Transfer learning-based package for image retrieval

Nov 2023

Introduction

- **Objective**:

  Develop a simple content-based image retrieval (**CBIR**) system that compares features extracted from a collection of images (image database) to those of an online image, identifying similarities and discerning patterns.

- **Approach**:

  Implement **Transfer Learning** (TL) technique through the **ResNet50** deep learning model pre-trained on ImageNet for feature extraction from both local and online images.

- **Application**:

  Focus on comparing local images against a target online image to find the most similar ones, based on **Cosine** similarity.

Image Preprocessing (first function)

- **Purpose**:

  To pre-process a collection of images (from an image database) to standardise them for feature extraction.

- **Resizing Images**:

  All images are standardised to a uniform size (i.e., 224x224 pixels for ResNet50) to ensure consistency in feature extraction.

- **Pre-processing** [1]:

  The images are converted from RGB to BGR, then each colour channel is zero-centred with respect to the ImageNet dataset, without scaling.

- **Feature Extraction** [2, 3, 4]:

  The ResNet50 model is adapted to extract features from the Average Pooling Layer, just before the fully connected layers.

Image Preprocessing (first function)

- **Feature Normalisation**:

  Features are normalised using the L2 norm. L2 normalisation scales the feature vector so that the sum of the squares of its elements equals 1.

- **Save features**:

  Normalised features are stored for quicker subsequent searches.

- **Multi-processing**:

  Parallel processing (default = 4 processes) is utilised to handle large volumes of images simultaneously, enhancing processing speed.

Image Indexing (second function)

- **Purpose**:

  To identify and sort similar images in a dataset by comparing them to a reference **online** image.

- **Feature extraction for Online image**:

  Features are extracted from the online image using the same ResNet50 model.

- **Similarity Metric**:

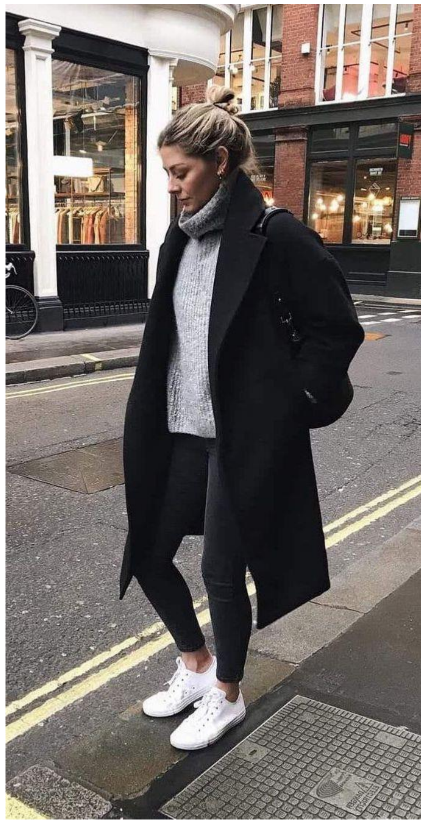
  Cosine similarity is employed to measure the closeness between feature sets of local and online images.

- **Output**:

  A list of tuples, each containing an image ID from the local collection and its corresponding similarity score, sorted in descending order of similarity.

Main Findings for top K using a fashion image dataset

- top K (id, **cosine**) = [(175, **0.651**), (338, **0.641**), (239, **0.616**), (279, **0.611**), (132, **0.603**)]


online image

id 175

id 338

id 239

id 279

id 132

Main Findings for top K

- The similarity scores ranged from 0.603 to 0.651, indicating a moderate to high level of similarity.
- Images with IDs 175, 338, 239, 279, and 132 were identified as the most similar to the online reference image.
- Image 175 scored the highest with a cosine similarity of 0.651, suggesting it is the closest match.
- The CBIR system seems to detect fashion products that are similar to the product shown in the online image.
- Beyond identifying similar products, it seems that it captures contextual elements within the images. For instance, it recognises patterns and similarities in backgrounds, such as roads, buildings, and urban settings.
- The case of ID 239 presents an interesting result where the CBIR system identified a high degree of similarity between the long coat in the online image and the long dress in the dataset local image. This may suggest that the system sometimes generalises features such as shape and length, which can lead to conflating distinct items like dresses and coats due to their visual similarities.
- Lastly, while ID 279 was ranked fourth, indicating a high similarity score, a closer visual inspection suggests a lower relevance when we consider specific attributes such as the style and the type of clothing.

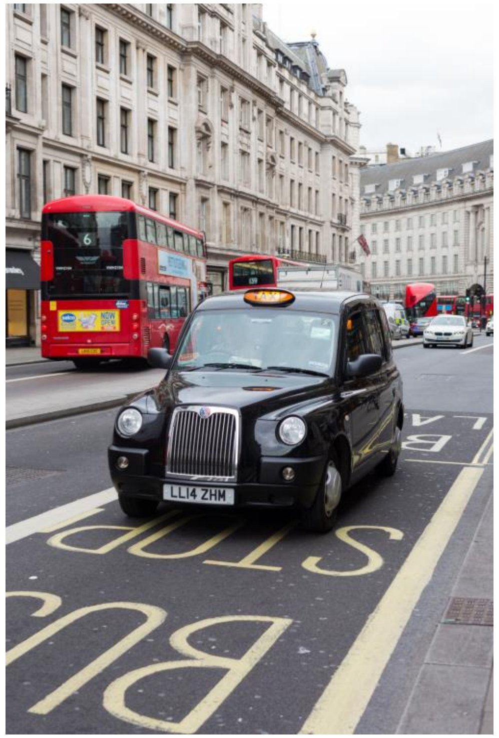
References

1. https://www.tensorflow.org/api_docs/python/tf/keras/applications/resnet50/preprocess_input
2. He, K., Zhang, X., Ren, S., & Sun, J. (2015). Deep Residual Learning for Image Recognition. arXiv preprint arXiv:1512.03385. Available at: https://arxiv.org/abs/1512.03385
3. Gkelios, S., Sophokleous, A., Plakias, S., Boutalis, Y., & Chatzichristofis, S. A. (2021). Deep convolutional features for image retrieval. Expert Systems with Applications, 177, https://doi.org/10.1016/j.eswa.2021.114940
4. https://keras.io/api/applications/#usage-examples-for-image-classification-models

Appendix

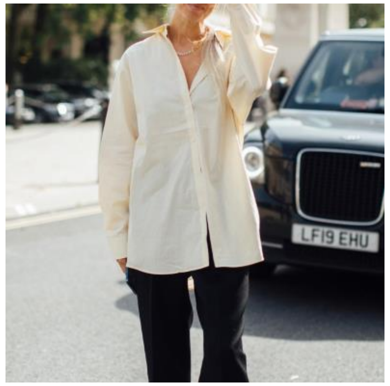
Non-fashion online image to check the impact of the image background

- top K (**id**, **cosine**) = [(**51**, **0.4974**), (**307**, **0.4846**), (**175**, **0.4662**), (**420**, **0.4440**), (**282**, **0.4228**)]
- In the second-ranked image, a black cab can be observed in the background.



online image

id 51



id 307

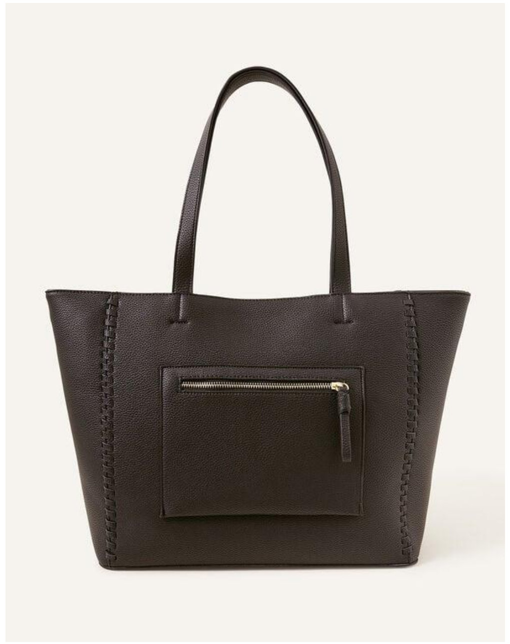
id 175

id 420

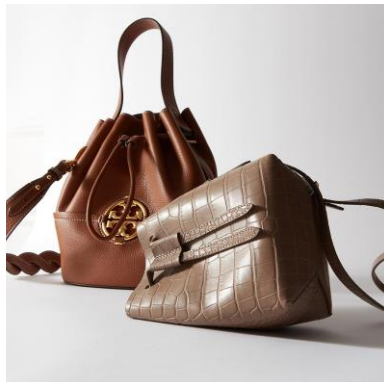
id 282

Women’s handbags

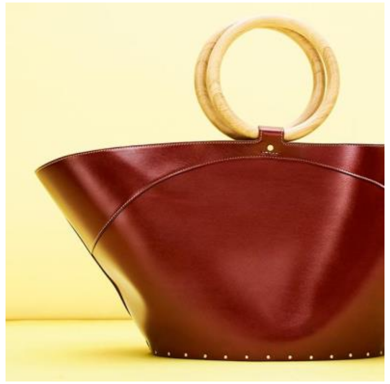
- top K (id, **cosine**) = [(116, **0.6249**), (382, **0.6074**), (357, **0.5312**), (458, **0.5212**), (428, **0.5199**)]



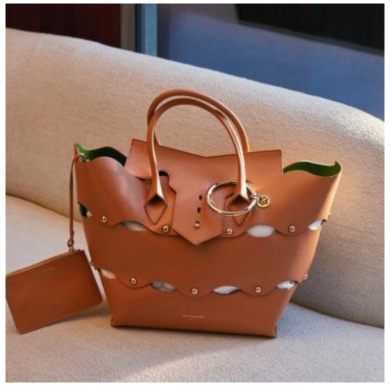
online image



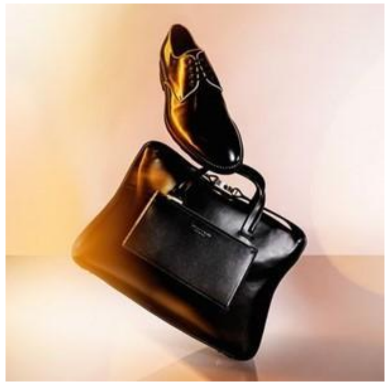
id 116



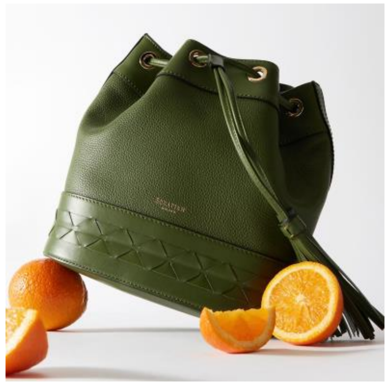
id 382



id 357



id 458

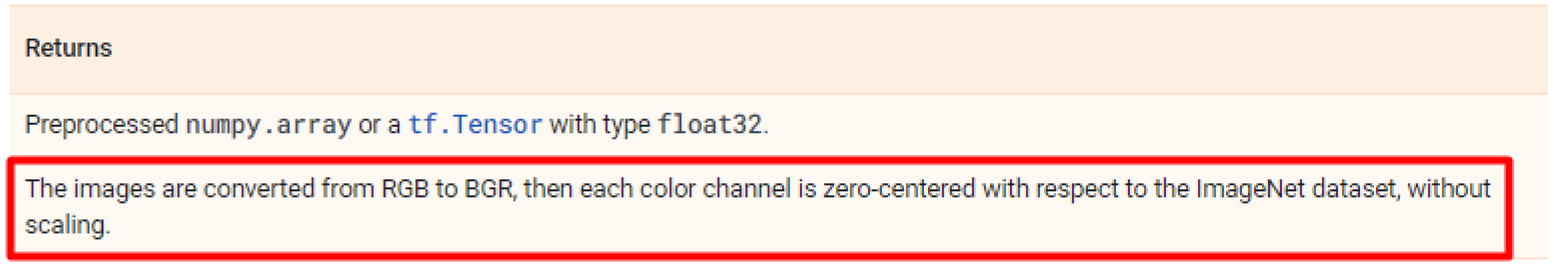


id 282

Preprocess input for ResNet50

- **Pre-processing** (reference):

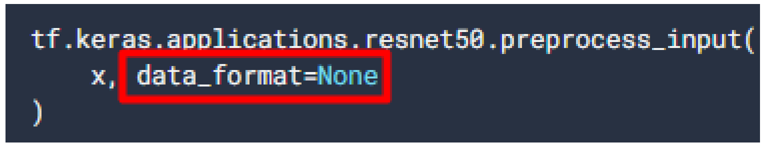
  The images are converted from RGB to BGR, then each colour channel is zero-centred with respect to the ImageNet dataset, without scaling.



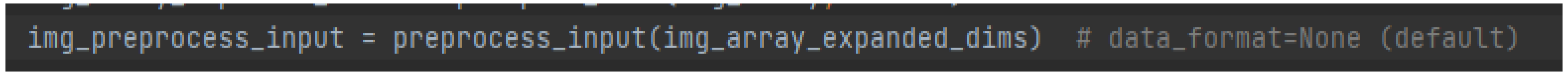
Returns

Preprocessed `numpy.array` or a `tf.Tensor` with type `float32`.

The images are converted from RGB to BGR, then each color channel is zero-centered with respect to the ImageNet dataset, without scaling.



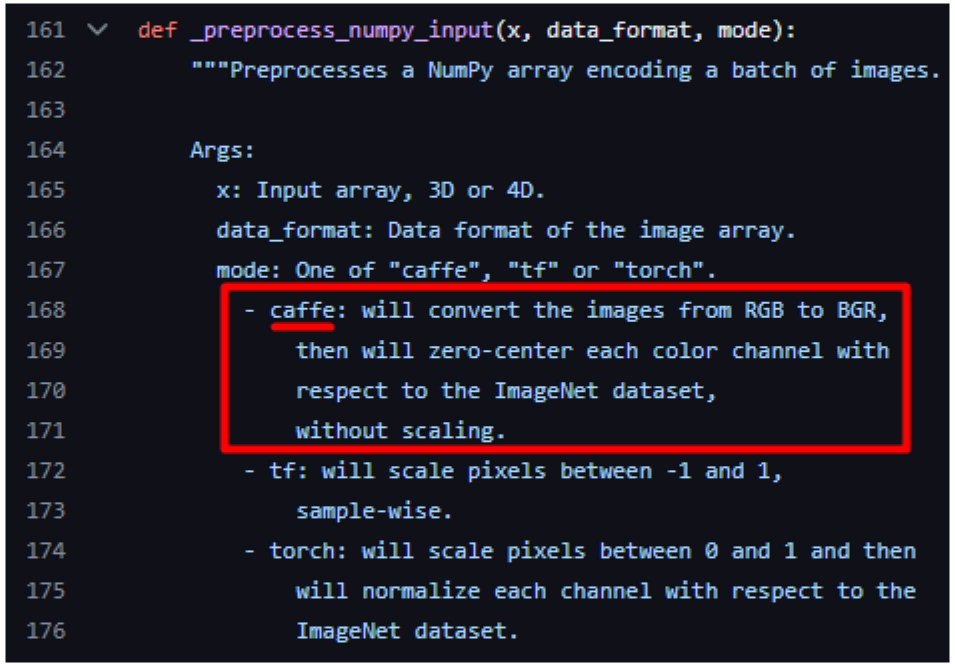
```
tf.keras.applications.resnet50.preprocess_input(
    x, data_format=None
)
```



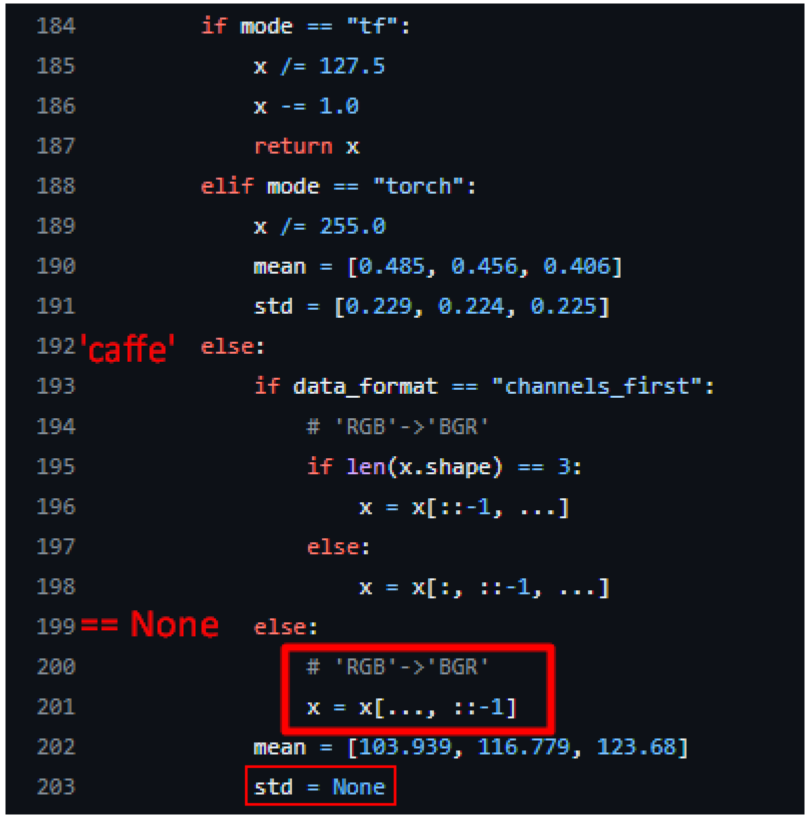
```
img_preprocess_input = preprocess_input(img_array_expanded_dims)  # data_format=None (default)
```

Preprocess input for ResNet50

- https://github.com/keras-team/keras/blob/master/keras/applications/imagenet_utils.py#L72



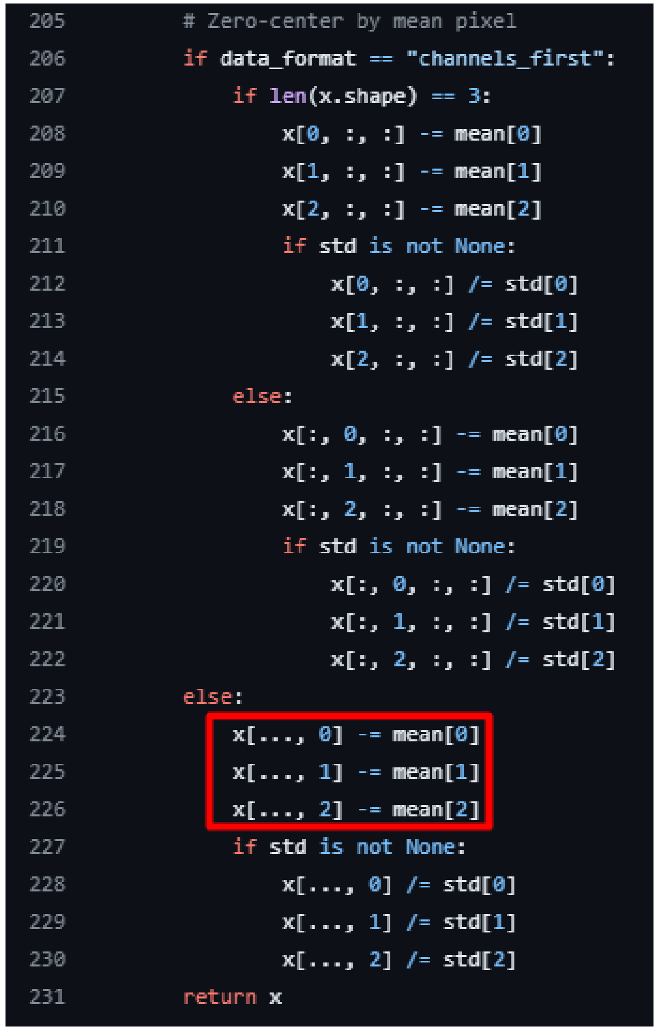
```
def _preprocess_numpy_input(x, data_format, mode):
    """Preprocesses a NumPy array encoding a batch of images.

    Args:
      x: Input array, 3D or 4D.
      data_format: Data format of the image array.
      mode: One of "caffe", "tf" or "torch".
        - caffe: will convert the images from RGB to BGR,
            then will zero-center each color channel with
            respect to the ImageNet dataset,
            without scaling.
        - tf: will scale pixels between -1 and 1,
            sample-wise.
        - torch: will scale pixels between 0 and 1 and then
            will normalize each channel with respect to the
            ImageNet dataset.
```



```
    if mode == "tf":
        x /= 127.5
        x -= 1.0
        return x
    elif mode == "torch":
        x /= 255.0
        mean = [0.485, 0.456, 0.406]
        std = [0.229, 0.224, 0.225]
    else:
        if data_format == "channels_first":
            # 'RGB'->'BGR'
            if len(x.shape) == 3:
                x = x[::-1, ...]
            else:
                x = x[:, ::-1, ...]
        else:
            # 'RGB'->'BGR'
            x = x[..., ::-1]
        mean = [103.939, 116.779, 123.68]
        std = None
```

Preprocess input for ResNet50

- https://github.com/keras-team/keras/blob/master/keras/applications/imagenet_utils.py#L72



```python
        # Zero-center by mean pixel
        if data_format == "channels_first":
            if len(x.shape) == 3:
                x[0, :, :] -= mean[0]
                x[1, :, :] -= mean[1]
                x[2, :, :] -= mean[2]
                if std is not None:
                    x[0, :, :] /= std[0]
                    x[1, :, :] /= std[1]
                    x[2, :, :] /= std[2]
            else:
                x[:, 0, :, :] -= mean[0]
                x[:, 1, :, :] -= mean[1]
                x[:, 2, :, :] -= mean[2]
                if std is not None:
                    x[:, 0, :, :] /= std[0]
                    x[:, 1, :, :] /= std[1]
                    x[:, 2, :, :] /= std[2]
        else:
            x[..., 0] -= mean[0]
            x[..., 1] -= mean[1]
            x[..., 2] -= mean[2]
            if std is not None:
                x[..., 0] /= std[0]
                x[..., 1] /= std[1]
                x[..., 2] /= std[2]
        return x
```

Preprocess input for ResNet50

- **Re-generate the results of ‘preprocess_input’**
  - **Reversing Colour Channels**: This line reverses the colour channels of the image. The original image is in RGB format (Red, Green, Blue), but ResNet50 was trained on images in BGR format (Blue, Green, Red). The [::-1] slices the array in the reverse order along the last dimension (colour channels).

    ```
    x = img_array[..., ::-1]
    ```

  - **Defining the Mean Values**: These are the mean values for each colour channel (BGR) that were used when the ResNet50 model was trained. They are used for mean subtraction, a form of normalisation.

    ```
    mean = [103.939, 116.779, 123.68]
    ```

  - Subtracting the Mean from Each Channel: Here, the mean for each channel (B, G, and R) is subtracted from the respective channels of the image. This mean subtraction is a common preprocessing step in deep learning for image data. It normalises the data, making the model less sensitive to dynamic range variations in the input images.

    ```
    x[..., 0] -= mean[0]
    x[..., 1] -= mean[1]
    x[..., 2] -= mean[2]
    ```

Preprocess input for ResNet50

- In summary, these steps are preparing an image for classification with the ResNet50 model by
  1. resizing it,
  2. converting it to an array,
  3. reordering the colour channels, and
  4. normalising the pixel values.

```
# Resize the image
img = image.load_img(input_arg, target_size=IMG_TARGET_SIZE)

# Convert it to an array
img_array = image.img_to_array(img)

# Reorder the colour channels
x = img_array[..., ::-1].copy()

mean = [103.939, 116.779, 123.68]

# Normalise the pixel values
x[..., 0] -= mean[0]
x[..., 1] -= mean[1]
x[..., 2] -= mean[2]
```

Preprocess input for ResNet50

- **Pre-processing** (reference):

  The images are converted from RGB to BGR, then each colour channel is zero-centred with respect to the ImageNet dataset, without scaling.

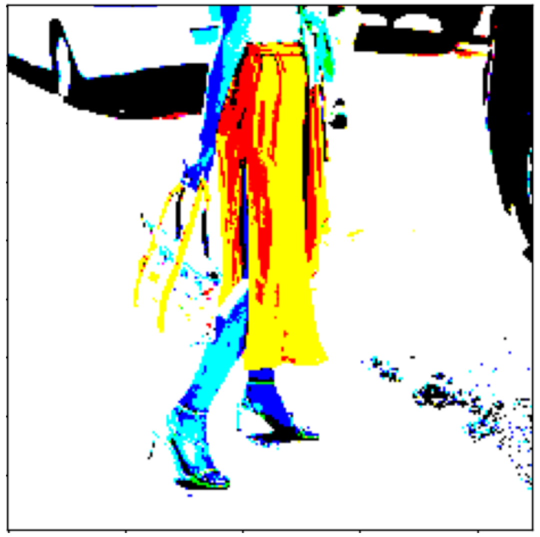
original

1080x1080

resize

224x224

preprocess_input for ResNet50



224x224

L2 norm

- The L2 norm, often referred to as the Euclidean norm, is a measure of the magnitude (or, length) of a vector. It is defined for a vector A = (a1, a2, …, ap) as the square root of the sum of the squared elements of the vector. Mathematically, it can be represented as:

$$||A|| = \sqrt{a_1^2 + a_2^2 + \cdots + a_p^2}$$

- Let's say, A = (5, 0, 3, 0, 2, 0, 0, 2, 0, 0), then $||A|| = \sqrt{5^2 + 3^2 + 2^2 + 2^2} = 6.4807$

- Normalise A vector with L2 norm, $A_{norm} = \frac{A}{||A||} = (0.7715, 0, 0.4629, 0, 0.3086, 0, 0, 0.3086, 0, 0)$

- The sum of squares of A_norm's elements equals 1 $\sum_i A_{norm,i}^2 = 0.7715^2 + 0.4629^2 + 0.3086^2 + 0.3086^2 = 1$

Cosine Similarity

- The cosine similarity formula is a mathematical equation used to measure the cosine of the angle between two non-zero vectors in an inner product space. This measure is used to determine how similar the vectors are to each other. The formula is given by:

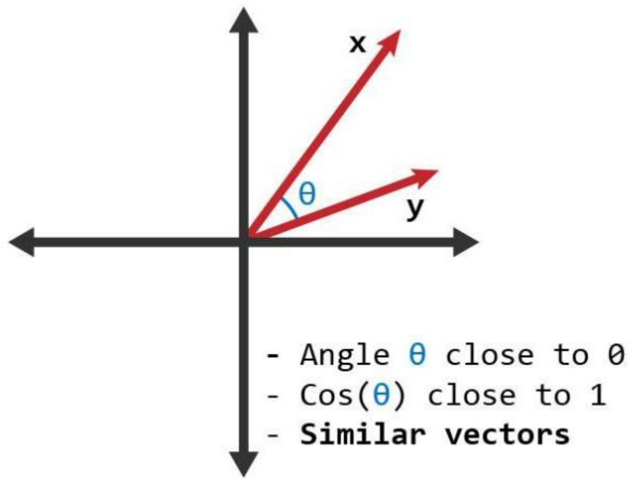
$$similarity(A,B) = cos(\theta) = \frac{A \cdot B}{||A||\,||B||}$$

- where $A \cdot B$ is the dot product of vectors A = (a1, a2, …, ap) and B = (b1, b2, …bp), defined as

$$A \cdot B = a_1 b_1 + a_2 b_2 + \cdots + a_p b_p$$

- where ||A|| is the Euclidean norm of vector A = (a1, a2, a3, …, ap), defined as

$$||A|| = \sqrt{a_1^2 + a_2^2 + \cdots + a_p^2}$$


x
y
θ
- Angle θ close to 0
- Cos(θ) close to 1
- Similar vectors