NATIONAL UNIVERSITY OF HO CHI MINH CITY
UNIVERSITY OF INFORMATION TECHNOLOGY

Project report

Subject: Multimedia Information Retrieval

Class: CS336.O11.KHCL - EN

Topic: Image Retrieval with Deep Learning

Students:
Nguyễn Thanh Phúc – 20521769
Trần Minh Phúc – 20521782
Quách Minh Triết – 20522057
Nguyễn Minh Nhật – 20521708
Nguyễn Anh Tuấn – 20522114

Teacher:
Ngô Đức Thành

Contents

1 Introduction

In the era of rapidly developing information technology, image retrieval is one of the essential needs of users. Image retrieval allows users to search for information by entering an image or describing an image. To perform image retrieval effectively, a powerful image processing and analysis system is required.

One of the most common methods for image retrieval is to use feature extraction. Feature extraction is the process of extracting features from images, which can describe the properties of the image, such as color, shape, structure, etc. These features are then used to compare with the features of images in the database to find the most relevant images to the query.

In this project, we will investigate the use of feature extraction in image retrieval. We will use 5 popular deep learning models, Xception, InceptionResNetV2, VGG19, DenseNet121, and NasNetMobile, to extract features from images. Then, we will use these features to perform image retrieval on a large image dataset.

2 Datasets

Oxbuild and Paris Dataset are two widely used image datasets of buildings in image retrieval research. Each dataset contains over 5,000 images of buildings in Oxford, England and Paris, France. The images were collected from Flickr and manually labeled.

The Oxbuild Dataset contains 5,062 images of 11 buildings in Oxford. The Paris Dataset contains 6,412 images of 12 buildings in Paris. The images in these two datasets are high-resolution and captured from multiple angles.

Oxbuild and Paris Datasets are useful datasets for image retrieval research. They provide a rich resource for training and evaluating image retrieval models.

Figure 1: Oxford Buildings Dataset

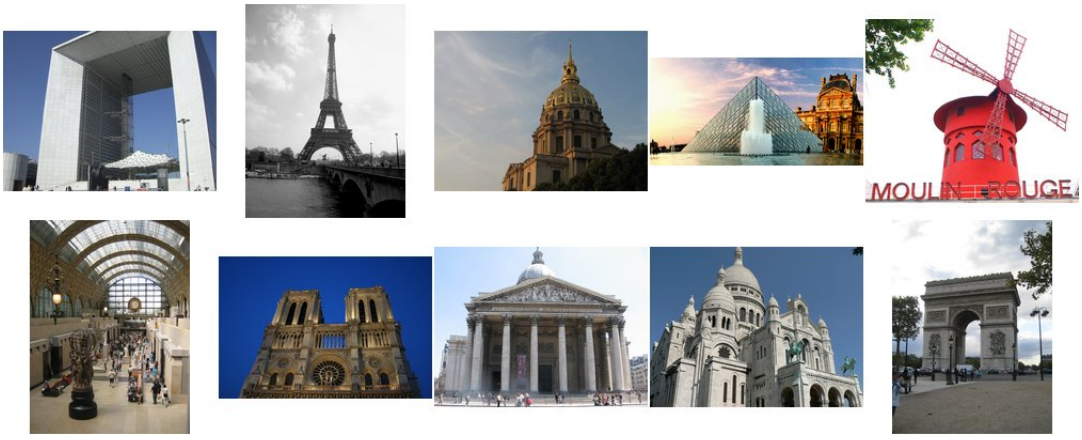

Figure 2: Paris Buildings Dataset

3 Approach to the problem

3.1 Workflow

By using input image as query, we perform feature extraction to convert it into a feature vector. The popular method is to employ the simple Bagof-Visual-Words methodology, which is both accessible and understandable. A recently developed and effective method, based on a deep learning model. In this project, our team will develop image retrieval system with 5 deep learning models namely: Xception, InceptionResNetV2, VGG19, DenseNet121, NasNetMobile and proceed to evaluate and compare them; however, when deploying system on the website, we will only use the most effective model (Xception).

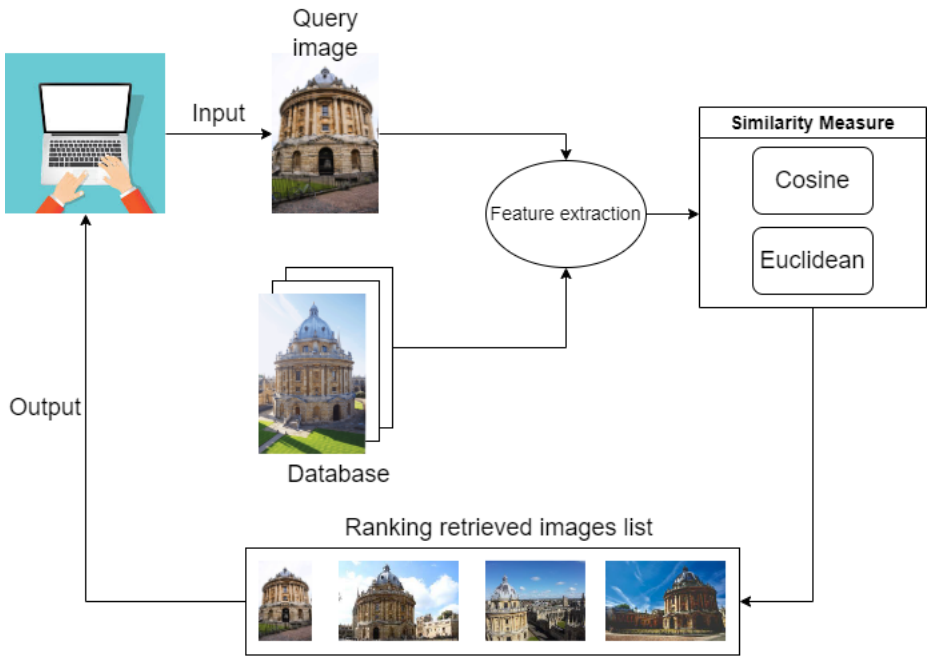

Figure 3: Workflow

The system handles the image retrieval problem with:

- Input: A image (maybe full or partial picture).
- Output: List of ranked relevant images.

Regarding to database for system, we will preprocess and use VGG-19 network to collect vector features and synthesize these vectors into a vector and stored.

For each query image, we perform the same feature extraction steps.

After obtaining the feature vector, calculate the similarity between the representation vector of the query image and the vector representing the image in the database.

The result returns to users that is ranking list of high similarity sorted descending order.

3.2 Image feature extraction

Image feature extraction is the process of identifying and extracting important visual patterns or features from an image, which can be used to represent the image in a compact and meaningful way. The process of feature extraction involves

transforming the raw image data into a set of feature vectors that capture the key characteristics of the image. There are various techniques used for feature extraction, including:

1. *Histogram of Oriented Gradients* (HOG): This technique involves computing a histogram of gradients in different regions of an image to represent the local edge directions.
2. *Scale-Invariant Feature Transform* (SIFT): This technique involves identifying key points in an image that are invariant to scale, rotation, and illumination changes.
3. *Convolutional Neural Networks* (CNN): CNNs are a type of deep learning technique that can automatically learn hierarchical features from raw image data.
4. *Local Binary Patterns* (LBP): This technique involves computing a binary code for each pixel in an image based on the values of its neighboring pixels.

In this project, we will ultilize some state-of-the-art of Convoluational Neural Networks to extract the feature of image that we use to build CBIR system. Specifically, our team will develop image retrieval systems based on some deep convolutional neural network models such as: Xception, InceptionResNetV2, VGG19, DenseNet121, NasNetMobile. We will present the basic knowledge of the methods and apply them to image feature extraction for image retrieval system.

4 Deep learning models are used

4.1 Xception

The Xception model was developed by Francois Chollet and published in the paper "Xception: Deep Learning with Depthwise Separable Convolutions" in 2017.

The Xception model was developed based on the Inception V3 model. The Inception V3 model has a complex architecture and requires a lot of resources to train. The Xception model was designed to simplify the architecture of the Inception V3 model, while still ensuring its effectiveness.

Theoretical basis

The Xception model uses depthwise separable convolutions instead of conventional convolutions. Depthwise separable convolutions are divided into two steps:

- **The first step** is to perform a convolution operation on each individual channel of the image.
- **The second step** is to perform a convolution operation on the output channels of the first step.

Using depthwise separable convolutions can reduce the number of convolution operations required, which can simplify the model structure and save resources.

Architecture of Xception

The overall architecture of Xception is clearly described in the figure below. The architecture consists of three parts: Entry flow, Middle flow, and Exit flow. There are a total of 14 modules in the architecture, with 4 modules in Entry flow, 8 modules in Middle flow, and 2 modules in Exit flow. Each module is a combination of depthwise separable convolution and pooling layer. The final layers of the architecture are fully connected layers.

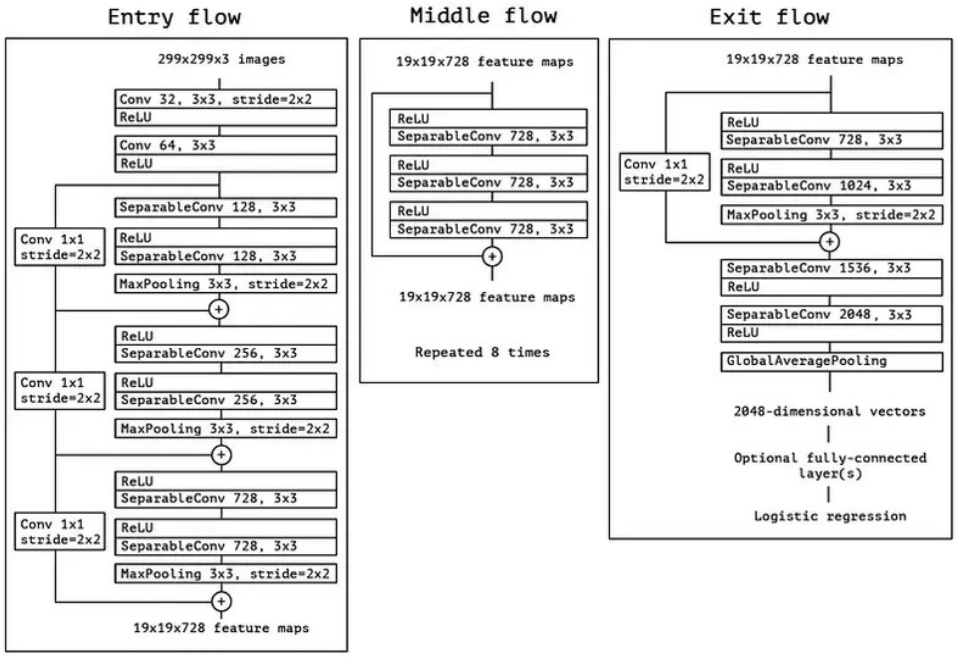

Figure 4: Overall architecture of Xception

4.2 Inception-ResNet-V2

Inception-ResNet-V2 is a convolutional neural network architecture that was developed by Christian Szegedy and colleagues at Google AI and published in the paper

"Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning" in 2016.

Inception-ResNet-V2 is a version of the Inception V3 architecture that is simplified by using residual connections. Inception V3 is a complex architecture that is computationally expensive to train. Inception-ResNet-V2 is designed to improve the learning ability of Inception V3 while still maintaining its effectiveness.

Theoretical basis

Inception-ResNet-V2 model uses conventional convolution operations and depthwise separable convolution operations. Conventional convolution operations are used to extract features from images. Depthwise separable convolution operations are used to reduce the number of convolution operations required, thereby simplifying the model structure and saving resources.

Architecture of Inception-ResNet-V2

The architecture of the Inception-ResNet-V2 model consists of 153 layers, divided into 7 blocks. Each block consists of a number of conventional convolution operations and depthwise separable convolution operations.

The first and last blocks of the Inception-ResNet-V2 model include resizing operations to adjust the size of the input and output images. The remaining blocks include conventional convolution operations and depthwise separable convolution operations to extract features from images.

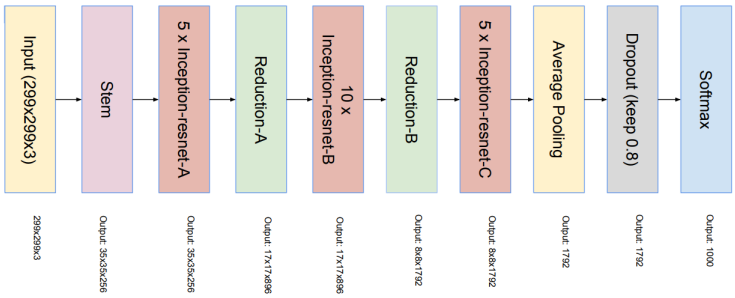

Figure 5: Schema for Inception-ResNet-v2

4.3 VGG-19

The VGG19 model was developed by Karen Simonyan and Andrew Zisserman at the University of Oxford and was published in the paper "Very Deep Convolutional

Networks for Large-Scale Image Recognition" in 2014.

The VGG19 model was developed based on the VGGNet model, which was developed by the same team of authors in 2014. The VGGNet model has a simpler structure than other CNN models, making it easier to understand and deploy. However, the VGGNet model has only 16 layers, while other CNN models typically have 20 layers or more.

The VGG19 model was developed to address this issue. The VGG19 model has 19 layers, which improves the model's image classification capabilities.

Theoretical basis

The VGG19 model uses conventional convolution operations to extract features from images. Conventional convolution operations are operations that compute the convolutions of neighboring values in an image.

The VGG19 model uses convolution operations with a filter size of 3 x 3. Convolution operations with a filter size of 3 x 3 can extract features from images in many different directions, which helps to improve the model's image classification capabilities.

Architecture of VGG-19

The architecture of the VGG19 model consists of 19 layers, divided into 5 blocks. Each block consists of a number of conventional convolution operations, followed by a max pooling layer.

- Layer 1: Image resizing to adjust the input image size from 224 x 224 x 3 to 112 x 112 x 64.
- Layer 2: Convolution operation with filter size of 3 x 3, number of output channels is 64.
- Layer 3, 4: Convolution operation with filter size of 3 x 3, number of output channels is 128.
- Layer 5, 6, 7, 8: Convolution operation with filter size of 3 x 3, number of output channels is 256.
- Layer 9, 10, 11, 12: Convolution operation with filter size of 3 x 3, number of output channels is 512.

- Layer 13, 14, 15, 16: Image resizing to reduce the output image size from 56 x 56 x 512 to 28 x 28 x 512.
- Layer 17, 18: Two fully-connected layers with 4096 nodes each.
- Layer 19: A fully-connected layer with 1000 nodes followed by a softmax layer to predict the probability of each label.

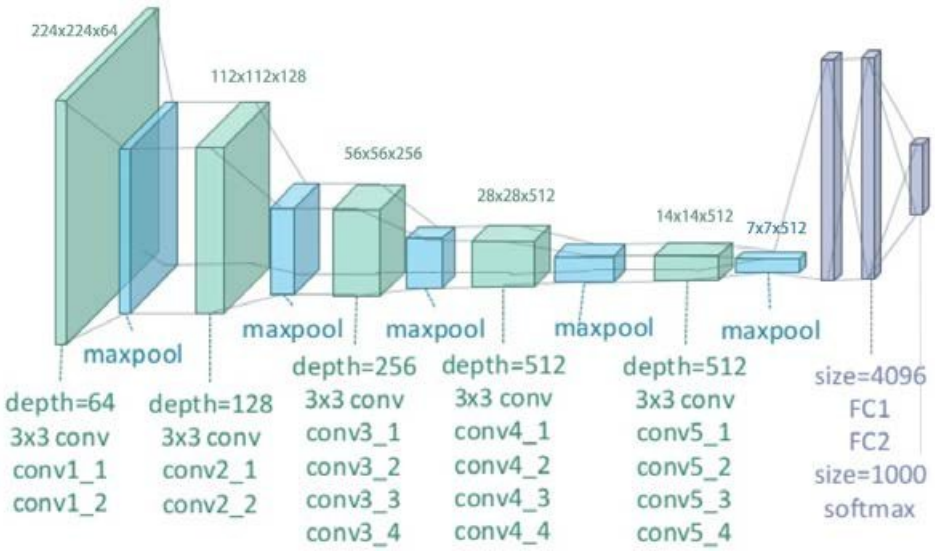

Figure 6: Architecture VGG19

4.4 DenseNet121

The DenseNet model was first proposed in the paper "Densely Connected Convolutional Networks" by Gao Huang, Zhuang Liu, Kaiming He, and Jian Sun at the International Conference on Machine Learning (ICML) in 2016.

Architecture of DenseNet121

DenseNet is a further development of DL in Computer vision. The DenseNet architecture is similar to ResNet, but it has a few key differences. DenseNet consists of Dense-blocks and transition layers, as shown in the figure below.

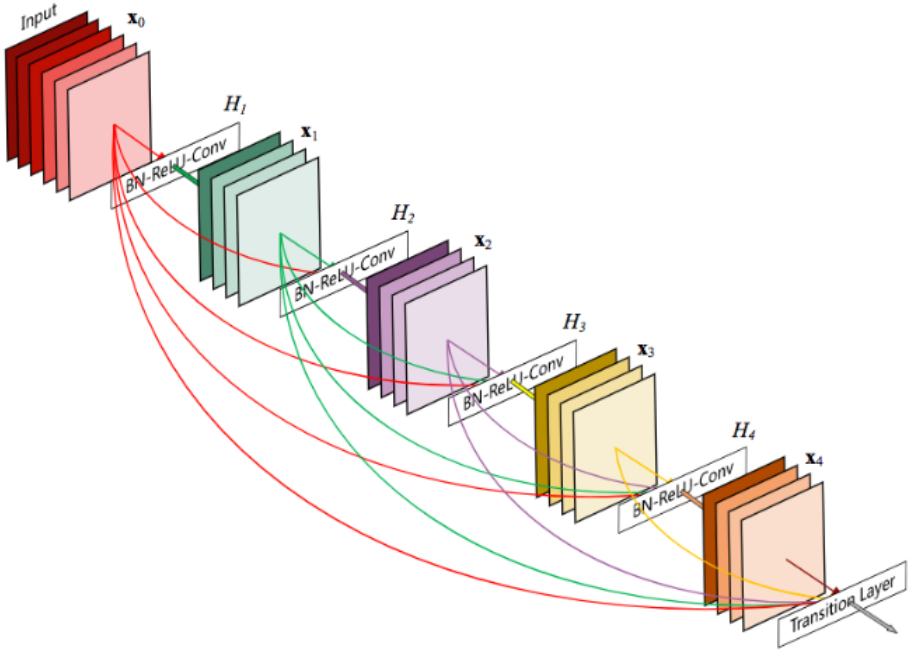

Figure 7: Dense-block and transition layer

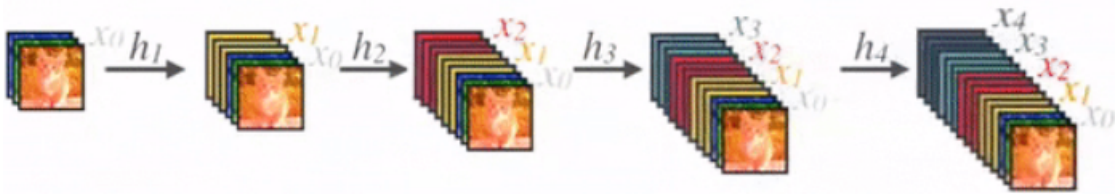

Figure 8: Dense-block and transition layer

A Dense-block is a group of layers that are connected to all the layers before it. A layer includes:

- Batch normalization
- ReLU activation
- 3×3 Conv

The authors found that using pre-activation (BN and ReLU before Conv) is more effective than post-activation. Unlike ResNet, DenseNet concatenates all feature maps together. On the other hand, concatenation is not possible when feature maps have different sizes. Therefore, in Dense blocks, feature maps in each layer have the same size.

However, downsampling is typically performed by CNNs. In DenseNet, it is performed by transition layers. The architecture consists of:

- Batch normalization
- 1×1 Conv
- 2×2 Average pooling

The DenseNet121 model consists of 121 layers, divided into 12 blocks. Each block consists of a Dense Block and a Transition Block.

The first block of the model is a Dense Block with 64 output channels. Blocks 2 through 11 are Dense Blocks with output channels of 128, 256, 512, 1024, 1024, 1024, 1024, and 1024, respectively. Block 12 is a Transition Block with 1024 output channels. Blocks 13 through 14 are Dense Blocks with output channels of 2048 and 2048, respectively. Block 15 is a Transition Block with 2048 output channels. Block 16 is a fully connected layer with 1000 nodes. Block 17 is a softmax layer to predict the probability of each label

4.5 NasNetMobile

NasNetMobile is a small, lightweight convolutional neural network (CNN) model developed by Google AI. It is a variant of the larger NasNet model, which was trained on the ImageNet dataset.

Architecture of NasNetMobile

Each block in NasNetMobile consists of the following components:

- Convolutional neural network (CNN) with filter size 3 x 3 or 1 x 1: This CNN applies convolution operations to extract features from the input image.
- Convolutional neural network (CNN) with filter size 1 x 1: This CNN reduces the number of output channels of the network.
- Linear convolutional neural network (CNN): This linear CNN applies linear transformations to combine features from previous layers.

NasNetMobile uses a "bottleneck" architecture to reduce the size of the output. This helps to improve the efficiency of the model and reduce the computational cost.

The blocks in NasNetMobile are connected to each other in a random way. This allows the model to find the optimal structure for the specific task.

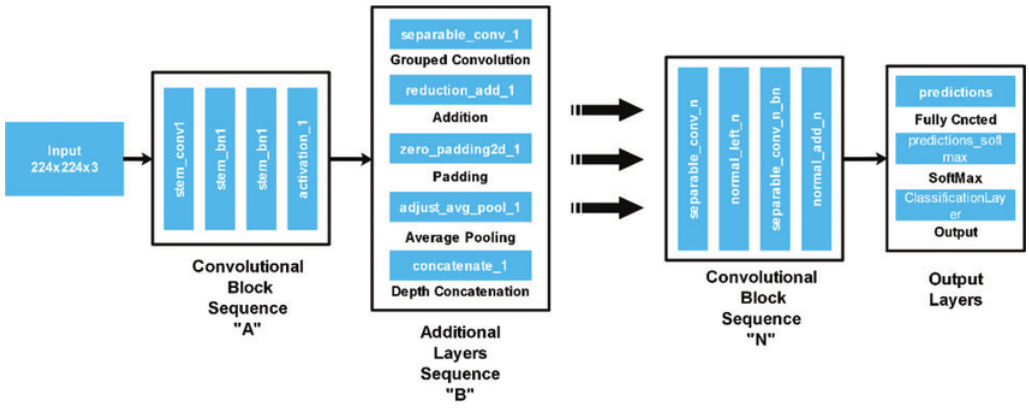

Figure 9: NasNetMobile architecture

5 Similarity calculation

Once we have a database of feature vectors, we need to determine the distance measure (or similarity measure) so that we can compare the feature vectors of the images in the corpus with the feature vectors of the query image, to find the images that are most similar to the query image.

Our image search system uses two common distance measures, Cosine Distance and Euclidean Distance. Feature vector of the query image after extraction will be compared with all available feature vectors in the database, the results obtained will be ranked from smallest to largest because the two images are more similar. when the distance between two feature vectors is smaller.

5.1 Cosine distance

Cosine similarity is the measure of similarity between two vectors, calculated as the *cosine* of the angle between those two vectors. Two vectors are more similar if the similarity approaches 1 (the angle between the two vectors approaches 0). Therefore, we define the *Cosine distance* as 1 minus the *cosine* of the angle between those two vectors.

With two feature vectors u and v, the cosine distance between them is calculated by the formula:

$$Similarity = 1 - cos(\phi) = \frac{u \cdot v}{\|u\|\|v\|}$$

However, Cosine similarity only calculates the angle between two vectors and in the direction, not using magnitude, so it will cause information loss leading to reduced performance.

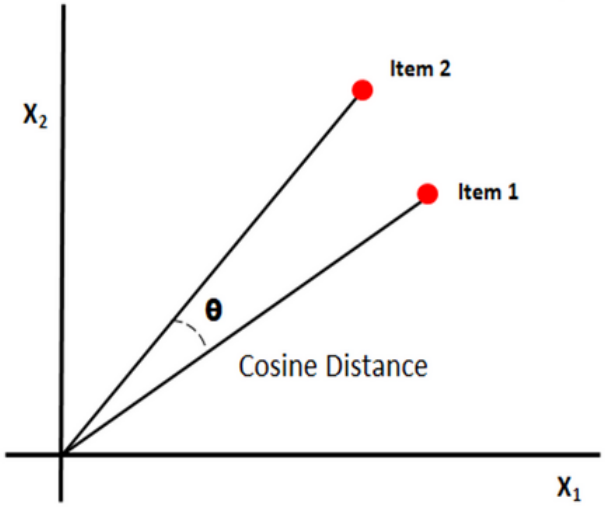

Figure 10

5.2 Euclidean distance

Euclidean Distance is the most common and easy to understand method in practice. Used to calculate the length distance of a line segment joining two points, commonly referred to as the L-2 distance.

The formula calculating Euclidean Distance:

$$D(u, v) = \sqrt{\sum_{i=1}^{n}(u_i - v_i)^2}$$

Euclidean distance is especially effective with low-dimensional data because it is easy to understand, easy to implement, and gives quite good results. However, *Euclidean Distance* is affected by the unit of feature and the number of dimensions of the vector, so when applied in practice with large and multidimensional datasets, *Euclidean Distance* becomes inefficient.

6 System evaluation

As introduced in the previous section, the dataset has 55 preassigned ground truth query images containing images that are considered relevant to the query, divided into 4 groups: Good, OK, Junk and Bad based on the level of view clarity of the object. In this project, we will combine two files "good" and "ok" to evaluate the system and ignore “bad” since it is unaffected; each image returned by the system

in both result files is counted as "relevant," and if none, it's counted as "irrelevant." On this basis, we will assess the system's performance using two datasets.

We use the *Mean Average Precision (mAP)* measure to evaluate the system as well as compare tested methods. It is a method of evaluating the performance of an information query system. In the process of querying information, the system will search for documents related to the query keyword and sort them by relevance.

mAP is used to measure the quality of an information retrieval system by calculating the average of the precision value at different recall thresholds. Precision is the ratio of the number of related documents returned to the total number of documents returned, and recall is the ratio of the number of related documents returned to the total number of related documents.

Mean average precision is calculated as the average of the average precision of the queries. Average precision is the average of precision at the positions the return is said to be relevant.

$$AP = \frac{\sum_{k=1}^{n} P(k) * rel(k)}{\text{number of relevent documents}}$$

$$mAP = \frac{\sum_{q=1}^{Q} AP(q)}{Q}$$

Beside the *non-interpolated mAP*, we carry out extra *interpolated mAP* which calculate based on *11-point interpolated average precision*. For each information need, the interpolated precision is measured at the 11 recall levels of 0.0, 0.1, 0.2, ..., 1.0. For the precision-recall curve in Figure 3. For each recall level, we then calculate the arithmetic mean of the interpolated precision at that recall level for each information need in the test collection. A composite precisionrecall curve showing 11 points can then be graphed. The *interpolated precision* at a certain recall level r is defined as the highest precision found for any recall level $r' \geq r$:

$$p_{inter\ p}(r) = max_{r' \geq r}\, p(r')$$

7 Results, analysis and inferences

After carrying out some experiments, we receive the evaluation results that were calculated based on a set of 55 queries over which an object retrieval system provided being as follows:

- Regarding to *The Oxford Building Dataset*:

Detailed results with 55 queries: The Oxford Building Evaluation

Table 1

Method	Non-interpolated MAP	Interpolated MAP	Time (s)
Xception	**0.68**	**0.34**	**31.95**
InceptionResNetV2	0.6	0.27	106.79
VGG19	0.53	0.22	30.86
DenseNet121	**0.72**	**0.33**	**31.18**
NasNetMobile	0.58	0.22	97.97

- Regarding to *The Paris Building Dataset*:

Detailed results with 55 queries: The Paris Building Evaluation

Table 2

Method	Non-interpolated MAP	Interpolated MAP	Time (s)
Xception	**0.86**	**0.54**	**75.98**
InceptionResNetV2	0.81	0.47	42.94
VGG19	0.8	0.45	40.9
DenseNet121	**0.87**	**0.53**	**35.14**
NasNetMobile	0.78	0.44	34.83

Through the results obtained from the two datasets above, we have some comments:

- Considering MAP (non-interpolation and interpolation), *DenseNet121* and *Xception* are two deep neural network models that bring higher results than other methods.

- However, for a retrieval system, the time to search for results is also an important factor in the evaluation, so we find *DenseNet121* somewhat superior to *Xception*.

From the comments, we have concluded that *DenseNet121* will be deploy into the website as image retrieval system with interface and some retrieval functions.

8 Technologies Used

8.1 Frontend (FE)

8.1.1 JavaScript (JS)

Dynamic and Interactive Elements: JS is chosen for its ability to create dynamic and interactive elements on the user interface, enhancing user engagement.

Asynchronous Operations: Enables asynchronous operations, allowing the frontend to fetch data from the backend without reloading the entire page, leading to a seamless user experience.

8.1.2 Bootstrap

Responsive Design: Bootstrap is employed to ensure a responsive design that adapts to various screen sizes and devices, providing a consistent and user-friendly layout.

Aesthetic Appeal: The framework's pre-built components and styling options contribute to an aesthetically pleasing design, saving development time while maintaining visual coherence.

8.1.3 jQuery

DOM Manipulation: jQuery streamlines DOM manipulation, simplifying complex tasks like traversing and manipulating the HTML document. This leads to cleaner and more readable code.

Event Handling: The use of jQuery for event handling enhances the user experience by simplifying the process of capturing and responding to user interactions.

8.2 Backend (BE)

Flask

Simplicity and Flexibility: Flask is chosen for its simplicity, flexibility, and ease of use. Its minimalistic approach allows developers to focus on the specific requirements of the application without unnecessary complexities.

Compatibility with Python: Being written in Python, Flask aligns with the team's expertise, enabling seamless integration with existing Python libraries and tools.

Robust Foundation: Flask provides a robust foundation for building RESTful APIs, making it well-suited for handling image retrieval requests in a scalable and efficient manner.

8.3 Application Architecture

8.3.1 Client-Server Architecture

The decision to adopt a client-server architecture is rooted in the separation of concerns. The frontend is responsible for user interactions, ensuring a responsive and dynamic user interface. In contrast, the backend handles the processing of image retrieval requests, interacting with the image retrieval engine to deliver relevant results.

8.3.2 Frontend Structure

The user interface is crafted using a combination of HTML, CSS, and Bootstrap. This choice is driven by the need for a visually appealing and responsive design. JavaScript enhances the interface's interactivity, creating a more engaging user experience. jQuery further streamlines development by simplifying DOM manipulation and event handling.

8.3.3 Backend Structure

Flask is selected as the backend framework due to its simplicity and flexibility. It provides a RESTful API that efficiently handles image retrieval requests. The backend serves as the mediator between the frontend and the image retrieval engine, ensuring a smooth flow of data and responses.

9 Conclusion

Image retrieval is a problem that has been research and has significant applications in today's life. Organizing and searching images quickly will assist people in accessing and capturing information better. Through this project, our team has studied, researched and developing deep learning network models to build image retrieval system. Besides the experiment and evaluation on the required dataset (The Oxford Building Dataset), we also built models on The Paris Building Dataset and obtained some positive results. Not only do we stop at building and evaluating methods, we also deploy them into a website with interface and image retrieval functions. In the future, we will develop more systems to increase the accuracy and speed of processing by studying and using more modern deep learning networks to extract high-level features, combined with the use of more sophisticated measures to compare feature vectors (such as triplet loss), and additional use of search techniques on large image databases.

References

[1] A. Rosebrock, "Keras: Feature extraction on large datasets with Deep Learning", 27 May 2019. [Online]. Available: https://pyimagesearch.com/2019/05/27/keras-feature-extraction-on-large-datasets-with-deep-learning/.

[2] "Thử làm Google Image Search (Content based Image Retrieval) - Mì AI", 2 October 2021. [Online]. Available: https://miai.vn/2021/10/02/thu-lam-google-image-search-content-based-image-retrieval-mi-ai/.

[3] "Keras Applications," [Online]. Available: https://keras.io/api/applications/#usage-examples-for-image-classification-models.

[4] Wan J, Wang D, Hoi SCH, Wu P, Zhu J, Zhang Y, Li J. Deep learning for content-based image retrieval: a comprehensive study. In: ACM international conference on multimedia 2014. [Paper]

[5] Huang W, Qiang W. Image retrieval algorithm based on convolutional neural network. In: Selected paper from Common Sense Media Awards 2017. [Paper]

[6] Hanan A. Al-Jubouri, Sawsan M. Mahmmod. "A comparative analysis of automatic deep neural networks for image retrieval". [Paper]

[7] Nikhil V Shirahatti, Kobus Barnard. "Evaluating ImageRetrieval". [Paper]

[8] Mayank R. Kapadia, Dr. Chirag N. Paunwala. "Content based medical image retrieval system for accurate disease diagnoses using modified multi feature fused Xception model". [Paper]