HELWAN UNIVERSITY
Faculty of Computers and Artificial Intelligence
Computer Science Department

Intelligent Interior Design Search Engine Using Multi-Modal AI Techniques

A graduation project dissertation by:

Ahmed Gamal Ali Elsayed (202000024)

Hasnaa Muhammed Abd El-Fattah Marei (202000274)

Shaden Khaled Sayed Hafez (202000421)

Omar Atef Wagdy Attia (202000601)

Mahmoud Osama Nabawy Ashour (202000839)

Nada Mohamed Soliman Mohamed (202000994)

Submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Computers & Artificial Intelligence, Helwan University

Supervised by:

Dr. Soha Ahmed Ehssan

June 2024

جامعة حلوان

كلية الحاسبات والذكاء الإصطناعي

قسم علوم الحاسب

التصميم الداخلى

رسالة مشروع تخرج مقدمة من:

احمد جمال علي السيد (202000024)

حسناء محمد عبد الفتاح (202000274)

شادن خالد سيد حافظ (202000421)

عمر عاطف وجدى عطيه (202000601)

محمود اسامه نبوى عاشور (202000839)

ندى محمد سليمان محمد (202000994)

رسالة مقدمة ضمن متطلبات الحصول على درجة البكالوريوس في الحاسبات والذكاء الإصطناعي،

بقسم علوم الحاسب، كلية الحاسبات والذكاء الإصطناعي، جامعة حلوان

تحت إشراف:

د/ سهى احسان احمد

يونيو / حزيران 2024

Table of Contents

Acknowledgments

We would like to show our gratefulness for all the people who gave us their hand of help.

First and foremost, we would like to express our deepest gratitude to our Supervisor **Dr. Soha Ahmed Ehssan** for her support, outstanding guidance and encouragement throughout our graduation project and her continuous support and help since the beginning of our project.

We are also so thankful for our beloved faculty,

Faculty of Computer Science & Artificial Intelligence – Helwan University,

For providing us a suitable environment for learning, practicing & flourishing in our career.

Abstract

Good interior design plays a significant role in creating comfortable living or working environments. A well-designed space can greatly enhance overall comfort and satisfaction of its occupants.

It is not easy to imagine your interior design without actually visualizing it with your own eyes, so we created an interior design Multi-Modal Web Application that helps users to visualize the interiors of an empty space, like Augmented Reality which helps you to imagine how furniture items will look in the space. It helps them make interiors that are reflective of their personality and taste. It also helps businesses to increase customer engagement.

Chapter 1: Introduction

1.1 Overview:

In this project, we address the challenge of helping users visualize and personalize their interior spaces by developing a multi-modal interior design search engine. This innovative tool combines advanced AI technologies to deliver a seamless and intuitive search experience that leverages both visual and textual inputs.

Project Motivation and Background

Interior design significantly impacts the comfort and aesthetic appeal of living and working environments. However, visualizing how different furniture items and decor elements will look in a given space can be difficult without the right tools. Traditional interior design processes often require the assistance of professional designers, which can be costly and time-consuming. Additionally, existing search engines for interior design are limited in their ability to integrate visual features from images with semantic understanding from textual descriptions. This limitation often results in suboptimal search experiences where users struggle to find designs or items that match their specific requirements and preferences.

Project Goals

The primary goal of our project is to create an advanced search engine for interior design that allows users to search for images and designs based on both visual and textual inputs. By combining state-of-the-art models for object detection, image encoding, and textual embeddings, our search engine aims to enhance the user experience by providing accurate, relevant, and aesthetically pleasing search results.

Significance and Impact

Our multi-modal interior design search engine represents a significant advancement in the field of AI-driven design tools. By addressing the limitations of existing search engines and providing a more integrated and intuitive search experience, our project offers several key benefits:

- **Empowering Users:** Allowing users to easily visualize and personalize their interior spaces, making design decisions more informed and enjoyable.

- **Enhancing Personalization:** Offering a high degree of personalization by enabling users to search for items and designs that match their unique tastes and preferences.

- **Supporting Businesses:** Helping businesses increase customer engagement by providing a powerful tool for visualizing and selecting interior design products.

1.2 Problem Statement:

Existing interior design search engines are often limited in their ability to simultaneously leverage visual features from images and semantic understanding from textual descriptions. This limitation results in suboptimal search experiences, where users struggle to find designs or items that match their specific requirements and preferences. Several key challenges and gaps in current solutions include:

Challenges:

- **Limited Visual Search Capabilities:**
  - Current search engines often rely on basic image recognition technologies that lack the ability to accurately identify and differentiate between various furniture items and interior decor elements within a complex scene. This results in imprecise search results that fail to meet user expectations.

- **Inadequate Textual Search Integration:**
  - Many interior design search platforms do not effectively integrate textual descriptions with visual data. As a result, users cannot refine their searches using specific keywords or phrases, leading to a less personalized and contextually relevant search experience.

- **Poor Feature Fusion:**
  - The inability to effectively combine visual and textual features limits the accuracy and relevance of search results. Current models struggle to merge these two types of data in a way that fully captures the nuances and details of user queries.

- **Suboptimal User Experience:**
  - The user interfaces of existing search engines are often not intuitive, making it difficult for users to input complex queries and navigate through search results. This reduces the overall effectiveness and satisfaction of the search process.

- **Lack of Personalization:**
  - Many search engines do not offer personalized recommendations based on user preferences and past interactions. This means that users must repeatedly refine their searches to find items that match their unique tastes and requirements.

- **Limited Dataset Diversity:**
  - The datasets used to train current models are often not comprehensive enough to cover the wide range of interior design styles and elements. This results in limited search capabilities and reduced accuracy in identifying and recommending relevant items.

Specific Issues:

- **Visualizing Interior Design:**
  - Users find it challenging to visualize how different furniture items and decor elements will look in their own spaces. Existing tools do not provide a seamless way to integrate and visualize these elements within a given room context.

- **Matching Aesthetic Preferences:**
  - It is difficult for users to find items that match their specific aesthetic preferences. Current search engines often fail to understand and match the style and design preferences described by users.

- **Time-Consuming and Costly Process:**
  - The traditional process of interior design, which often involves hiring professional designers, is both time-consuming and costly. Users need a more efficient and affordable way to explore design options and make informed decisions.

- **Combining Multi-Modal Queries:**
  - Users want to combine visual inputs (e.g., uploading a photo of a room) with textual descriptions (e.g., "modern black chair") to refine their search. Current systems lack the capability to handle such multi-modal queries effectively.

1.3 Scope and Objectives:

Scope:

The scope of this project is centered on developing a multi-modal search engine tailored for interior design, leveraging advanced artificial intelligence technologies to enhance the user's ability to visualize and personalize their living spaces. The key aspects of the project scope include:

Dataset Collection and Annotation:

- **Image Acquisition:** Gathering a comprehensive dataset of images depicting various interior items and room scenes from multiple sources, primarily through web scraping and manual collection.
- **Textual Descriptions:** Creating detailed textual descriptions for each image, providing semantic context and enhancing the search engine's ability to understand user queries.
- **Categorization:** Organizing the dataset into 14 distinct categories based on room classes and items (e.g., living room, bedroom, chair, clock), facilitating targeted searches and training.

Model Integration and Training:

- **Object Detection with YOLOv8:** Implementing the YOLOv8 model to detect and localize objects within interior images, identifying furniture and decor elements with high accuracy.
- **Image Encoding with ConvNeXt:** Utilizing ConvNeXt to extract high-dimensional feature vectors from images, capturing their visual characteristics for effective similarity comparisons.
- **Textual Embeddings with SBERT and RoBERTa:** Generating semantically meaningful embeddings for textual descriptions using SBERT and RoBERTa models, enabling the search engine to interpret and match textual queries.

Feature Fusion and Search Implementation:

- **Multi-Modal Feature Fusion:** Developing algorithms to combine visual and textual features, creating a unified representation that leverages the strengths of both modalities.
- **Similarity Measurement:** Using cosine similarity to measure the similarity between query embeddings and dataset embeddings, ensuring accurate and relevant search results.
- **Search Engine Interface:** Designing a user-friendly interface that allows users to input both visual and textual queries, and view the search results in an intuitive manner.

Evaluation and Optimization:

- **Performance Metrics:** Implementing rigorous evaluation metrics, such as accuracy and mean Average Precision (mAP), to assess the performance of the search engine.
- **Parameter Tuning:** Continuously refining and optimizing model parameters and search algorithms to enhance performance and efficiency.

Deployment and User Interaction:

- **Web Application Development:** Creating a web-based application that provides an interactive platform for users to perform searches, view results, and visualize interior design concepts.

Objectives:

The primary objectives of this project are to:

Develop a Comprehensive Dataset:

- Collect and annotate a diverse set of interior design images, ensuring a rich dataset that covers various items and room scenes.
- Provide detailed textual descriptions for each image to enhance the semantic understanding of the search engine.

Integrate Advanced AI Models:

- Implement YOLOv8 for accurate object detection, identifying multiple furniture classes and their bounding boxes within images.
- Utilize ConvNeXt for effective image encoding, extracting high-dimensional visual features from images.
- Leverage SBERT and RoBERTa models for generating high-quality textual embeddings, enabling the search engine to understand and process textual queries.

Implement Feature Fusion Techniques:

- Develop algorithms to combine visual and textual features, creating a unified multi-modal representation that enhances search accuracy and relevance.
- Ensure that the feature fusion process effectively balances the contributions of both modalities, leading to robust and meaningful search results.

Design an Intuitive Search Interface:

- Create a user-friendly interface that supports multi-modal queries, allowing users to input both images and text to perform searches.
- Ensure that the search results are presented in a clear and visually appealing manner, facilitating easy exploration and interaction.

Evaluate and Optimize the Search Engine:

- Implement comprehensive evaluation metrics to assess the performance of the search engine, focusing on accuracy, relevance, and user satisfaction.
- Continuously optimize model parameters and search algorithms to improve the efficiency and effectiveness of the search process.

Enhance User Experience and Accessibility:

- Develop additional features to improve user interaction, such as voice-based search, mobile application support, and multi-language capabilities.
- Ensure that the search engine provides a personalized experience, adapting to user preferences and feedback to deliver more relevant results.

By achieving these objectives, we aim to create a state-of-the-art multi-modal search engine for interior design that significantly enhances the user's ability to visualize and personalize their spaces. Our project combines advanced AI technologies with a rich and diverse dataset, delivering a powerful tool that meets the needs of both individual users and businesses in the interior design domain. Through continuous evaluation, optimization, and user-centric enhancements, we strive to offer a cutting-edge solution that revolutionizes the way users interact with interior design concepts.

1.3 Work Methodology:

Our work methodology encompasses several key phases, each involving specific tasks and processes to ensure the successful development and deployment of our multi-modal interior design search engine. Below is a detailed description of the methodology we followed:

1.3.1 Data Collection and Preparation

Data Collection:

- **Web Scraping:** We collected a diverse set of 2900 images from various online sources, focusing on interior design items and room scenes. Images were categorized into 14 distinct classes such as living rooms, bedrooms, chairs, clocks, etc.
- **Manual Annotation:** Each image was accompanied by detailed textual descriptions created manually to provide semantic context. This included descriptions of the room scene or individual item, its style, color, and any other relevant details.

Data Preprocessing:

- **Data Cleaning:** Ensuring all images were of high quality and appropriately labeled. Removing duplicates and correcting any mislabeled data.
- **Annotation Format Conversion:** Converting annotations to the required YOLO format for object detection, which involves specifying the class ID and bounding box coordinates (class_id x_center y_center width height).

1.3.2 Model Selection and Integration

Object Detection with YOLOv8:

- **Model Initialization:** Using the pre-trained YOLOv8 model for its robust real-time object detection capabilities.
- **Training:** Fine-tuning the YOLOv8 model on our custom dataset to accurately detect and classify interior items and decor elements.

- **Parameter Optimization:** Adjusting parameters such as the detection confidence threshold and non-max suppression to improve detection accuracy.

Image Encoding with ConvNeXt:

- **Model Initialization:** Utilizing ConvNeXt for extracting high-dimensional feature vectors from images, capturing intricate visual details.
- **Feature Extraction:** Passing images through ConvNeXt to obtain embeddings that represent the visual characteristics of each image.

Textual Embeddings with SBERT and RoBERTa:

- **Model Initialization:** Implementing SBERT and RoBERTa models to generate semantically meaningful embeddings for textual descriptions.
- **Text Encoding:** Processing the textual descriptions to create embeddings that encapsulate the semantic content of the text.

1.3.3 Feature Fusion and Similarity Search

Multi-Modal Feature Fusion:

- **Combining Visual and Textual Features:** Developing algorithms to fuse features from ConvNeXt (visual) and SBERT/RoBERTa (textual). This involves techniques such as weighted averaging and attention mechanisms to ensure a balanced contribution from both modalities.
- **Normalization:** Ensuring all feature vectors are normalized, typically to unit length, to facilitate accurate similarity calculations.

Similarity Measurement:

- **Cosine Similarity:** Using cosine similarity to measure the similarity between query embeddings and dataset embeddings. This metric is chosen for its effectiveness in high-dimensional spaces.

- **Ranking Results:** Sorting the results based on similarity scores to provide the most relevant search results at the top.

1.3.4 Implementation and Deployment

Search Engine Interface:

- **User Interface Design:** Creating an intuitive and user-friendly web interface that supports multi-modal queries. Users can input both images and text to perform searches.
- **Interactive Features:** Implementing features such as drag-and-drop image uploads, voice-based search, and interactive result displays to enhance user engagement.

Backend Integration:

- **API Development:** Developing RESTful APIs to handle search queries, process inputs, and return results. Ensuring seamless communication between the frontend interface and backend processing.

1.3.5 Evaluation and Optimization:

Evaluation Metrics:

- **Mean Average Precision (mAP):** Using mAP to evaluate the overall performance across multiple queries, considering both precision and recall.

By following this structured methodology, we ensure that our multi-modal interior design search engine is developed systematically, with a focus on accuracy, user experience, and continuous improvement. This comprehensive approach allows us to deliver a cutting-edge tool that meets the diverse needs of users in the interior design domain.

Chapter 2: Related work

2.1 Background:

- The proposed multi-modal search engines for interior design combines visual and textual queries. The goal of the engine is to retrieve interior objects, eg: furniture or wall clocks, that share visual and aesthetic similarities with the query.

2.2 Literature Survey:

2.2.1 Textual Search:

- First methods proposed to address textual information retrieval have been based on
  token counts, ex: **Bag-of-Words** or **TF-IDF**

- **Drawback:** The scalability of those methods is very limited and when using such representations long sequences (documents) tend to have similar token distributions which results in lower discriminative power of the representation and lower retrieval precision.

- To avoid those problems is to apply a **SVD decomposition** of the token cooccurrence matrix and, hence, reduce the dimensionality of a representation vector.

- **Drawback:** SVD assumes that the relationships between features are linear, which can limit its ability to capture complex non-linear relationships.

- To handle those shortcomings, a new type of representation called word2vec has been proposed “Continuous Bag of Words (**CBOW**) and **Skip-Grams**”.

- They allow the token representation to be learned based on its local context

- **Drawback:** The CBOW model averages the context of a word and it is a unidirectional model, which means that it only considers the context words in the forward direction. Skip-Grams model tends to have higher memory requirements.

2.2.2 Visual Search:

- Traditionally, image-based search methods drew their inspiration from textual retrieval systems.

- By using **K-Means Clustering** method in the space of local feature descriptors, such as SIFT, they are able to mimic textual word entities with the so-called visual words.

- **Video Google** was one of the first visual search engines that relied on this concept that checks for geometrical correctness of initial query and eliminates the results that are not geometrically plausible.

- **R-MAC** technique is an image representation based on the outputs of CNNs that could be computed in a fixed layout of spatial regions.

- **Siamese networks** are proved successful when applied to content-based image retrieval
- **Drawback:**
  They do not take into account the contextual and stylistic similarity of the retrieved objects, which yields their application to the problem of infeasibility of interior design items retrieval.

2.3 Analysis of Related Work:

- YOLO9000 was used as an object detection model integrated with RESNET-50 that they will be able to get higher performance on overcoming the previous drawbacks.

- **Drawback:** Accuracy was very low (48%) for detecting the objects in the images.

Chapter 3: Proposed Solution

3.1 Framework:

3.1.1 Architecture Overview:

The multi-modal interior design search engine follows a client-server architecture with a well-defined data pipeline. The frontend, built with HTML, CSS, and JavaScript, provides a user-friendly interface for image and text input. User queries are processed by the Flask-based backend, which interacts with the pre-trained models through a model API.

Data Pipeline:

- **Image Collection:**
  - **Web Scraping:** Images are sourced from various online sources using web scraping techniques. Publicly available datasets are also utilized to gather a diverse set of interior design images.
  - **Manual Collection:** Additional images are collected manually to ensure a comprehensive dataset.

- **Data Preprocessing:**
  - **Image Standardization**: Images are resized to a consistent dimension and normalized to ensure uniformity across the dataset.
  - **Data Augmentation:** Techniques such as rotation, flipping, and color adjustments are applied to augment the dataset, increasing its diversity and robustness (in dataset that use to train YOLOv8).

- **Annotation:**
  - **Manual Annotations:** Detailed textual descriptions and annotations are created for each image, providing semantic context and enhancing the dataset's usability.
  - **Bounding Box Annotation:** Objects within images are annotated with bounding boxes, specifying class labels and coordinates. This is essential for training the YOLOv8 object detection model.

- **Object Detection:**
  - **Model Initialization:** YOLOv8 is initialized and fine-tuned on the custom dataset to detect various interior design objects (e.g. door, window, wardrobe) and also use the default weights to detect specific classes(e.g. bed, chair, couch).
  - **Detection and Labeling:** The YOLOv8 model detects objects in images and labels them with class IDs and bounding box coordinates.
  - **ROI Extraction:** Regions of interest (ROIs) are extracted based on the detected bounding boxes, allowing for focused analysis on specific objects.

- **Feature Extraction:**
  - **Image Embedding:** ConvNeXt model extracts high-dimensional visual features from images and ROIs. These embeddings capture the intricate details of each image, facilitating accurate similarity comparisons.
  - **Text Embedding:** SBERT and RoBERTa models generate textual embeddings from the descriptions, capturing the semantic content and nuances of the text.
- **Storage:**
  - **embedding storage:** The embedding is saved in the storage space and enables efficient similarity searches.

Search Engine Components:

- **Frontend:** Provides a user interface for image and text input, visualizes search results, and handles user interactions. Built using HTML, CSS, and JavaScript for a responsive and interactive user experience.
- **Backend:** The Flask-based backend processes user queries, calls models through the API, performs similarity searches, and returns ranked results to the frontend.
- **Model API:** Facilitates seamless interaction between the backend and the models (YOLOv8, ConvNeXt, SBERT, RoBERTa).

Ensures efficient communication and processing of model inferences.

- **Similarity Search Algorithm:** The system calculates cosine similarity between the query embeddings and database embeddings. Results are ranked based on these combined similarity scores to ensure relevance and accuracy.

3.1.2 Software Stack:

The search engine leverages the following software technologies:

- **Language:** Python for its extensive libraries and ease of integration.
- **Web Framework:** Flask for building a robust and scalable backend.
- **Deep Learning Library:** PyTorch for model training and inference.
- **Object Detection Model:** YOLOv8 for detecting and localizing objects within images.
- **Image Embedding Model:** ConvNeXt for extracting high-dimensional visual features.
- **Text Embedding Models:** SBERT and RoBERTa for generating meaningful textual embeddings.
- **Frontend Technologies:** HTML, CSS, JavaScript for building the user interface.
- **Other Libraries:** Additional Python libraries such as **`timm`** for model integration, **`sklearn`** for similarity calculations, and **`pandas`** for data handling.

3.1.3 Deployment Considerations:

Hardware: A GPU (e.g., NVIDIA GeForce RTX 30 series) is recommended for efficient model training and inference, ensuring high performance and quick response times.

This framework section provides a clear understanding of the technology behind our search engine and helps to demonstrate its robustness and scalability. By utilizing state-of-the-art models and technologies, our system is designed to deliver high accuracy and efficiency in interior design searches.

3.2 Proposed Model:

In this section, we present the architecture and workflow of our multi-modal Style Search Engine, which is designed to enhance the search experience for interior design by leveraging both visual and textual inputs. The system operates as follows:

- **Input Query:**
  - The system accepts two types of inputs: an image of an interior (e.g., a picture of a dining room) and a textual query specifying the search criteria (e.g., "black chair").

- **Object Detection:**
  - An object detection algorithm, specifically YOLOv8, is applied to the uploaded image. This algorithm identifies and locates objects of interest within the image, such as chairs, tables, or sofas.
  - Both a default YOLOv8 model and a custom-trained YOLOv8 model are employed to detect standard objects as well as additional classes that are relevant to interior design.

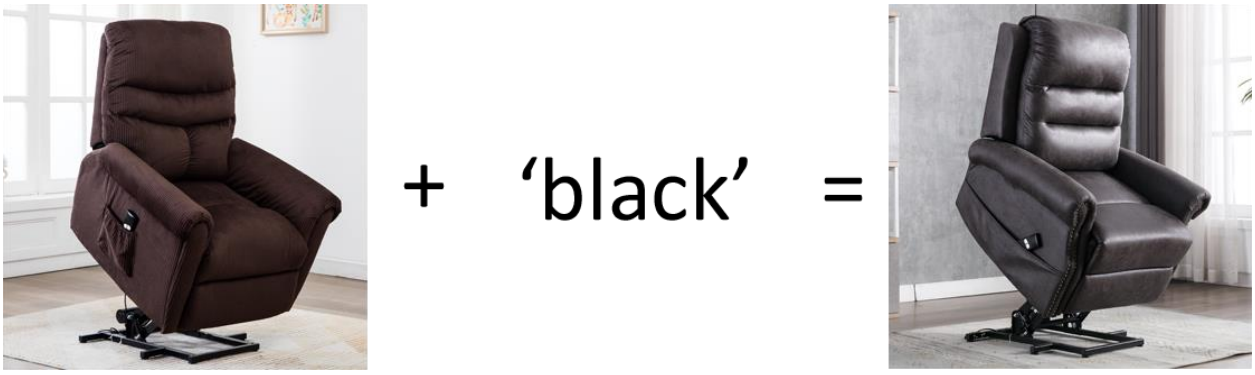
- **Feature Extraction:**
  - Once the objects are detected, their regions of interest (ROIs) are extracted as image patches.
  - The ConvNeXt model is used to extract high-dimensional features (embeddings) from these image patches as well as from the full image.

- **Textual Embedding:**
  - Simultaneously, the textual query is processed using advanced NLP models, SBERT and RoBERTa, to generate high-quality sentence embeddings that capture the semantic meaning of the query.

- **Feature Fusion:**
  - The features from the full image and the ROIs are combined using an attention mechanism. This mechanism assigns higher weights to features from regions with higher detection confidence, creating robust image embeddings.
  - The embeddings from SBERT and RoBERTa are ensembled to leverage the strengths of both models, resulting in a more comprehensive semantic representation of the text.

- **Similarity Calculation and Ranking:**
  - The combined image and text embeddings are used to calculate similarity scores with precomputed embeddings from the dataset.
  - Cosine similarity is employed to measure the similarity between the query embeddings and the dataset embeddings. The similarity scores from the visual and textual modalities are then combined using a late fusion technique.
  - The search results are ranked based on these combined similarity scores, ensuring that the most relevant items are presented to the user.

- **Visualization:**
  - The top-ranked search results are visualized, providing an intuitive interface for users to see the most relevant images and items based on their query.



This integrated approach ensures that the search engine can accurately retrieve and rank interior design items that match both the visual and textual criteria specified by the user. Below, we describe each part of the engine in more detail.

3.2.1 Visual Search:

The visual search part of the code uses ConvNeXt for image feature extraction and YOLOv8 for object detection to identify regions of interest (ROIs). Features from the full image and ROIs are combined using an attention mechanism to form robust image embeddings. These embeddings are used in conjunction with text embeddings from SBERT to perform a similarity search, which ranks images based on their combined visual and textual similarities to the query. The top results are then visualized, providing a clear and intuitive interface for the search outcomes.

- **Image Embedding with ConvNeXt:**
  - ConvNeXt is a convolutional neural network architecture designed for image classification. In this code, ConvNeXt is used to extract features (embeddings) from images.

Code Implementation:

- **Model Initialization:**

```
model = timm.create_model('convnext_base', pretrained=True)
model.eval()
```

The **`timm`** library is used to create and load a pre-trained ConvNeXt model. The model is set to evaluation mode using **`model.eval()`**.

- **Image Transformation:**

```
transform = transforms.Compose([
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

Images are transformed to a consistent size (224x224 pixels) and normalized using mean and standard deviation values typical for ImageNet pre-trained models.

- **Image Feature Extraction:**

```
full_image_tensor = transform(image).unsqueeze(0).to(device)
with torch.no_grad():
    full_image_features = model(full_image_tensor).squeeze(0).cpu().numpy()
```

The transformed image tensor is passed through the ConvNeXt model to obtain feature embeddings. The **`torch.no_grad()`** context is used to avoid calculating gradients, as this is an inference step.

- ## Object Detection with YOLOv8:

  - YOLO (You Only Look Once) is a real-time object detection system. In this code, both a default YOLO model and a custom-trained YOLO model are used for object detection.

Code Implementation:

- **Model Initialization:**

```
default_yolo_model = YOLO('yolov8s.pt')
custom_yolo_model = YOLO('/content/drive/MyDrive/shared GP/YOLOv8 train/best (2).pt')
```

The default YOLO model is loaded from a pre-trained weights file, while the custom YOLO model is loaded from a file containing weights trained on a specific dataset.

- **Object Detection:**

```
results_default = default_yolo_model(img)
results_custom = custom_yolo_model(img)
```

The models are used to detect objects in the image. The results contain bounding boxes, class labels, and confidence scores for detected objects.

- ## Region of Interest (ROI) Expansion and Feature Extraction:

  - When objects are detected, regions of interest (ROIs) around these objects are expanded and features are extracted from these regions.

Code Implementation:

- **ROI Expansion:**

```
def expand_roi(x1, y1, x2, y2, image_size, expansion_factor=1.5):
    width, height = image_size
    new_width = (x2 - x1) * expansion_factor
    new_height = (y2 - y1) * expansion_factor
    x1 = max(0, x1 - (new_width - (x2 - x1)) / 2)
    y1 = max(0, y1 - (new_height - (y2 - y1)) / 2)
    x2 = min(width, x1 + new_width)
    y2 = min(height, y1 + new_height)
    return int(x1), int(y1), int(x2), int(y2)
```

The **`expand_roi`** function expands the bounding box around detected objects by a specified factor. This helps to capture more context around the object.

- **ROI Feature Extraction:**

```
cropped_image = image.crop((x1, y1, x2, y2))
image_tensor = transform(cropped_image).unsqueeze(0).to(device)
with torch.no_grad():
    features = model(image_tensor).squeeze(0).cpu().numpy()
```

The expanded ROI is cropped from the image, transformed, and passed through the ConvNeXt model to obtain feature embeddings.

▪ Combining Features with Attention Mechanism:

- Features from the whole image and ROIs are combined using an attention mechanism. This mechanism assigns higher weights to features from regions with higher detection confidence.

Code Implementation:

- **Attention-Based Feature Combination:**

```
def combine_features_with_attention(full_image_features, roi_features, roi_confidences, full_weight=0.2):
    roi_confidences = np.array(roi_confidences)
    attention_weights = F.softmax(torch.tensor(roi_confidences), dim=0).numpy()
    weighted_features = [full_image_features * full_weight]
    weights = [full_weight]
    detection_weight = (1 - full_weight) / len(roi_features)
    for features, weight in zip(roi_features, attention_weights):
        weighted_features.append(features * weight * detection_weight)
        weights.append(weight * detection_weight)
    combined_features = np.sum(weighted_features, axis=0) / sum(weights)
    return combined_features
```

This function uses SoftMax to convert detection confidences into attention weights. The full image features and ROI features are combined based on these weights to form a single feature vector.

▪ Similarity Search:

- The similarity between the query image (and its description) and all other images (and their descriptions) is calculated using cosine similarity. The search results are ranked based on the combined similarities.

Code Implementation:

- **Image Query Embedding:**

```
image = Image.open(image_path).convert('RGB')
full_image_tensor = transform(image).unsqueeze(0).to(device)
with torch.no_grad():
    full_image_features = model(full_image_tensor).squeeze(0).cpu().numpy()
```

The query image is processed to obtain its embedding using the ConvNeXt model.

- **Text Query Embedding:**

```
text_embedding = model.encode([text_query])[0]
```

The text query is encoded into an embedding using SBERT.

- **Similarity Calculation:**

```
image_similarities = cosine_similarity(image_embedding.reshape(1, -1), image_embeddings).flatten()
text_similarities = cosine_similarity(text_embedding.reshape(1, -1), text_embeddings).flatten()
combined_similarities = 0.5 * image_similarities + 0.5 * text_similarities
ranked_indices = np.argsort(-combined_similarities)[:top_k]
```

Cosine similarity is calculated between the query embeddings and the precomputed embeddings for all images and texts. Similarities are combined and used to rank the results.

- **Visualization of Search Results:**

  - The top search results are visualized.

Code Implementation:

- **Visualization:**

```
def visualize_results(ranked_results):
    fig, axes = plt.subplots(nrows=1, ncols=len(ranked_results), figsize=(5, 5))
    for idx, (result_path, similarity) in enumerate(ranked_results, 1):
        img = mpimg.imread(result_path)
        axes[idx-1].imshow(img)
        # axes[idx-1].set_title(f"Rank {idx}\nSimilarity: {similarity:.2f}")
        axes[idx-1].axis('off')
    plt.tight_layout()
    plt.show()
```

The function **`visualize_results`** creates a plot with the top search results. Each result image is displayed without axis labels for a clean visualization.

3.2.2 Textual Search:

The code utilizes two powerful NLP models, SBERT and RoBERTa, to encode textual descriptions into high-dimensional embeddings. These embeddings are then combined to leverage the strengths of both models. The combined embeddings are used for various tasks like similarity search and evaluation. The approach ensures robust and semantically meaningful representations, improving the performance of downstream tasks like image and text similarity search.

- **Sentence-BERT (SBERT):**
  - Sentence-BERT is a modification of the BERT (Bidirectional Encoder Representations from Transformers) model designed for sentence embeddings. SBERT uses Siamese and triplet network structures to derive semantically meaningful sentence embeddings that can be compared using cosine similarity.

Code Implementation:

- **Model Initialization:**

```
sbert_model = SentenceTransformer('all-MiniLM-L6-v2')
```

Here, we use a specific variant of SBERT named 'all-MiniLM-L6-v2', which is a smaller, faster version suitable for embedding sentences into fixed-size vectors.

- **Encoding Descriptions:**

```
sbert_embeddings = sbert_model.encode(desc, show_progress_bar=False)
```

The **'encode'** method is used to convert a list of text descriptions into embeddings. These embeddings represent the sentences in a high-dimensional vector space where semantically similar sentences are close to each other.

- **RoBERTa (Robustly Optimized BERT Approach):**
  - RoBERTa is a robustly optimized version of BERT, improving upon the original model by training with more data, longer sequences, and removing the Next Sentence Prediction (NSP) objective.

Code Implementation:

- **Model and Tokenizer Initialization:**

```
tokenizer = RobertaTokenizer.from_pretrained('roberta-base')
roberta_model = RobertaModel.from_pretrained('roberta-base')
```

This initializes the tokenizer and model for RoBERTa using the pre-trained 'roberta-base' variant.

- **Encoding Descriptions:**

```
def encode_with_roberta(descriptions, model, tokenizer):
    encoded_inputs = tokenizer(descriptions, padding=True, truncation=True, return_tensors="pt", max_length=512)
    with torch.no_grad():
        model_output = model(**encoded_inputs)
    embeddings = model_output.last_hidden_state[:, 0, :].cpu().numpy()
    return embeddings
```

passes them through the RoBERTa model to obtain embeddings. The embeddings are derived from the last hidden state of the model.

- **Ensembling Embeddings:**
  - Ensembling refers to the technique of combining multiple models or representations to improve performance. In this case, embeddings from SBERT and RoBERTa are combined to create more robust text representations.

Code Implementation:

- **Ensemble Embeddings:**

```
def ensemble_embeddings(sbert_embeddings, roberta_embeddings, weights=[0.5, 0.5]):
    roberta_embeddings_reduced = roberta_embeddings[:, :sbert_embeddings.shape[1]]
    combined_embeddings = (weights[0] * sbert_embeddings + weights[1] * roberta_embeddings_reduced)
    return combined_embeddings
```

The **`ensemble_embeddings`** function takes the embeddings from SBERT and RoBERTa and combines them using specified weights(in our project we chose 50% for every model). The RoBERTa embeddings are reduced to the same dimensionality as SBERT embeddings before combining. This weighted average of embeddings helps leverage the strengths of both models.

- ## Usage in Main Function:

  - ### Text Embedding Calculation:

    The SBERT and RoBERTa embeddings are calculated and saved if they don't already exist:

```
if not os.path.exists(sbert_encoded_file):
    sbert_embeddings = sbert_model.encode(desc, show_progress_bar=False)
    np.save(sbert_encoded_file, sbert_embeddings)

if not os.path.exists(roberta_encoded_file):
    roberta_embeddings = encode_with_roberta(desc, roberta_model, tokenizer)
    np.save(roberta_encoded_file, roberta_embeddings)
```

  - ### Combining Embeddings:

    After loading the embeddings, they are ensembled:

```
sbert_embeddings = np.load(sbert_encoded_file)
roberta_embeddings = np.load(roberta_encoded_file)
nlp_embeddings.append(ensemble_embeddings(sbert_embeddings, roberta_embeddings))
```

- ## Fusion for Similarity Search:

  - ### Text Embedding for Query:

```
text_embedding = model.encode([text_query])[0]
```

    The text query is encoded using SBERT.

  - ### Similarity Calculation:

```
text_similarities = cosine_similarity(text_embedding.reshape(1, -1), text_embeddings).flatten()
```

    Cosine similarity is calculated between the query embedding and the precomputed text embeddings.

3.2.3 Fusion:

The fusion part in this code refers to the process of combining different feature representations (embeddings) from various models to enhance the overall performance and robustness of the system. Here, fusion is implemented in two primary contexts: feature combination with attention and late fusion for similarity calculation.

The fusion part of the code involves two key techniques:

- **Combining Features with Attention:** This technique merges full image features with ROI features weighted by their confidence scores to create a robust feature representation.
- **Late Fusion:** This method averages the similarity scores from different modalities (image and text) to leverage their strengths for better retrieval performance.

These fusion techniques help in enhancing the robustness and accuracy of the retrieval system by effectively combining information from multiple sources.

- This method integrates features from the whole image and regions of interest (ROI) detected by YOLO models. The idea is to give more importance (weight) to features from the ROI based on their confidence scores while retaining some information from the entire image.

Code Implementation:

- **Combining Features with Attention:**

```
def combine_features_with_attention(full_image_features, roi_features, roi_confidences, full_weight=0.2):
    roi_confidences = np.array(roi_confidences)
    attention_weights = F.softmax(torch.tensor(roi_confidences), dim=0).numpy()
    weighted_features = [full_image_features * full_weight]
    weights = [full_weight]
    detection_weight = (1 - full_weight) / len(roi_features)
    for features, weight in zip(roi_features, attention_weights):
        weighted_features.append(features * weight * detection_weight)
        weights.append(weight * detection_weight)
    combined_features = np.sum(weighted_features, axis=0) / sum(weights)
    return combined_features
```

Compute attention weights using the SoftMax function on ROI confidence scores and combine full image features with weighted ROI features and normalizes the combined features to produce a single feature vector representing the image.

- # Late Fusion for Similarity Calculation:

  - Late fusion is used to combine similarity scores from different embeddings (image and text) to enhance retrieval performance. This technique averages the similarity scores from different modalities to leverage their strengths.

    ## Code Implementation:

    - **Late Fusion:**
    - ```
      def late_fusion(cv_similarities, nlp_similarities):
          return (cv_similarities + nlp_similarities) / 2
      ```
    - The **`late_fusion`** function simply takes the cosine similarity scores from both image and text embeddings and averages them. This combined score is used for ranking in the retrieval process.

- # Usage in Main Function:

  - **Searching and Ranking:**

```
def search_engine(image_query_path, text_query, image_embeddings, text_embeddings, all_image_paths, top_k=5):
    transform = transforms.Compose([
        transforms.Resize((224, 224)),
        transforms.ToTensor(),
        transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
    ])
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = timm.create_model('convnext_base', pretrained=True).to(device)
    model.eval()

    image = Image.open(image_query_path).convert('RGB')
    image_tensor = transform(image).unsqueeze(0).to(device)
    with torch.no_grad():
        image_embedding = model(image_tensor).squeeze(0).cpu().numpy()
    model = SentenceTransformer('all-MiniLM-L6-v2')
    text_embedding = model.encode([text_query])[0]
    image_similarities = cosine_similarity(image_embedding.reshape(1, -1), image_embeddings).flatten()
    text_similarities = cosine_similarity(text_embedding.reshape(1, -1), text_embeddings).flatten()
    combined_similarities = 0.5 * image_similarities + 0.5 * text_similarities
    ranked_indices = np.argsort(-combined_similarities)[:top_k]
    ranked_results = [(all_image_paths[idx], combined_similarities[idx]) for idx in ranked_indices]

    return ranked_results
```

In the search engine function:

- The image query is processed, and its embedding is obtained using ConvNeXt.
- The text query is encoded using SBERT.
- Cosine similarity scores are calculated separately for image and text embeddings.
```

- Similarity scores are combined using the late_fusion function, which averages the scores.
- The images are ranked based on the combined similarity scores to return the top results.

3.2.4 Similarity Calculation and Ranking:

This approach ensures that the search results are ranked by their overall relevance, taking into account both the visual and textual aspects of the query. This method provides a robust way to retrieve and rank images that are most similar to the input query, considering the multi-modal nature of the data.
Cosine similarity is a measure of similarity between two non-zero vectors of an inner product space. It is defined as the cosine of the angle between the vectors, which can be calculated using the dot product and the magnitude (or norm) of the vectors:

$$cosine\ similarity = \cos(\theta) = \frac{A.B}{\|A\|\|B\|} + \frac{\sum_{A=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Where:

- A and B are the vectors.
- A.B is the dot product of A and B.
- ‖A‖ and $\|B\|$ are the magnitudes (norms) of vectors A and B, respectively.
- θ is the angle between the vectors.

- **Calculating Similarity:**

  - The code computes cosine similarity for both image and text embeddings separately.

Code Implementation:

- **Image Similarity Calculation:**

```
image_similarities = cosine_similarity(image_embedding.reshape(1, -1), image_embeddings).flatten()
```

Here, **`image_embedding`** represents the feature vector of the query image, and **`image_embeddings`** contains the feature vectors of all images in the dataset. The **`cosine_similarity`** function from **`sklearn.metrics.pairwise`** is used to compute the similarity between the query image and all dataset images. The result is a 1D array of similarity scores.

- **Text Similarity Calculation:**

```
text_similarities = cosine_similarity(text_embedding.reshape(1, -1), text_embeddings).flatten()
```

Here, **`image_embedding`** represents the feature vector of the query image, and **`image_embeddings`** contains the feature vectors of all images in the dataset. The **`cosine_similarity`** function from **`sklearn.metrics.pairwise`** is used to compute the similarity between the query image and all dataset images. The result is a 1D array of similarity scores.

• Combining Similarities:

- The code combines image and text similarities to leverage both visual and textual information for ranking.

Code Implementation:

- **Combining Similarities:**

```
combined_similarities = 0.5 * image_similarities + 0.5 * text_similarities
```

The image and text similarity scores are averaged with equal weights (0.5 each). This approach ensures that both visual and textual information contribute equally to the final similarity score.

• Ranking the Results:

- Once combined similarities are calculated, the code ranks the dataset images based on these scores.

Code Implementation:

- **Ranking:**

```
ranked_indices = np.argsort(-combined_similarities)[:top_k]
ranked_results = [(all_image_paths[idx], combined_similarities[idx]) for idx in ranked_indices]
```

The **`np.argsort`** function is used to sort the combined similarities in descending order (hence the negative sign). The top **`k`** indices (corresponding to the most similar items) are selected. These indices are then used to retrieve the paths of the top **`k`** images and their respective similarity scores.

3.2.5 Evaluation:

Evaluation metrics provide a comprehensive assessment of the multi-modal search system's performance, highlighting its ability to retrieve and rank relevant images based on both visual and textual input queries.

- **Mean Average Precision (mAP) Calculation:**
  - mAP is the mean of the average precision scores for all queries. It provides an overall measure of the system's performance.

Code Implementation:

- **mAP Calculation:**

```
def calculate_map(all_image_paths, relevant_images_dict, cv_embeddings, nlp_embeddings):
    ap_scores = []
    path_to_index = {path: idx for idx, path in enumerate(all_image_paths)}
    for image_path, relevant_images in relevant_images_dict.items():
        relevant_indices = [path_to_index[path] for path in relevant_images]
        query_cv_embedding = cv_embeddings[path_to_index[image_path]]
        query_nlp_embedding = nlp_embeddings[path_to_index[image_path]]
        fused_similarities = late_fusion(cosine_similarity([query_cv_embedding], cv_embeddings)[0], cosine_similarity([query_nlp_embedding], nlp_embeddings)[0])
        sorted_indices = np.argsort(-fused_similarities)
        is_relevant = np.array([index in relevant_indices for index in sorted_indices])
        precisions = np.cumsum(is_relevant) / (np.arange(len(is_relevant)) + 1)
        ap = np.sum(precisions * is_relevant) / np.sum(is_relevant)
        print(f"AP for {image_path}: {ap}")
        ap_scores.append(ap)
    map_score = np.mean(ap_scores)
    return map_score
```

The **`calculate_map`** function computes the AP for each query and averages these scores to obtain the mAP. It uses a dictionary to map image paths to indices and iterates over each query to calculate the fused similarities, sorted indices, and precision values.

- **Text Similarity Calculation:**

```
image_similarities = cosine_similarity(image_embedding.reshape(1, -1), image_embeddings).flatten()
```

Here, **`image_embedding`** represents the feature vector of the query image, and **`image_embeddings`** contains the feature vectors of all images in the dataset. The **`cosine_similarity`** function from **`sklearn.metrics.pairwise`** is used to compute the similarity between the query image and all dataset images. The result is a 1D array of similarity scores.

3.3 Dataset:

In order to evaluate our proposed Style Search Engine, we collected our dataset images from Google.

Content:

- The dataset includes various interior items such as chairs, beds, and other furniture.
- It also contains room scenes, including bedrooms and living rooms.
- Each room scene image and individual item image is accompanied by a textual description.

Since no publicly available dataset meets our specific requirements, we created our own dataset by recursively scraping many websites on Google. This process allowed us to download 2900 images of room scenes and individual items. Initially, these images did not have descriptions, so we manually wrote descriptions for each image.

Dataset Composition:

The dataset is divided into 14 categories based on the room class and items: Living Room, Bedroom, Chair, Clock, Bed, Couch, Dining Table, Potted Plant, Door, Dresser, Lamp, Wardrobe, Window, table.

Training YOLO:

The Object Detection YOLO model was previously trained on COCO Data Set, but that was not enough for our project and the classes assigned to us. It was missing some classes and was trained on others, so we had to train it on the rest of the classes, including (window, door, wardrobe, dresser, lamp).

To train YOLO on this dataset, we use a YAML configuration file that specifies the paths to the images, the number of classes, and the class names. Let's review the provided **'custom_data.yaml'** file to confirm its structure.

Example YAML Configuration:

```yaml
path: /content/drive/MyDrive/GPModel/yolo dataset/dataset yolo

train:
  - /content/drive/MyDrive/GPModel/yolo dataset/dataset yolo/door/2/train
  - /content/drive/MyDrive/GPModel/yolo dataset/dataset yolo/dresser/train
  - /content/drive/MyDrive/GPModel/yolo dataset/dataset yolo/lamp/1/train
  - /content/drive/MyDrive/GPModel/yolo dataset/dataset yolo/wardrobe/1/train
  - /content/drive/MyDrive/GPModel/yolo dataset/dataset yolo/wardrobe/2/train
  - /content/drive/MyDrive/GPModel/yolo dataset/dataset yolo/window/1/train
  - /content/drive/MyDrive/GPModel/yolo dataset/dataset yolo/window/2/train

val:
  - /content/drive/MyDrive/GPModel/yolo dataset/dataset yolo/door/2/val
  - /content/drive/MyDrive/GPModel/yolo dataset/dataset yolo/lamp/1/test
  - /content/drive/MyDrive/GPModel/yolo dataset/dataset yolo/lamp/1/valid
  - /content/drive/MyDrive/GPModel/yolo dataset/dataset yolo/wardrobe/1/test
  - /content/drive/MyDrive/GPModel/yolo dataset/dataset yolo/wardrobe/1/valid
  - /content/drive/MyDrive/GPModel/yolo dataset/dataset yolo/wardrobe/2/test
  - /content/drive/MyDrive/GPModel/yolo dataset/dataset yolo/wardrobe/2/valid
  - /content/drive/MyDrive/GPModel/yolo dataset/dataset yolo/wardrobe/3/test
  - /content/drive/MyDrive/GPModel/yolo dataset/dataset yolo/window/1/test
  - /content/drive/MyDrive/GPModel/yolo dataset/dataset yolo/window/1/valid
  - /content/drive/MyDrive/GPModel/yolo dataset/dataset yolo/window/2/test
  - /content/drive/MyDrive/GPModel/yolo dataset/dataset yolo/window/2/valid

nc: 5

names: ['door', 'dresser', 'lamp', 'wardrobe', 'window']
```

This YAML file includes:

- Paths to the training, validation, and test images.
- The number of classes **(`nc`)**, which is 5 in this case.
- The class names **(`names`)**, representing the different categories in the dataset.

Loading and Preparing the Dataset

For training YOLO, the dataset needs to be organized correctly. This involves having images and their corresponding annotation files. The annotation files should be in YOLO format, where each line represents a bounding box with the format: **`class_id x_center y_center width height`**.

Steps for Dataset Preparation:

- **Organize Images and Annotations:**
  - Ensure that each image has a corresponding annotation file.
  - Annotation files should be in YOLO format, specifying the class and bounding box coordinates for each object in the image.
- **Create YAML Configuration File:**
  - Define the paths to the training, validation, and test images.
  - Specify the number of classes and their names.

Example Python Code for Training YOLO:

Using the YAML configuration, we can train the YOLO model as follows:

Training YOLO:

```
from ultralytics import YOLO

model = YOLO('yolov8s.pt')

save_dir = '/content/drive/MyDrive/GPModel'

model.train(
    data='/content/custom_data (1) (1) (2).yaml',
    epochs=50,
    imgsz=640,
    batch=8,
    resume=True,
    save_period=2,
    save_dir=save_dir
)
```

Explanation:

- **Load Dataset Configuration:**

  ```
  data='/content/custom_data (1) (1) (2).yaml',
  ```

  The dataset configuration is loaded from the specified YAML file

- **Initialize YOLO Model:**

```
model = YOLO('yolov8s.pt')
```

A pre-trained YOLO model **(`yolov8s.pt`)** is loaded. This model serves as a starting point for training.

- **Train the Model:**

```
model.train(
    data='/content/custom_data (1) (1) (2).yaml',
    epochs=50,
    imgsz=640,
    batch=8,
    resume=True,
    save_period=2,
    save_dir=save_dir
)
```

The YOLO model is trained using the dataset specified in the YAML file. Training parameters include:

  - **`epochs=50`**: Number of training epochs.
  - **`imgsz=640`**: Image size for training.
  - **`batch=8`**: Batch size for training.
  - **`data='/content/custom_data (1) (1) (2).yaml'`:** Directory to save training results.

By following these steps, we ensure that the YOLO model is effectively trained on the custom dataset, leveraging the provided configuration and annotations. This process prepares the model for accurately detecting objects in interior design scenes, which is crucial for the proposed Style Search Engine.

Chapter 4: Results and Discussion

Results:

- ## Training Details (YOLOv8):

  The training was conducted for a total of 50 epochs with a batch size of 8 and an image size of 640 for 5 classes (door, dresser, lamp, wardrobe, window). The optimizer was set to auto, with AdamW chosen and a learning rate of 0.001111. Mixed precision training was enabled using AMP, and a patience parameter of 100 was set to manage early stopping. The model was pretrained, and various data augmentation techniques were employed, including Blur, MedianBlur, ToGray, CLAHE, Mosaic, and others. Additional YOLO-specific parameters included settings such as box=7.5, cls=0.5, dfl=1.5, pose=12.0, among others. The model architecture consisted of 225 layers, with 11,137,535 parameters and 28.7 GFLOPs. The training time for one epoch was approximately 230 seconds using a Tesla T4 graphics card. The AdamW optimizer was used with an initial learning rate determined automatically (lr=0.001111). TensorBoard was enabled for monitoring. The model employed various data augmentation techniques such as Blur, MedianBlur, ToGray, CLAHE, and Mosaic. The final model configuration included details like specific layers, parameters, and module arguments.

Below are the performance metrics captured during the selected epochs:

Epoch	GPU Memory	Box Loss	Classification Loss	DFL Loss	Instances	Precision (P)	Recall (R)	mAP50	mAP50-95
46	2.37G	0.6525	0.6079	1.182	7	0.646	0.641	0.684	0.468
47	2.35G	0.6526	0.5832	1.183	7	0.686	0.586	0.684	0.469
48	2.38G	0.6351	0.552	1.171	9	0.708	0.615	0.688	0.481
49	2.38G	0.6175	0.5281	1.139	6	0.74	0.587	0.694	0.476
50	2.35G	0.6241	0.5301	1.146	7	0.72	0.606	0.692	0.481

Below are the visualizations of the training results:

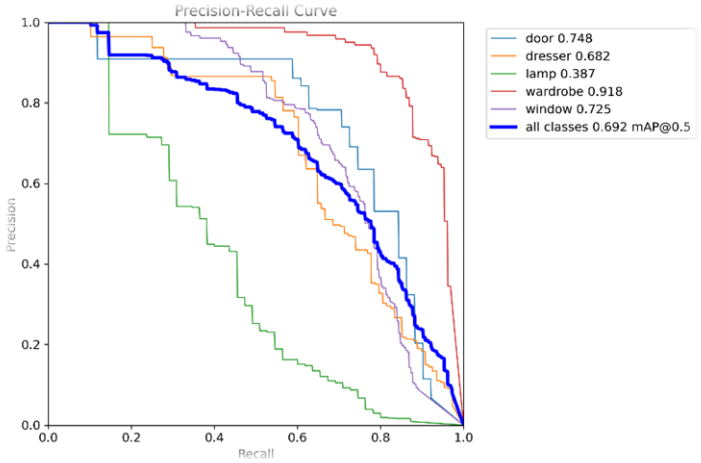


Figure 1: This curve plots precision against recall for different classes. It shows the trade-off between precision and recall at various thresholds. A larger area under the curve indicates better overall performance, as it means the model maintains high precision.

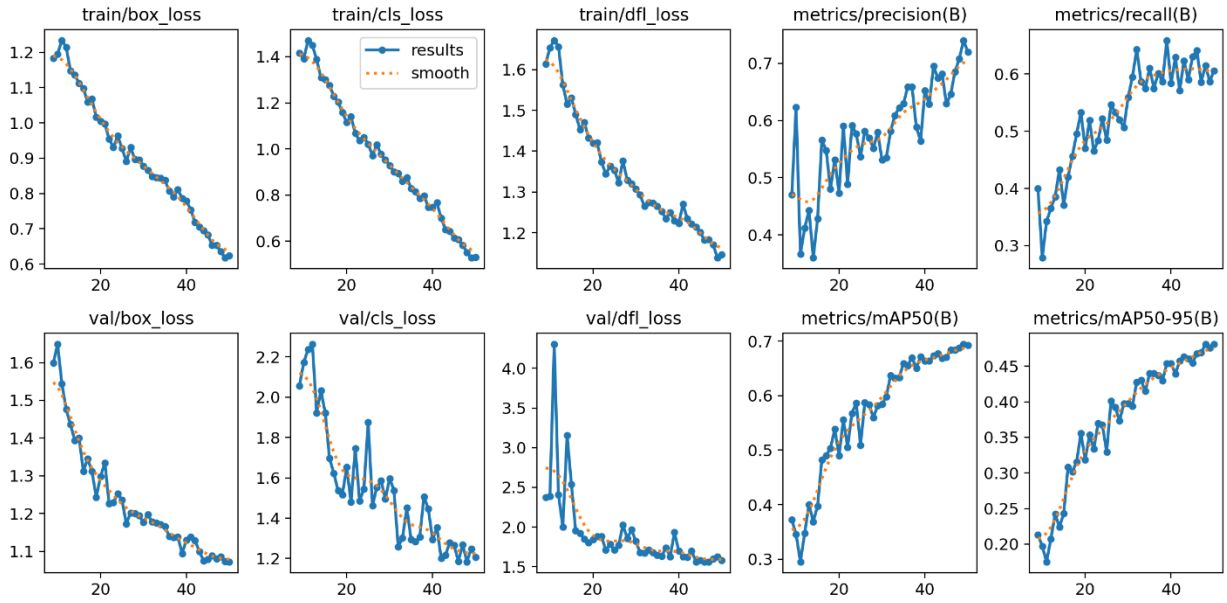


Figure 2: This set of graphs illustrates the training and validation losses, as well as the precision, recall, and mean Average Precision (mAP) metrics over the epochs. The top row shows the training losses for box, classification, and DFL (Distribution Focal Lo

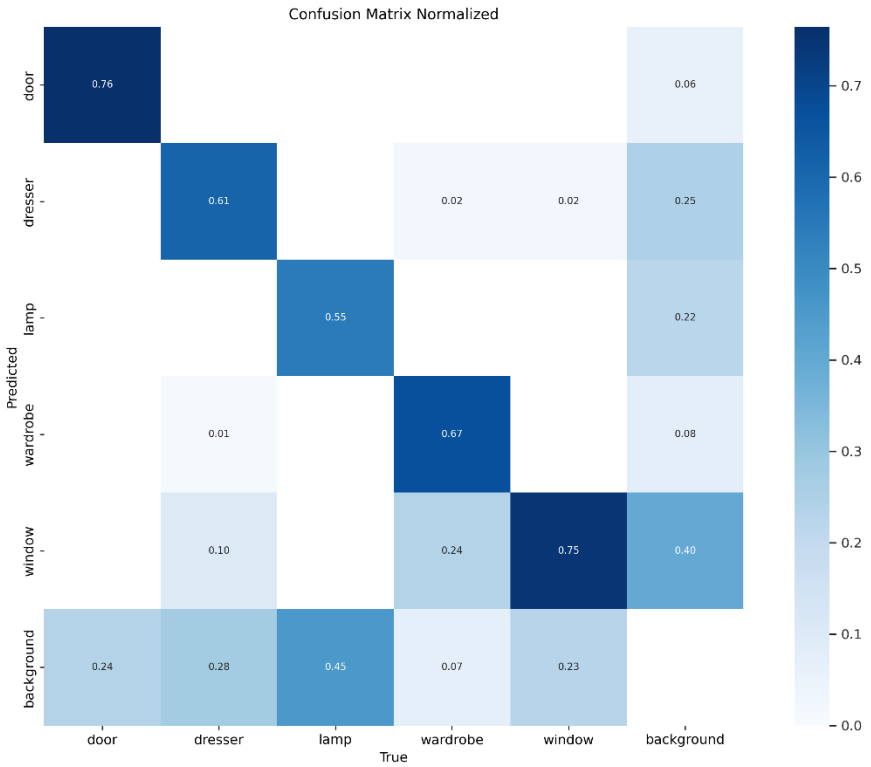


Figure 3: The normalized confusion matrix provides insights into the classification performance of the model across different classes. Each cell in the matrix shows the proportion of actual class instances that were correctly or incorrectly predicted. High values

- # Visual Search:

  The visual search system demonstrated robust performance in detecting and retrieving relevant objects and descriptions. The combination of advanced object detection models, attention-based feature extraction, and effective similarity calculation techniques contributed to the overall success of the system. The integration of multimodal embeddings and late fusion further enhanced the retrieval accuracy, providing a comprehensive solution for visual search tasks.

Object Detection and Feature Extraction:

- ### Object Detection:

  The ConvNeXTv1 model was used to extract high-dimensional features from the entire image. In addition, the regions of interest (ROIs) detected by YOLOv8 were expanded and passed through

ConvNeXTv1 to extract detailed features. We chose ConvNeXTv1 because of its efficiency in feature extraction compared to other feature extraction models.

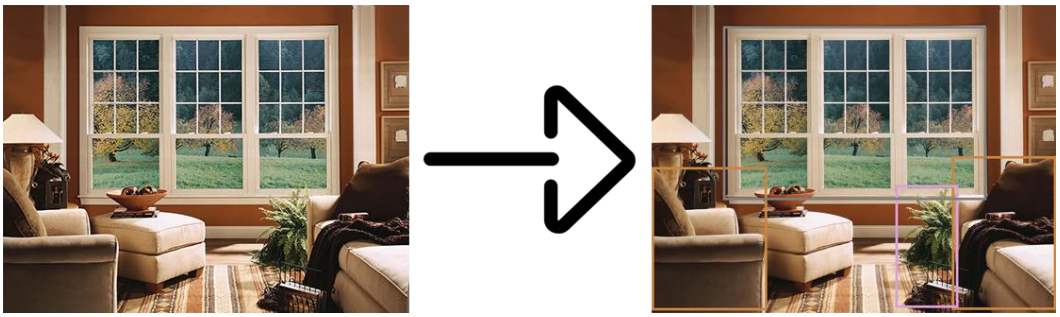


Figure 4: Comparison of original image (left) with detected objects highlighted by YOLOv8 model (right).

- **Feature Extraction:**

  The default YOLOv8 model was utilized to detect objects from the COCO dataset, focusing on key classes such as 'bed', 'chair', 'couch', 'dining table', 'clock', and 'potted plant'. Additionally, a custom-trained YOLOv8 model was specifically designed to detect additional classes like 'door', 'dresser', 'lamp', 'wardrobe', and 'window'. The models were evaluated based on their ability to detect these objects in various images, with high-confidence detections visualized with bounding boxes and annotated with the class name and confidence score.

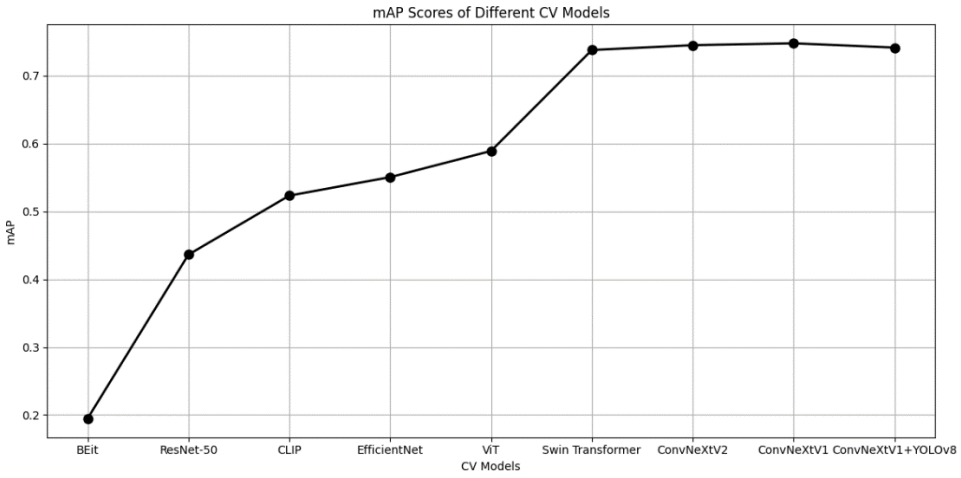


Figure 5: Line plot showing mean Average Precision (mAP) scores of different computer vision models. The plot compares various models, such as BEiT, ResNet-50, CLIP,

EfficientNet, ViT, Swin Transformer, ConvNeXtV2, ConvNeXtV1, and ConvNeXtV1 combined with YOLOv8.

- # Textual Search:

  The textual search system demonstrated strong performance in retrieving relevant text descriptions based on semantic similarity. The combination of advanced NLP models (SBERT and RoBERTa) and their ensemble embeddings significantly improved the accuracy and relevance of the search results. The use of late fusion further enhanced the retrieval performance, making the system robust and effective for textual search tasks. The visualizations provided clear evidence of the semantic clustering capabilities of the embeddings, supporting the overall effectiveness of the textual search system.

Text Embedding and NLP Models:

- ### Text Descriptions:

  The text descriptions were encoded using two advanced NLP models, SBERT (Sentence-BERT) and RoBERTa. SBERT is specifically designed for creating semantically meaningful sentence embeddings, while RoBERTa is an optimized version of BERT, which is robust in handling diverse text inputs. By using these models, high-dimensional embeddings for each text description were generated. To take advantage of the strengths of both SBERT and RoBERTa, their imputations were combined using a weighted averaging technique. This combined approach aims to capture a more comprehensive semantic representation of text descriptions.

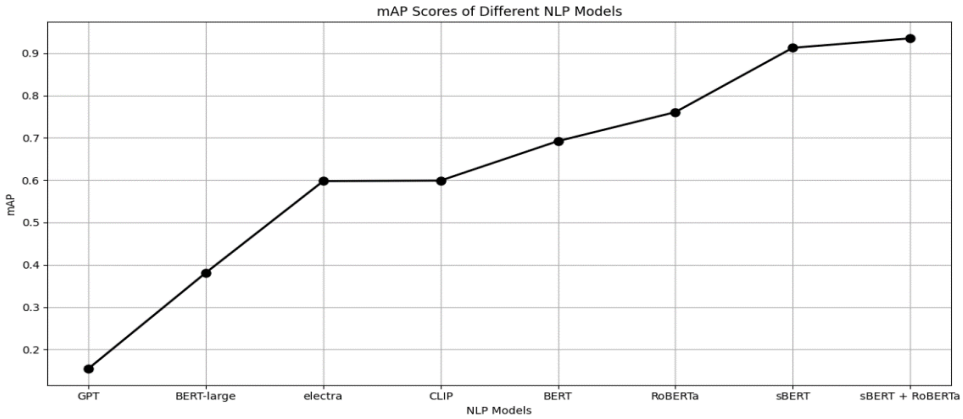


Figure 6: Line plot showing mAP scores of different NLP models. Models compared

include GPT, BERT-large, Electra, CLIP, BERT, RoBERTa, SBERT, and a combination of SBERT and RoBERTa.

- # Fusion:

  The fusion techniques employed in the system significantly improved the robustness and accuracy of the feature representations and embeddings. The attention-based feature combination effectively integrated global and local information, while the ensemble embeddings leveraged the strengths of both SBERT and RoBERTa models. Late fusion further enhanced retrieval performance by combining similarity scores from different modalities, resulting in a comprehensive and effective multimodal search system. The visualizations provided clear evidence of the improved semantic clustering and class separability achieved through fusion, supporting the overall effectiveness of the system.

Late Fusion for Similarity Calculation:

- ### Similarity Scores:

  Late fusion involved combining similarity scores from different modalities (image and text). This was achieved by averaging the similarity scores from the image embeddings and the text embeddings. The resulting combined similarity score provided a more accurate measure of similarity, improving the overall retrieval performance.

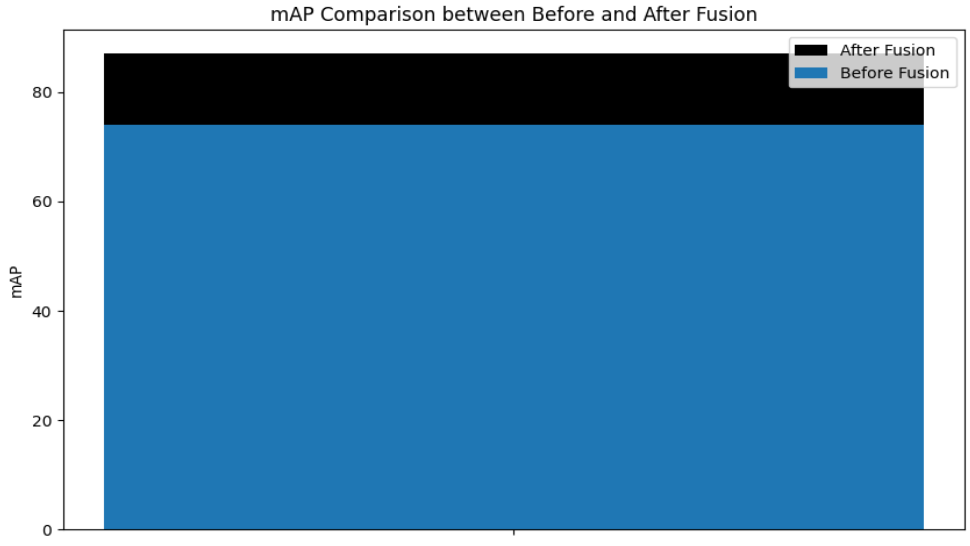


Figure 7: compares the mean Average Precision (mAP) before and after a fusion process(CV).

- **Fusion Techniques:**

  Various fusion techniques were compared, including Hybrid, Deep Cross-Modal, Hierarchical, Early, Intermediate, Slow, Tensor, and Late fusion. Late fusion outperformed the other methods, suggesting that integrating modalities at a later stage preserves more information from each modality, leading to better performance.

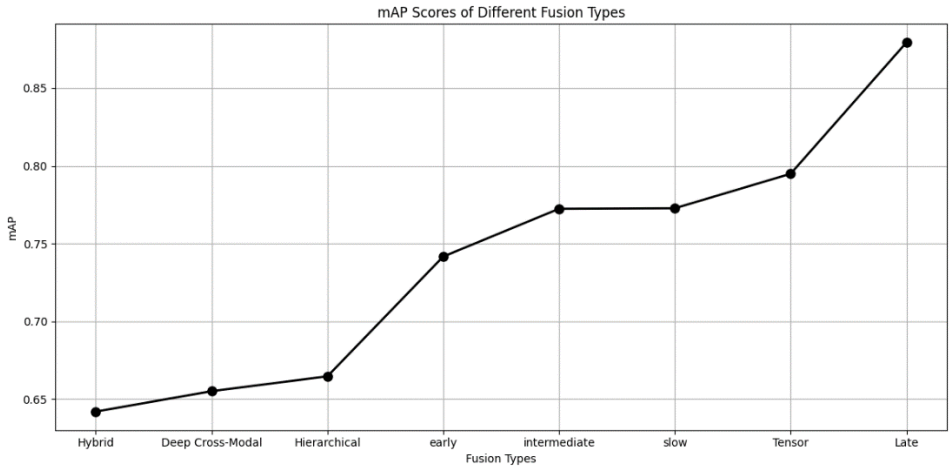


Figure 8: Line plot showing mAP scores of different fusion types.This graph explores various fusion techniques, such as Hybrid, Deep Cross-Modal, Hierarchical, Early, Intermediate, Slow, Tensor, and Late Fusion.

- **t-SNE Visualization of Fused Embeddings::**

  The t-SNE plot was used to visualize the fused embeddings in a 2D space. Each cluster in the t-SNE plot represents a different class, indicating how well the fusion process maintained class separability and semantic grouping.

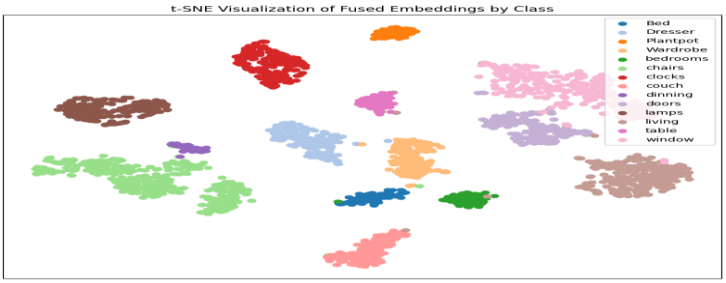


Figure 9:Each color represents a different class, indicating how well the embeddings cluster similar items together.

Chapter 5: Web Application

5.1 Application:

Visualizable is a multimodal search engine website dedicated to enhancing user experience in the field of interior design. Utilizing HTML, CSS, JavaScript, python(flask) as a backend within Visual Studio Code, the site is composed of four interconnected pages that facilitate intuitive navigation and advanced functionalities. The project integrates with a backend model to provide dynamic and personalized content.

Key Features and Pages:

1. **Homepage (Home Page)**:
    - **Introduction**: A welcoming interface for users.
    - **Primary Buttons**: main button, "Search", direct users to the respective functionalities.
2. **Navigation Bar**:
    - **Home**: Directs to the Home page.
    - **Search Engine**: Directs to the search functionality page.
    - **About**: Directs to the About page.
3. **Search Page**:
    - **Form**: Users can input a photo and text description and specify the number of images they want to see.
    - **Submit Button**: On clicking submit, the user is redirected to a results page.
4. **Display Results Page**:
    - **Image Slider**: Switch images chosen by the user in a slider format.
5. **About Us Page**:
    - **Team Members**: Introduces the team behind VisualiseVibe and their roles.
    - **Project Background**: Information about the project's purpose and objectives.

User Experience (UX) Focus:

The design of VisualiseVibe emphasizes ease of use and aesthetic appeal. Key UX features include:

- **Intuitive Navigation**: Clear and logical page transitions ensure users can easily find and utilize all site functionalities.
- **Responsive Design**: Ensures accessibility and a pleasant experience across various devices.
- **Interactive Elements**: Enhances engagement with interactive forms, image sliders, and dynamic content.

Backend Integration

The backend of VisualiseVibe is built using Flask, a lightweight WSGI web application framework in Python. It supports various functionalities including a search engine for images based on text and image inputs, as well as an image generation feature.

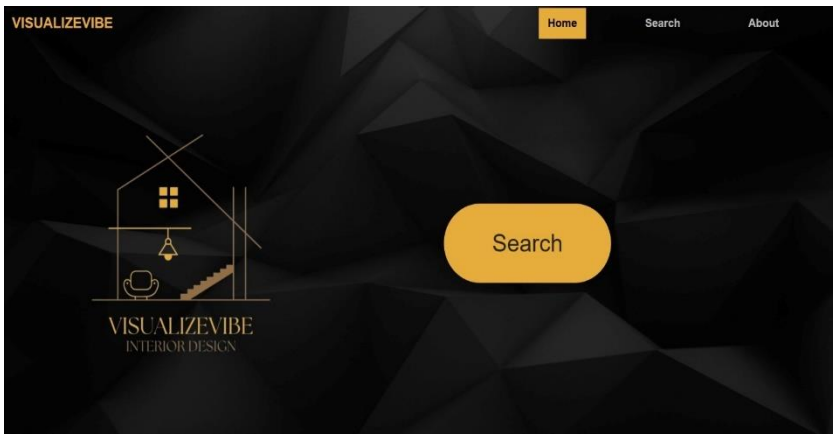
Color Scheme

- **Primary Colors**: Black and golden-yellow are used throughout the CSS for a sleek and vibrant appearance that aligns with the interior design theme.

Page Details:

Home Page

- **Contents**: Introduction text, main buttons ("Search").

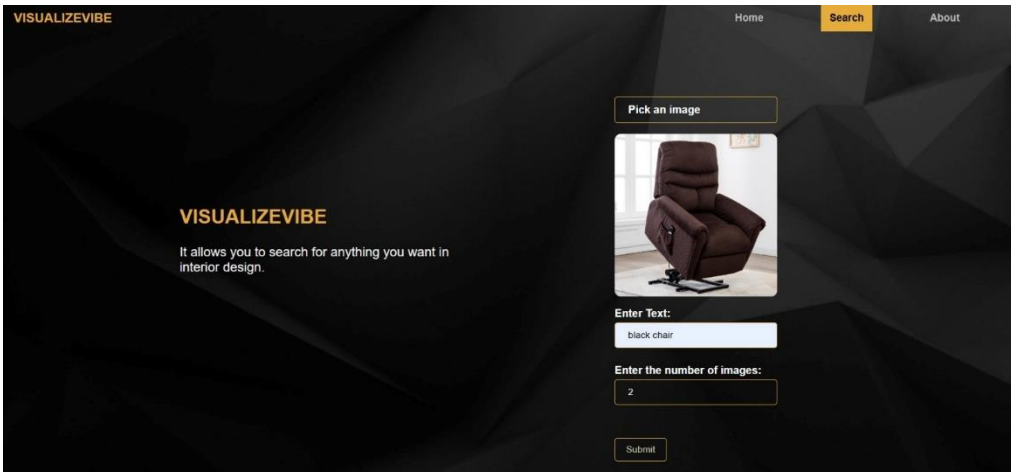


Search Page:

Method : POST

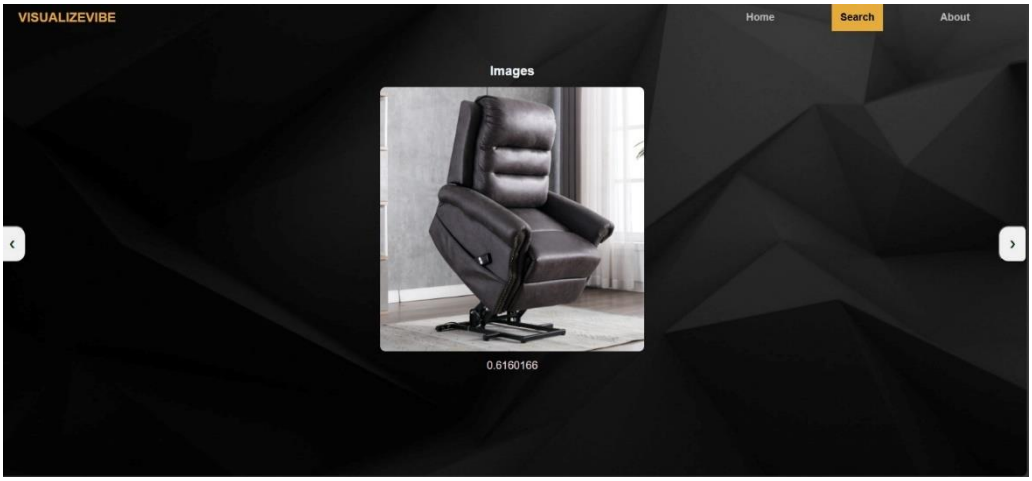
Description: Handles the search functionality. Accepts an image file and text input, processes them, and returns the similar images based on combined image and text embeddings. Results are rendered on the results page (display.html).

- **Contents**:
  - **Form Elements**: File input for photo, text input for description, number input for image count.
  - **Submit Button**: Triggers search and redirects to the results page.



Results Page

- **Contents**:
  - **Image Slider**: Displays images in a carousel format.



5.2 Software Tools and Technologies:

Visual Studio Code IDE:

- It is a free source-code editor made by Microsoft, its Features include supporting for debugging, syntax highlighting, intelligent code completion, code refactoring and embedded Git, and it supports web languages.

- It is a creative launching pad that you can use to edit, debug, and build code, and then publish an app

Google Colab:

- Colab is a hosted Jupyter Notebook service that requires no setup to use and provides free access to computing resources, including GPUs and TPUs.

- Colab is especially well suited to machine learning, data science, and education.

Python:

- It best fits machine learning due to its independent platform and its popularity in the programming community.
-

HTML:

- Hyper Text Markup Language (HTML) is the standard markup language for documents designed to be displayed in a web browser. It defines the content and structure of web content.

CSS:

- CSS (Cascading Style Sheets) is used to style and layout web pages.

JavaScript:

- JavaScript (JS) is a cross-platform, object-oriented programming language used by developers to make web pages interactive.

Flask Python:

- Flask is a popular web framework for Python that is used for building web applications and APIs. It makes it simple to create RESTful APIs with its flexible routing system and request handling with make it easier to make

Chapter 6: Conclusion & Future work

6.1 Conclusion:

The development of our multi-modal interior design search engine represents a significant step forward in helping users visualize and personalize their living spaces. By integrating advanced AI technologies such as YOLOv8 for object detection, ConvNeXt for image encoding, and SBERT and RoBERTa for textual embeddings, we have created a robust platform that effectively combines visual and textual search capabilities.

Our project addresses the limitations of existing search engines by leveraging both visual features from images and semantic understanding from textual descriptions, providing users with a more intuitive and efficient search experience. The custom dataset we created, with 2900 images and detailed descriptions, forms a solid foundation for training and evaluating our models, ensuring high accuracy and relevance in search results.

Key achievements of our project include:

- **Integration of Advanced Models:**
  Our project successfully integrates state-of-the-art models for both object detection and feature extraction. YOLOv8's real-time object detection capabilities, combined with ConvNeXt's high-dimensional image feature extraction, allow for precise identification and representation of various interior design elements. The use of SBERT and RoBERTa models for textual embeddings ensures that semantic understanding from textual descriptions is effectively incorporated, making the search engine highly intuitive and efficient.

- **Comprehensive Custom Dataset:**
  The creation of a custom dataset with 2900 images and detailed textual descriptions was a critical aspect of our project. This dataset includes various interior items and room scenes, divided into 14 categories based on room class and items. The manual annotation of descriptions for each image enhances the dataset's quality, ensuring that the models are trained on

relevant and accurate data. This foundation is essential for achieving high accuracy and relevance in search results.

- **Multi-Modal Search Engine:**
  Our implementation of a multi-modal search engine supports queries based on both images and text. This dual-modality approach enhances user interaction and satisfaction by allowing searches that leverage visual features from images and semantic understanding from textual descriptions. Users can now search for interior design items and room scenes more effectively, finding results that closely match their specific requirements and preferences.

- **Feature Fusion:**
  By combining features from visual and textual modalities, our search engine achieves improved accuracy and relevance. Attention-based feature combination and multi-modal embedding fusion techniques ensure that the most relevant features are emphasized in the final feature vector. This sophisticated approach to feature fusion enhances the search engine's ability to deliver high-quality results.

- **Evaluation and Performance:**
  We employ rigorous evaluation methods to assess the performance of our models. Accuracy calculation and mean Average Precision (mAP) metrics provide comprehensive assessments of the search engine's effectiveness. These evaluation techniques demonstrate the robustness of our approach and validate the high performance of our multi-modal search engine.

- **Impact and Potential:**
  Our approach demonstrates the potential of AI-driven solutions in transforming the way users interact with and visualize interior design concepts. The ability to accurately detect and represent various interior design elements, combined with the seamless integration of visual and textual search capabilities, offers users a personalized and engaging experience. Our project significantly enhances the search process for interior design items and room scenes, providing a valuable tool for both consumers and businesses.

- **Scalability and Future Growth:**
  The modular architecture of our search engine allows for scalability and future growth. As new models and techniques are developed, they can be integrated into our existing framework, ensuring that the search engine remains at the forefront of technology. This flexibility enables continuous improvement and adaptation to evolving user needs and technological advancements.

In summary, our multi-modal interior design search engine represents a significant achievement in leveraging advanced AI technologies to create a user-friendly and efficient tool for visualizing and personalizing interior spaces. The project's success is a testament to the power of integrating cutting-edge models, creating a comprehensive dataset, and employing sophisticated feature fusion techniques. We are confident that our search engine will continue to evolve and provide immense value to its users, transforming the way they approach interior design.

6.2 Recommendations:

To further **enhance** the capabilities of our project, we recommend the following:

- **Utilize Google Colab Pro:**
  - **L**everaging Google Colab Pro will provide access to higher RAM runtimes and more powerful GPUs and TPUs. This upgrade will significantly improve the efficiency of model training and inference processes. The increased computational resources will enable us to handle larger datasets, perform more complex data augmentation, and experiment with deeper and more sophisticated neural network architectures.

- **Expand the Dataset:**
  - Adding more variety to our dataset, including additional room scenes and furniture items, will improve the model's ability to generalize and provide accurate search results across a wider range of interior design scenarios. Sourcing images from different styles and periods (e.g., modern, vintage, minimalist) will ensure the model can handle diverse user preferences and interior design trends.
  - Partnering with interior design platforms and online furniture retailers to access proprietary datasets could provide richer and more varied image data, enhancing the robustness and applicability of our search engine.

- **Enhance Data Annotations:**
  - Improving the quality and granularity of annotations can help in training more precise models, particularly in distinguishing between similar items and styles. Detailed annotations, including attributes such as color, material, and design style, will allow for more nuanced search capabilities.
  - Implementing a semi-automated annotation tool using active learning can help in efficiently generating high-quality annotations. This tool can

suggest annotations based on model predictions, which human annotators can then verify and correct.

- **Optimize Model Parameters:**
  - Continuously refining and optimizing the parameters of our models (e.g., detection confidence threshold in YOLOv8) can lead to better performance and more accurate detections. Regularly conducting hyperparameter tuning experiments will ensure the models operate at their peak performance.
  - Exploring advanced optimization techniques such as Bayesian optimization or grid search can systematically identify the best combination of hyperparameters for our models.

- **User Feedback Integration:**
  - Implementing mechanisms to collect and integrate user feedback will help refine the search engine and improve its relevance and usability. Users can provide feedback on the accuracy of search results, which can be used to fine-tune the models and update the dataset.
  - Creating a user-friendly feedback interface within the application can encourage users to share their experiences and suggestions. This feedback loop can be used to iteratively improve the system.

- **Incorporate Data Augmentation Techniques:**
  - Applying data augmentation techniques, such as random cropping, color jittering, and rotation, can help create a more diverse training dataset. This will improve the model's ability to generalize to new and unseen images.
  - Synthetic data generation, such as using Generative Adversarial Networks (GANs) to create realistic variations of interior design images, can further enhance the training dataset.

- **Explore Transfer Learning and Fine-Tuning:**
  - Leveraging transfer learning by fine-tuning pre-trained models on our custom dataset can enhance model performance. Pre-trained models on large datasets can be fine-tuned with our specific dataset to improve detection and classification accuracy.

- Experimenting with different layers and stages of fine-tuning can help determine the optimal approach for our specific use case.

- **Enhance Model Interpretability:**
  - Developing tools and techniques to improve the interpretability of our models will help users understand the reasoning behind search results. Visual explanations of why certain images are retrieved can increase user trust and satisfaction.
  - Implementing techniques such as Grad-CAM (Gradient-weighted Class Activation Mapping) to highlight regions of images that the model focuses on can provide valuable insights into model behavior.

- **Improve Real-Time Performance:**
  - Optimizing the inference time of our models to ensure real-time performance is critical for user experience. Techniques such as model quantization, pruning, and efficient hardware utilization can reduce latency and improve responsiveness.
  - Implementing asynchronous processing and pre-fetching of data can further enhance the real-time capabilities of the application.

- **Enhance Security and Privacy:**
  - Ensuring that user data, including search queries and preferences, is handled securely and privately is essential. Implementing robust encryption and access control mechanisms will protect user data.
  - Regularly auditing and updating the security protocols of our application will help maintain user trust and compliance with data protection regulations.

By focusing on these areas, we aim to continuously improve our application, ensuring it meets the evolving needs of our users and remains at the forefront of interior design search technologies. Our commitment to innovation and user satisfaction drives our ongoing efforts to enhance the capabilities and reach of our multi-modal search engine.

6.3 Future Plans:

Our future plans for the project include several enhancements aimed at increasing accessibility, usability, and functionality. These plans are focused on improving user experience, expanding the application's capabilities, and ensuring its adaptability to emerging technologies.

- **Search by Voice Note:**
  - **Voice Recognition Integration:** Implementing voice-based search functionality using advanced speech recognition technologies like Google Speech-to-Text or Amazon Transcribe. This will allow users to describe their search queries verbally, making the application more accessible and user-friendly, particularly for users with disabilities or those who prefer voice commands over typing.
  - **Natural Language Processing:** Enhancing the natural language processing (NLP) capabilities to understand and interpret spoken queries accurately. This involves training models to handle various accents, speech patterns, and contextual nuances.
- **Mobile Application:**
  - **Cross-Platform Development:** Developing a mobile application that is available on both iOS and Android platforms using frameworks like Flutter or React Native. This will provide users with greater flexibility and convenience, enabling them to use the search engine on-the-go.
  - **User Interface Design:** Creating a user-friendly and intuitive interface that adapts to different screen sizes and resolutions, ensuring a seamless user experience on mobile devices.
  - **Offline Capabilities:** Incorporating offline functionalities that allow users to access previously viewed items and perform searches without an active internet connection.

- **Language Translation:**
  - **Multilingual Support:** Adding support for multiple languages to make the application more inclusive, allowing users from different linguistic backgrounds to interact with the platform. This can be achieved by integrating translation APIs such as Google Translate API or Microsoft Translator API.
  - **Localized Content:** Providing localized content, including translated descriptions and user interfaces, to cater to the cultural and language preferences of diverse user groups.

- **Search History:**
  - **User Profiles:** Implementing user profiles that store search history, preferences, and personalized settings. This will allow users to revisit previous searches, enhancing the user experience by providing quick access to past results.
  - **Recommendation System:** Developing a recommendation system that suggests items based on the user's search history and preferences, leveraging collaborative filtering and content-based filtering techniques.

- **Enhanced Visualization Tools:**
  - **3D Visualization:** Developing more sophisticated visualization tools, including 3D visualizations and augmented reality (AR) features. This will allow users to virtually place items in their real-world environments, providing a more immersive and interactive experience.
  - **Interactive Features:** Incorporating interactive features such as zooming, rotating, and customizing items within the visual search results. This will help users explore different aspects of the items in detail.

- **Integration with E-commerce Platforms:**
  - **Direct Purchase Links:** Linking the search engine with e-commerce platforms to allow users to directly purchase items found in search results. This

will bridge the gap between inspiration and acquisition, making the process more convenient for users.

- **Price Comparison:** Adding price comparison features that display prices from different sellers, helping users find the best deals and make informed purchasing decisions.
- **Inventory Management:** Integrating inventory management systems to provide real-time availability information, ensuring that users can see whether an item is in stock before making a purchase.

- **Advanced Personalization:**
  - **Custom Recommendations:** Using machine learning algorithms to provide custom recommendations based on user behavior, preferences, and interactions with the platform. This will enhance the personalization aspect of the application, making it more tailored to individual users.
  - **Adaptive Learning:** Implementing adaptive learning techniques that continuously improve the recommendation engine based on user feedback and evolving preferences.

- **Scalability and Performance Optimization:**
  - **Cloud Infrastructure:** Migrating to cloud-based infrastructure to handle increased user load and ensure high availability and scalability of the application. This can be achieved using platforms like AWS, Google Cloud, or Azure.
  - **Performance Tuning:** Continuously monitoring and optimizing the performance of the application to reduce latency, improve response times, and ensure a smooth user experience.
- **Community and Collaboration Features:**
  - **User Reviews and Ratings:** Allowing users to leave reviews and ratings for items, which can help other users make informed decisions and provide feedback to improve the quality of the search results.
  - **Collaborative Design Boards:** Introducing collaborative design boards where users can share their favorite items and designs, collaborate on projects, and get feedback from friends, family, or design professionals.

- # Image generation:

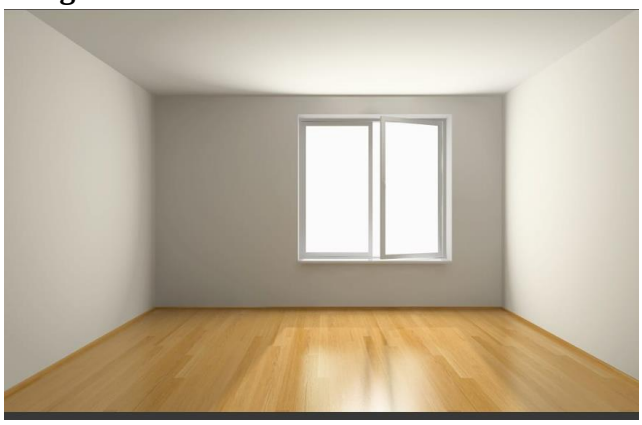
  ## We can use an image generation model to let our users imagine their items or their rooms

  - **Image to image**: It transforms an image into another one using Stable Diffusion, It takes an input image and text prompt as the input and the generated image will be conditioned by both the input image and the text prompt.

    Input:

    Text: "living Room, Sofa, chai, Grey walls ,tv"

    Image:

    Output:

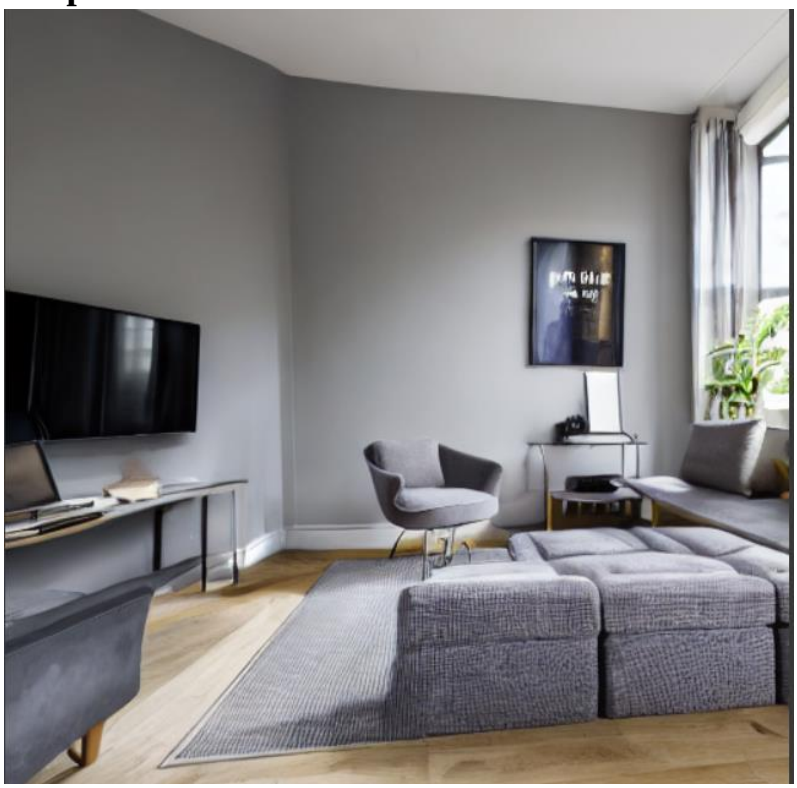
    

- **Text to image**: stable diffusion generates a random tensor in the latent space. You control this tensor by setting the seed of the random number generator.

  Input:

  Text: "An image of living room with a tv and chair and sofa with grey walls"

  Output:



By focusing on these areas, we aim to continuously improve our application, ensuring it meets the evolving needs of our users and remains at the forefront of interior design search technologies. Our commitment to innovation and user satisfaction drives our ongoing efforts to enhance the capabilities and reach of our multi-modal search engine.

References:

Tautkute, Ivona, et al. "What looks good with my sofa: Multimodal search engine for interior design." 2017 Federated Conference on Computer Science and Information Systems (FedCSIS). IEEE, 2017.

Liu, Yinhan, et al. "Roberta: A robustly optimized bert pretraining approach." arXiv preprint arXiv:1907.11692 (2019).

Reimers, Nils, and Iryna Gurevych. "Sentence-bert: Sentence embeddings using siamese bert-networks." arXiv preprint arXiv:1908.10084 (2019).

Reis, Dillon, et al. "Real-time flying object detection with YOLOv8." arXiv preprint arXiv:2305.09972 (2023).

Liu, Zhuang, et al. "A convnet for the 2020s." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2022.

Tautkute, Ivona, et al. "Deepstyle: Multimodal search engine for fashion and interior design." *IEEE Access* 7 (2019): 84613-84628.

Vaswani, Ashish, et al. "Attention is all you need." *Advances in neural information processing systems* 30 (2017).