With the feature vectors extracted, the similarity between any two images in a dataset (relative to a specific encoder) can be computed using cosine similarity. This metric is calculated between a query image and all other images, allowing the dataset to be sorted by similarity to the query.

To automate this, a script is created that loads the file containing all feature vectors as a NumPy array. Each vector (row) is normalized by dividing it by its own norm. Cosine similarity between every pair of images is then computed efficiently via the dot product of the normalized matrix with its own transpose, resulting in a similarity matrix.

To retrieve the most similar images for any given query, an index matrix is generated. For each feature vector (row), this matrix contains the indices of all vectors sorted from most to least similar, obtained by applying argsort to the similarity matrix rows.

Since the feature vectors are stored in the same order as the images in the dataset, these indices directly map to both the corresponding feature vectors and their associated images.

The top 10 most similar images for a given query can be visualized by locating its index in the index matrix and retrieving the images corresponding to the indices in that row.

To evaluate retrieval performance, the mean average precision (mAP) is computed for each combination of encoder and dataset. This involves first calculating the average precision (AP) for every image in the dataset. A script is developed to iterate through the index matrix: for each row (representing a query image), it checks whether the images at each indexed position share the same label as the query. When a relevant image is found, its precision is computed.

Once all relevant retrievals for a query are processed, the average precision for that image is calculated and stored. After processing the entire dataset, these values are averaged to obtain the mean average precision (mAP) for the selected encoder and dataset.
With the average precisions computed for all encoders and datasets, these values can be used to identify the best and worst retrieval results, where a higher average precision indicates better performance.

By applying argsort to the array of average precisions, a new array of indices is obtained, sorting the images from lowest to highest precision. The first five indices in this sorted array can then be used as queries to visualize the worst retrieval cases.

To retrieve the five best results, the precision values are multiplied by -1 before applying argsort, effectively sorting them in descending order.

To generate a precision-recall curve for each dataset and encoder, individual image precisions must be interpolated to align with recall values ranging from 0 to 1 in steps of 0.1. These interpolated values are then averaged to produce a single precision array for plotting.

During the computation of average precision for each image, the code also builds an interpolated precision list. For each recall step, it assigns the highest precision value available from retrievals where the recall is greater than or equal to the current step. This process results in a list of 11 precision values, one for each recall level.

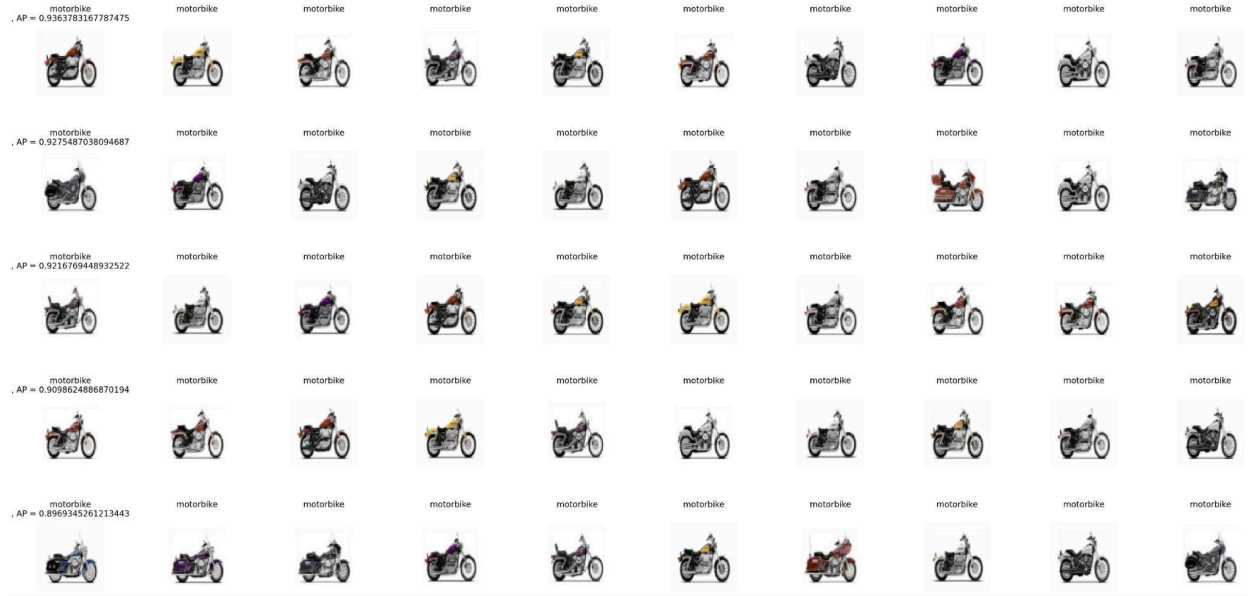
After processing all images, these lists are stacked into a NumPy array, and the mean precision across all images is calculated for each recall step. The resulting 1D array serves as the y-axis (precision) of the graph, while recall values from 0 to 1 (in 0.1 increments) form the x-axis.

3. Experimental Results

3.1 Simple1k Dataset

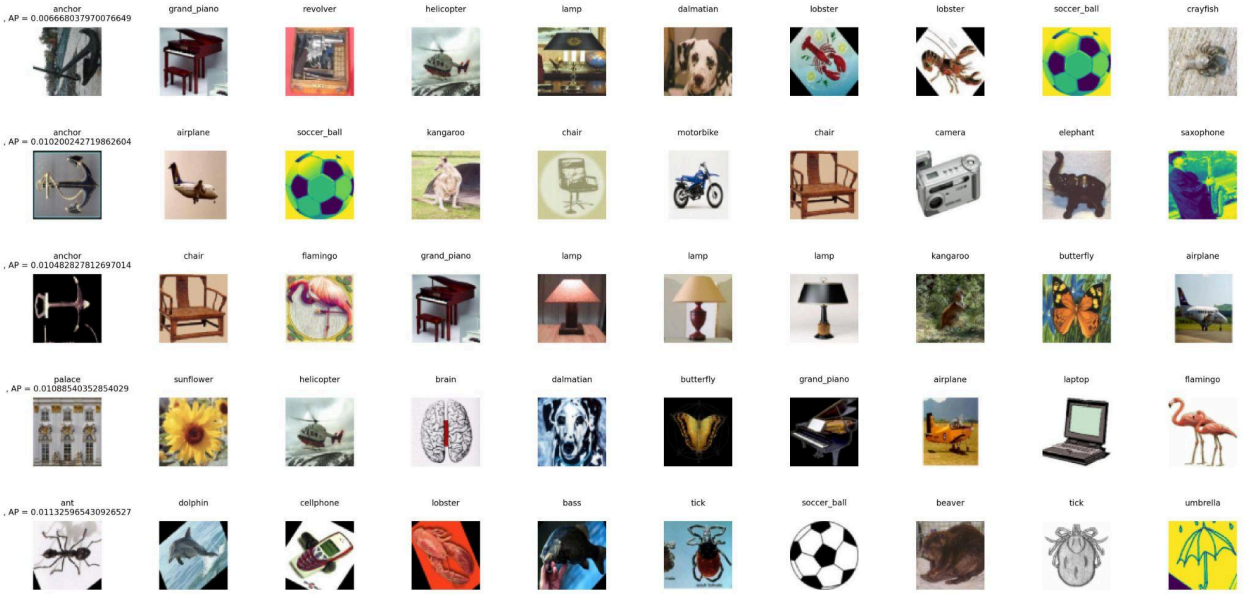
3.1.1 ResNet18

Figure 3.1: Five Best Retrievals Simple1K - ResNet_18



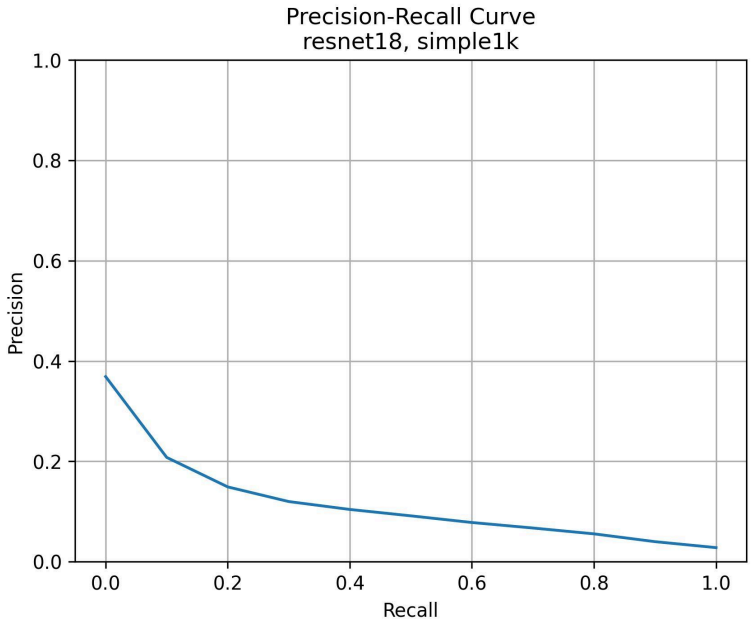
Source: Self-produced

Figure 3.2: Five Worst Retrievals Simple1K - ResNet_18



Source: Self-produced

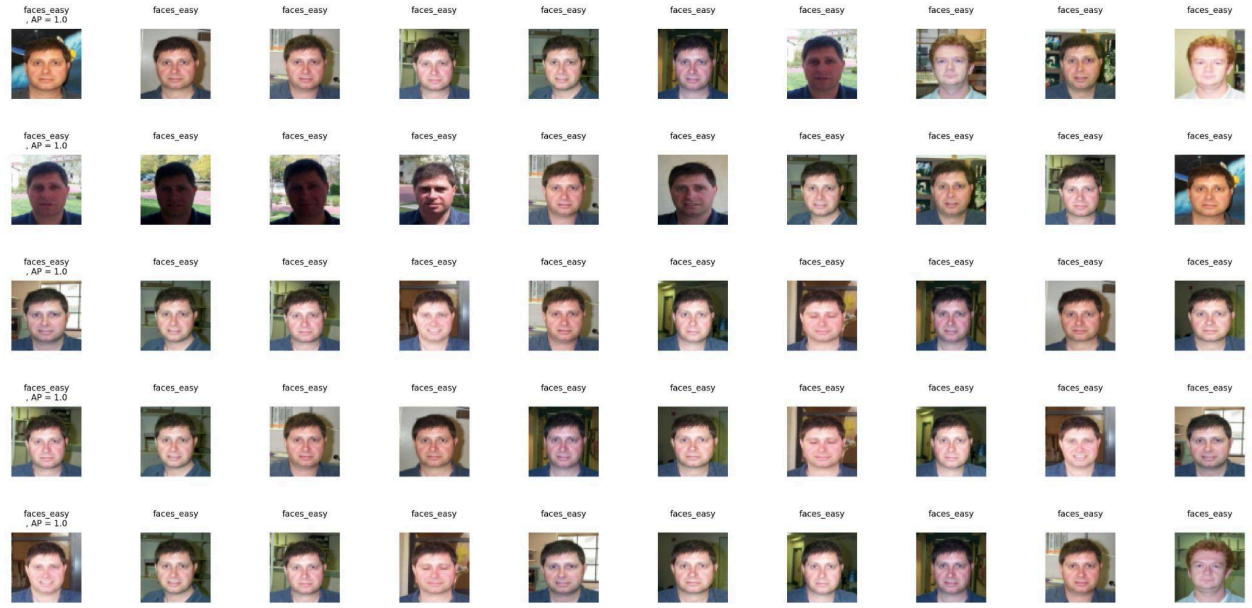
Figure 3.3: Precision - Recall Curve Simple1K - ResNet_18



Source: Self-produced

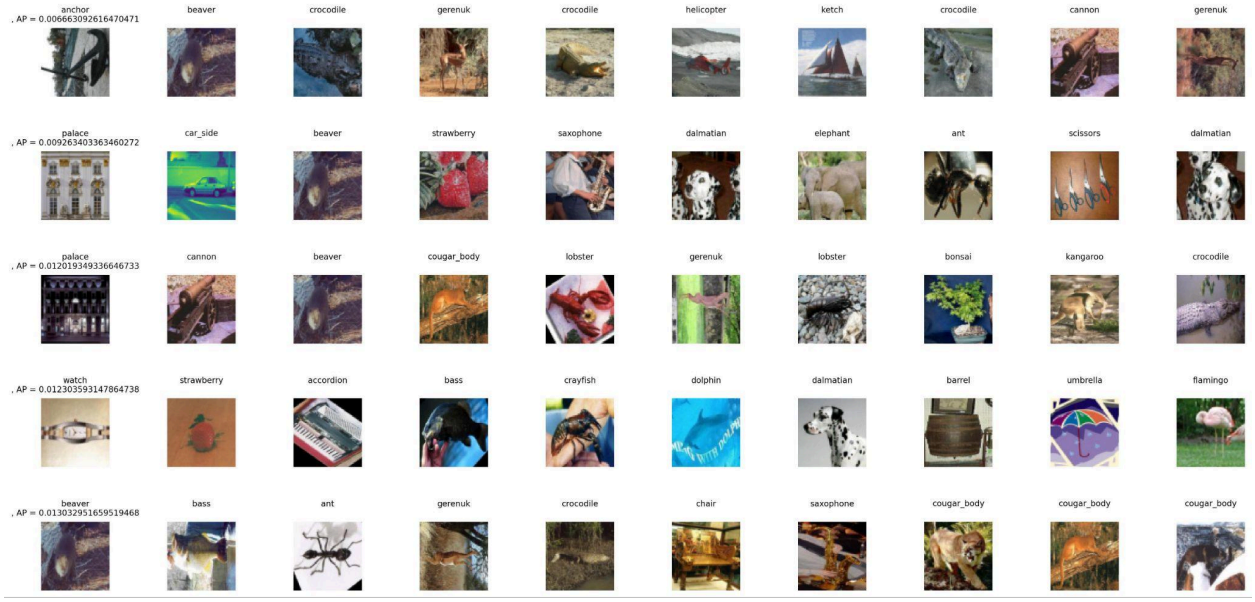
3.1.2 ResNet34

Figure 3.4: Five Best Retrievals Simple1K - ResNet_34



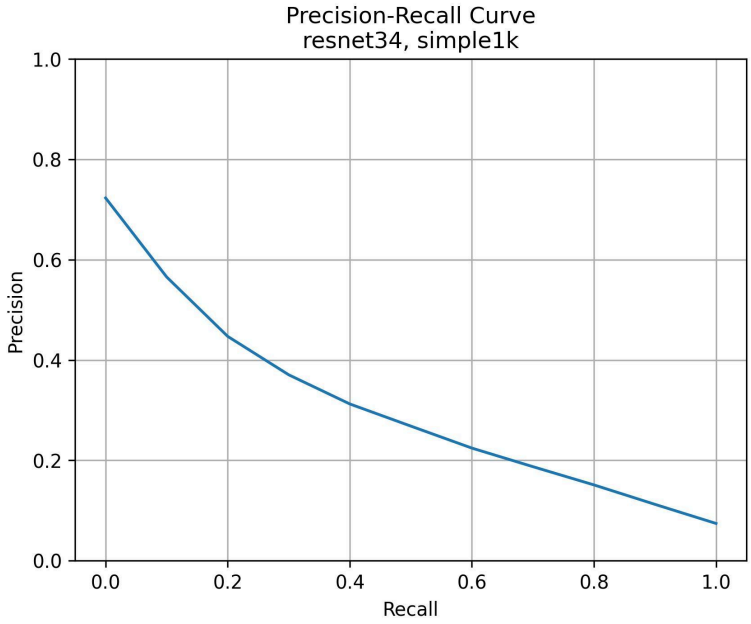
Source: Self-produced

Figure 3.5: Five Worst Retrievals Simple1K - ResNet_34



Source: Self-produced

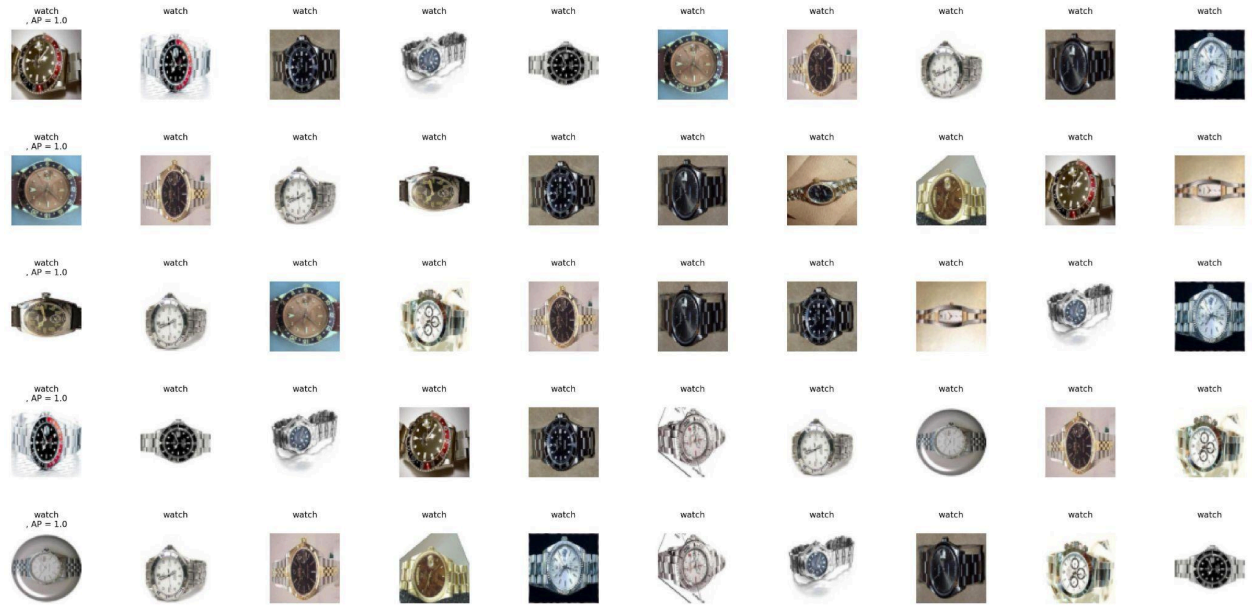
Figure 3.6: Precision - Recall Curve Simple1K - ResNet_34



Source: Self-produced

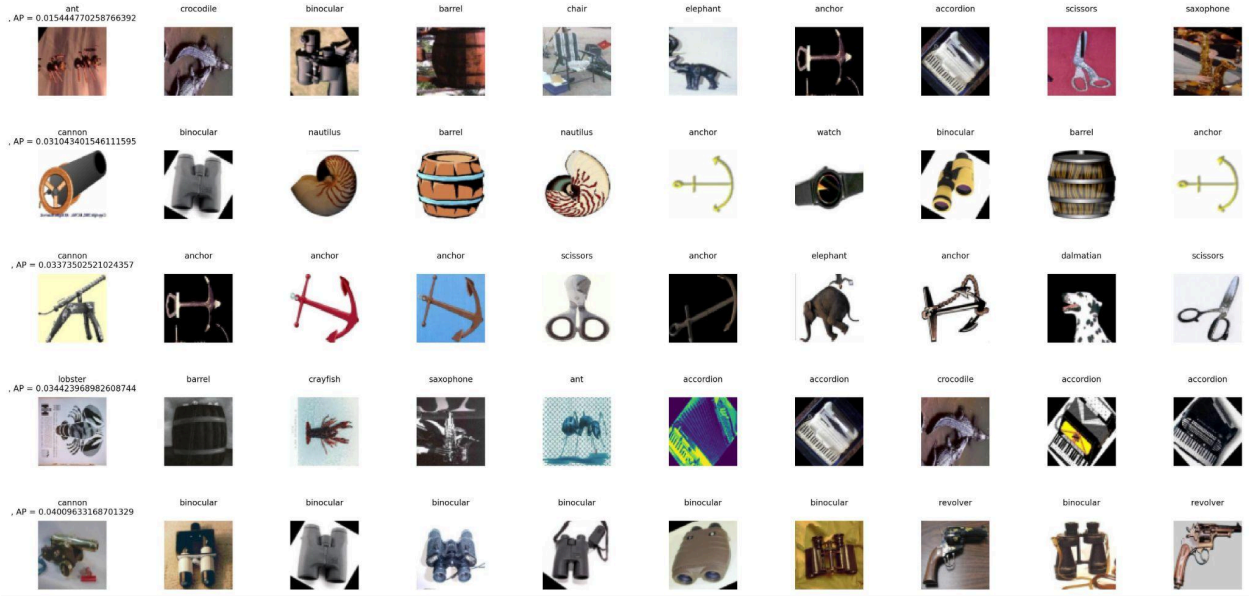
3.1.3 CLIP

Figure 3.7: Five Best Retrievals Simple1K - CLIP



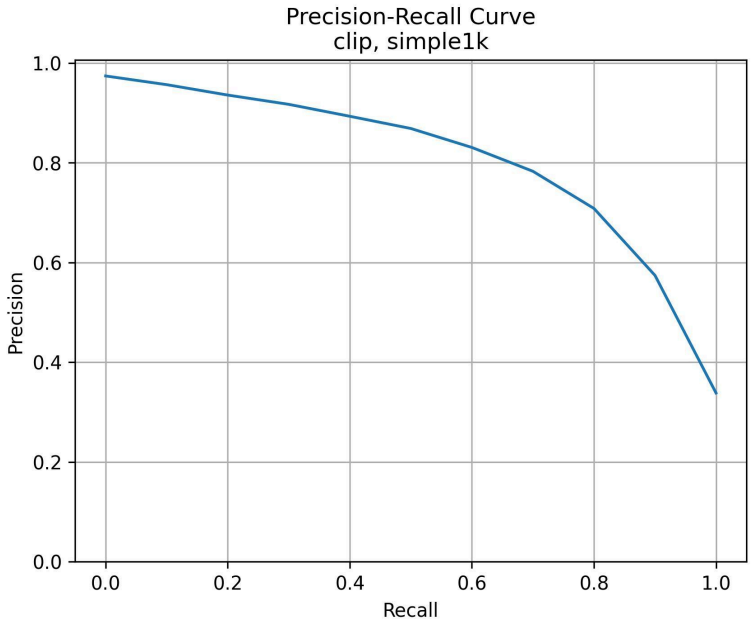
Source: Self-produced

Figure 3.8: Five Worst Retrievals Simple1K - CLIP



Source: Self-produced

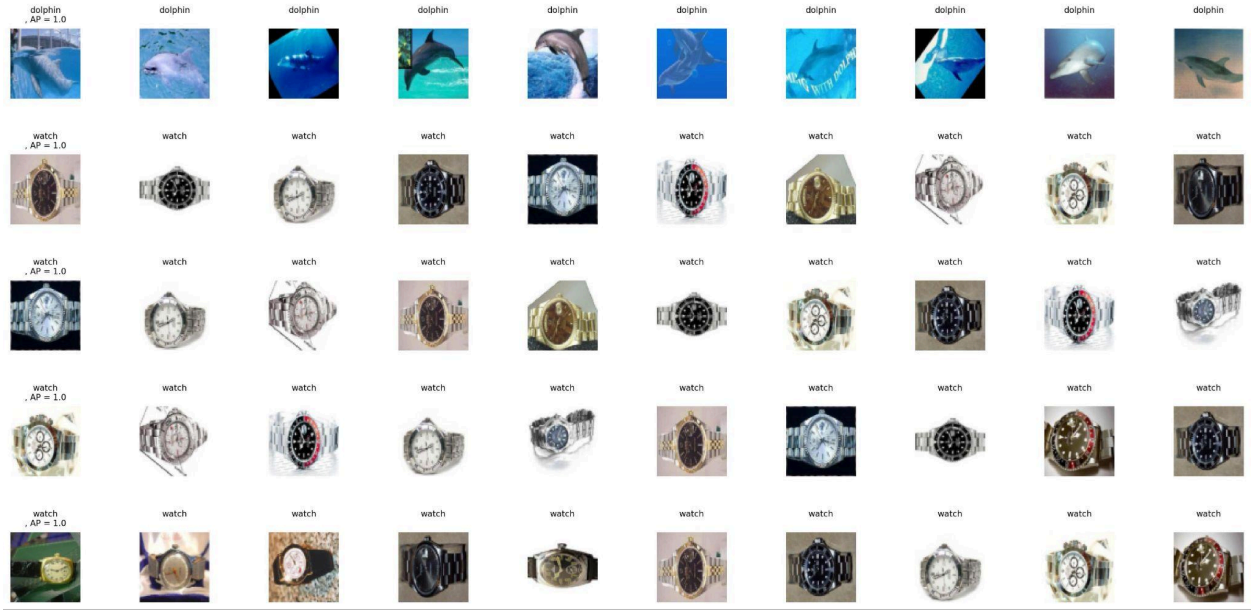
Figure 3.9: Precision - Recall Curve Simple1K - CLIP



Source: Self-produced

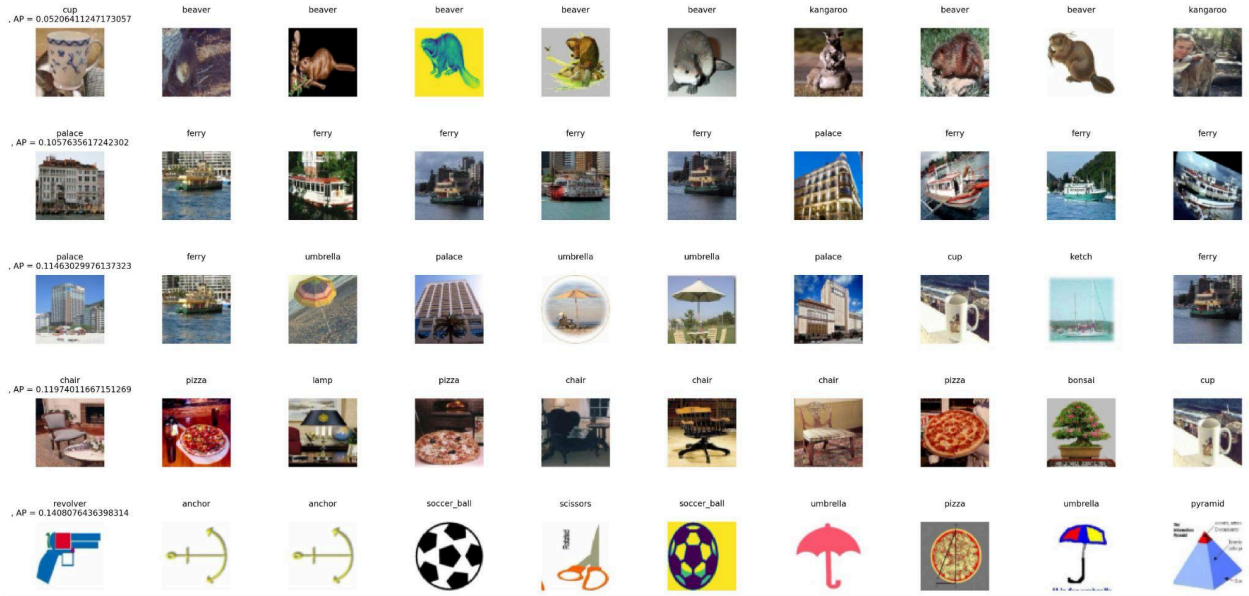
3.1.4 DINOv2

Figure 3.10: Five Best Retrievals Simple1K - DINOv2



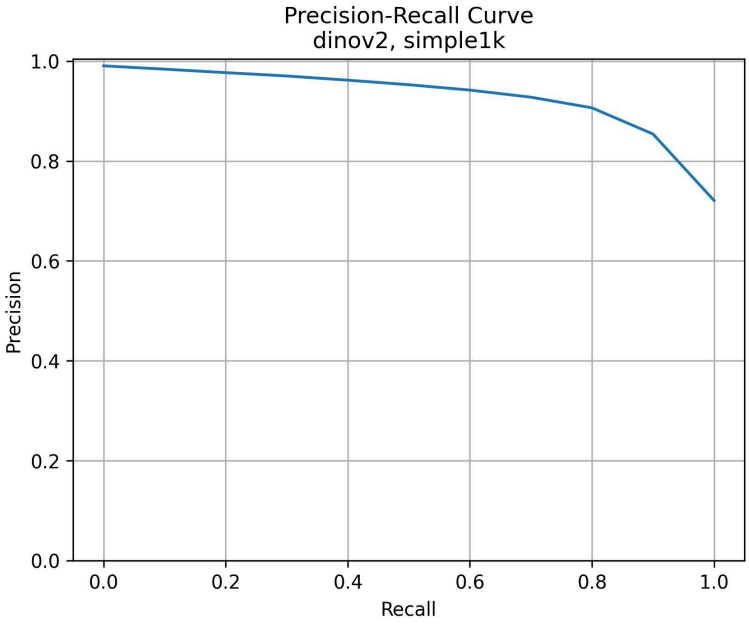
Source: Self-produced

Figure 3.11: Five Worst Retrievals Simple1K - DINOv2



Source: Self-produced

Figure x: Precision - Recall Curve Simple1K - DINOv2

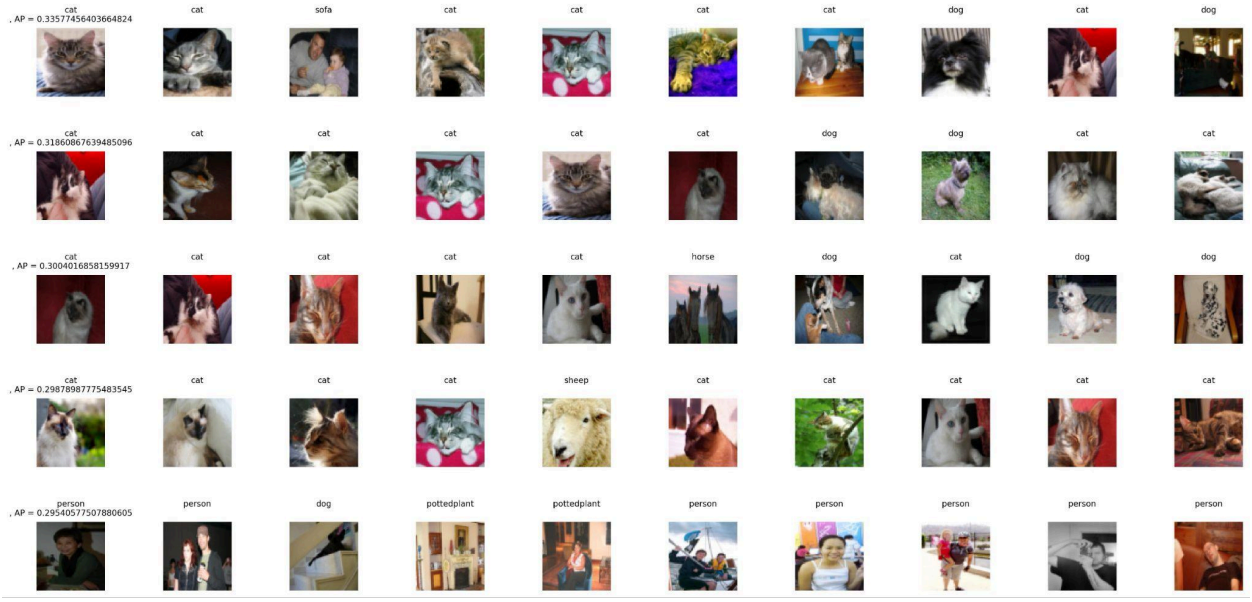


Source: Self-produced

3.2 VOC_val Dataset

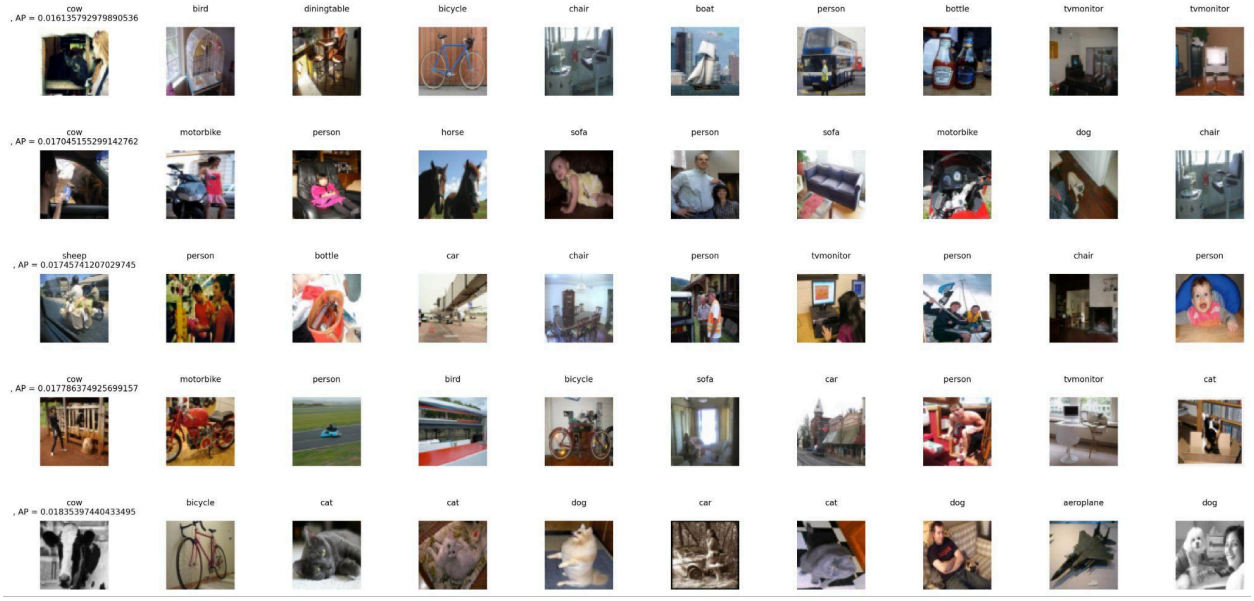
3.2.1 ResNet18

Figure 3.12: Five Best Retrievals VOC_val - ResNet_18



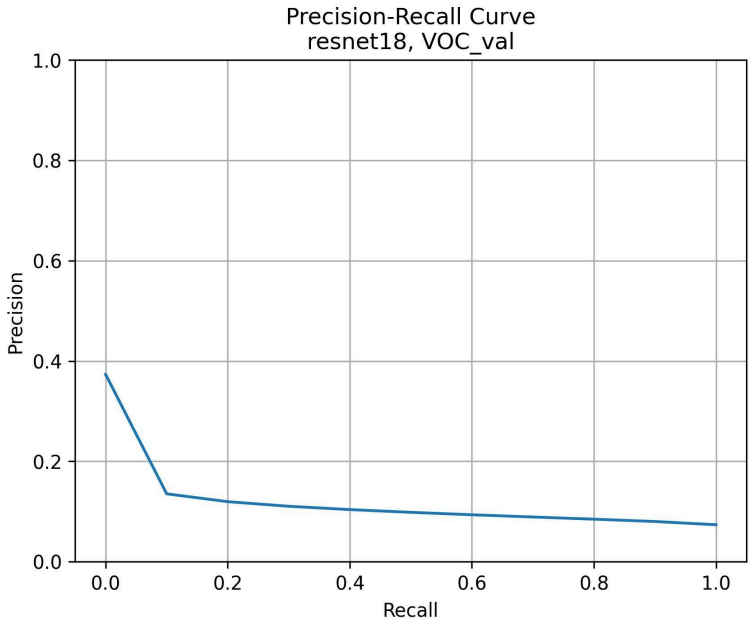
Source: Self-produced

Figure 3.13: Five Worst Retrievals VOC_val - ResNet_18



Source: Self-produced

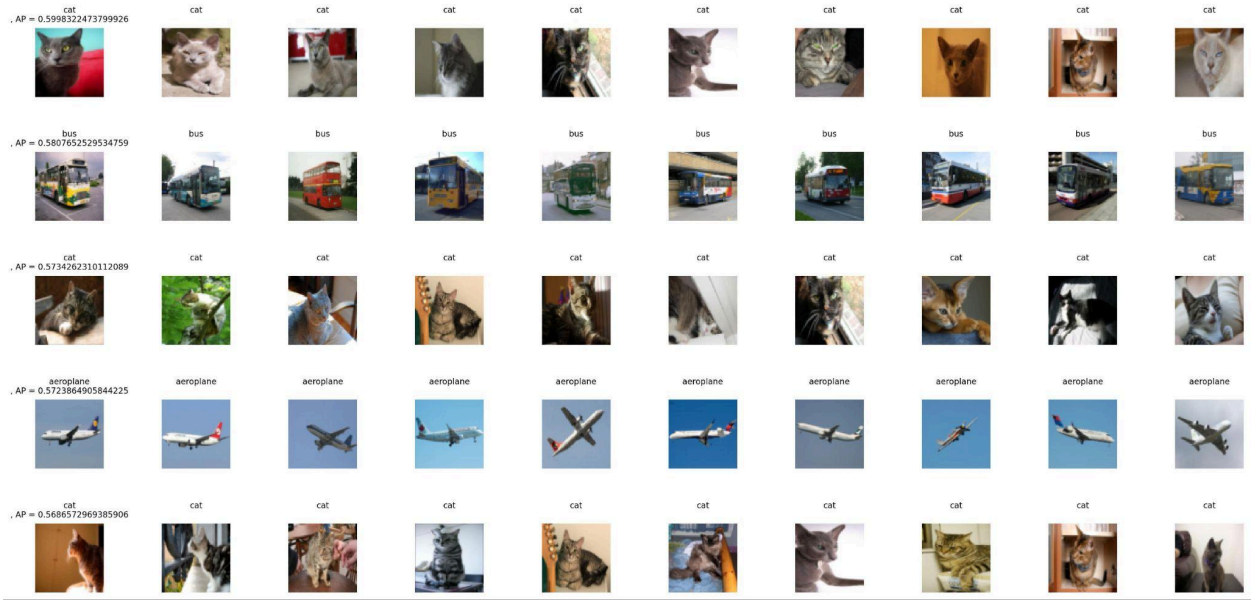
Figure 3.14: Precision - Recall Curve VOC_val - ResNet_18



Source: Self-produced

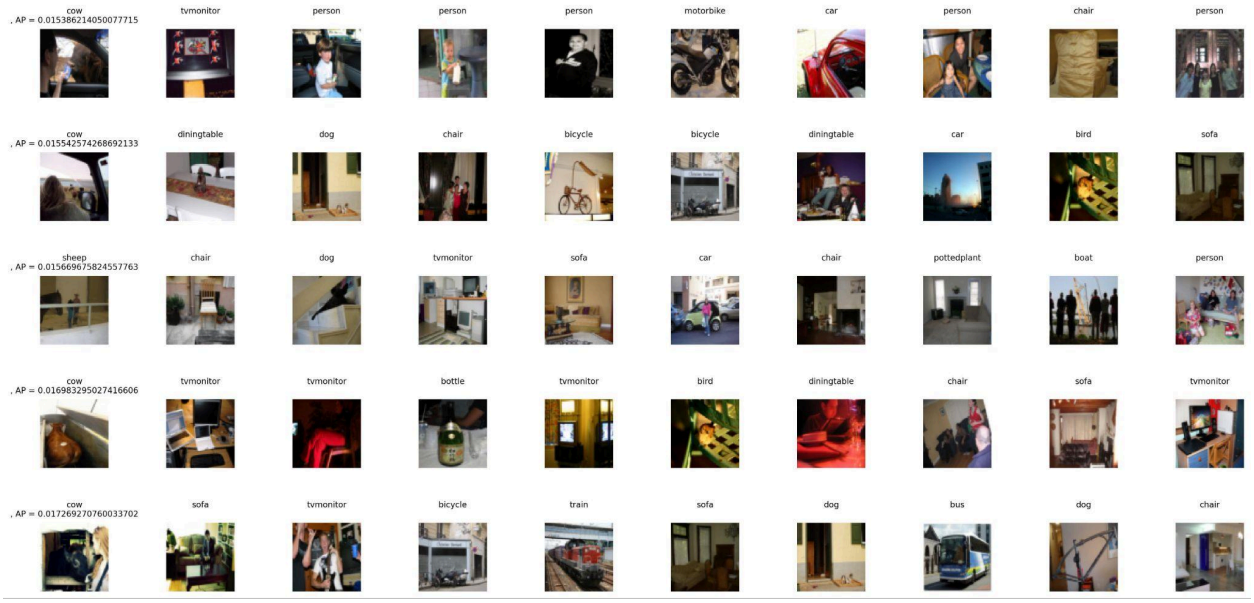
3.2.2 ResNet34

Figure 3.15: Five Best Retrievals VOC_val - ResNet_34



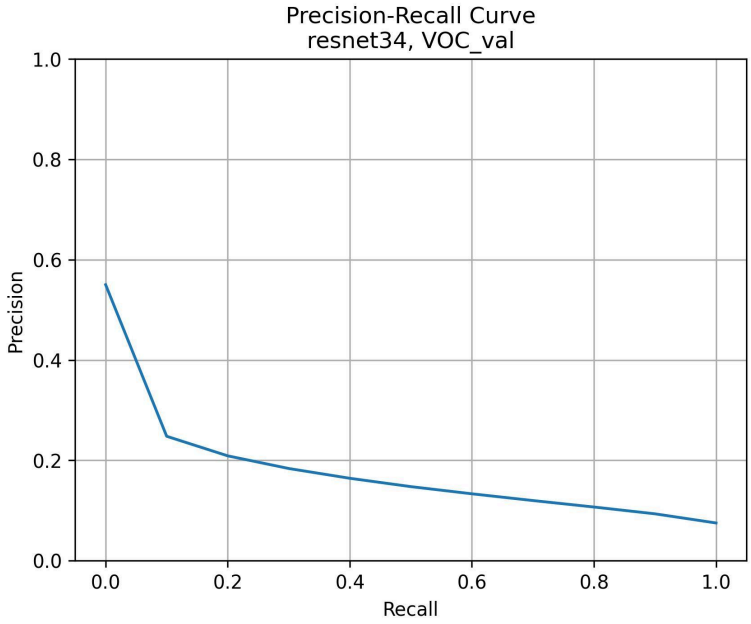
Source: Self-produced

Figure 3.16: Five Worst Retrievals VOC_val - ResNet_34



Source: Self-produced

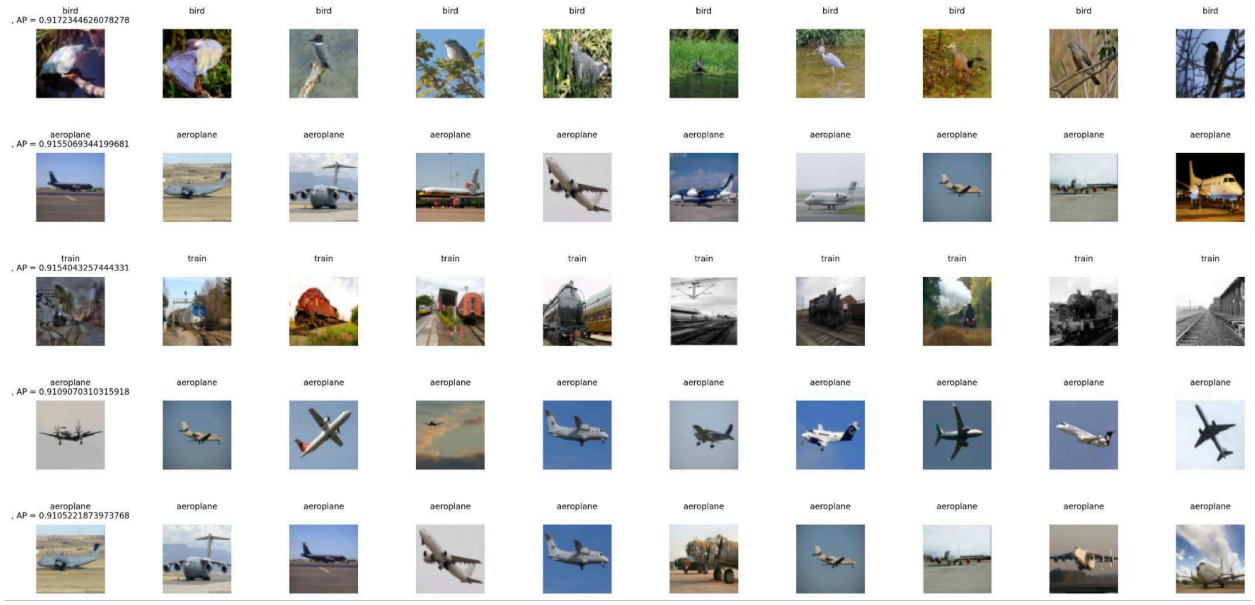
Figure 3.17: Precision - Recall Curve VOC_val - ResNet_34



Source: Self-produced

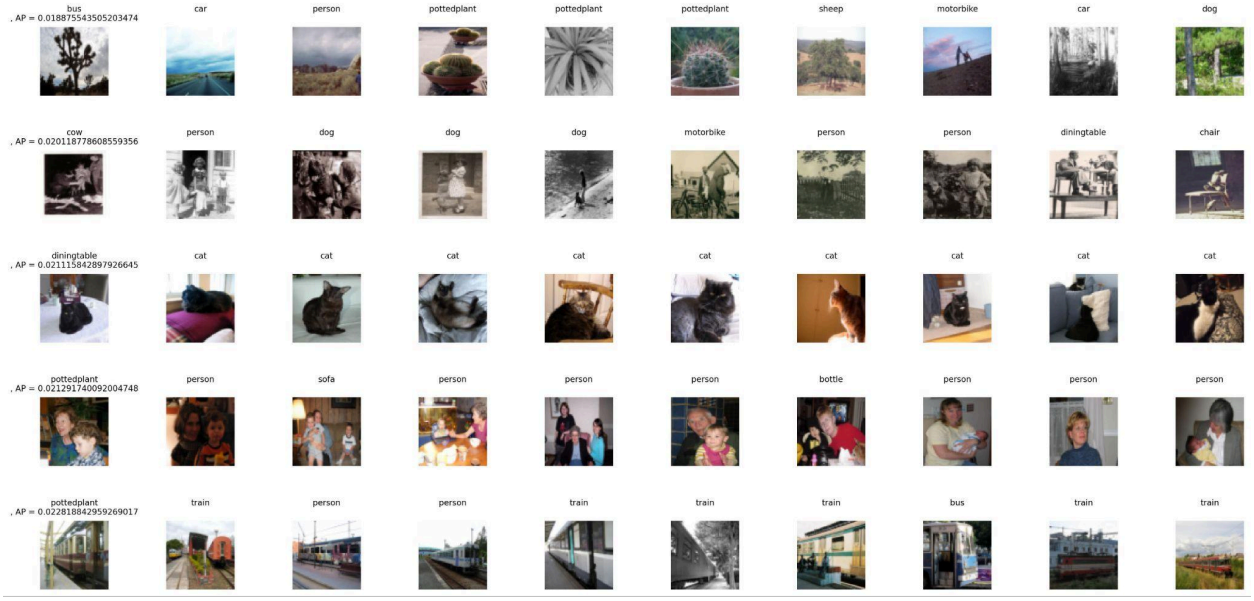
3.2.3 CLIP

Figure 3.18: Five Best Retrievals VOC_val - CLIP



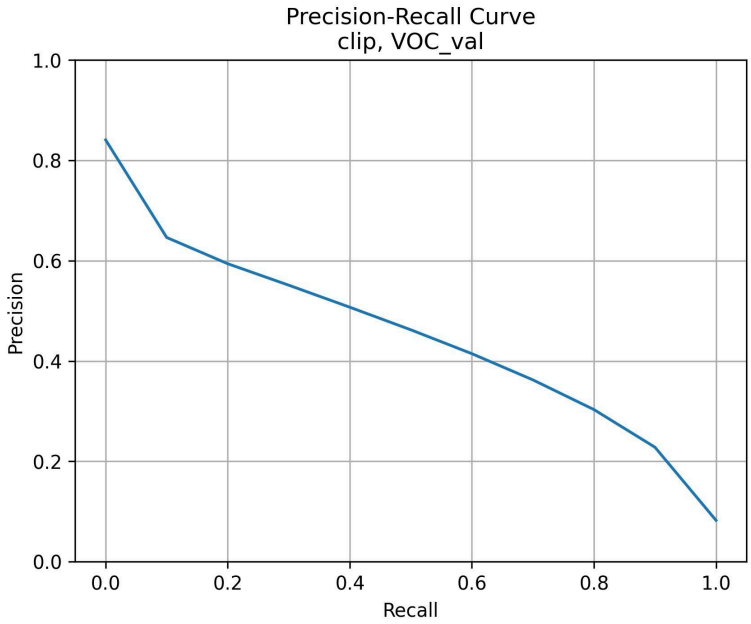
Source: Self-produced

Figure 3.19: Five Worst Retrievals VOC_val - CLIP



Source: Self-produced

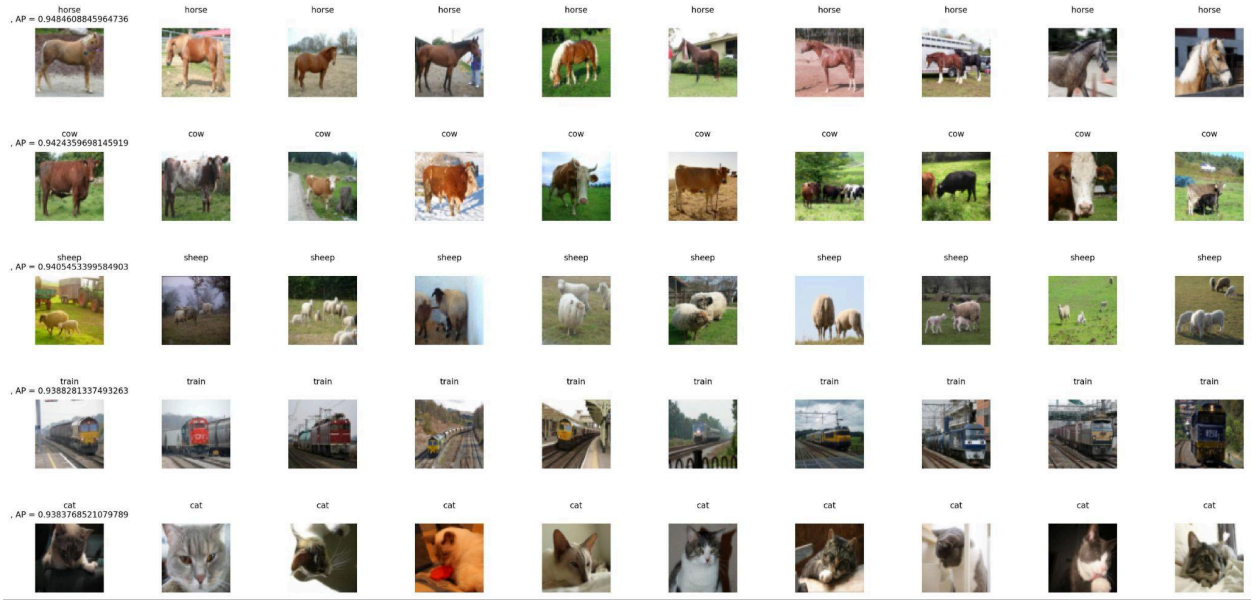
Figure 3.20: Precision - Recall Curve VOC_val - CLIP



Source: Self-produced

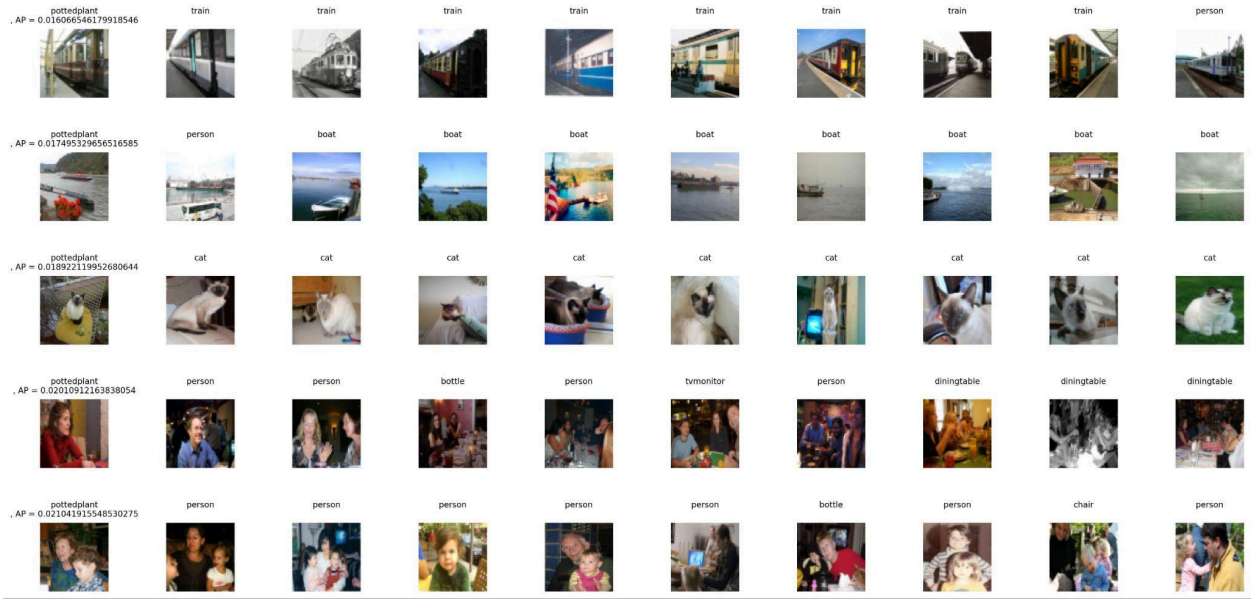
3.2.4 DINOv2

Figure 3.21: Five Best Retrievals VOC_val - DINOv2



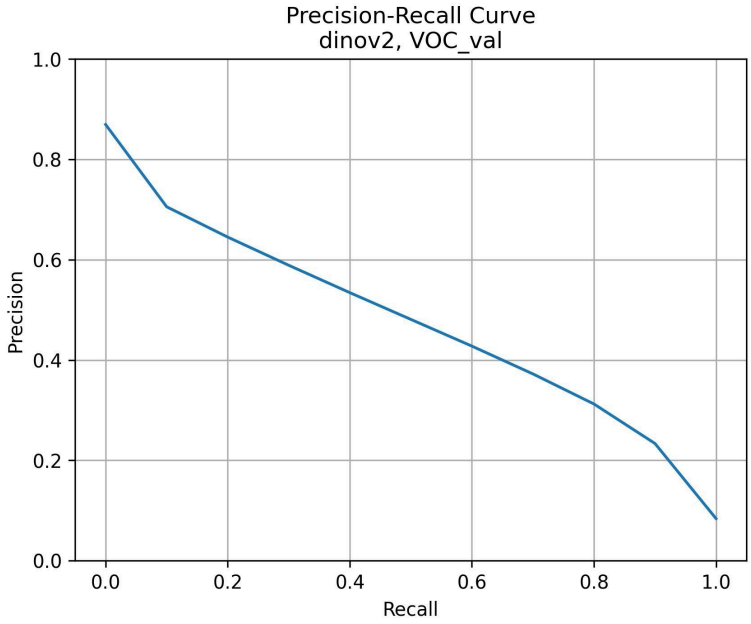
Source: Self-produced

Figure 3.22: Five Worst Retrievals VOC_val - DINOv2



Source: Self-produced

Figure 3.23: Precision - Recall Curve VOC_val - DINOv2

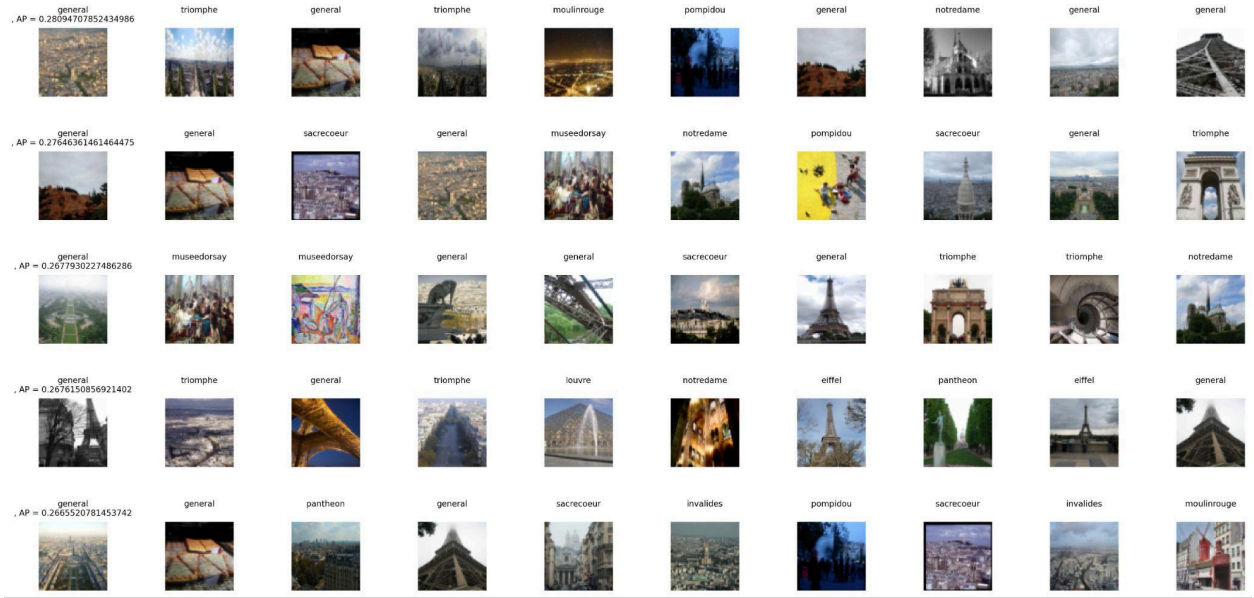


Source: Self-produced

3.3 Paris Dataset

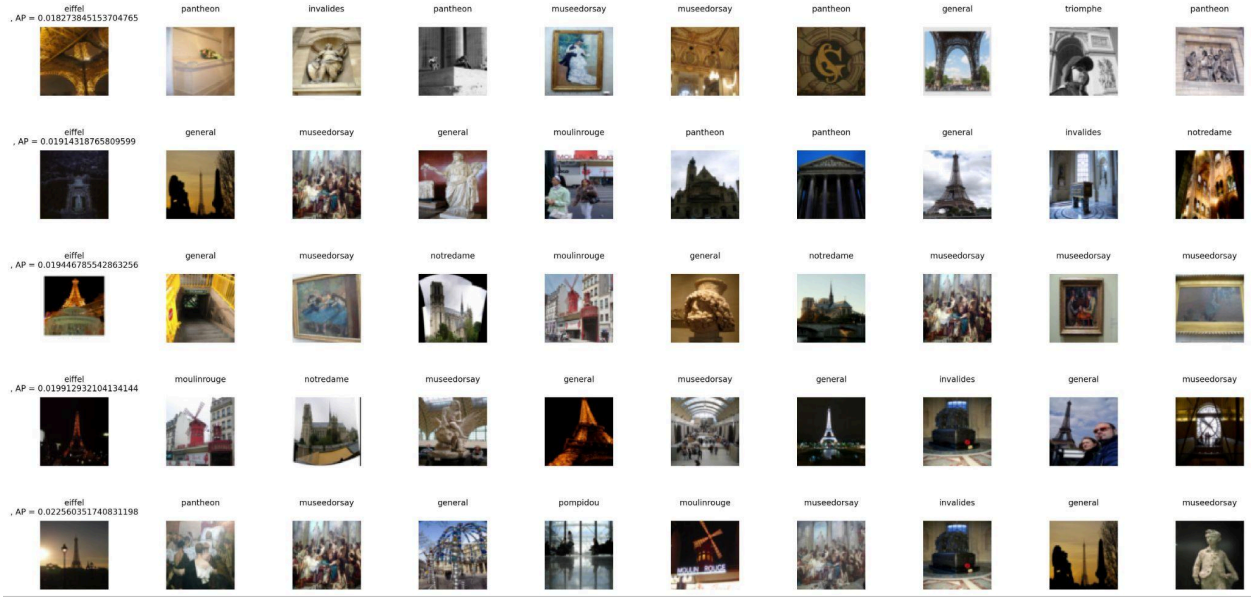
3.3.1 ResNet18

Figure 3.24: Five Best Retrievals Paris - ResNet_18



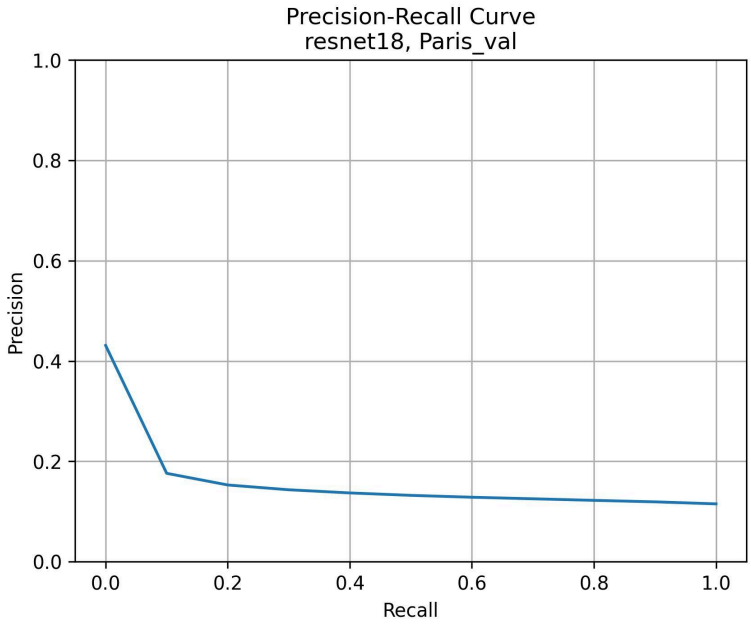
Source: Self-produced

Figure 3.25: Five Worst Retrievals Paris - ResNet_18



Source: Self-produced

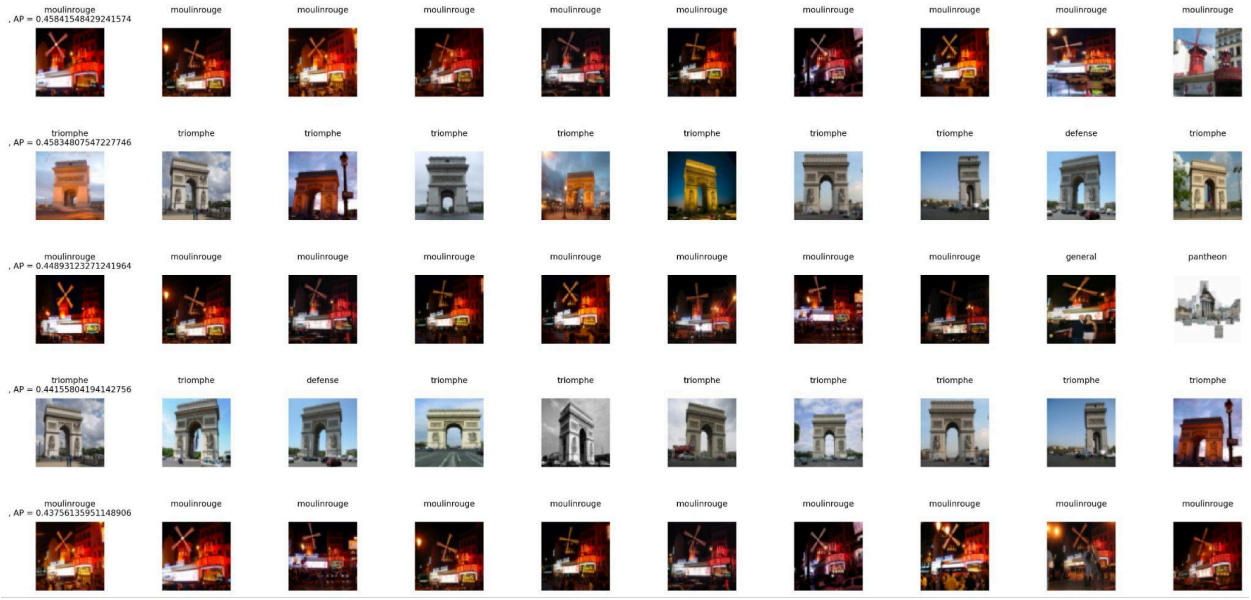
Figure 3.26: Precision - Recall Curve Paris - ResNet_18



Source: Self-produced

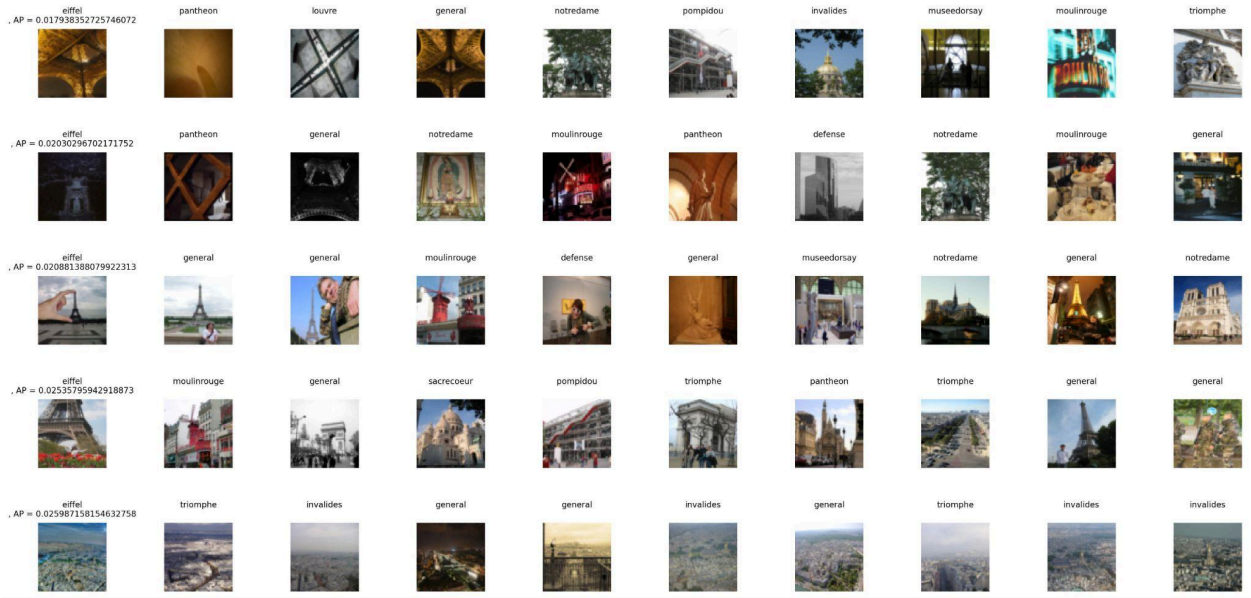
3.3.2 ResNet34

Figure 3.27: Five Best Retrievals Paris - ResNet_34



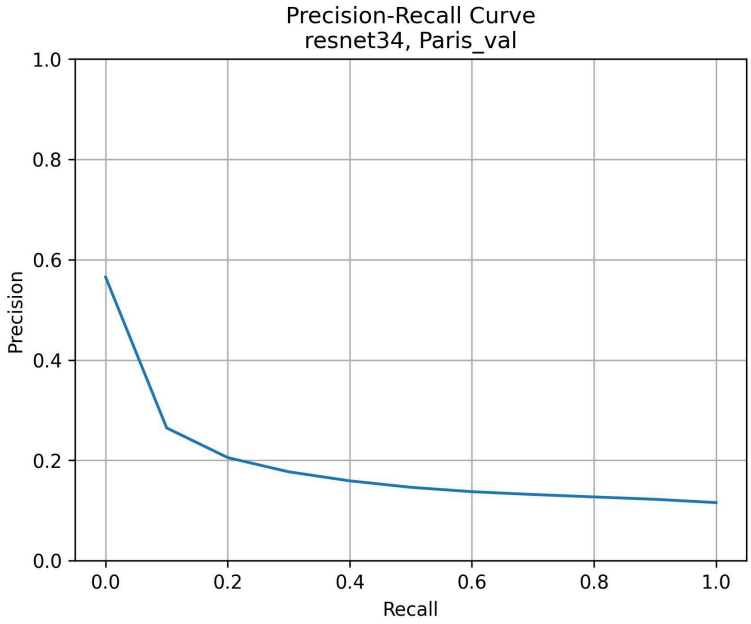
Source: Self-produced

Figure 3.28: Five Worst Retrievals Paris - ResNet_34



Source: Self-produced

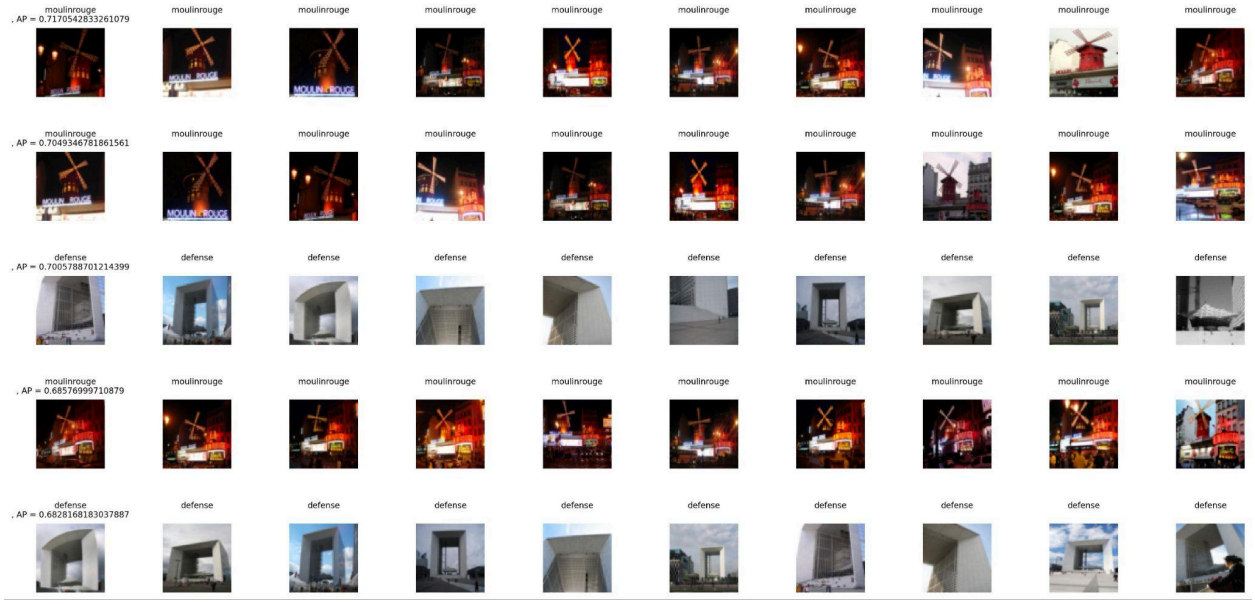
Figure 3.29: Precision - Recall Curve Paris - ResNet_34



Source: Self-produced

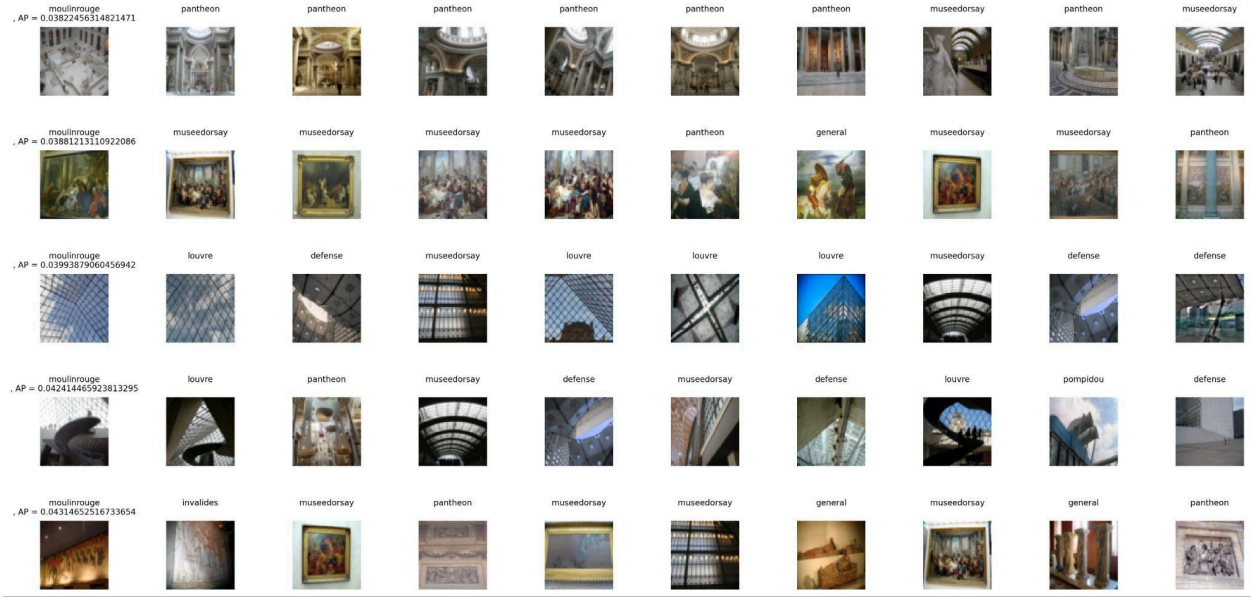
3.3.3 CLIP

Figure 3.30: Five Best Retrievals Paris - CLIP



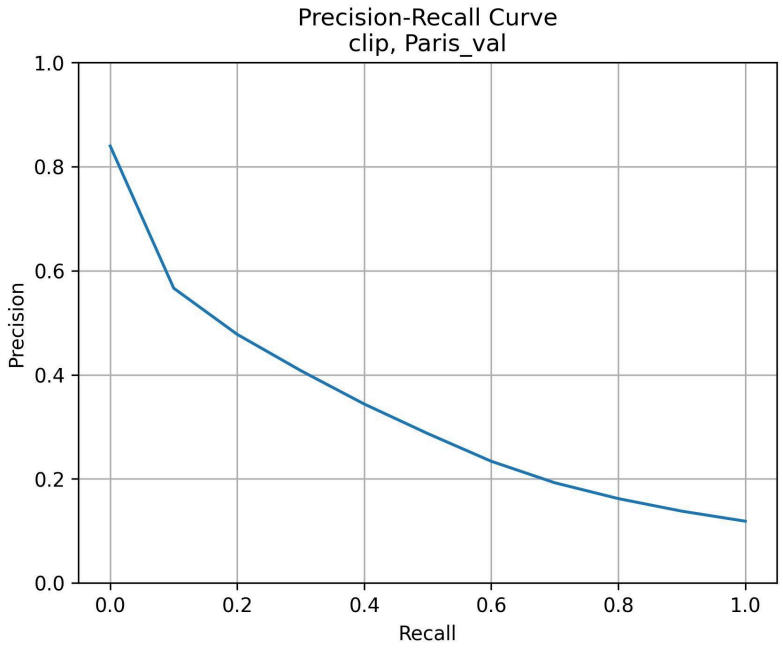
Source: Self-produced

Figure 3.31: Five Worst Retrievals Paris - CLIP



Source: Self-produced

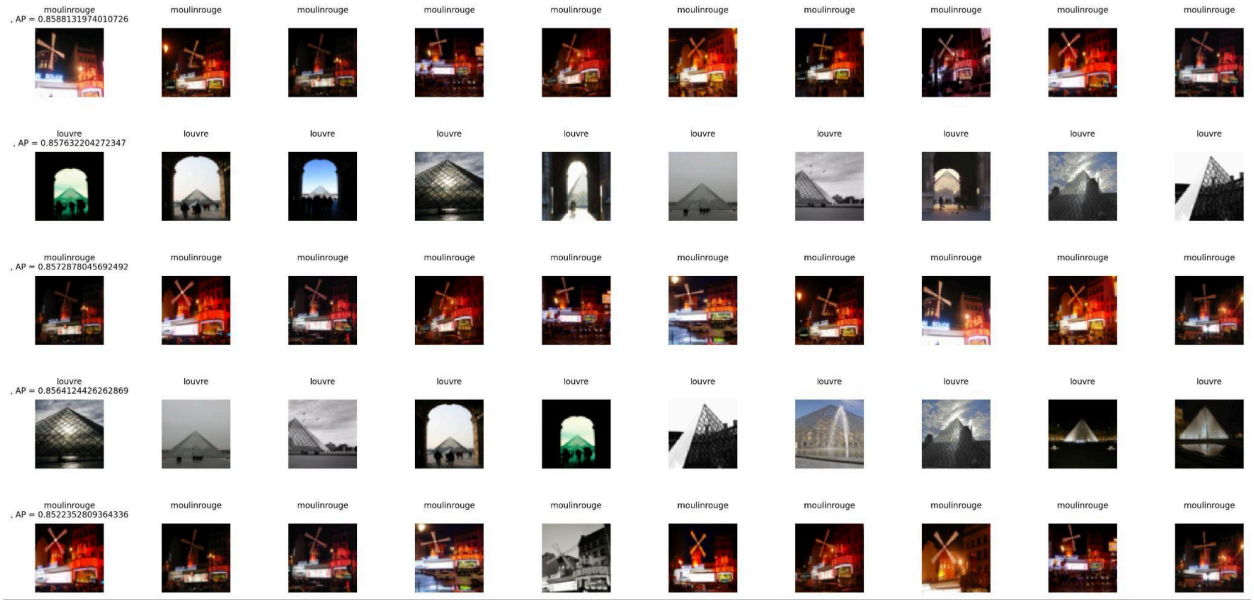
Figure 3.32: Precision - Recall Curve Paris - CLIP



Source: Self-produced

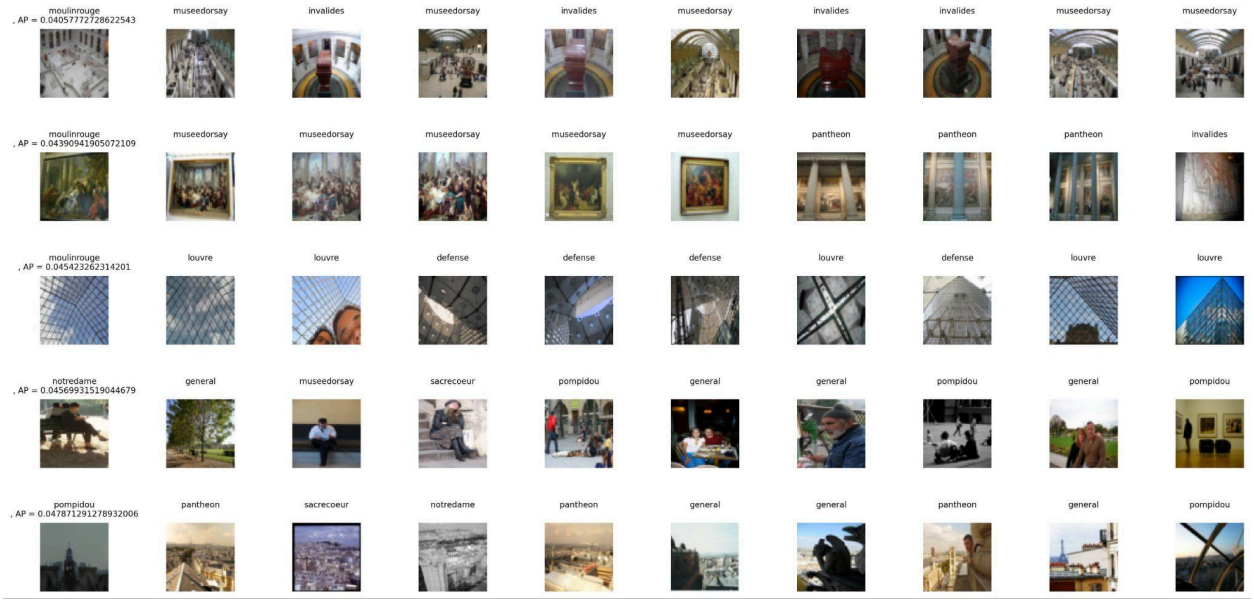
3.3.4 DINOv2

Figure 3.33: Five Best Retrievals Paris - DINOv2



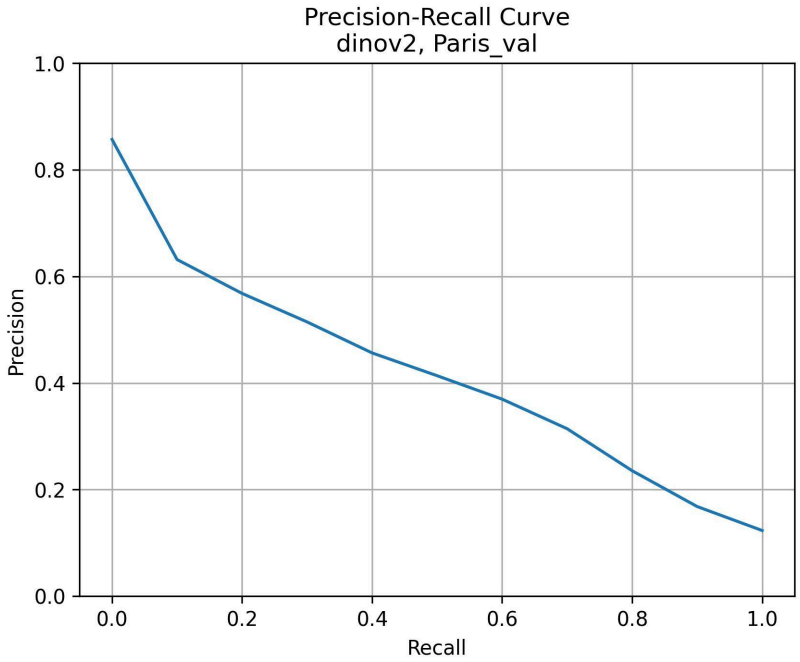
Source: Self-produced

Figure 3.34: Five Worst Retrievals Paris - DINOv2



Source: Self-produced

Figure 3.35: Precision - Recall Curve Paris - DINOv2



Source: Self-produced

3.4. Mean Average Precision

In the table shown below it is possible to see the mAP of every encoder with every dataset, where the last column has the relative performance compared to the model with the best performance (DINOv2)

Model	Simple1K	VOC_val	Paris	Comparative Performance
ResNet_18	0.1053	0.1018	0.1336	19.283%
ResNet_34	0.2945	0.1586	0.1661	34.4%
CLIP	0.8082	0.4447	0.3134	87.04%
DINOv2	0.9310	0.4695	0.3990	100%

4. Discussion

4.1 ResNet18

This version of the ResNet model consistently shows the poorest performance across all datasets, as evidenced by its low average precision scores and the corresponding precision-recall curves.

Analysis of the best and worst retrievals reveals that ResNet performs adequately on simpler images, such as those in the Simple1K dataset. Its best retrievals involve images with structures closely matching the query (similar angles, shapes, and backgrounds). This suggests that even slight variations in these factors could significantly reduce the model's ability to generate effective feature vectors for images of the same label.

For more complex datasets, like VOC_val, the model struggles. Although some correct retrievals are observed (for example, images of cats), irrelevant images (such as people) still appear among the top results. This indicates difficulty in capturing nuanced features of more complex objects.

Similarly, in the Paris dataset, the retrieved images often share only basic elements, such as the presence of the sky or a similar viewing angle, failing to capture finer details of architectural features. This further highlights the model's limitations in representing complex visual patterns for retrieval tasks.

4.2 ResNet34

This ResNet variant demonstrates better performance than the 18-layer architecture, though it still falls short of models like CLIP and DINOv2. The improvement is evident in its ability to generate feature vectors that remain consistent across images of the same object or structure, even when variations in angle, background, or lighting are present.

The most notable results appear in the Simple1K and Paris datasets. In Simple1K, the model successfully retrieves images of the same person despite minor alterations such as shading differences or background changes. For the Paris dataset, it accurately retrieves images of the same structure (or visually similar ones) from different angles and with varying tones.

While its worst retrievals still include some mismatches, they are not as severe as those from ResNet-18. The retrieved images often share similar color schemes or structural elements, even if they belong to different object categories, indicating a moderate level of feature generalization.

4.3 CLIP

CLIP ranks as the second-best performing model overall, showing a significant improvement over the previous architectures. Its average precision (AP) sees a substantial increase, particularly on the Simple1K dataset, where AP jumps from around 0.3 to 0.8.

This improvement is also reflected in the precision-recall curves, which now exhibit a higher initial precision at recall = 0.0 and a much slower decline as recall increases. Unlike earlier models, where the curves started low and dropped steadily, CLIP maintains higher precision across more of the recall range.

The retrieval results highlight CLIP's capability to consistently identify images of the same object type or structure, even in more challenging cases, such as retrieving images of birds in the VOC_val dataset. Its performance remains strong across datasets, successfully retrieving watches in Simple1K and correctly identifying architectural structures in Paris, despite variations in angle and distance. However, CLIP’s good performance on the Paris dataset also reveals a challenge: while some retrieved images are visually very similar, label discrepancies in the dataset can negatively affect precision scores. As a result, even highly similar images may be considered irrelevant, leading to a lower AP despite the retrievals being visually correct.

4.4 DINOv2

DINOv2 stands out as the best-performing model, achieving the highest average precision (AP) across all datasets. Its strongest results are seen on the Simple1K dataset, reaching an AP of 0.9, with an exceptional precision-recall curve that maintains precision between 0.8 and 0.9 across most recall levels, only dropping slightly at the end.

DINOv2 consistently retrieves correct images, even for complex queries involving multiple animals in datasets like VOC_val. It also performs very well on Paris_val and Simple1K, often retrieving the same images that previous models identified as their best cases. This suggests

that, for more advanced models, these particular images are inherently easier to represent and retrieve accurately. However, DINOv2 faces the same challenge as CLIP regarding label inconsistencies in datasets like Paris and VOC_val, where visually correct retrievals are penalized due to mismatched labels.

An unusual case appears in Simple1K's worst retrievals, where the retrieved images are clearly different from the query. Given DINOv2 otherwise strong performance, this likely reflects a limitation in the dataset itself, where few, if any, relevant images are available for those specific queries, rather than a failure of the model.

5. Conclusions

The use of encoders enables rapid machine learning implementations, especially when utilizing pre-trained models. These pre-trained encoders significantly outperform non-pretrained ones, eliminating the need for time-consuming training and validation datasets. This not only shortens development time but also reduces costs, as curating a suitable dataset is often resource-intensive.

All four encoders tested successfully extracted the relevant image features to retrieve images similar to the input. However, their performance varied considerably, as expected due to the differences in their underlying architectures. Each model has its pros and cons, which are usually a trade-off between precision and architecture complexity and training time.

ResNet18 delivered the lowest performance among the encoders. While it was capable of retrieving relevant images in some cases, its precision-recall curve reveals a steep drop in recall, indicating inconsistent retrieval as recall increases. In contrast, ResNet32, a deeper model, performed approximately 50% better in terms of precision.

DINOv2 and CLIP both showed significantly higher performance compared to the ResNet models. Based on the mean average precision (mAP), DINOv2 outperformed CLIP by about 13%. Notably, both models were at least four times better than ResNet18 and at least twice as effective as ResNet32, with DINOv2 reaching up to three times the performance of the latter. When comparing DINOv2 and CLIP, their precision-recall curves suggest that CLIP’s precision degrades more rapidly as recall increases, although both start from a similar precision level. Their strong performance can likely be attributed to their shared Vision Transformer architecture, with DINOv2’s self-supervised training approach appearing to provide an edge over CLIP.

In conclusion, pre-trained Vision Transformer-based encoders, particularly DINOv2, offer a highly effective solution for image retrieval tasks based on similarity, outperforming traditional convolutional encoders like ResNet. Their superior performance, combined with the ease of deployment, makes them excellent candidates for practical, cost-efficient image retrieval systems.