COL764 - Project

Final Report

Atharv Dabli & Gurarmaan S. Panjeta
2020CS10328 & 2020CS50426

Feat. Multi-modal Querying, Visual Feature Extraction and Word Embeddings

November 18, 2023

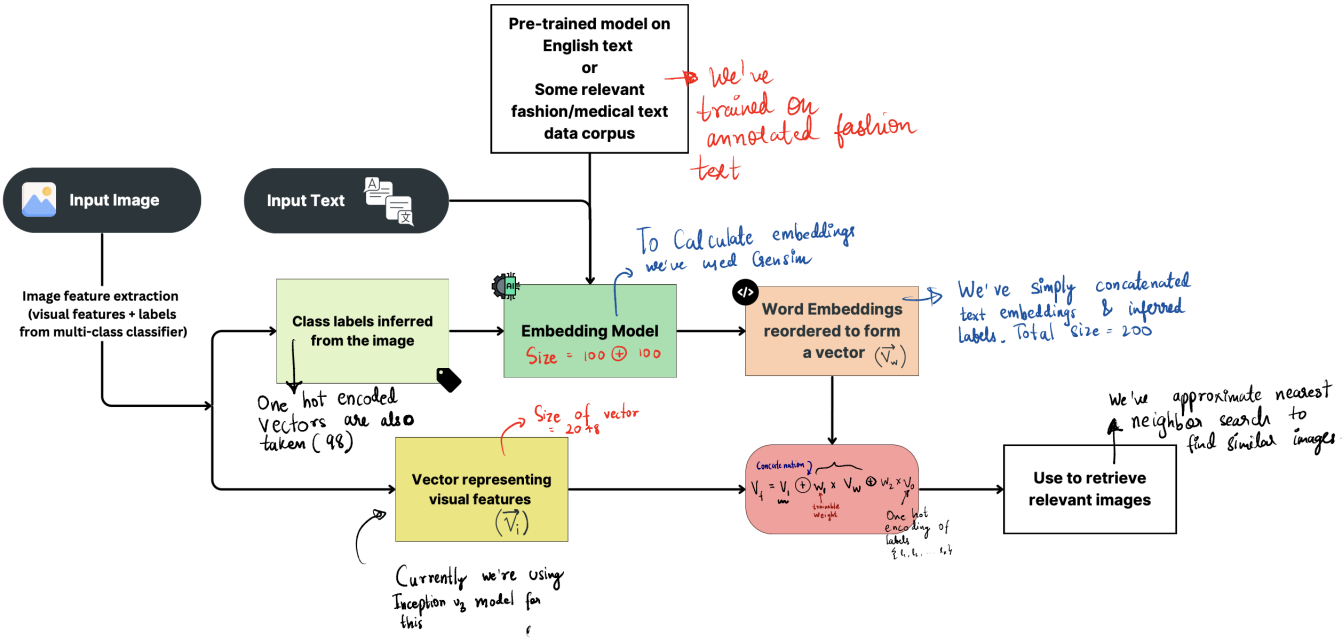
November 18, 2023

Introduction

We build an efficient, fast and resilient query system for textual and image data.

This Domain Agnostic Search fits applications like web-shopping, and performs a directed visual search by incorporating textual evidence. We experiment with and tune the architecture and the relative weights to be given to the various features, and provide a user friendly interface to interact with the system.

Base Pipeline



- We extract visual features from the input images using the Inception v3 model, which helps us identify various objects and characteristics within the images.
- To process textual data, we make use of the Gensim library to compute word embeddings and generate embeddings for our query text.
- To generate vector embeddings for input images, our approach involves two distinct strategies, both of which we have integrated into our model:

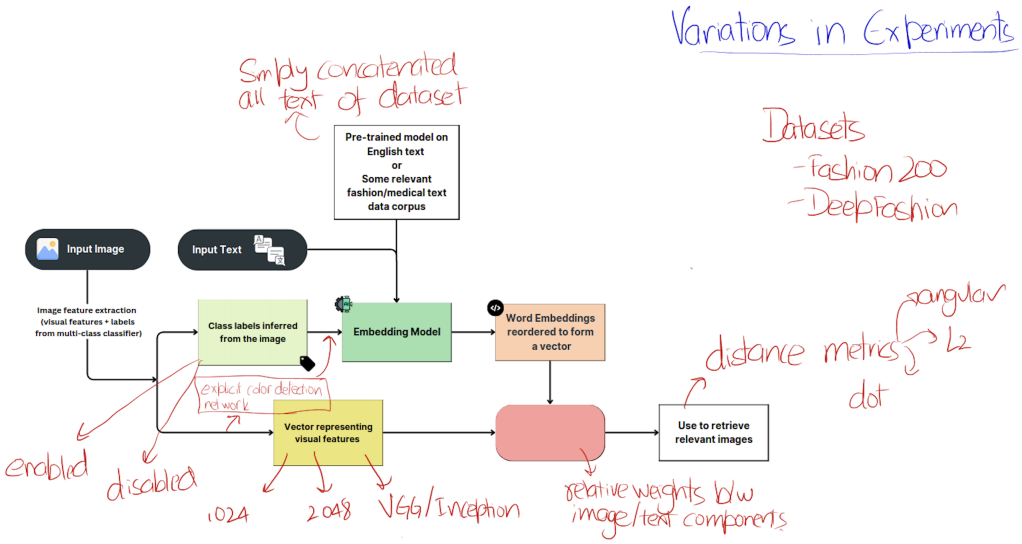
  Utilizing a one-hot encoded vector representation: We create binary vectors where the presence or absence of specific class labels determines the values (0 or 1) within the vector.

  Leveraging class labels as textual tokens: We treat the class labels as words and concatenate them with the input text query, creating a unified representation.
- The resulting vector representation combines multiple elements, including visual features (2048 dimensions), one-hot encoded class labels (98 dimensions), textual query

embeddings (100 dimensions), and label words embeddings (100 dimensions).

- Subsequently, we utilize this comprehensive data, comprising images, their associated annotations, and the 2346-dimensional vector representation for each image, to train an approximate nearest neighbor search data space model.
- The final step in our retrieval process involves obtaining vector representations for both the query text and query images using the aforementioned pipeline and then identifying their nearest neighbors (vis-a-vis Spotify's Annoy algorithm) within the dataset. These nearest neighbors serve as the search results for the given query.

Experimental Variations



Over the base pipeline, we've improved and experimented the pipeline over the following parameters, which are analysed in detail in the subsequent sections :

a) *Enable/Disable Class Inference* - While incorporating inferred class labels would theoretically increase correlation between the two modes, we experimentally verify it by removing this step for the Fashion200k dataset.

b) *Visual feature representation* - Choice over a vector of length 1024 or 2048, and VGG19 or Inception model were experimented with. We find that all combinations qualitatively similar and accurate results, probably because these models are well trained and can disambiguate between these (relatively) small datasets well.

c) *Relative Weights between modalities* - These weights govern the relative importance given to the modalities at query time, hence we tune it for a good default, and allow the user to alter it if the need be.

d) *Distance Metrics* - For efficient dataset construction for the k-nearest neighbours paradigm and the metric actually used as a proxy for the relevance of the result to the query's information need.

e) *Incorporating Color* - Specially for Fashion, color is an important metric, but our models were underperfoming in matching color similarities, probably because they are classifiers. Hence, we explicitly added a network to detect and inform the retrieval system of the colors in the image at both training and query time.

Incorporating color in the search

- To enhance the retrieved results, we have integrated the ColourThief tool to extract dominant colors from images. We basically find the dominant colours of the image in RGB format first, and then convert these RGB into 11 dominant colors. These are white, red, orange, blue, black, purple, yellow and so on.
- First we get the dominant colours in the image in RGB form, top 2-3. Subsequently, we have developed a neural network model trained to classify RGB color values into 11 distinct color categories. This model aids in associating relevant color labels with textual captions present in our training data.
- We then append these colours with the captions that we already had in training data, and then train the model with colors in the captions.
- When user fires a query with a color associated with it, our vector search model would now also include the colors in the training data as a token and its embedding would be used while retrieval.

Approximate Nearest Neighbour Search

For building a vector space model and finding the nearest neighbour of query text and image, we have used Approximate Nearest Neighbour Search which is a technique used to find points in a high-dimensional space that are approximately close to a given query point. The goal is to significantly reduce the search time compared to an exhaustive search through all data points. For this, we have used an implementation by spotify - annoy.

- **Tree Construction**: : ANNOY builds a tree structure to organize the data points. At each intermediate node in the tree, a random hyperplane is chosen. This hyperplane effectively divides the space into two subspaces. The selection of this hyperplane involves sampling two points from the subset represented by that node and positioning the hyperplane equidistant from these sampled points.
- The random selection of hyperplanes is repeated (k times), resulting in a diverse set of trees in the forest.
- **Trade-off with Parameter k:** The parameter k, which represents the number of trees in the forest, is a critical factor in ANNOY's performance. It is a tunable parameter, allowing us to find balance between precision and computational efficiency. We have used a k of 10. As k increases, precision increases, but so does computation time.

1 Weights

Vectors saved while training, per image are simply the concatenation of $V_{visual}, V_{captions}$ and $V_{features}$. At retrieval time, we find the k-nearest neighbours to the vector $V_{queryimage} + w \cdot V_{querytext}$. This w decides the relative importance to the text given. We feel that this design decision is important because:

- To bring both the modes at par in their ability to influence results.
- in case the user has custom confidence, someone may want to find something more/less associated with the image and less/more with the text, they can set this as a custom parameter.
- Default values of $w = 5000$ for inception and 1000 for VGG for a balanced retrieval have been set.
- An instance of the effect of retrieved results by modulating w has been documented in "Results" section.

Analysis of Distance Measures

When processing the Data for saving the database, and at query-time, we experiment with a number of distance measures that can be employed to compute the relevance between query prompts and database entries. Note that this is a non-trivial decision - both text and visual components of the entries must be weighed in. This is partially tuned with the "weight" parameter w analysed above. The other half is tuned using the distance metric.

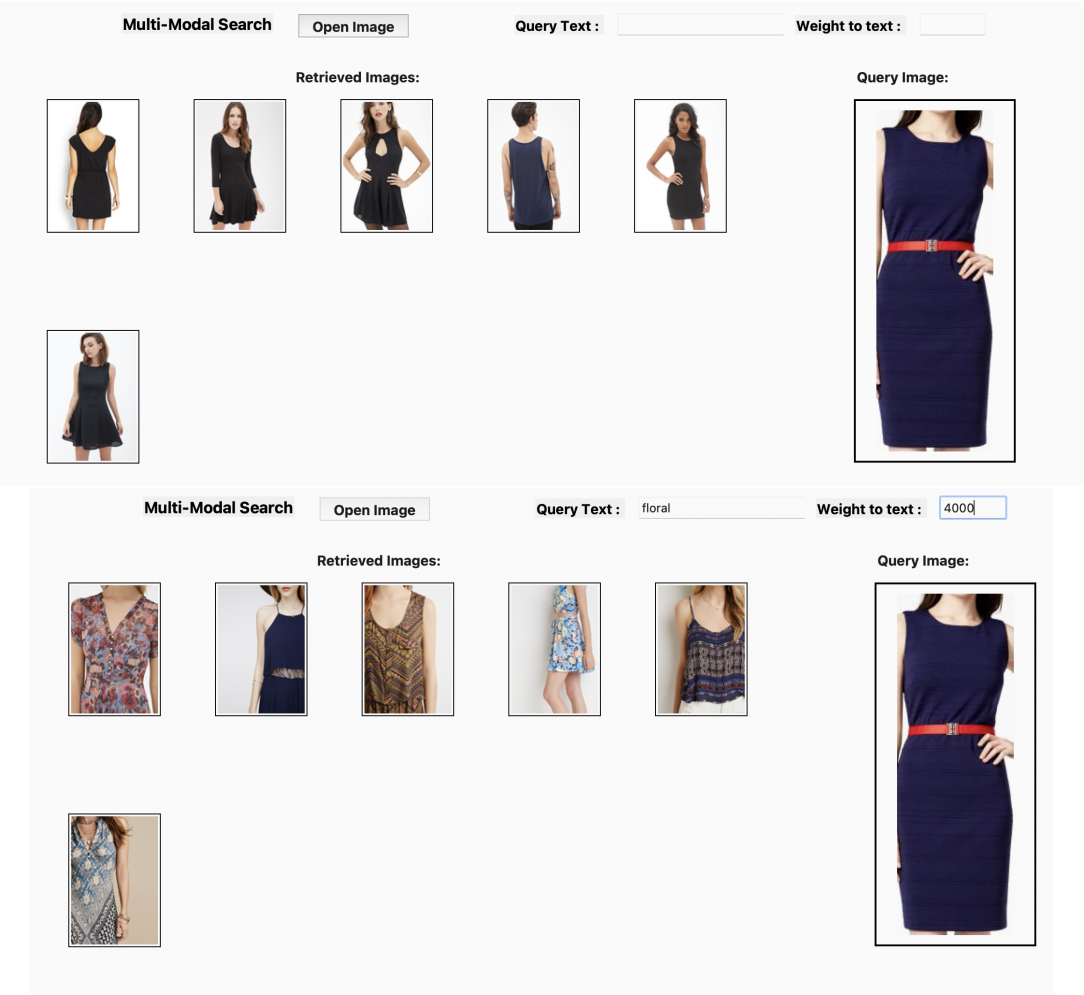
- Dot Product: $\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i$- While this seems like a good approach at face value - closer the entry to the query value, the better - it is inherently flawed. It is sensitive to the magnitudes of the vectors and Even though it reflects the similarity in both direction and magnitude between vectors, it has certain flaws:It is sensitive to the magnitudes of the vectors. If one vector has significantly larger magnitudes than the other, it may dominate the similarity measure. Also, it does not explicitly account for differences in the scales of individual dimensions.
- Angular Similarity: $\left(\frac{\mathbf{a}}{\|\mathbf{a}\|} - \frac{\mathbf{b}}{\|\mathbf{b}\|}\right)^2$. It normalizes the vectors by their magnitudes, making the similarity measure less sensitive to differences in vector magnitudes and it emphasizes the direction or orientation of vectors rather than their magnitudes. This may not perform well in situations where the orientation matters less than the magnitudes. However, in our case, this is more relevant since two different vectors represent an image, hence it makes sense to normalize. Hence we have used it.

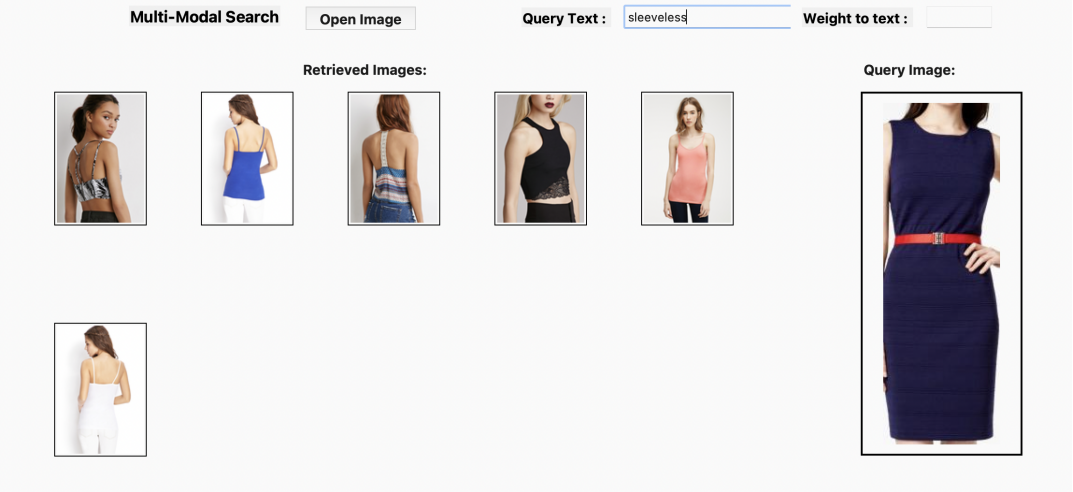
Results

Some samples of our trained models' outputs on (query Image, query Text) are demonstrated:

1.1 Variation in Informational Needs

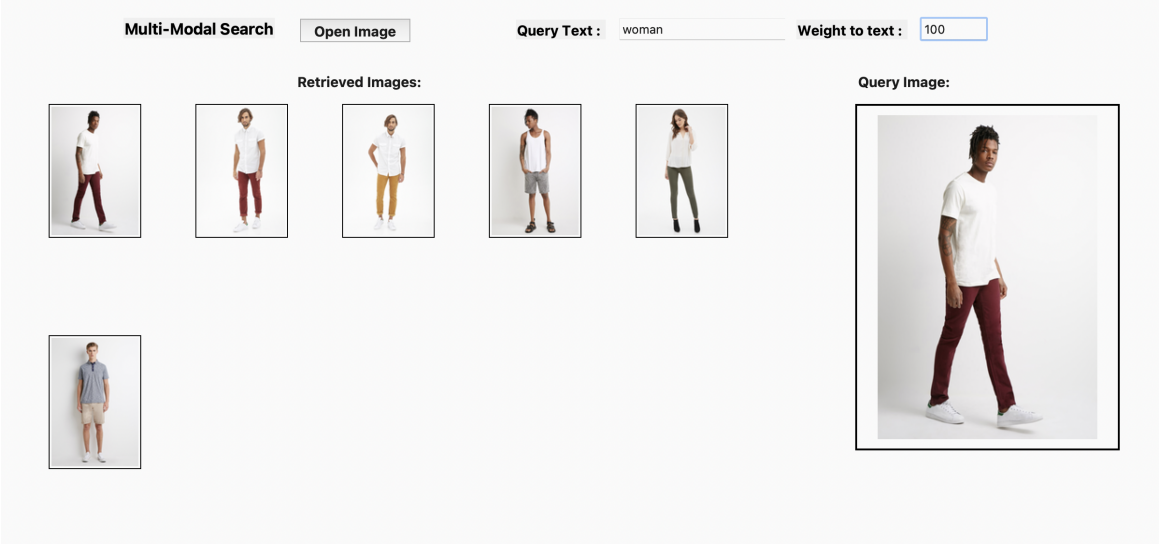
The retrieval system is able to cater to a diverse variety of dimensions. The DeepFashion Dataset has annotations for sleeve lengths, necklines and print patterns, so the user can search along any of these dimensions via text! The images below show the default results when just the image is provided (first result), and contrasted with the "tuning" that happens when textual prompt is provided as well in the later two results, even when the image prompt is the same.

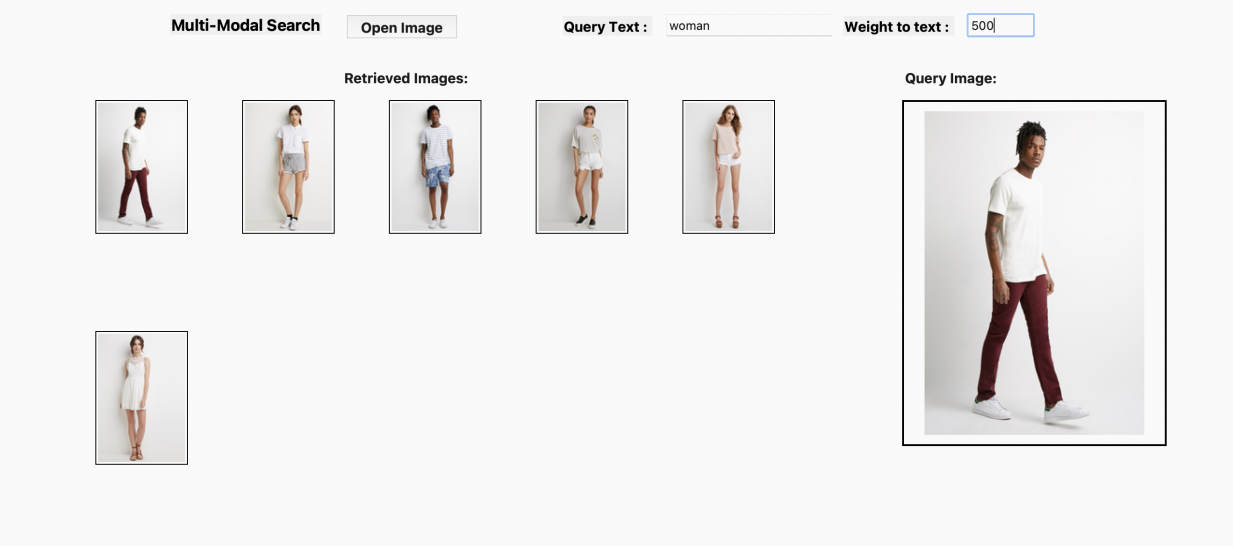


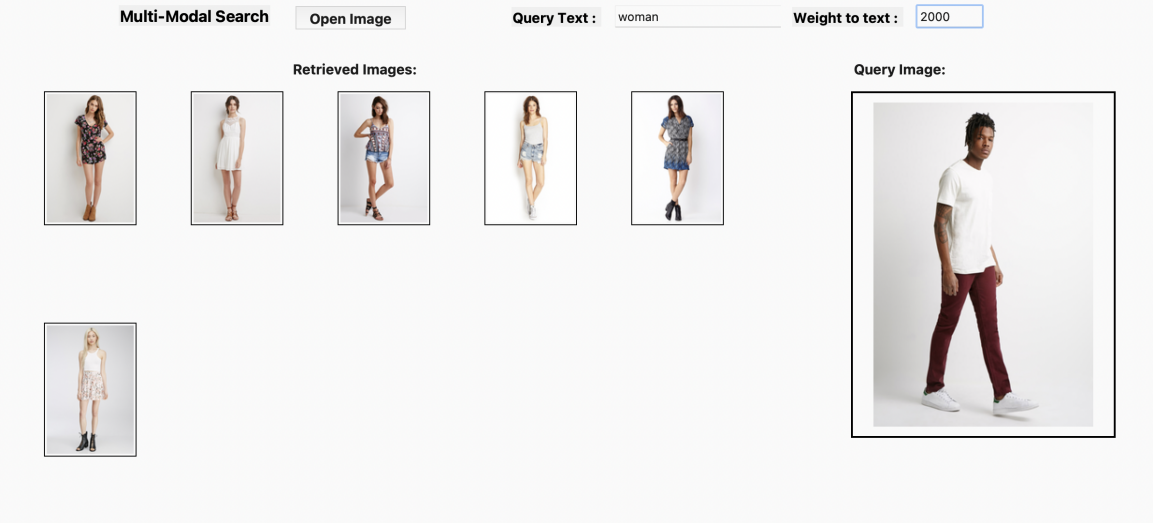


Weights Influencing Results

Gradually Increasing the weight given to text, shows up in the retrieved images. The retrieved results go from men in white t-shirts and pants (low weight, image similarity) to men and women with similar color distribution to the query image (mid weight, both text and image similarity) to only women (high weight, text similarity) Interesting to note how the textual features weigh so much for w=2000 that the even the query image (from the same database, for illustration) is not shown in the top results!

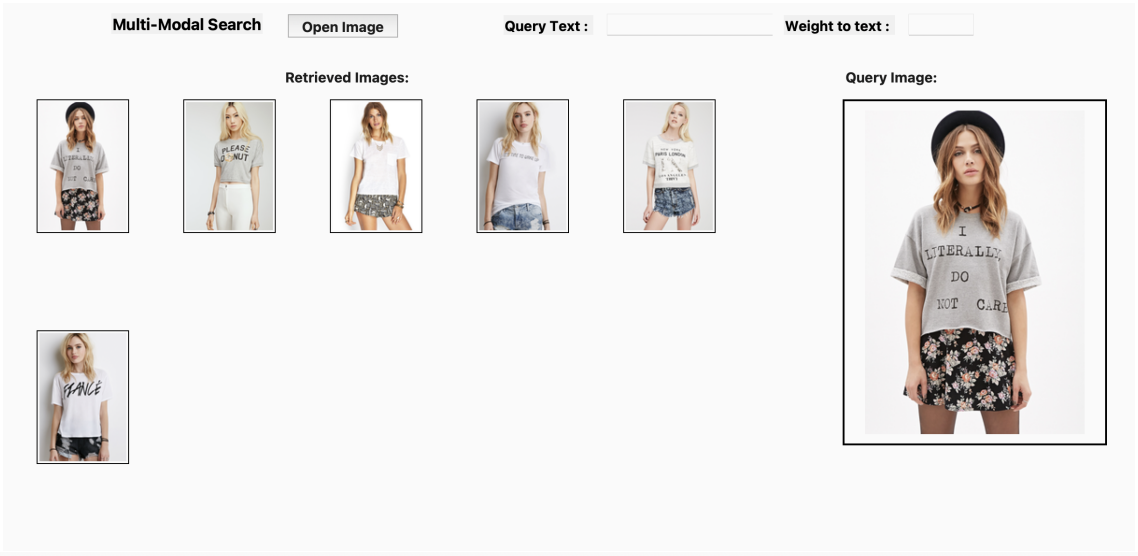


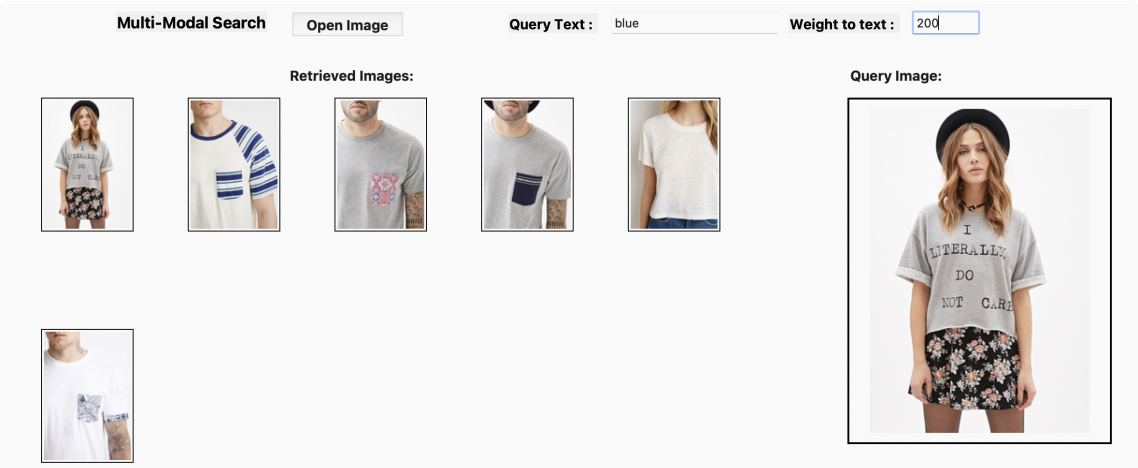


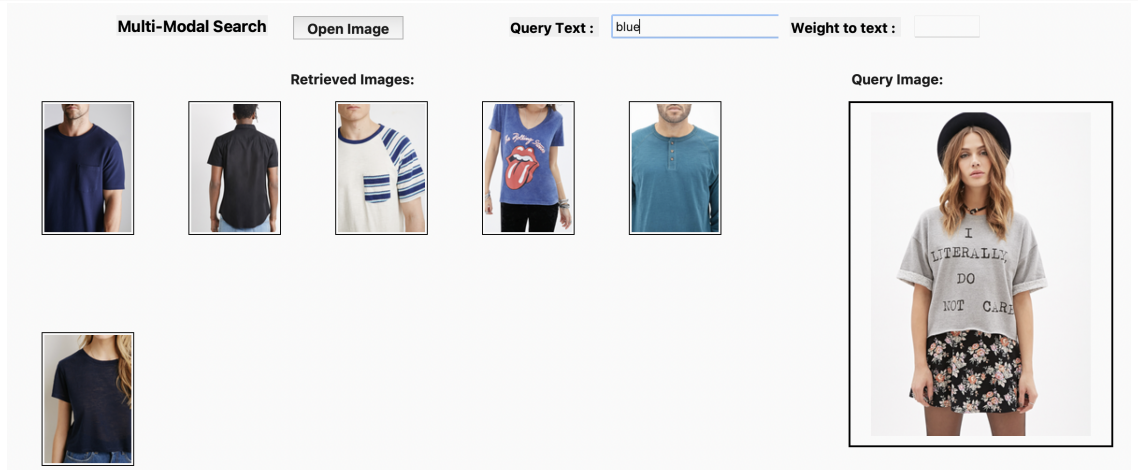


1.2 Incorporating Color!

In the baseline implementation, the learned weights of these models seemed to be abstracting out colors. We explicitly incorporated colors via a color-detector model, appending to captions for building the connection between the (color) word and the image. The results below show how the model can now react to color prompt! These three images show no color prompt, color prompt with low weight, color prompt with high weight (default). One can visually appreciate the change in information need as interpreted by the retriever.

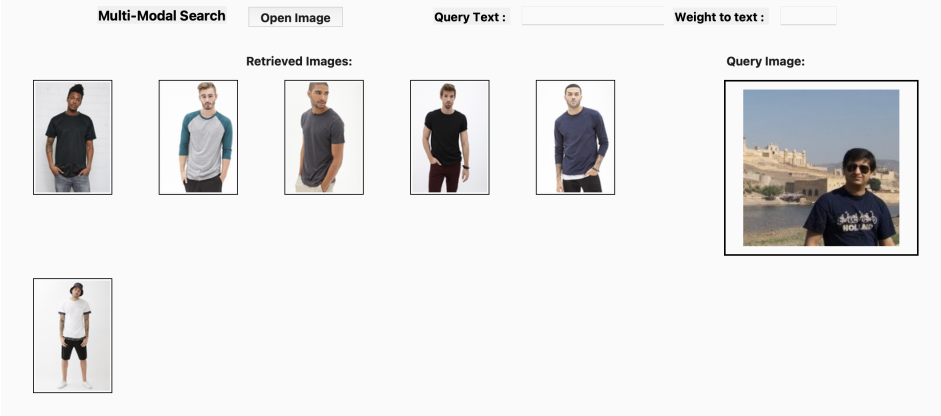

Multi-Modal Search
Open Image
Query Text :
Weight to text :
Retrieved Images:
Query Image:


Multi-Modal Search
Open Image
Query Text :
blue
Weight to text :
200
Retrieved Images:
Query Image:


Multi-Modal Search
Open Image
Query Text :
blue
Weight to text :
Retrieved Images:
Query Image:

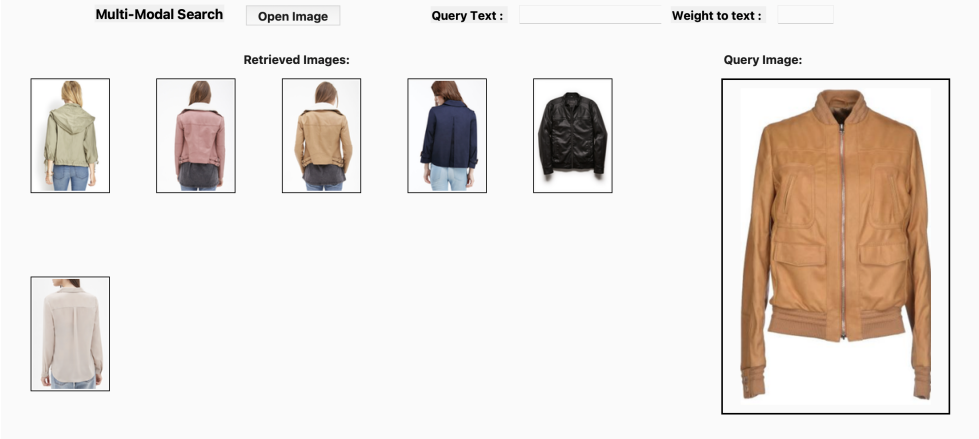
Background effects

Query Image sourced from the internet. We can see that the retrieved images resemble the model's positioning and capture the structure pretty well. Also note that the retriever seems to be able to tune out the background of the query image pretty well.



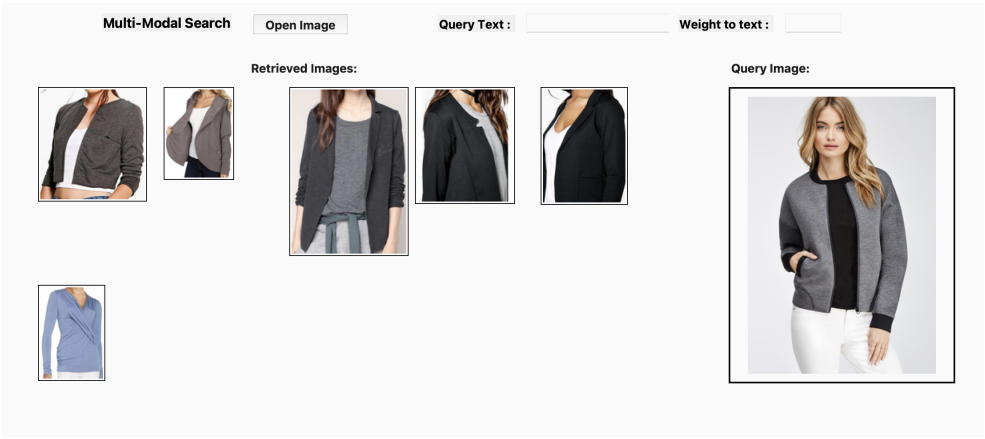
Using a Different Model (VGG)

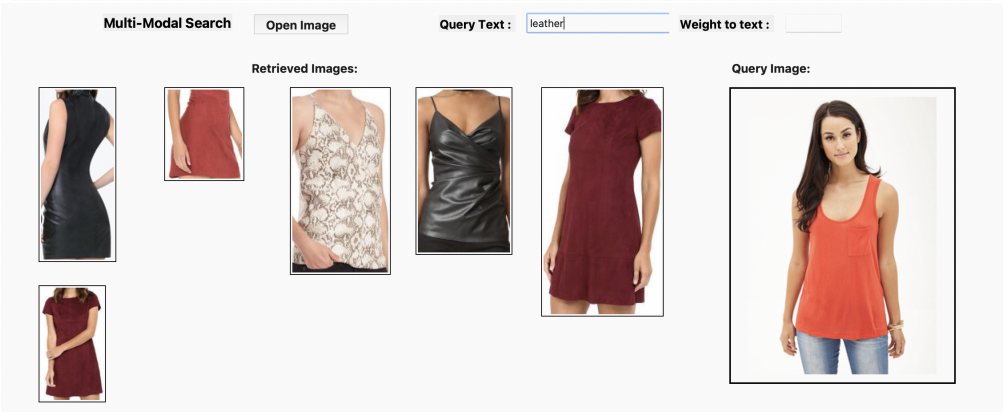
We perform similar queries, but over a different model architecture, that captures the visual features differently. The three results below are (1) a sample demonstration, (2) without any text (3) with text. Interesting to note is the way the model has captured the notion of "striped" (the query text) as variations in colour, and cannot hence distinguish the fine difference between ripples on the fabric and stripes in the fabric in (3).

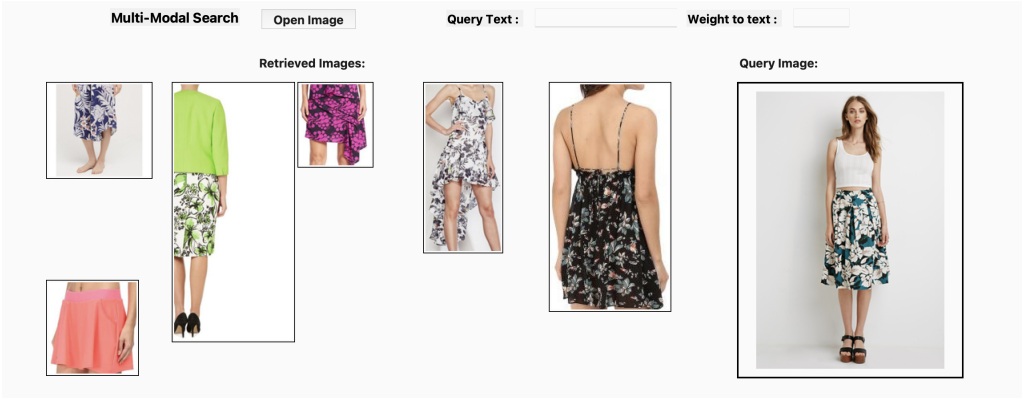


Generalisability and the Fashion200

Fashion200 does not include the faces of people, and only includes parts of clothing. We are still able to focus on the relevant parts of the query image and retrieve relevant results even when query images were full shots of people. The first 2 results show the retriever auto focusing on the upper half of the body because of the way the query image emphasises it, and the last result shows how the retriever focuses on the lower half.







Fashion 200

To compare what advantage our approach of deriving text promts from images had helped, we tried carriying out the retrieval without the classification on the Fashion200 Dataset . Our Results show very similar performance, rather visually beats the Deepfashion systems because the images in F200 are cropped images of individual clothing items, hence whenever the retrieval system finds a match, it is because that component has a high similarity with the query image.

Metric for Evaluation

Assessing the outcomes of similar item searches within a dataset of comparable items poses a significant challenge in terms of objective scoring. Furthermore, the sheer scale of the database renders manual identification of the most similar items impractical. Therefore, we came up with a cosine-similarity index.

The Vector used for retrieving images was weighted for better retrievals and custom importance. However, once these images had been retrieved, comparing their captions + their visual features with the query text and visual features (unweighted) could provide us how similar the query is to the dataset, via the cosine angle between the vectors!

This metric captured our intuition well. For very similar images, the score was as high as 0.99, and for results where text mentioned incorporated other features not in the image, the retrievals' image vectors were distant , and the score would even drop down to 0.51. However, from visual analysis the metric seems to be a monotonic function of the similarity of the retrieved image with the information need.

```
Angle found between query and retrieved :  0.999787663528396
Angle found between query and retrieved :  0.9222292594266835
Angle found between query and retrieved :  0.9201511129931276
Angle found between query and retrieved :  0.9179004015957378
Angle found between query and retrieved :  0.8927831894734813
Angle found between query and retrieved :  0.8896860507292612
Final score efficiency =  0.923756279624448
```

```
Angle found between query and retrieved :  0.5343561247427194
Angle found between query and retrieved :  0.4655441208364559
Angle found between query and retrieved :  0.5543435591831685
Angle found between query and retrieved :  0.4934150958636179
Angle found between query and retrieved :  0.5550747806737748
Angle found between query and retrieved :  0.5017500299274
Final score efficiency =  0.5174139518711894
```

Data Sets

The DeepFashion Dataset, includes 44,096 high-resolution human images, manually annotated for cloths, textures and accessories. We found that the Dataset was extremely well-annotated, overall complete (very few missing annotations) and actually fit well to our problem at hand.

One weakness, however, was the absence of color-annotations (blue, green, etc.) in the text annotations, which meant the model's text embeddings learnt (and the text-image mapping learnt) can not differentiate between colors. However, it can differentiate between textures and fabrics, say between leather and cotton, and can take thus also textual inputs to tune results. As mentioned in the Section" "incorporating Color in the Search", and demonstrated in the "Results", we are able to mitigate this weakness by explicit detection of color, and appending colors per-image to the caption corpus.

Fashion200 Dataset - Includes 300k images, with annotations for all wearables. This dataset was incomplete for analysis of the full picture - captions generated here were not standardised as feature vectors, and were rather English sentences. This meant that the classifier could not be trained on predicting features, and we had to disable the concatenation of vectors that were to be generated from the inferred features, thereby reducing correlation between the two modes.

However, the english sentences were fairly detailed, and thus query text could still find relevant images. This was a larger dataset, (more than 300k pictures), and we were thoroughly impressed with the retrieval speed - less than 2 seconds!

Goals Accomplished

- Fine tuned the weights and sizes used during concatenation of the visual and textual parameters.
- Diversification to two datasets (DeepFashion and Fashion200) , and obtaining similar results in both.
- Incorporated an explict color-feature. This enables text query of ”blue” to yield blue images (despite this dataset lacking color annotations)
- Diversifying across two models (VGG19 and Inception) and achieving similar results.
- Investigated result variations when different metrics were used for evaluating relevance (dot product and cosine similarity).
- Developed and evaluated a metric to report relevance of image to query, and to report the overall performance of the system.
- Created a smooth pipeline to streamline training and querying phases, and a user-friendly platform to query/display results.

References

[1] Jonghwa Yim, Junghun James Kim, Daekyu Shin , “One-Shot Item Search with Multimodal Data”

[2] The DeepFashion - Multimodal Dataset Repository

[3] The Fashion200k Dataset repository

[4] Spotify’s Annoy - Approximate Nearest Neighbors in C++/Python optimized for memory usage and loading/saving to disk

Codebase

Our code is available here!.