AI VIETNAM
All-in-One Course
(TA Session)

Image Retrieval

Project

Dinh-Thang Duong – TA

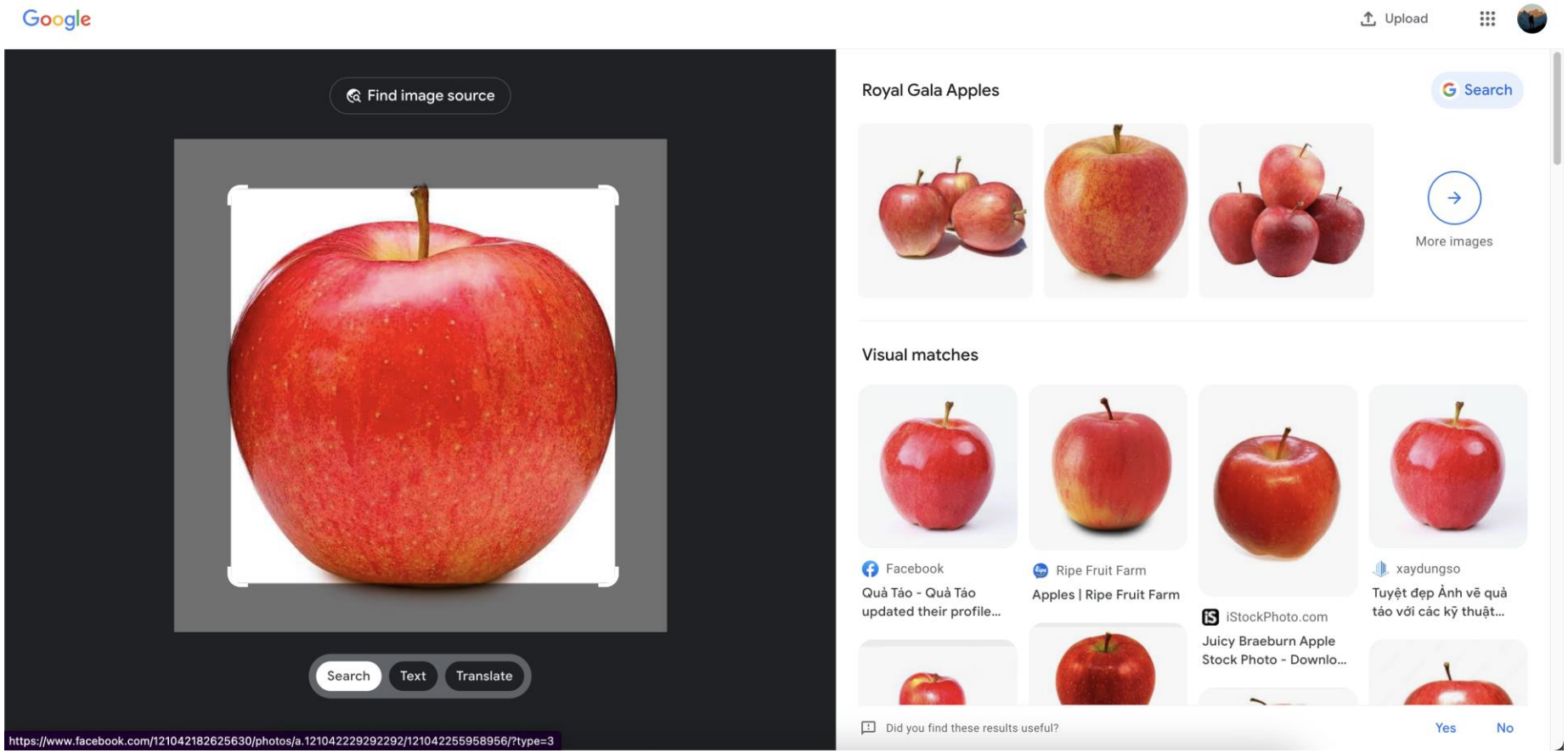
Year 2023

Outline

- Introduction
- Baseline
- Apply pre-trained ViT
- Question

Introduction

❖ Getting Started



Introduction

❖ Getting Started

Input Image

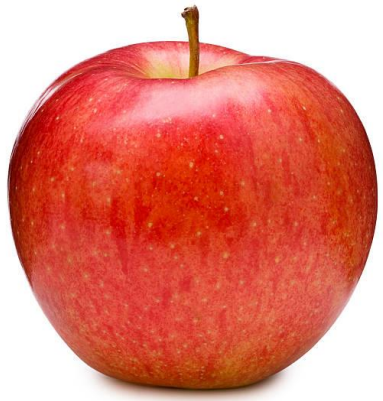
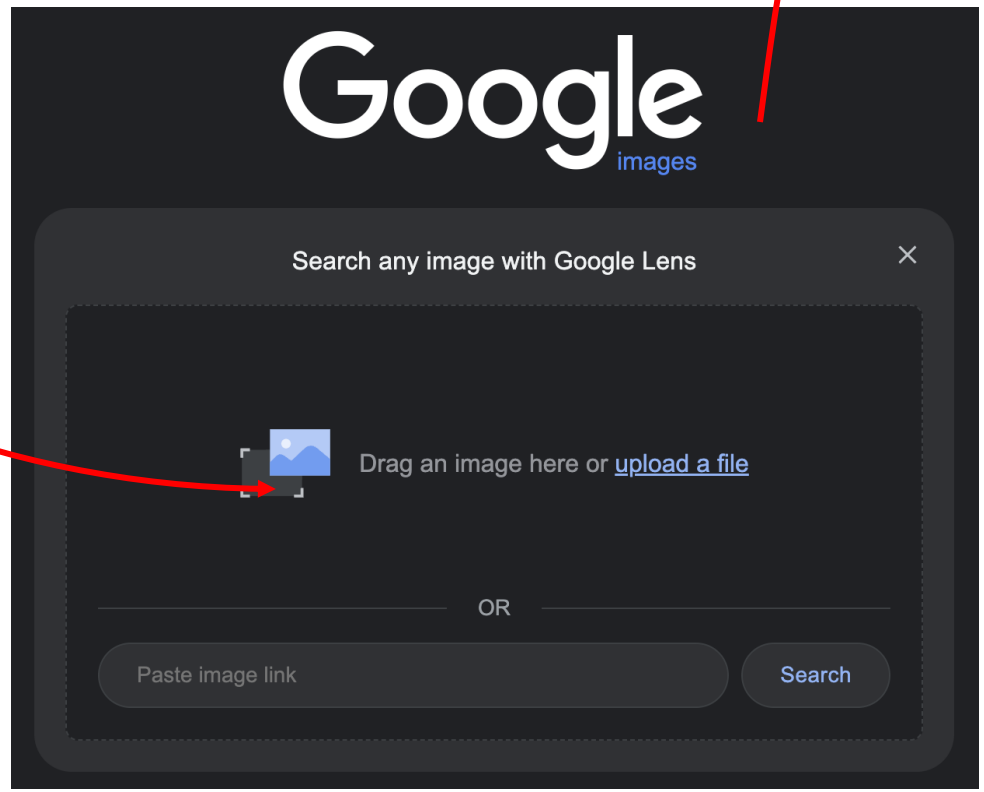
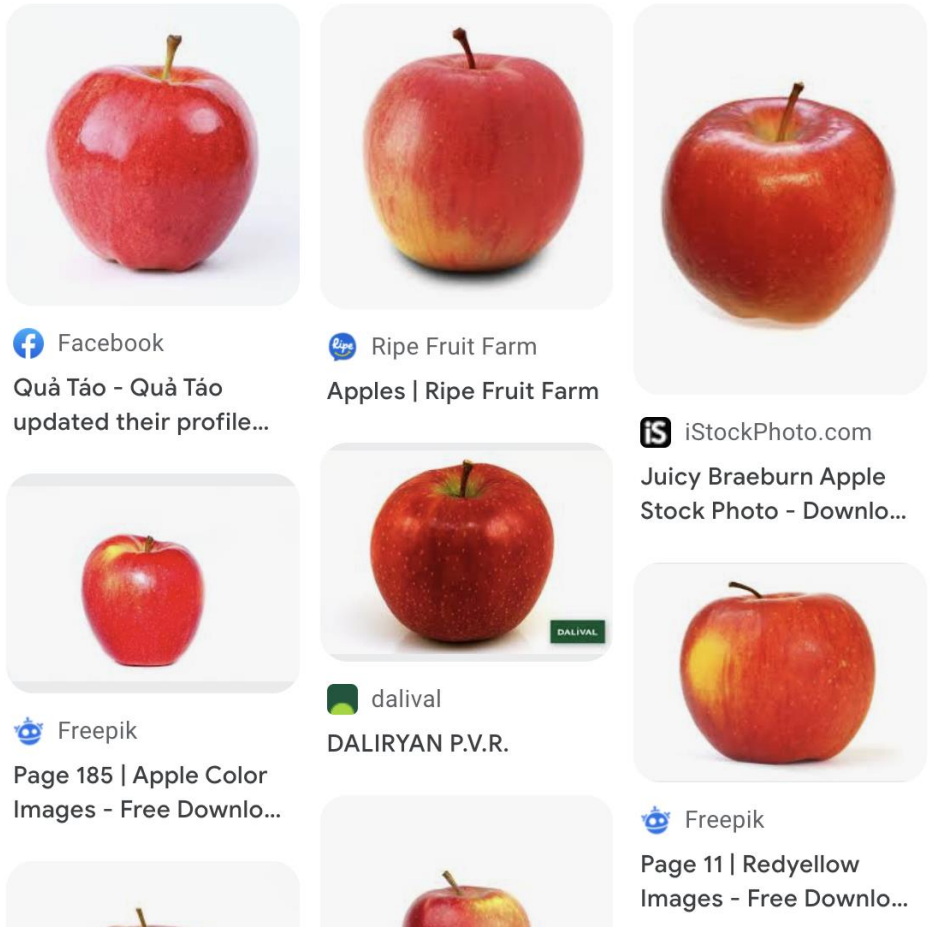


Image Search Engine

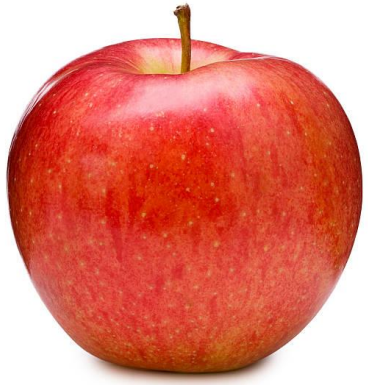


Results

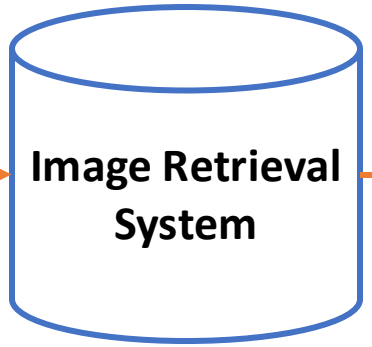


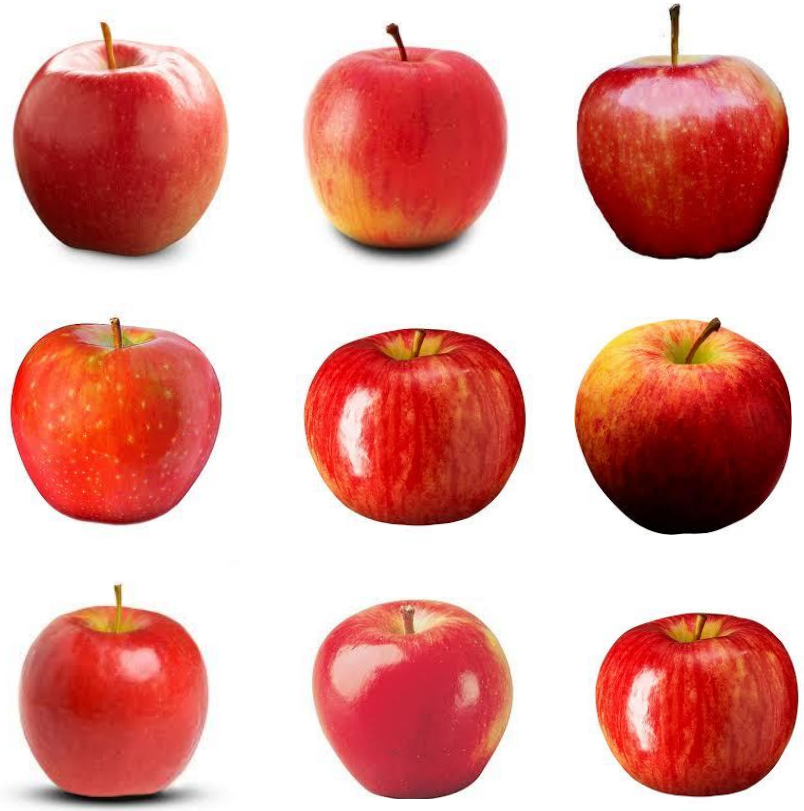
Introduction

❖ Image Retrieval



Query Image





Top K most similar images

An **image retrieval** system is a computer system used for browsing, searching and retrieving images from a large database of digital images.

https://en.wikipedia.org/wiki/Image_retrieval

Introduction

❖ Applications

Google

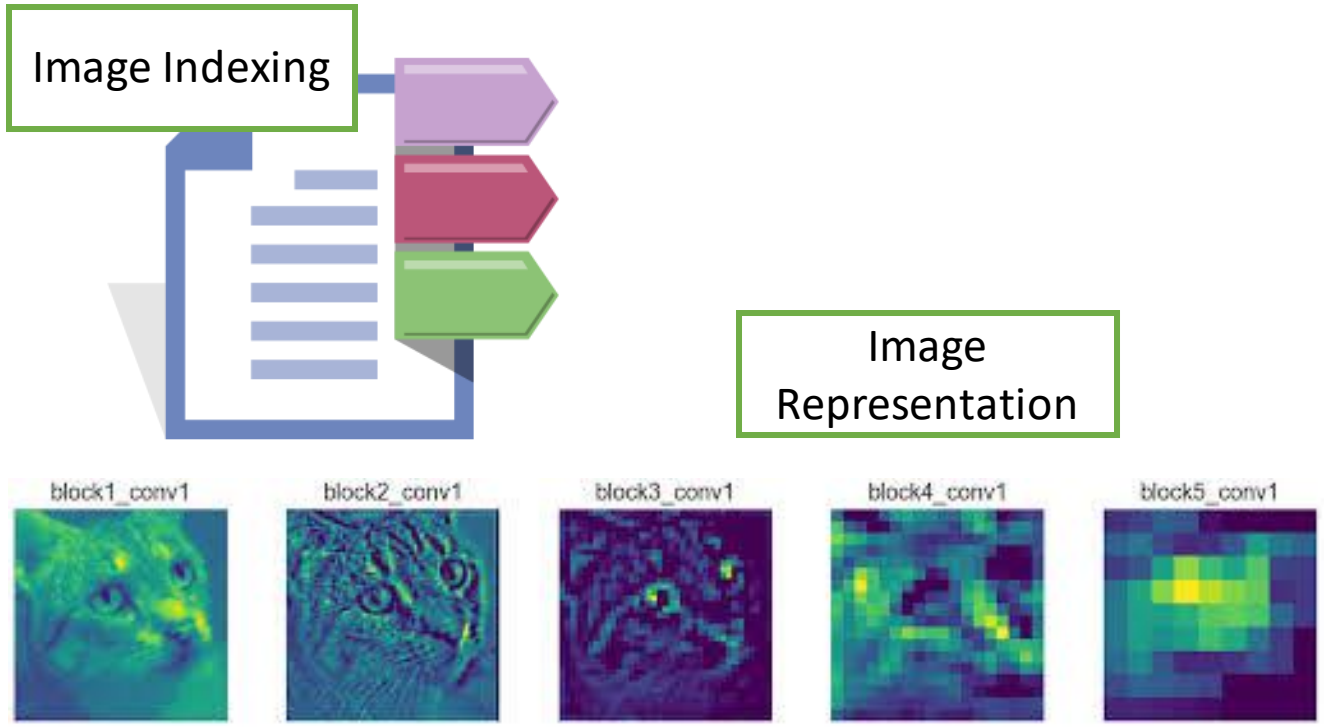
Baidu

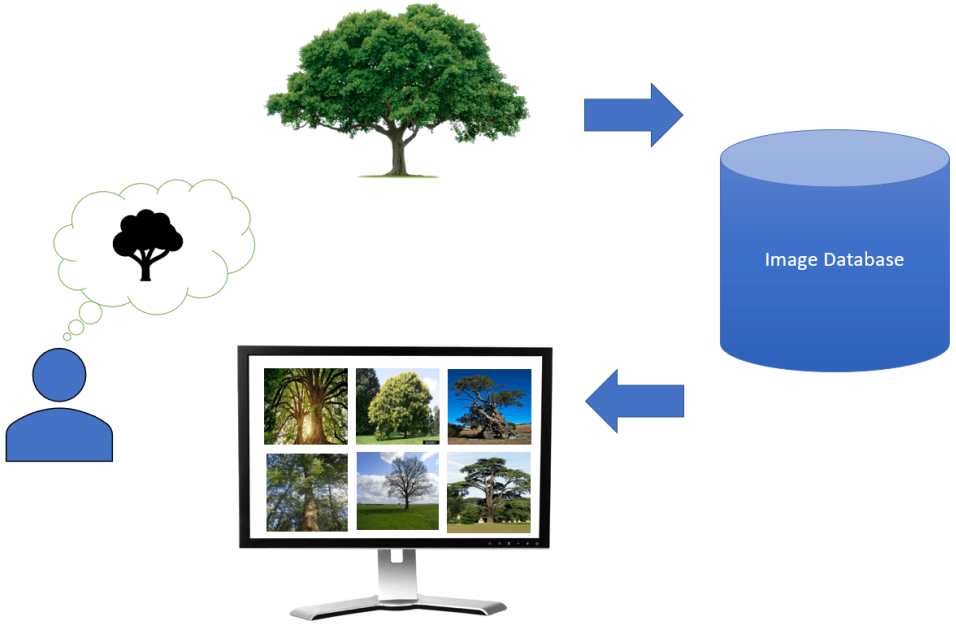
Yandex

Bing

Introduction

❖ Challenges





Share the same some IR challenges

How to satisfy information need?

Introduction

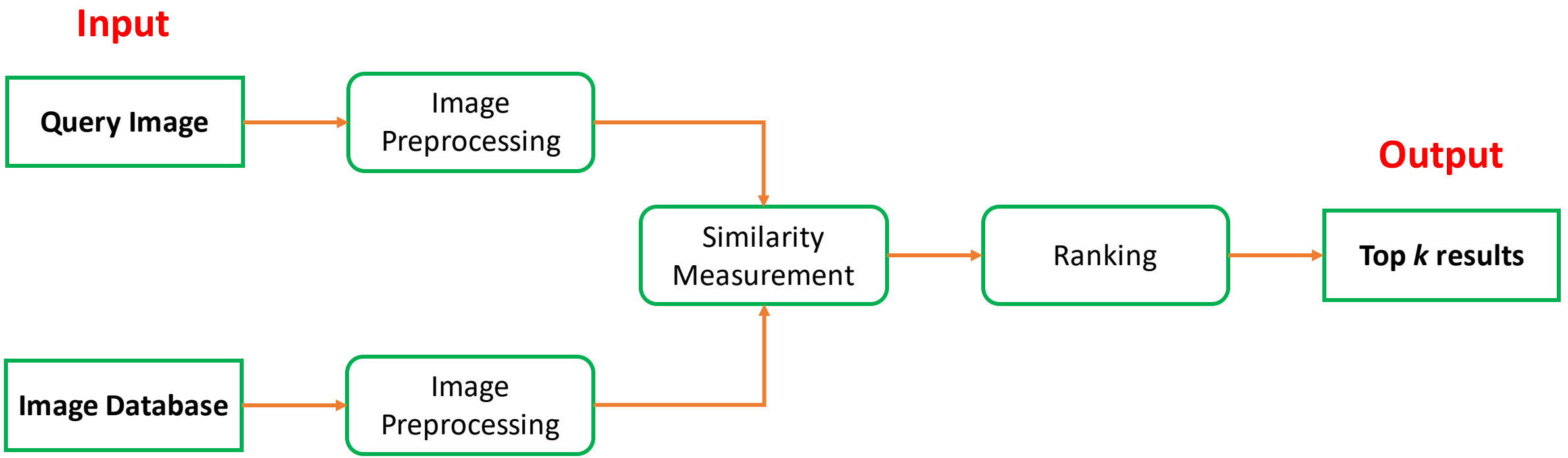
❖ Description

Project Statement: Given a list of images, build an image retrieval system that finds relevant or similar images among the source images compared to the query image.

Download the dataset here

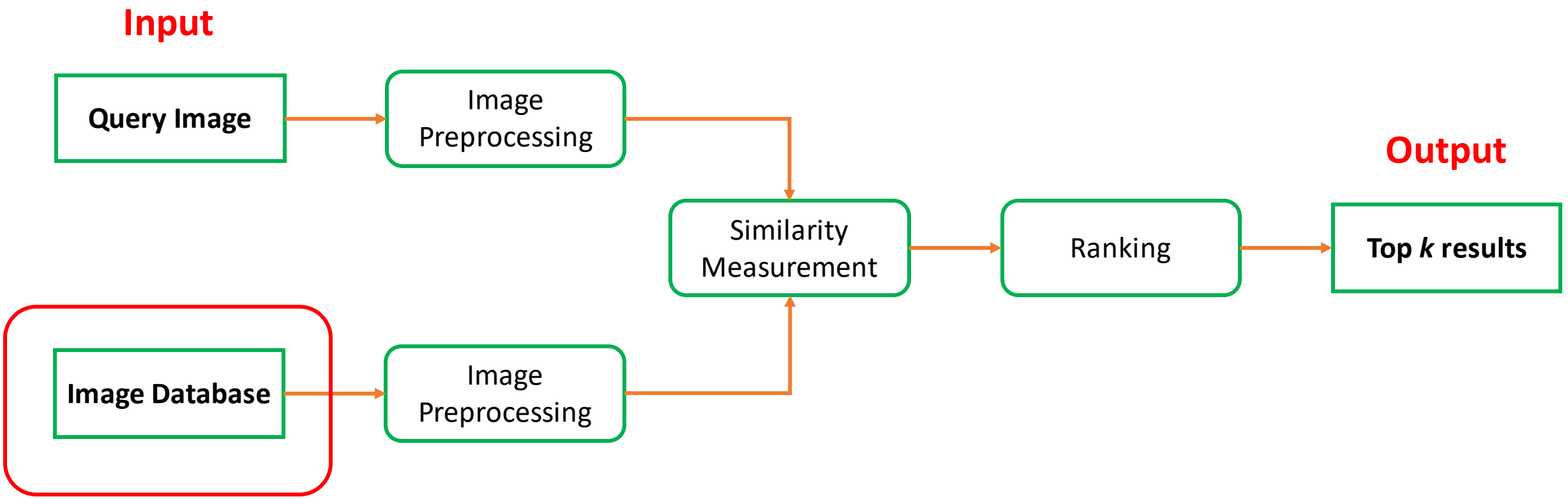
Baseline

❖ Project Pipeline



Baseline

❖ Project Pipeline



Baseline

❖ Step 1: Import libraries and get filenames

1. Import necessary libraries

```
1 import os
2 import numpy as np
3 import cv2
4 import matplotlib.pyplot as plt
```

2. Get image filenames

```
1 # Declare source images directory path
2 dataset_dir = 'images_mr'
3
4 # Get all filenames inside "dataset_dir"
5 image_filenames = os.listdir(dataset_dir)
```

`os.listdir(path='.')`

Return a list containing the names of the entries in the directory given by *path*. The list is in arbitrary order, and does not include the special entries `'.'` and `'..'` even if they are present in the directory. If a file is removed from or added to the directory during the call of this function, whether a name for that file be included is unspecified.

```
1 image_filenames
```

```
['9385.jpg',
 '5052.jpg',
 '6271.jpg',
 '5362.jpg',
 '5852.jpg',
 '6164.jpg',
 '8962.jpg',
 '3608.jpg',
 '6438.jpg',
 '6670.jpg',
 '5018.jpg',
 '8697.jpg',
```

```
1 print(
2     'Number of files: ',
3     len(image_filenames)
4 )
```

```
Number of files:  9908
```

Baseline

❖ Step 2: Read images

```python
# Declare empty list
src_images = []

for filename in image_filenames:
    # Create filepath of current image
    filepath = os.path.join(
        dataset_dir,
        filename
    )

    image = cv2.imread(filepath) # Read image
    image = cv2.cvtColor(
      image,
      cv2.COLOR_BGR2RGB
    ) # Convert to RGB
    src_images.append(image)
```

`os.path.join(path, *paths)`

Join one or more path segments intelligently.

Example:

- dataset_dir = “image_sr”
- filename = “01.jpg”

=> filepath = “image_sr/01.jpg”

Baseline

❖ Step 2: Read images

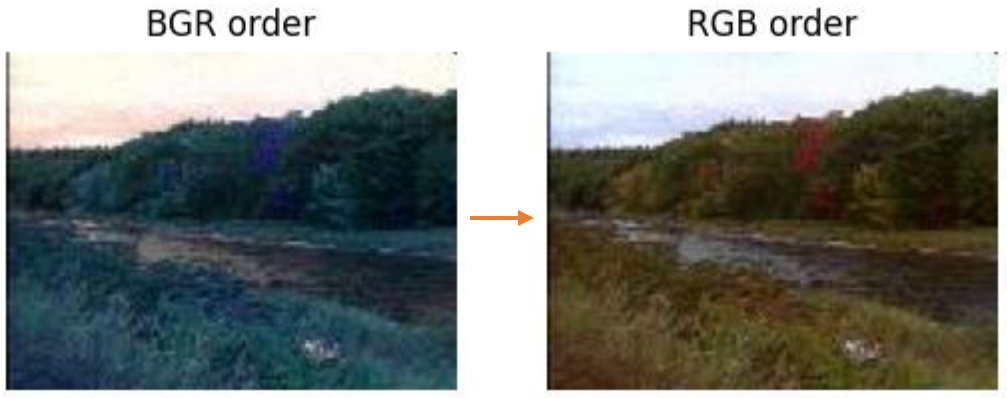
```python
# Declare empty list
src_images = []

for filename in image_filenames:
    # Create filepath of current image
    filepath = os.path.join(
        dataset_dir,
        filename
    )

    image = cv2.imread(filepath) # Read image
    image = cv2.cvtColor(
      image,
      cv2.COLOR_BGR2RGB
    ) # Convert to RGB
    src_images.append(image)
```

`cv2.imread(path, flag)`

Loads an image from a specified file (channels stored in B G R order for color images).

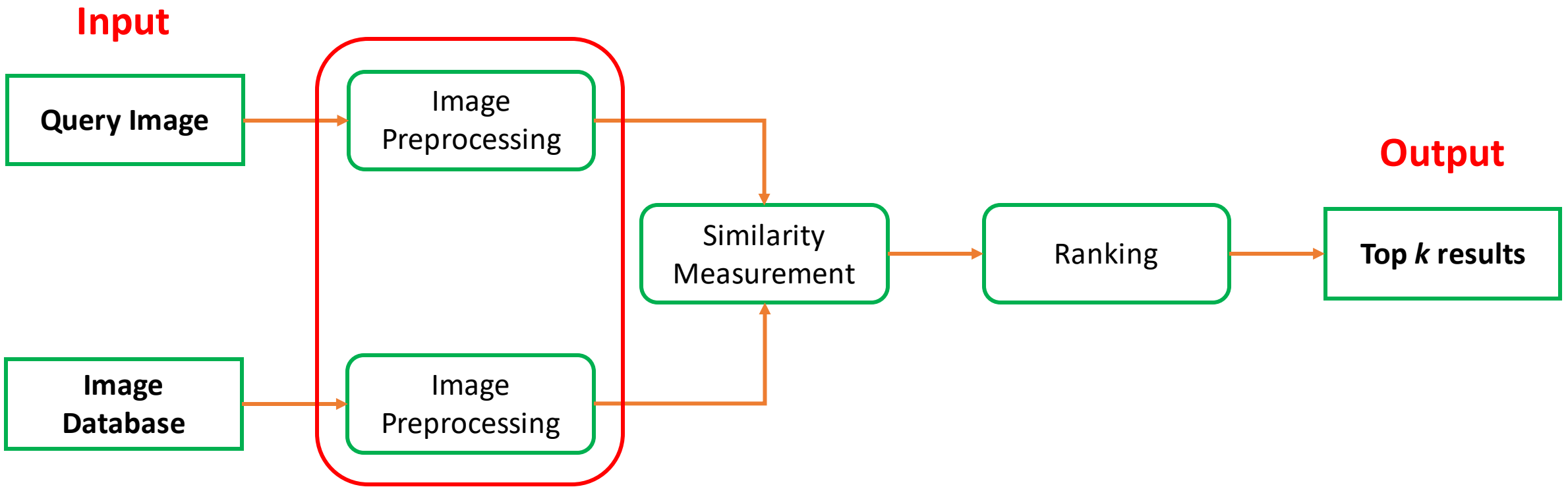
`cv2.cvtColor(src, code)`

Convert an input image from one color space to another.



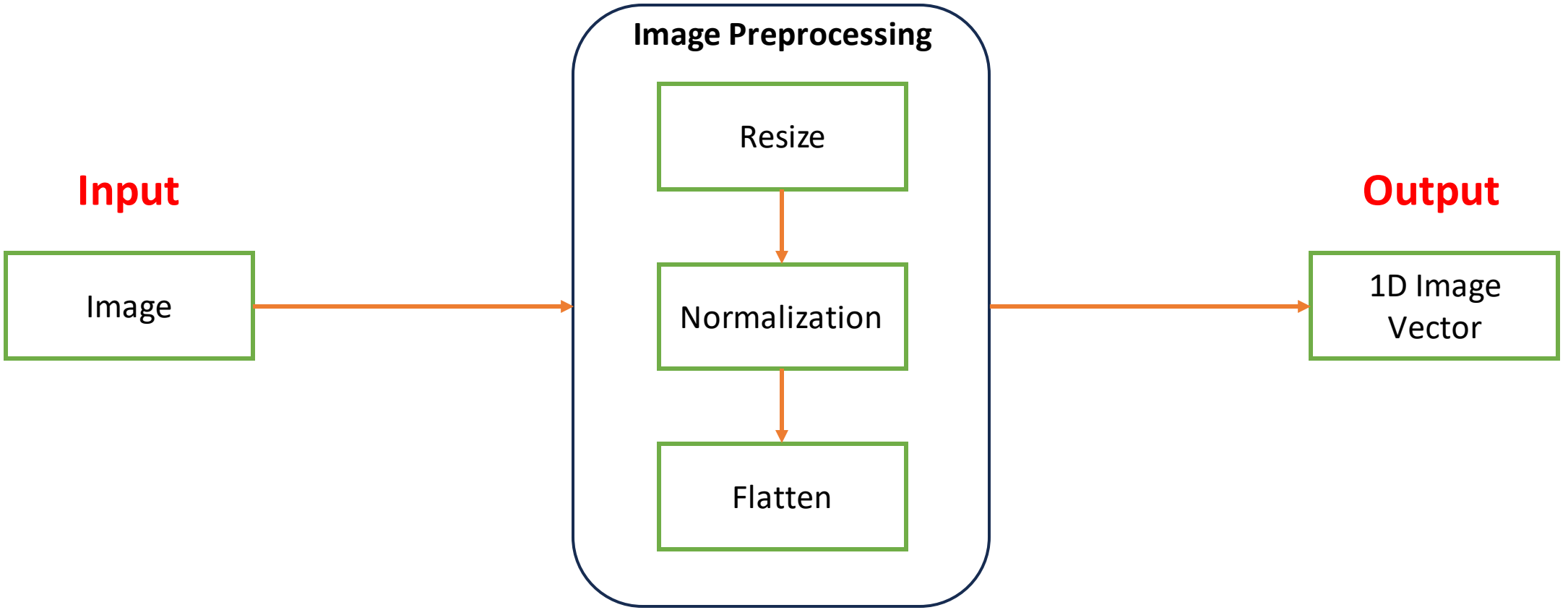
Baseline

❖ Project Pipeline



Baseline

❖ Step 3: Preprocess image



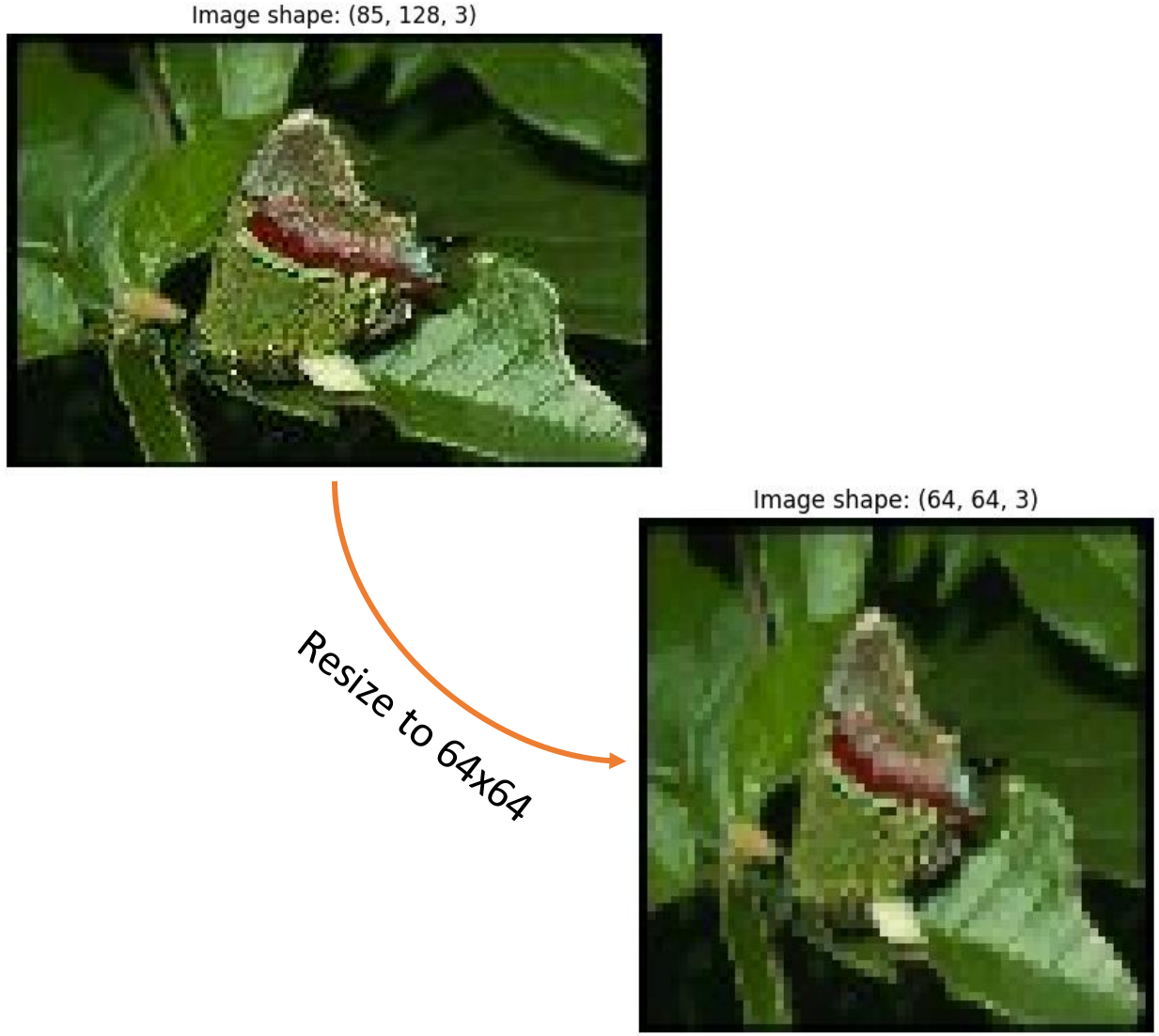
Baseline

❖ Step 3.1: Resize image

`cv2.resize(src, dsize)`

Resize an image into desired (height, width).

```python
def image_resize(images, target_size=(64, 64)):
    resized_image = cv2.resize(
      images,
      target_size
    )

    return resized_image
```



Baseline

❖ Step 3.2: Normalize image

Image Normalization: Transforms the pixel values of image into a common scale that might boost the performance of system.

Z-score Normalization (Standardization): A technique that transforms data so that it has a mean of 0 and a std of 1.

$$X_{normalized} = \frac{X - mean}{std}$$

```
[[[  4   6   9]          [[[-1.20057116 -1.15850123 -0.97975352]
  [  0   6   0]            [-1.25822794 -1.15850123 -1.11004419]
  [  1  10   0]            [-1.24381375 -1.09741939 -1.11004419]
  ...                      ...
  [  0   8   0]            [-1.25822794 -1.12796031 -1.11004419]
  [  4   7   8]            [-1.20057116 -1.14323077 -0.99423026]
  [  3   1  21]]           [-1.21498535 -1.23485353 -0.80603264]]

 [[  0   2   2]           [[-1.25822794 -1.21958307 -1.08109071]
  [ 17  26  11]            [-1.01318662 -0.85309204 -0.95080004]
  [ 26  40  11]            [-0.88345886 -0.6393056  -0.95080004]
  ...                      ...
  [ 10  23   2]            [-1.11408599 -0.89890342 -1.08109071]
  [  3   7   5]            [-1.21498535 -1.14323077 -1.03766049]
  [  0   1  14]]           [-1.25822794 -1.23485353 -0.90736982]]

 [[  0   2   1]           [[-1.25822794 -1.21958307 -1.09556745]
  [ 38  47  26]            [-0.71048852 -0.53241238 -0.73364894]
  [ 49  65  24]            [-0.55193237 -0.25754411 -0.76260242]
  ...                      ...
  [ 60  80  36]            [-0.39337623 -0.02848721 -0.58888154]
  [ 11  20   5]            [-1.09967179 -0.9447148  -1.03766049]
  [  0   3   6]]           [-1.25822794 -1.20431261 -1.02318375]]

 ...                      ...
```

Baseline

❖ Step 3.2: Normalize image

```python
def calculate_mean_std(images):
    mean = np.mean(images, axis=(0, 1, 2))
    std = np.std(images, axis=(0, 1, 2))

    return mean, std

def image_std_normalize(images, mean, std):
    normalized_image = (images - mean) / std

    return normalized_image
```

Z-score Normalization (Standardization): A technique that transforms data so that it has a mean of 0 and a std of 1.

$$X_{normalized} = \frac{X - mean}{std}$$

Baseline

❖ Step 3.3: Flatten image

```python
def image_flatten(images, is_batch=False):
    # For flatten a bunch of images
    if is_batch:
        flattened_image = images.reshape(
            images.shape[0], -1
        )
     # For flatten a single image
    else:
        flattened_image = images.reshape(-1)

    return flattened_image
```

ndarray.reshape(*shape*, *order*='C')

Returns an array containing the same data with a new shape.

Refer to `numpy.reshape` for full documentation.

See also

`numpy.reshape`
equivalent function

```python
normalized_image = image_std_normalize(
    image_resized,
    mean, std
)
flattened_image = image_flatten(normalized_image)

print('Shape before flattening: ',normalized_image.shape)
print('Shape after flattening: ', flattened_image.shape)
```

```
Shape before flattening:  (64, 64, 3)
Shape after flattening:  (12288,)
```

Baseline

❖ Step 3.4: Final image preprocessing function

```python
def preprocess_batches(images):
    resized_images = [
        image_resize(image) for image in src_images
    ]
    images_arr = np.array(resized_images)
    mean, std = calculate_mean_std(images_arr)
    normalized_images = image_std_normalize(
        images_arr,
        mean, std
    )
    flattened_images = image_flatten(
        normalized_images,
        is_batch=True
    )

    return flattened_images, mean, std
```

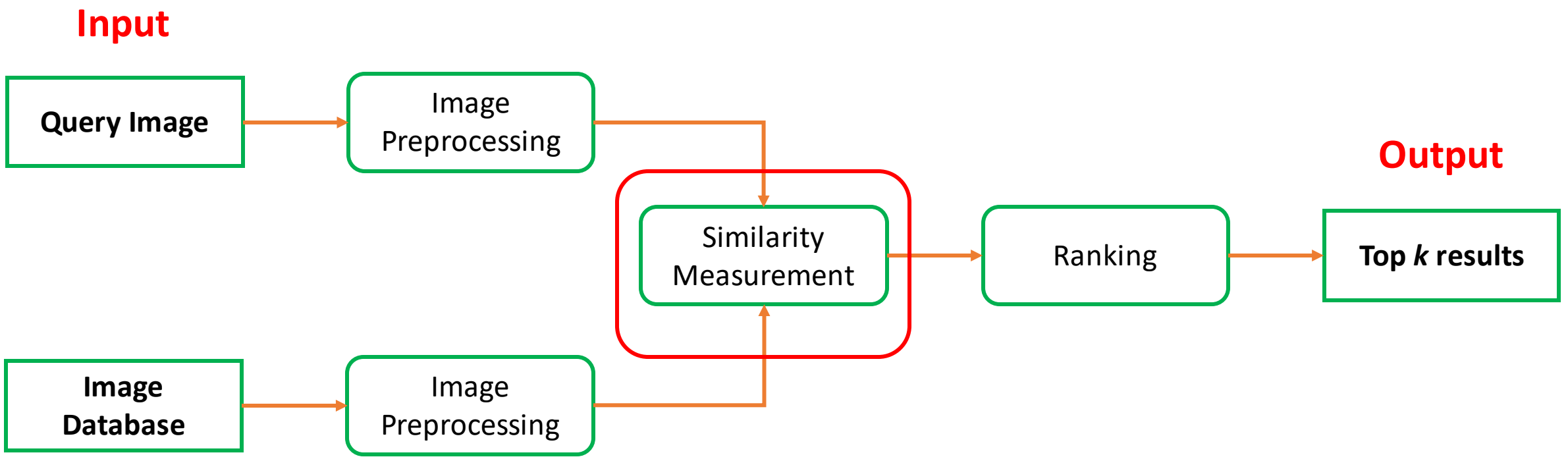
Function to process the whole dataset

```python
def preprocess_query(image):
    resized_image = image_resize(image)
    normalized_image = image_std_normalize(
        resized_image,
        mean, std
    )
    flattened_image = image_flatten(normalized_image)

    return flattened_image
```

Function to process a single image

Baseline

❖ Project Pipeline

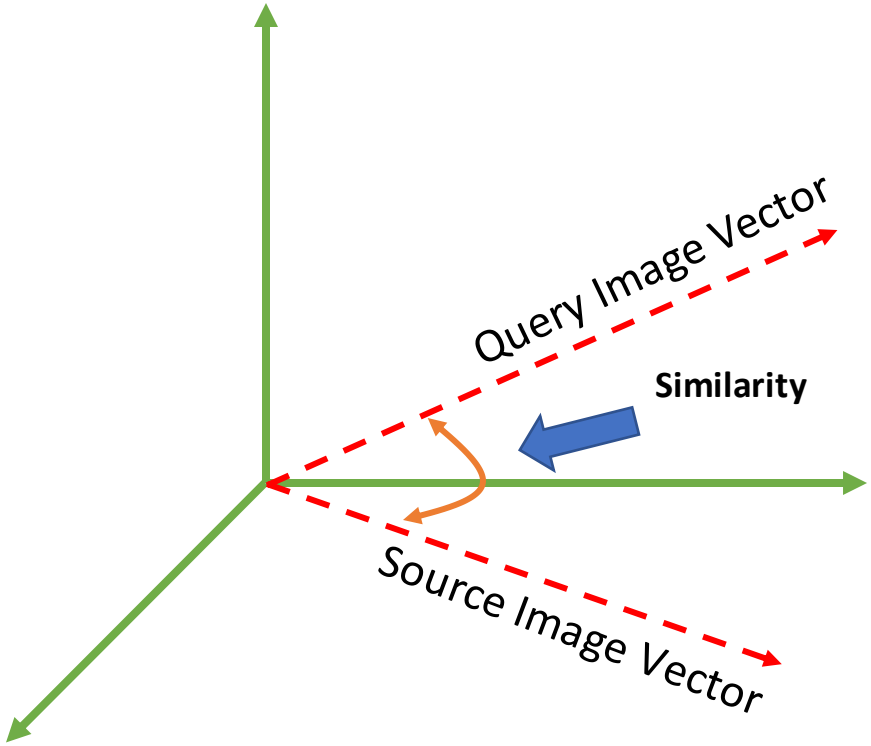


Baseline

❖ Step 4: Define similarity metrics

Similarity Calculation: Define a function to find the similarity between two vectors. In this project, we will define 4 metrics:

1. Mean Absolute Error (L1 Loss)
2. Mean Squared Error (L2 Loss)
3. Cosine Similarity
4. Correlation Coefficient



Baseline

❖ Step 4: Mean Absolute Error

```python
def mean_absolute_error(query_vector, src_vectors):
    abs_diff = np.abs(src_vectors - query_vector)

    mae = np.mean(abs_diff, axis=1)

    return mae
```

Given two 1D image vectors that have ***n*** elements, the Mean Absolute Error is calculated as follows:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|a_i - b_i|$$

Baseline

❖ Step 4: Mean Squared Error

```python
def mean_squared_error(query_vector, src_vectors):
    squared_diff = (src_vectors - query_vector) ** 2

    mse = np.mean(squared_diff, axis=1)

    return mse
```

Given two 1D image vectors that have ***n*** elements, the Mean Squared Error is calculated as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(a_i - b_i)^2$$

Baseline

❖ Step 4: Cosine Similarity

```python
def cosine_similarity(query_vector, src_vectors):
    query_norm = np.linalg.norm(query_vector)
    normalized_query = query_vector / query_norm
    src_norms = np.linalg.norm(src_vectors, axis=1)
    normalized_src = src_vectors / src_norms[:, np.newaxis]

    cosine_similarity = np.dot(
      normalized_src,
      normalized_query
    )

    return cosine_similarity
```

Given two 1D image vectors that have ***n*** elements, the Cosine Similarity is calculated as follows:

$$cosine\_similarity = \frac{a \cdot b}{\|a\|\|b\|}$$

Baseline

❖ Step 4: Correlation Coefficient

```python
def correlation_coefficient(query_vector, src_vectors):
    corr_coef =  np.corrcoef(
      query_vector,
      src_vectors
    )[:-1, -1]

    return corr_coef
```

numpy.corrcoef(*x, y=None, rowvar=True, bias=<no value>, ddof=<no value>, *, dtype=None*) [source]

Return Pearson product-moment correlation coefficients.

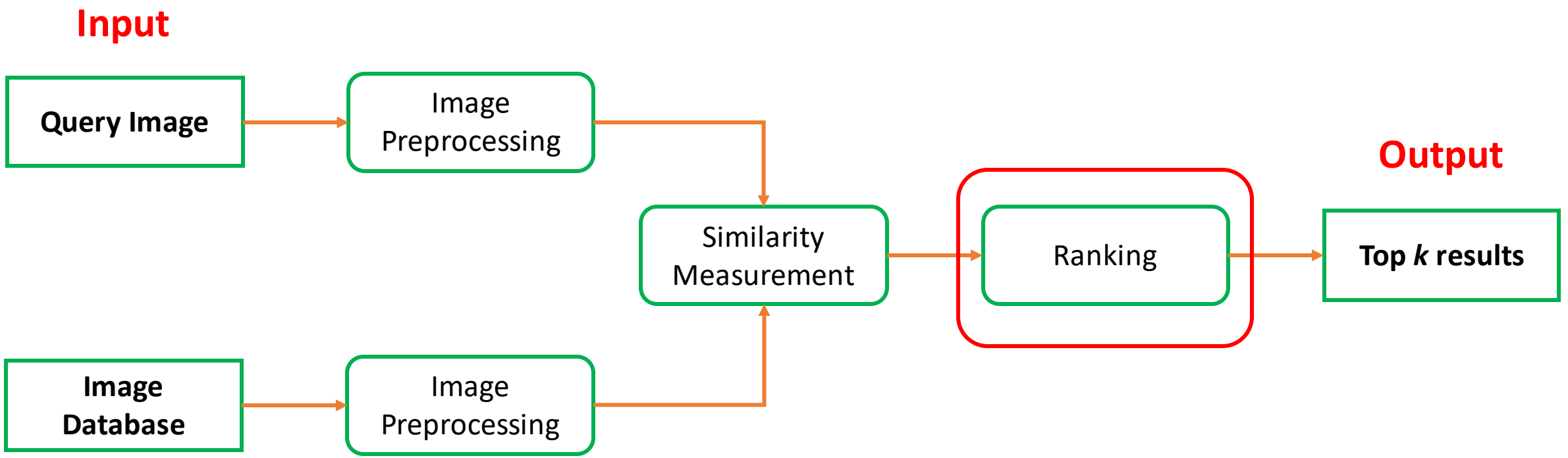
Please refer to the documentation for `cov` for more detail. The relationship between the correlation coefficient matrix, *R*, and the covariance matrix, *C*, is

$$R_{ij} = \frac{C_{ij}}{\sqrt{C_{ii} C_{jj}}}$$

The values of *R* are between -1 and 1, inclusive.

Baseline

❖ Project Pipeline



Baseline

❖ Step 5: Ranking

```python
def ranking(
    preprocessed_query_image,
    preprocessed_src_images,
    top_k=10
):
    scores = cosine_similarity(
        preprocessed_query_image,
        preprocessed_src_images
    )
    ranked_list = np.argsort(scores)[::-1][:top_k]
    scores = scores[ranked_list]

    return ranked_list, scores
```

Baseline

❖ Ranking program

```
query_image_paths = [
    '/content/images_mr/1.jpg'
]
top_k = 10

for query_image_path in query_image_paths:
    query_image = cv2.imread(query_image_path, 1)
    query_image = cv2.cvtColor(query_image, cv2.COLOR_BGR2RGB)
    preprocessed_query_image = preprocess_query(query_image)

    ranked_list, scores = ranking(
        preprocessed_query_image,
        preprocessed_src_images,
        top_k
    )

    print('Query Image')
    plt.figure(figsize=(3, 3))
    plt.imshow(query_image)
    plt.axis('off')
    plt.show()
    print(f'Top {top_k} results')
    for idx in range(len(ranked_list)):
        src_image_idx = ranked_list[idx]
        similarity_score = scores[idx]
        plt.figure(figsize=(3, 3))
        plt.imshow(src_images[src_image_idx])
        plt.title(f'Similarity: {similarity_score}', fontsize=10)
        plt.axis('off')
        plt.show()
```

Similarity: 1.0

Similarity: 0.6072830969370191

Similarity: 0.6172511516359848

Similarity: 0.5779375079555944

Baseline

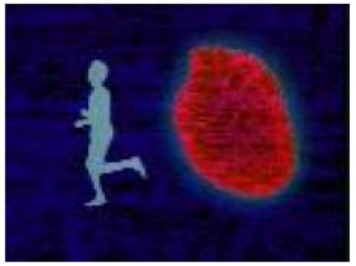
❖ Ranking program: How to visualize

Matplotlib: A library for creating static, animated, and interative visualizations in Python.










Baseline

❖ Ranking program: How to visualize

matplotlib.pyplot.figure

```
matplotlib.pyplot.figure(num=None, figsize=None, dpi=None, *,
facecolor=None, edgecolor=None, frameon=True, FigureClass=<class
'matplotlib.figure.Figure'>, clear=False, **kwargs)
```

[source]

Create a new figure, or activate an existing figure.

```
1 plt.figure()
2 plt.show()
3
4 # <Figure size 640x480 with 0 Axes
```

matplotlib.pyplot.show

```
matplotlib.pyplot.show(*, block=None)
```

[source]

Display all open figures.

```
1 plt.figure(figsize=(12, 12))
2 plt.show()
3
4 # <Figure size 1200x1200 with 0 Axes>
```

Baseline

❖ Ranking program: How to visualize

```
1 plt.figure(figsize=(4, 4))
2 image_path = 'images_mr/1.jpg'
3 image = cv2.imread(image_path)
4 image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
5 plt.imshow(image)
6 plt.show()
```

matplotlib.pyplot.imshow

matplotlib.pyplot.imshow(*X*, *cmap=None*, *norm=None*, ***, *aspect=None*, *interpolation=None*, *alpha=None*, *vmin=None*, *vmax=None*, *origin=None*, *extent=None*, *interpolation_stage=None*, *filternorm=True*, *filterrad=4.0*, *resample=None*, *url=None*, *data=None*, ***kwargs*) [source]

Display data as an image, i.e., on a 2D regular raster.

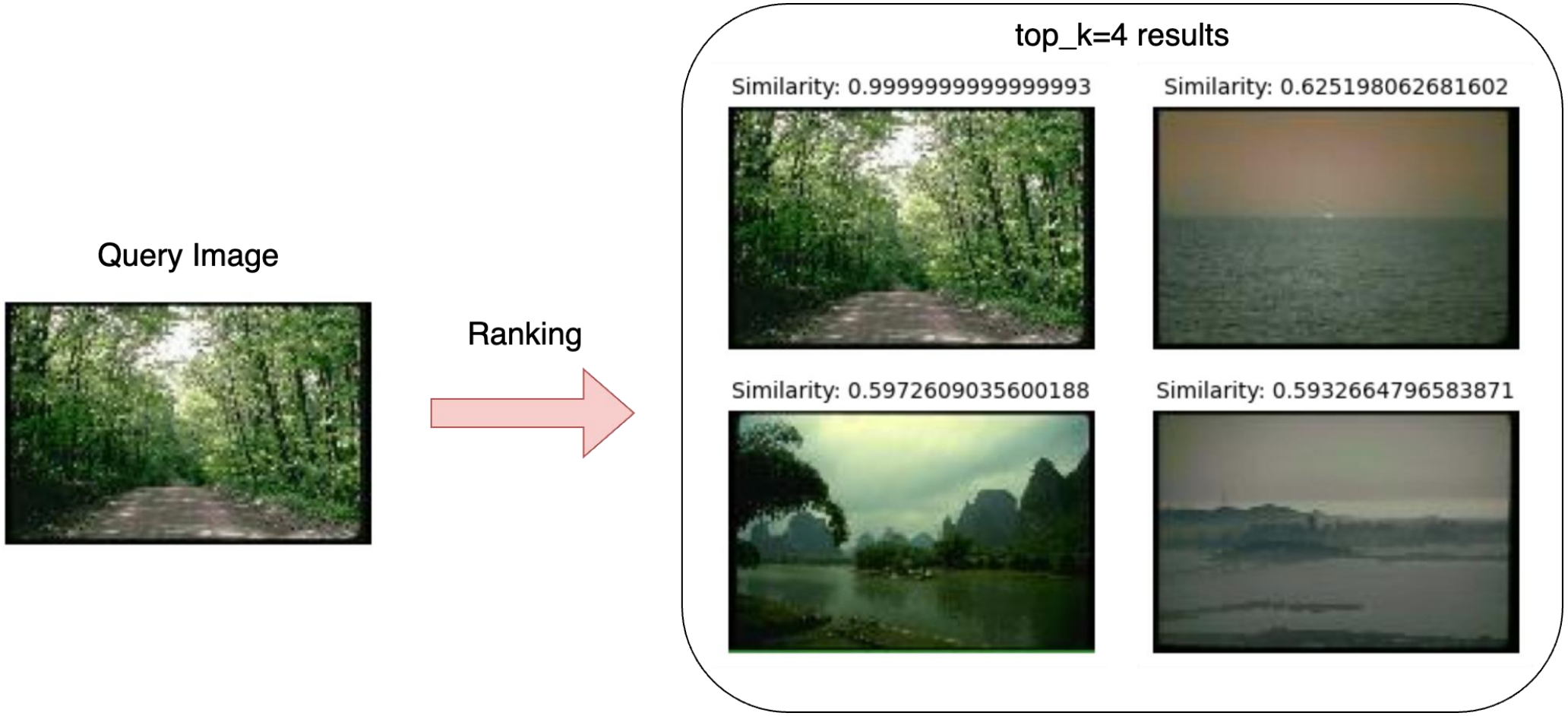
Baseline

❖ Ranking program: How to visualize

```
plt.figure(figsize=(4, 4))
image_path = 'images_mr/1.jpg'
image = cv2.imread(image_path)
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(image)
plt.axis('off')
plt.title(f'Image of {image_path}')
plt.show()
```

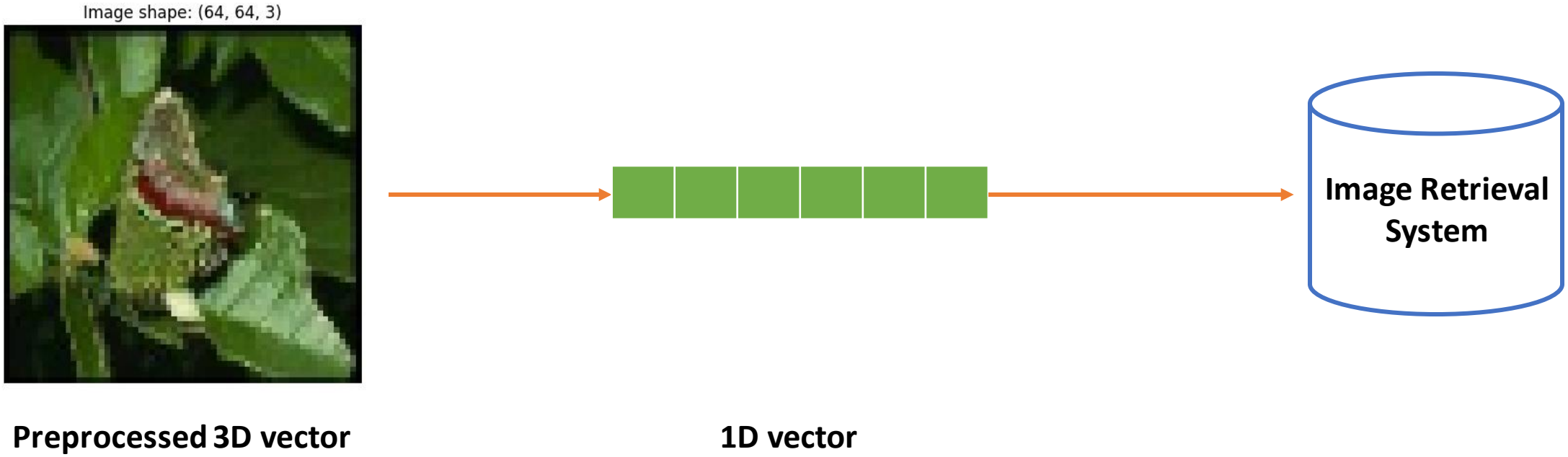
Baseline

❖ Baseline results



Apply pre-trained ViT

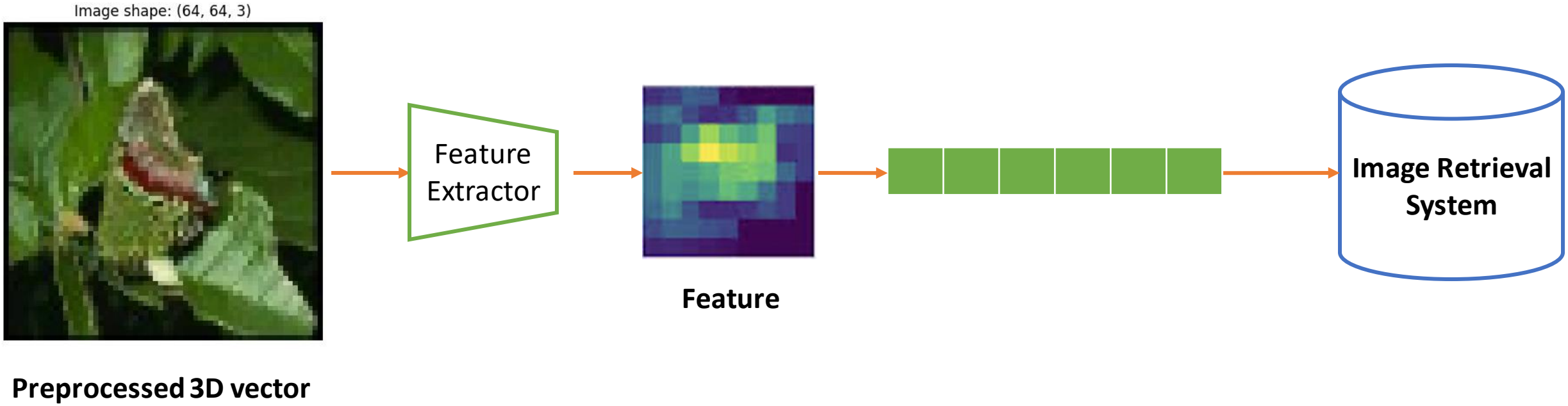
❖ Getting Started



Current Approach

Apply pre-trained ViT

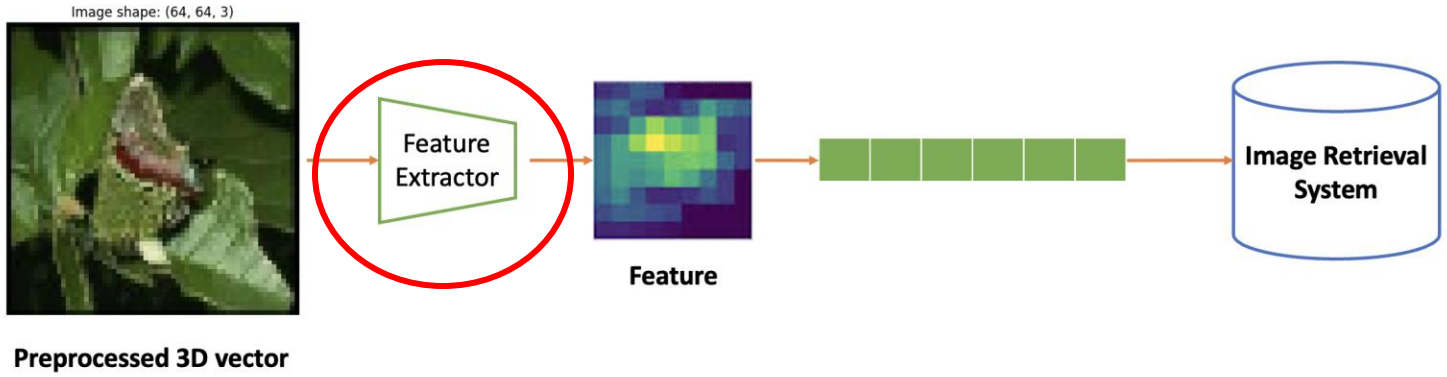
❖ Getting Started

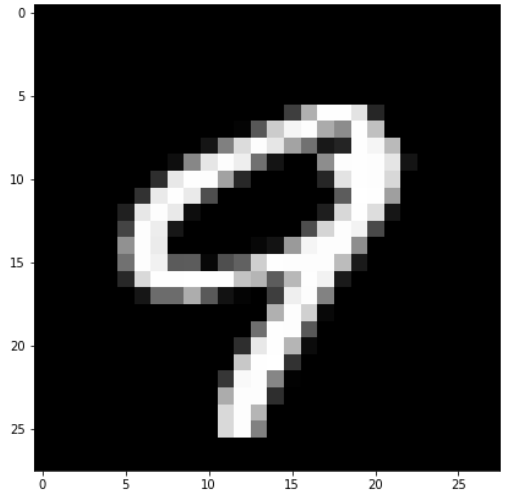


Modern approach with the use of ML/DL model

Apply pre-trained ViT

❖ Getting Started

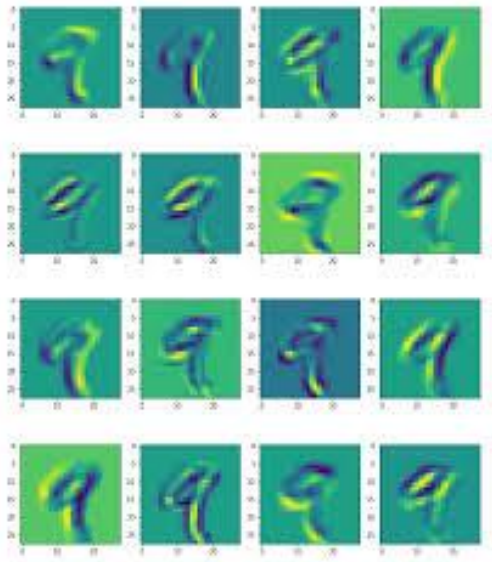




Raw Image

Feature Extractor: A component (typically a network) that is automatically identifying and extracting meaningful features from raw image data. Some well-known extractor are:

- **Digital Image Processing:** HOG, SIFT…
- **Traditional Machine Learning:** Bag of Visual Words, PCA…
- **Deep Learning:** CNNs, Autoencoder, ViT…



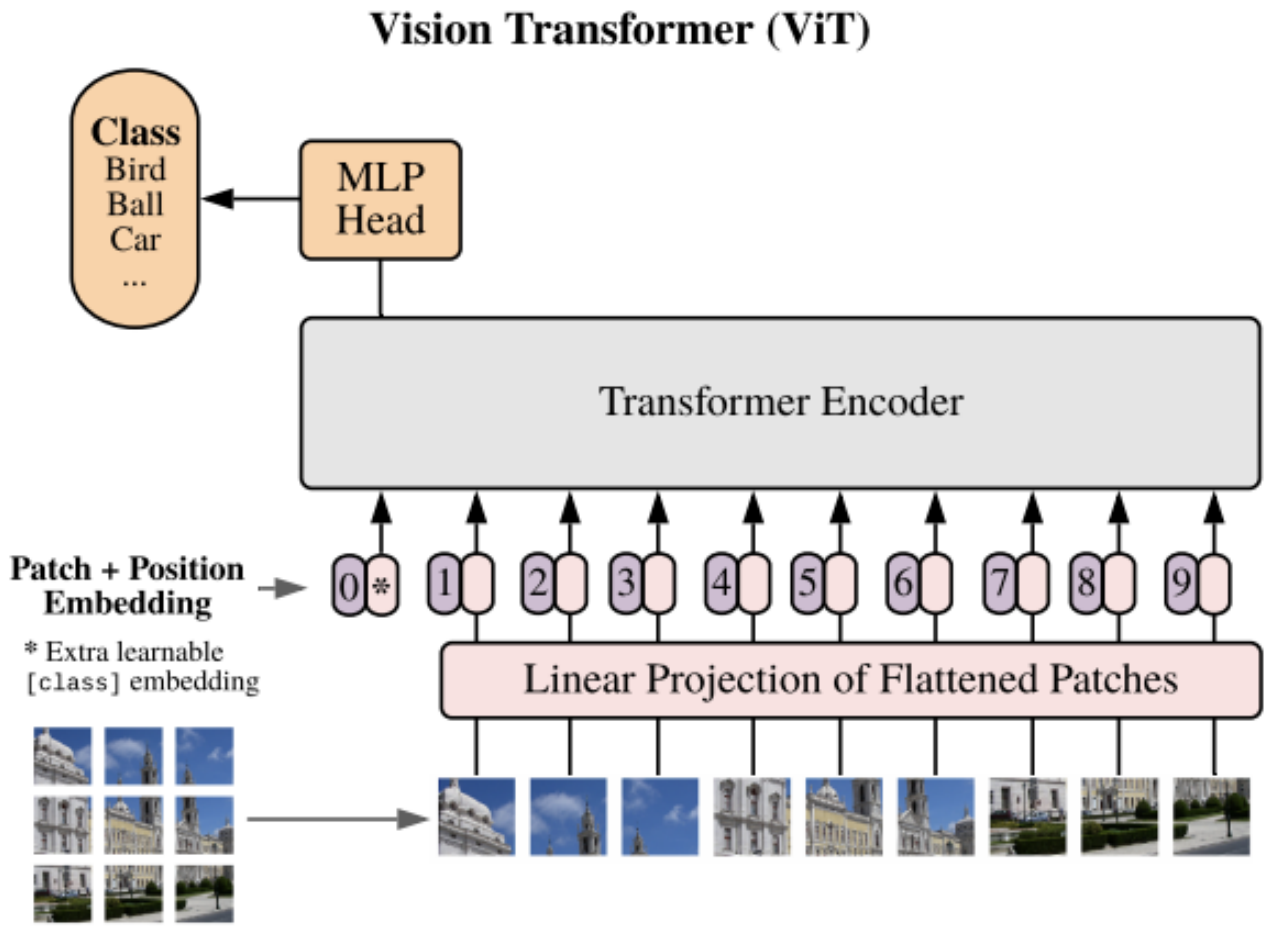
Feature Maps (resulted from Feature Extractor)

Apply pre-trained ViT

❖ Introduction

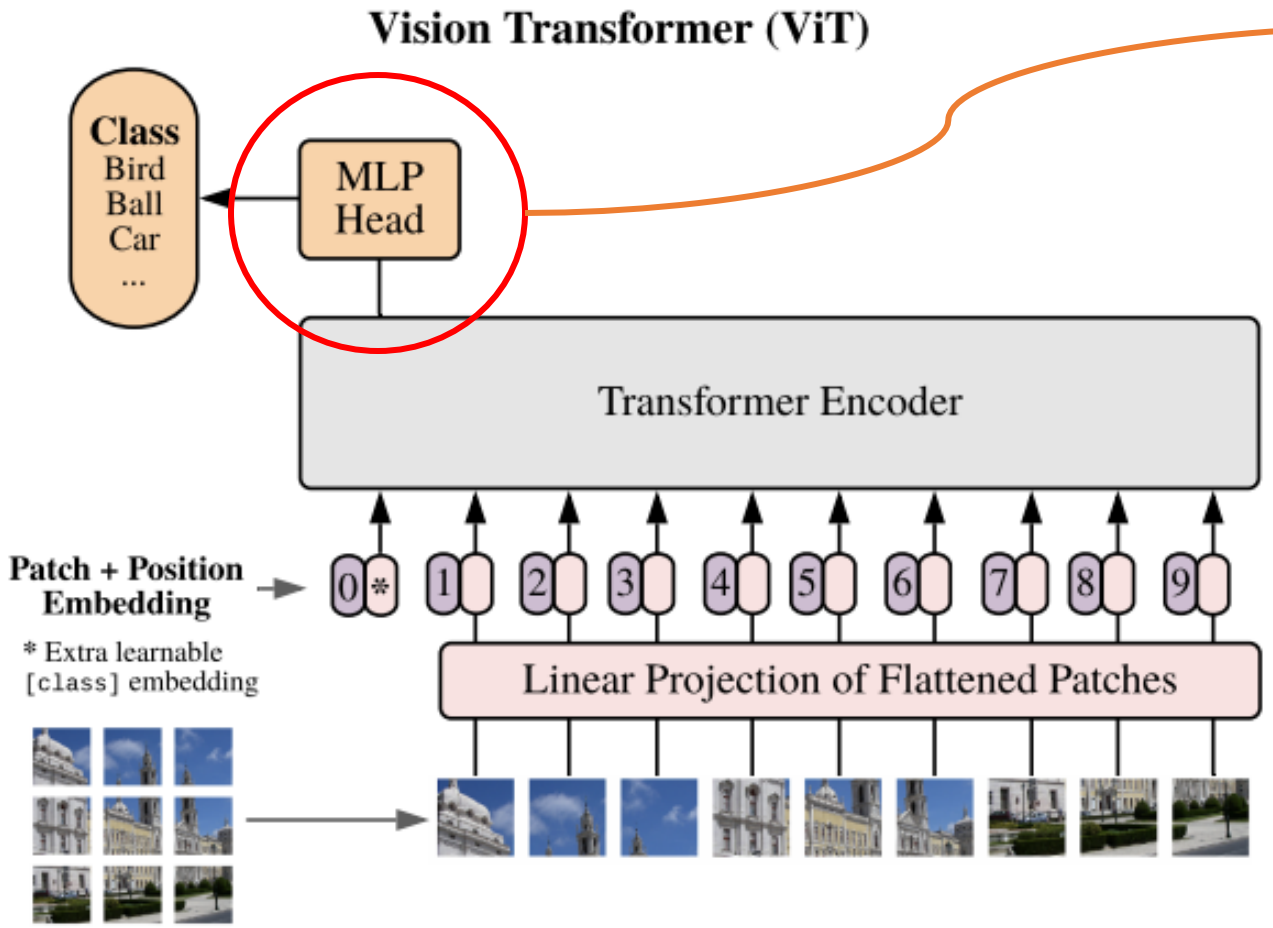
Vision Transformer (ViT): A deep learning architecture designed for image classification tasks. ViT deviates from traditional CNNs by using a transformer-based architecture.

❖ Paper: https://arxiv.org/pdf/2010.11929.pdf



Apply pre-trained ViT

❖ Idea

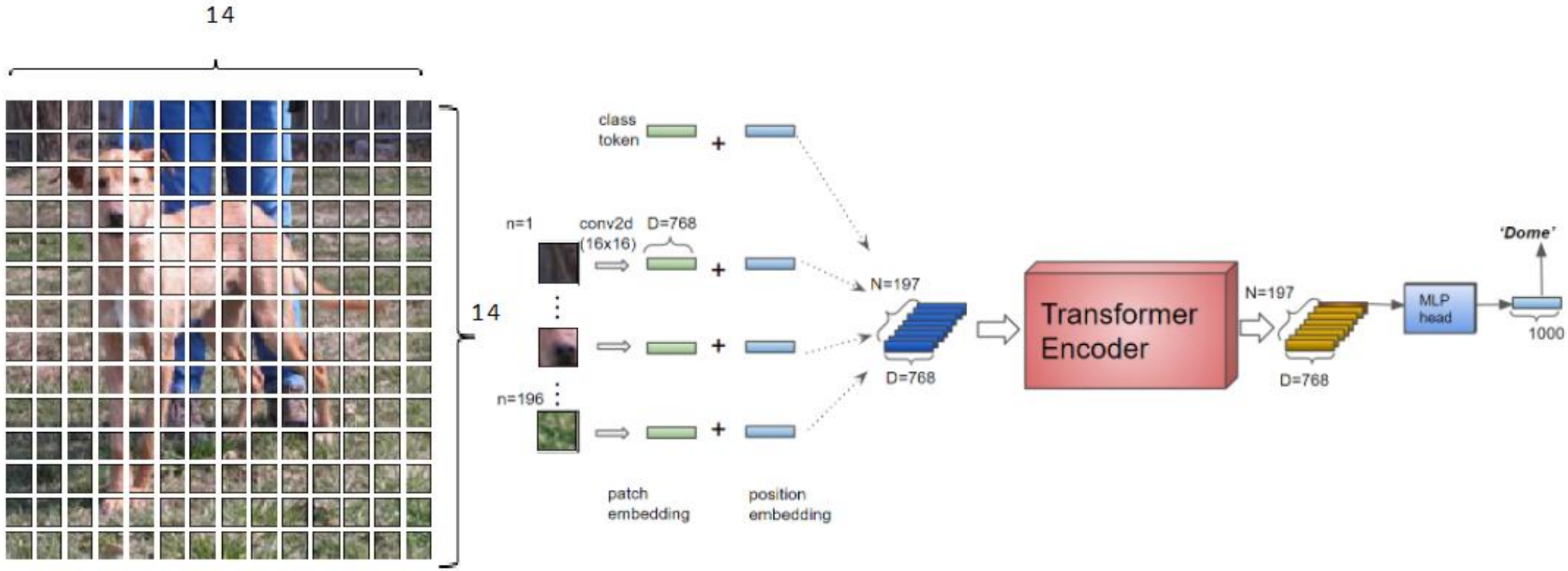


Using **[CLS] token** as vector representation for image.

Add **ViT model inference** in preprocessing image function.

Apply pre-trained ViT

❖ Idea



Using **[CLS] token** as vector representation for image.

https://dkamatblog.home.blog/2021/08/05/vision-transformers-vit/

Apply pre-trained ViT

❖ Step 1: Import libraries and get filenames

1. Import necessary libraries

```python
import os
import numpy as np
import cv2
import matplotlib.pyplot as plt
import torch
from transformers import ViTImageProcessor,
ViTForImageClassification
```

```python
!pip install transformers==4.31.0 -q
```

2. Get image filenames

```python
# Declare source images directory path
dataset_dir = 'images_mr'

# Get all filenames inside "dataset_dir"
image_filenames = os.listdir(dataset_dir)
```

```python
image_filenames
```

```
['9385.jpg',
 '5052.jpg',
 '6271.jpg',
 '5362.jpg',
 '5852.jpg',
 '6164.jpg',
 '8962.jpg',
 '3608.jpg',
 '6438.jpg',
 '6670.jpg',
 '5018.jpg',
 '8697.jpg',
```

```python
print(
    'Number of files: ',
    len(image_filenames)
)
```

```
Number of files:  9908
```

Note: Turn on GPU before start notebook.

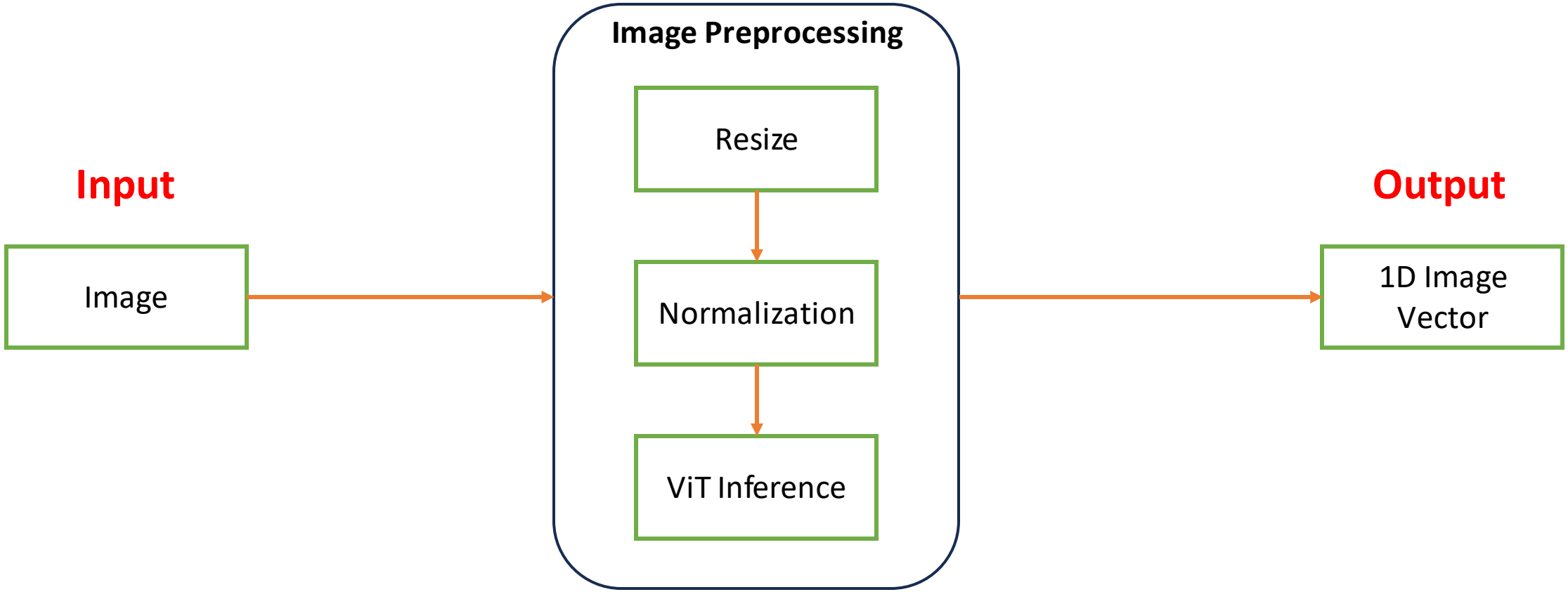
Apply pre-trained ViT

❖ Step 2: Load image dataset

```
dataset_dir = 'images_mr'
image_filenames = os.listdir(dataset_dir)[:500]
src_images = []
for filename in image_filenames:
    filepath = os.path.join(
        dataset_dir,
        filename
    )

    image = cv2.imread(filepath)
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
    src_images.append(image)
```

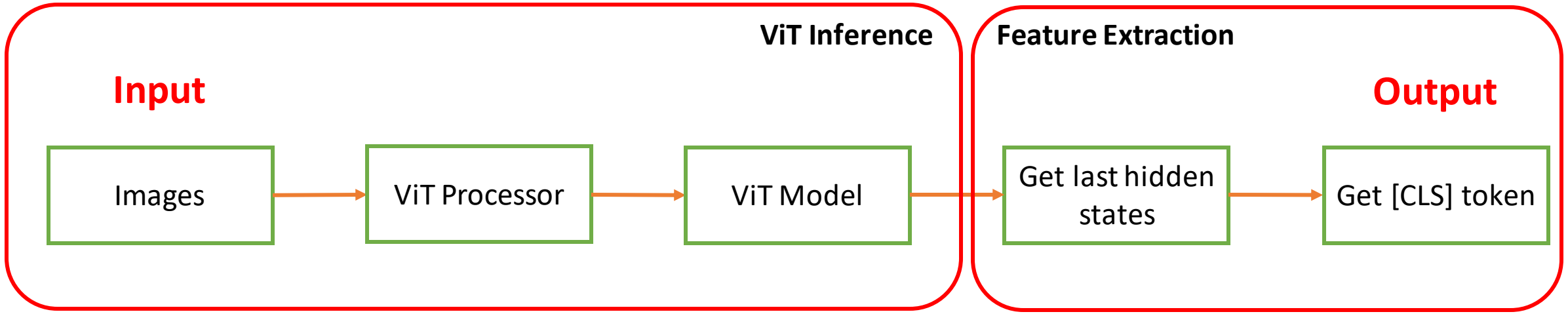
Apply pre-trained ViT

❖ Step 3: Define preprocessing function



Apply pre-trained ViT

❖ Step 3: Define preprocessing function



New preprocessing pipeline with the use of ViT

Apply pre-trained ViT

❖ Step 3: Define preprocessing function

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
processor = ViTImageProcessor.from_pretrained(
  "google/vit-base-patch16-224")
model = ViTForImageClassification.from_pretrained(
  "google/vit-base-patch16-224").to(device)
```

Pre-trained ViT can be taken from **Huggingface** library.

Apply pre-trained ViT

❖ Step 3: Define preprocessing function

```python
def preprocessing(images):
    inputs = processor(
        images,
        return_tensors='pt'
    ).to(device)

    with torch.no_grad():
        output = model(
            **inputs,
            output_hidden_states=True
        ).hidden_states[-1][:, 0, :].detach().cpu().numpy()

    return output
```

Apply pre-trained ViT

❖ Step 4, 5: Define similarity and rank function

```python
def mean_absolute_error(query_vector, src_vectors):
    abs_diff = np.abs(src_vectors - query_vector)
    mae = np.mean(abs_diff, axis=1)

    return mae

def mean_squared_error(query_vector, src_vectors):
    squared_diff = (src_vectors - query_vector) ** 2
    mse = np.mean(squared_diff, axis=1)

    return mse

def cosine_similarity(query_vector, src_vectors):
    query_norm = np.linalg.norm(query_vector)
    normalized_query = query_vector / query_norm
    src_norms = np.linalg.norm(src_vectors, axis=1)
    normalized_src = src_vectors / src_norms[:, np.newaxis]

    cosine_similarity = np.dot(normalized_src, normalized_query)

    return cosine_similarity

def correlation_coefficient(query_vector, src_vectors):
    return np.corrcoef(query_vector, src_vectors)[:-1, -1]
```

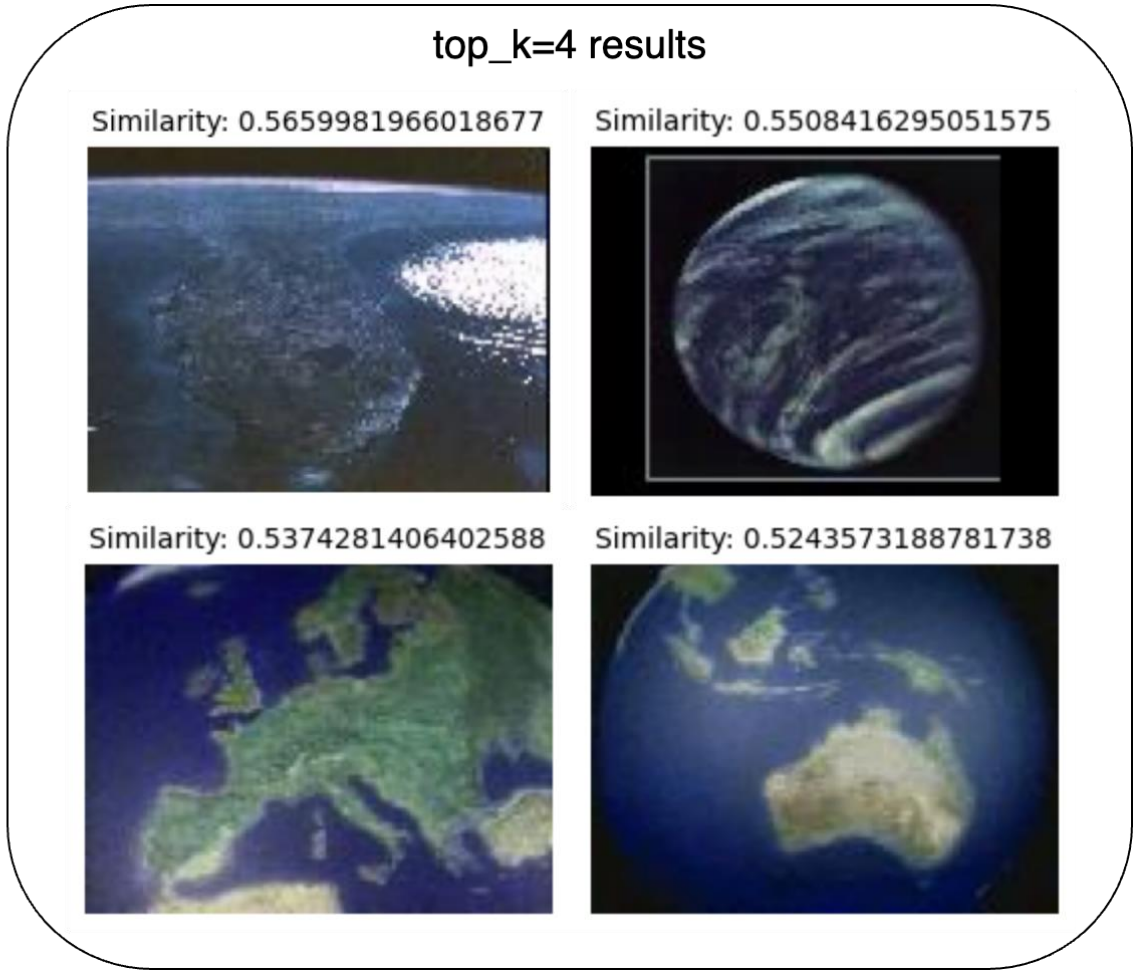
```python
def ranking(
  preprocessed_query_image,
  preprocessed_src_images,
  top_k=10
):
    scores = cosine_similarity(
        preprocessed_query_image,
        preprocessed_src_images
    )
    ranked_list = np.argsort(scores)[::-1][:top_k]
    scores = scores[ranked_list]

    return ranked_list, scores
```

Apply pre-trained ViT

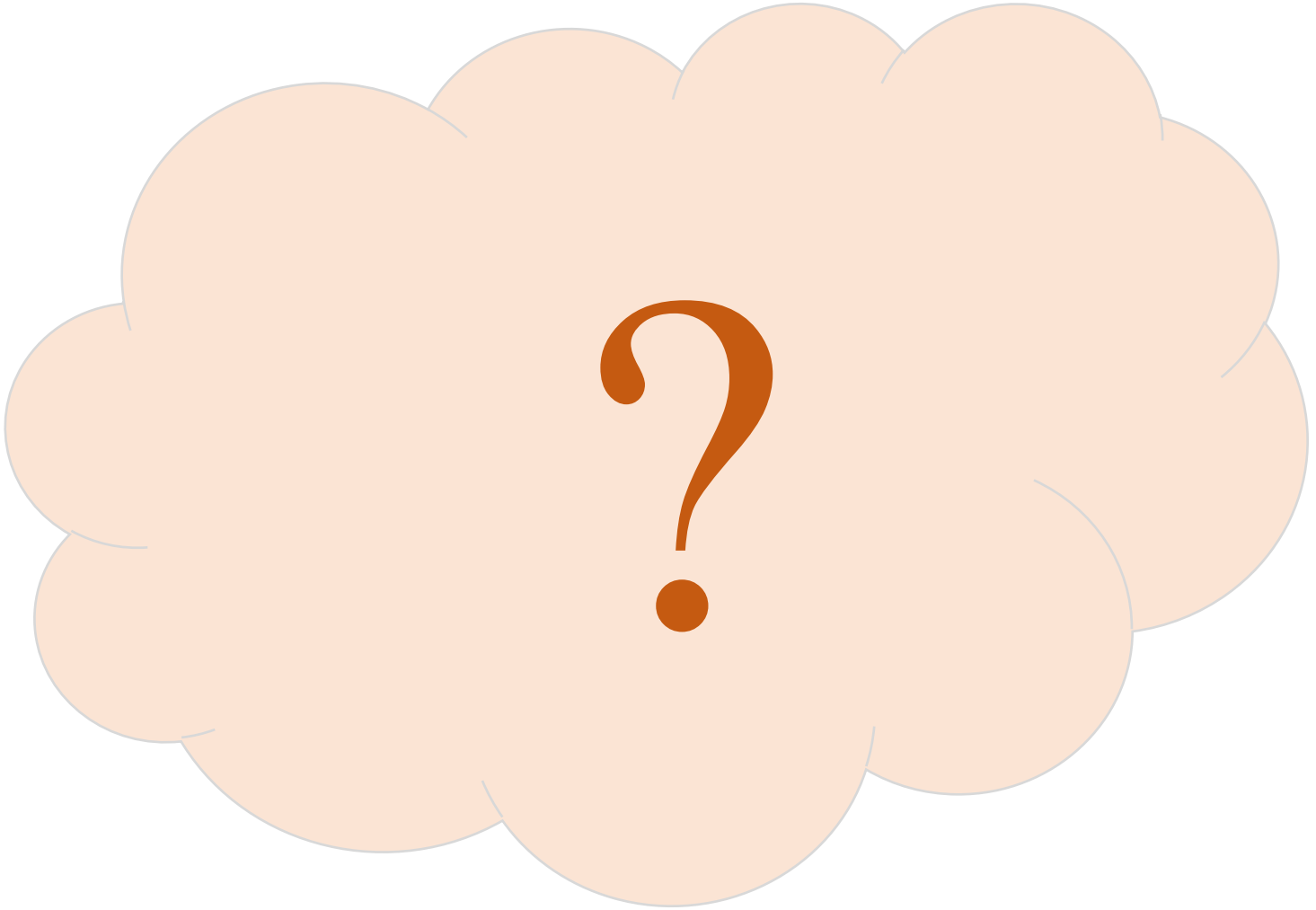
❖ ViT results

Query Image

Ranking



Question




Thank you!