CS6200 Information Retrieval - Final Project Report
Image Search Project

NUID:001302261 - Zihao Zhang

Github Source Code: https://github.com/asadafa123/CS6200-project/tree/master

1. Introduction

Image has always been an important type of media on the internet. Most internet applications nowadays cannot avoid interactions with images. In fact, there's a lot of trendy apps or websites that take images as their core.

For example, Google/Bing provides traditional services of image searching, while websites like Flickr and Pinterest are more focused on image retrieval. Social media like twitter and facebook cannot live without images while there are such applications like Instagram that build social networks based on images. Image retrieval system and a properly functioning image search engine is the common need of all the applications mentioned above.

In this project, a demo of Image based social network web application is built, while an image search engine is implemented, offered and evaluated.

2. Problem Identified

a. Relevance issue

For a traditional full-text search engine, documents are collected by the engine and the text inside are indexed by certain methods in order to calculate relevant scores for certain queries. However, in particular image searching circumstances, the image itself sometimes is not the target of the search engine. The object that is recognized as document and indexed by the

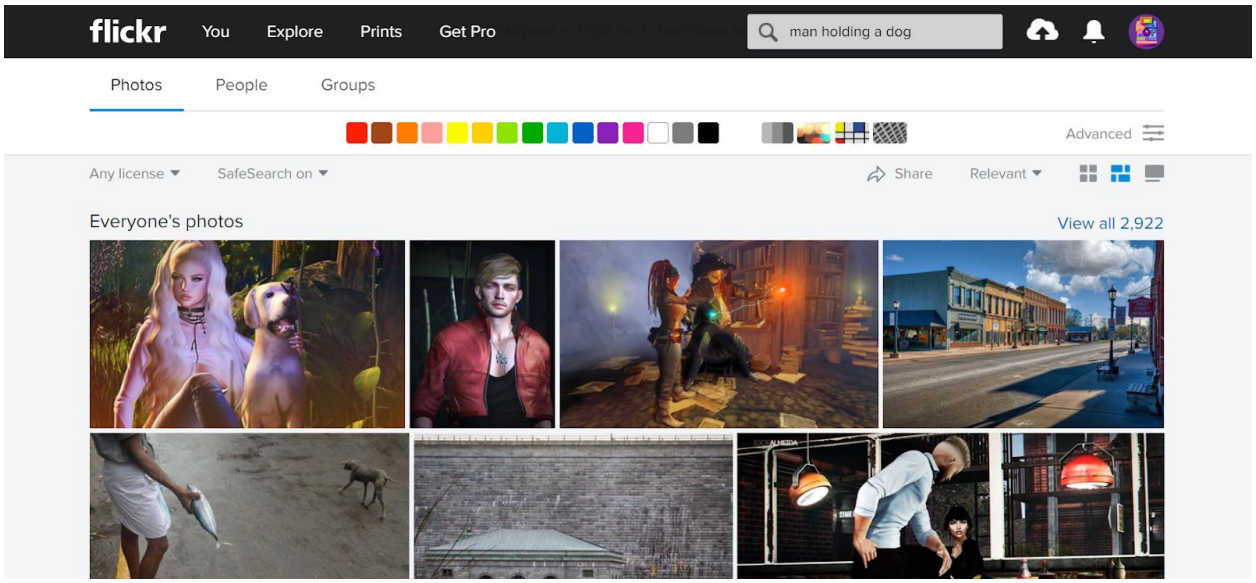
search engine is not the image, but may be the titles, articles and tags along with it. The content of the image is less valued in some image search engine.

The problem is how to index the content of images. By introducing deep learning algorithms, the engine can extract certain features from the pictures and index those features directly. But this kind of approach has limited support from traditional search engine solutions, and may cost more resources on the server. Another way is to extract the semantic information of such images as part of the document, then apply traditional full-text search. The extraction step is not limited to machine learning methods but also can be done in traditional ways, like manually labeling. In this project, the second approach is implemented and evaluated.

b. Example

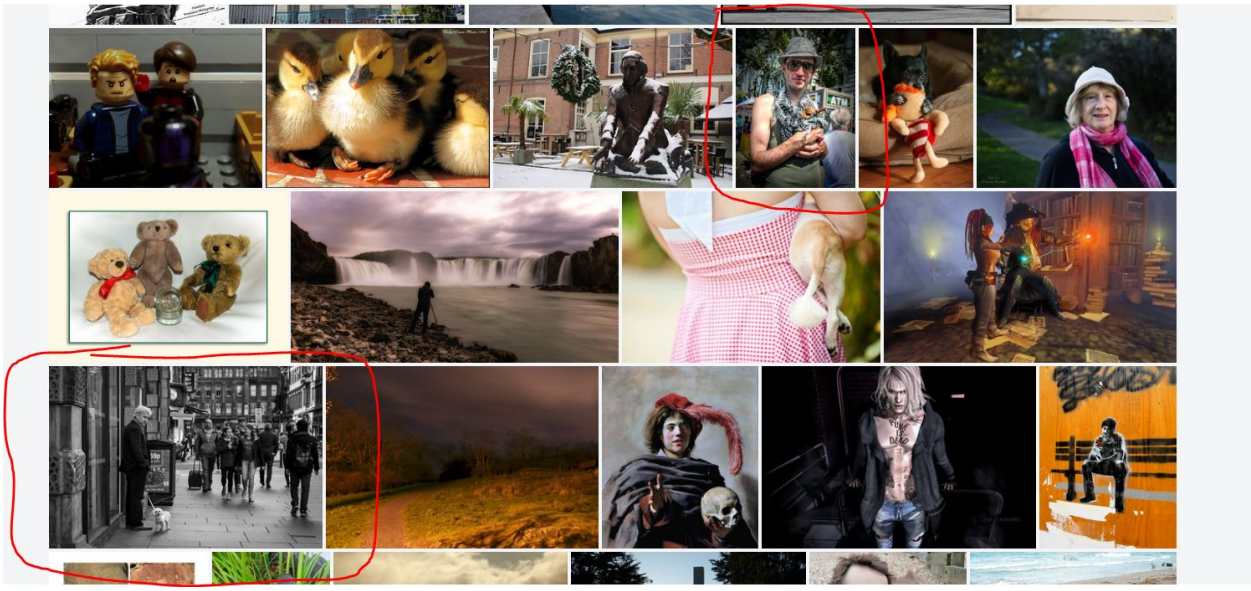
This example issue is spotted on www.flickr.com. Flickr is a photo-sharing platform and social network where users upload photos for others to see. It provides a strong image search engine with interesting functionalities like search pictures by its tone of colors, or by patterns. However, the issue of image content relevance is identified in this searching system.

By searching "**man holding a dog**" in Flickr, we can examine the results it returns.
The top 10 relevant result seems bad:



From the appearance of these, though we can identify some elements like **'man'** or **'dog'**, the results are not satisfying at all. Since we are sorting the results by its relevance to the query **"man holding a dog"**, there must be some reason that these results popped to the top.

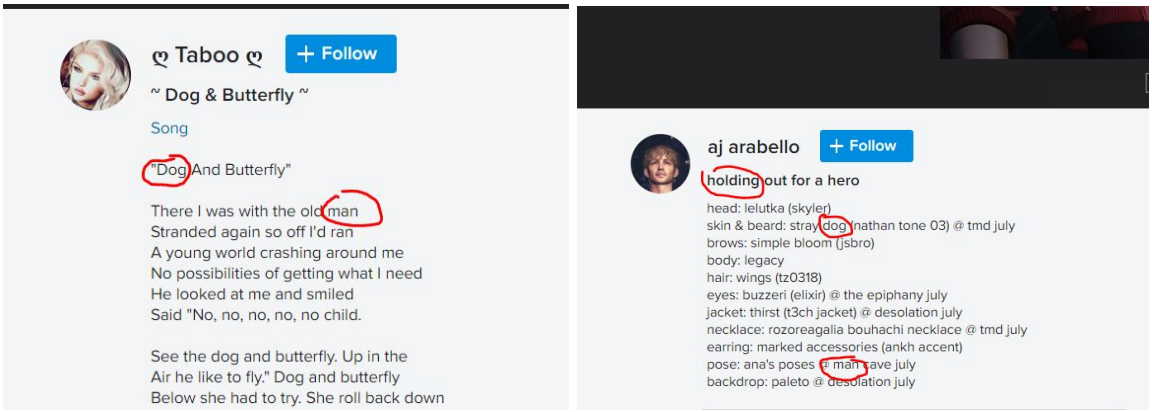
The image that seems relevant to the description" **man holding a dog**" shows until 9th 11th 26th, which are circled out in the snapshot below.



c. Problem analysis

The reason that the pictures on top 5 have that high values of relevance to the query is because the images are posted with **short articles** written by the users, which may contain **keywords "man, hold, dog"**. Since we are under a circumstance of social network application, apparently some of those articles can be arbitrary. They can be users sharing their moods, lines of poetry, or just random words at all. On the other hand, if we want to do queries based on the content of such posted images, we usually cannot get the ideal results. This phenomenon indicates that the semantic content of the image in this search engine does not take into account or at least does not have enough weight on relevance calculation.

As shown below, the top1 relevant result is posted along with an poetry called "Dog and Butterfly". Keyword "Dog" showed up in both the title and article, while "man" also can be found in the article. Thus this post can be rated high in this search engine.



E.g. the article that posts along with the top 1st & 2nd result.

3.Solution Proposed

As mentioned above, we try to address this problem by introducing the semantic information of images and make the search engine take use of them. A post consists of

{author, title, article, image_url, date_posted},

In traditional full-text search engines, under this situation, a document is often defined as

{author, title, article, date_posted},

This kind of definition can perfectly handle full-text queries. But as mentioned above, it cannot handle image content related queries.

In this project, a document is defined as

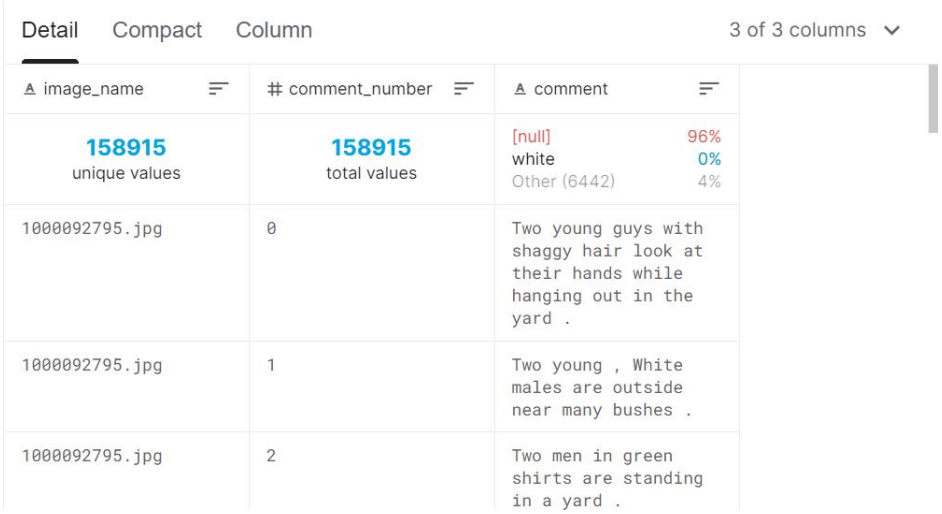
{author, title, article, ***description,*** date_posted},

So frankly this is also a full-text search engine, which takes description of the posted pictures into account.

The problem now comes to how to get the description of those pictures. In this project, the dataset itself contains several descriptions of the according image. For general user posts, the author can manually add descriptions of the image, while other viewers can also add descriptions to the post. The added descriptions are indexed in real-time by the search engine.

4.Dataset

The Flickr30k dataset has become a standard benchmark for sentence-based image description. This paper(http://bryanplummer.com/Flickr30kEntities/) presents Flickr30k Entities, which augments the 158k captions from Flickr30k with 244k coreference chains, linking mentions of the same entities across different captions for the same image, and associating them with 276k manually annotated bounding boxes. Such annotations are essential for continued progress in automatic image description and grounded language understanding. They enable us to define a new benchmark for localization of textual entity mentions in an image. We present a strong baseline for this task that combines an image-text embedding, detectors for common objects, a color classifier, and a bias towards selecting larger objects. While our baseline rivals in accuracy are more complex state-of-the-art models, we show that its gains cannot be easily parlayed into improvements on such tasks as image-sentence retrieval, thus underlining the limitations of current methods and the need for further research.



Cited from https://www.kaggle.com/hsankesara/flickr-image-dataset

The dataset contains 31784 pictures which are mainly daily photographs about people or pets, doing things that can be easily described in a couple of sentences. Each picture is mapped to 1-4 sentences of description, the mapping is stored in a csv file.

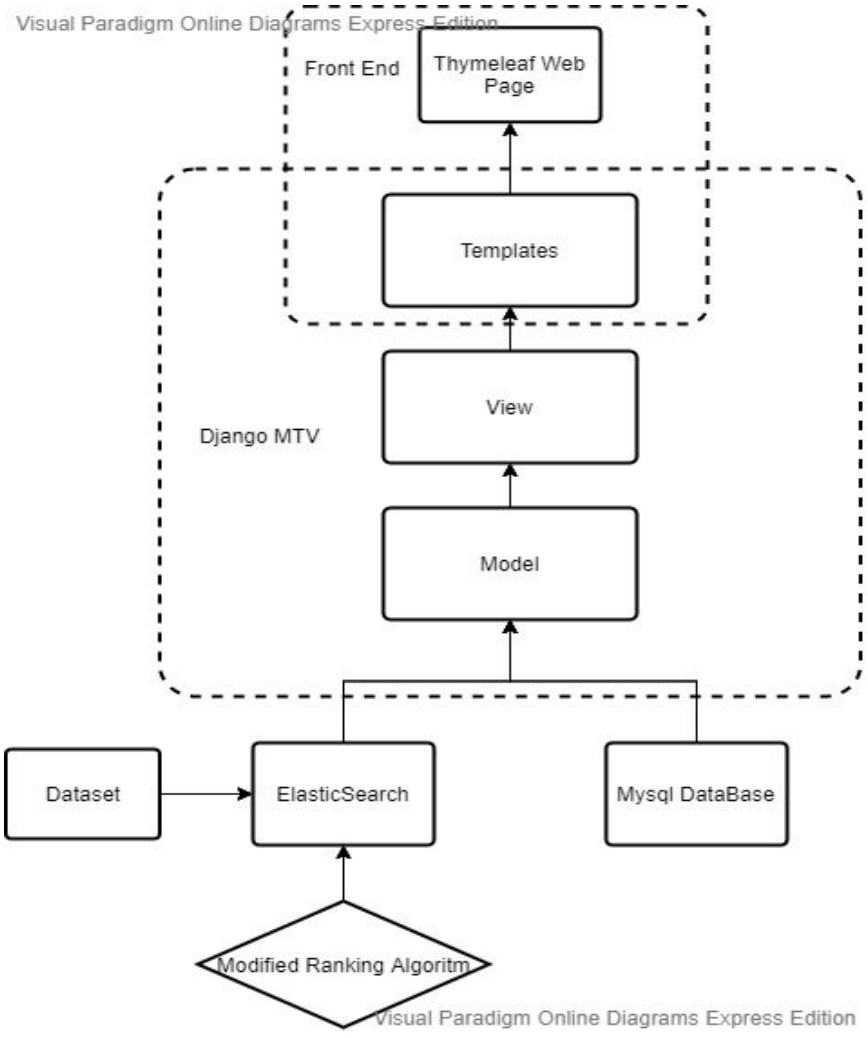
In this project the dataset is pre-processed, combining those descriptive statistics together and mapped to the images accordingly for further initializations mentioned later.

5. Project Structure

Images in the Flickr30k dataset are combined with semantic description of its content, which can firstly help me to build a demo search engine that can keep track of the semantics of the images and build a more proper ranking algorithm according to the problem described above.

a. Structure

For this demo image retrieval system, I will use django as the basic framework of this whole application, and ElasticSearch as the core search engine, sqlite3 as the database. For the front design, I'll use Django template language combined with Thymeleaf, html, css, javascript to create a stylish user interface. The searching result should be nicely placed (Like what is done in Pinterest or Flickr) and ranked by relevance.



b. Expansion on the ranking algorithm:

Indexing: The “document” in the dataset should be composed of Title, text (can be seen on the website) and the semantic description of the image(hidden metadata). In this project the search engine created an index on both of these three parts, introducing “description of the image” into relevance calculation. This should improve the problem mentioned above.

Ranking: Primarily I will calculate the TF-IDF value for each term in the query and calculate cousin similarity between the query and documents in the dataset. According to the performance, I’m going to adjust the weight of the three parts(title, article, description) for better precision and recall rate.

6.Implement Details

For this part of the report I’ll split the detail introduction into two parts, Back-end and Front-end.

a. Back-end

Since the framework chosen for this project is Django, the description of back-end will follow the structure of the framework: models, views, templates.

1) Models

Model in Django represents the mapping between the objects of classes we are creating in the project and what entity is stored in the database. Mainly we have “User”, “Post” as main entities. Also “UserConnection”, “Like”, “Comment” are added for auxiliary functions.

The **“User”** class is inherited from Django class “AbstractUser”. Its main functionality is nearly identical to a Django user class. The choice of this is because Django provides additional administrative functions for users, I can simply manipulate the users attributes in the Django admin system.

The **“Post”** class is inherited from Django class “models.Model”. Which means this class is created from draft. The field of it can be described as:

{author, title, article, description, image, date_posted},

Where **“author”** is a ForeignKey of the User model, title, article, description are all Text fields, image is a ProcessedImageField to store the url of a pre-processed Image for later use, date_posted is a DateTime field.

What’s more, the **UserConnection** class is designed for storing the relationship between users, and **Like, Comment** are auxiliary models for posts to record if a post is commented or liked by others.

Each model in django can be mapped to a table in Mysql or other relational database. By implementing models, the Django framework can store the data or objects we are using in the project to the database and make them persistent. In other words it’s an interface that the framework interacts with the database.

2) Views:

Views in django process Http requests and link them to logic functions. It catches the Http request sent from the front-end, does the logic and process, redirects to another webpage and shows the results on it.

For example, we take **“search(request)”** to explain the idea. In Insta/urls.py,

path('search/',search, name='search'),

Directs the http get request with url http://localhost:8000/Insta/search/ to the function search(request) in views.py. The function here extracts the query inside the request, and the searching options it chose.

The search() method interacts with the elasticsearch local server and processes the query sent to Django backend. The search result then is sent back to Django, wrapped up, and sent to the front end. At the mean time the user would be redirected to a newly rendered “search_result.html” page to see the results.

3). Database and ElasticSearch

The project chose **sqlite3**, which is a light and portable relational database. It functions similarly to Mysql database and well adapted by the Django framework.

For the search engine, ElasticSearch is chosen in this project. As long as the elasticsearch server is running, django has a bunch of supportive modules for django-elasticsearch

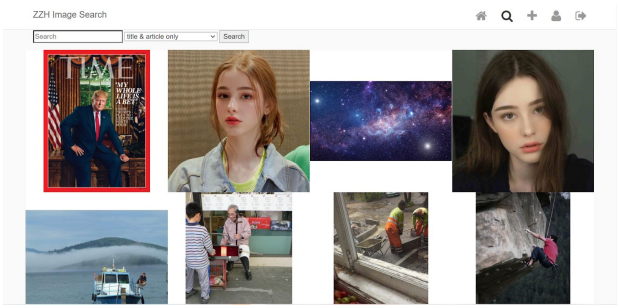
interaction. In this project, we imported “elasticsearch”, “elasticsearch_dsl” to implement such interaction.

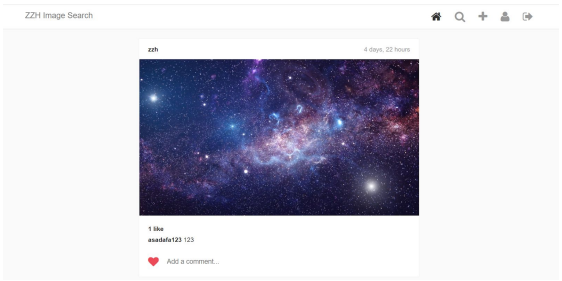
Module “elasticsearch” provide basic communication between Django and ElasticSearch server, while “elasticsearch-dsl” provides basic classes for indexing. It can easily index all the existing posts in the database as well as automatically index the newly posted ones.

b. Front-end

For the front-end, this project is trying to demonstrate a stylish UI like Instagram Or Flickr. On Insta/posts/ page the design of posts elements are like Instagram while in the searching page this project tried to make the result better presented, like Flickr does.

The introduction of the front-end can be simple. Here I’m implementing basic interacting functions like “pressing Like”, “add comments & descriptions” or “follow” with jquery, and make sure the import elements like pictures or posts are restricted inside a box element, so the UI should look nicer.

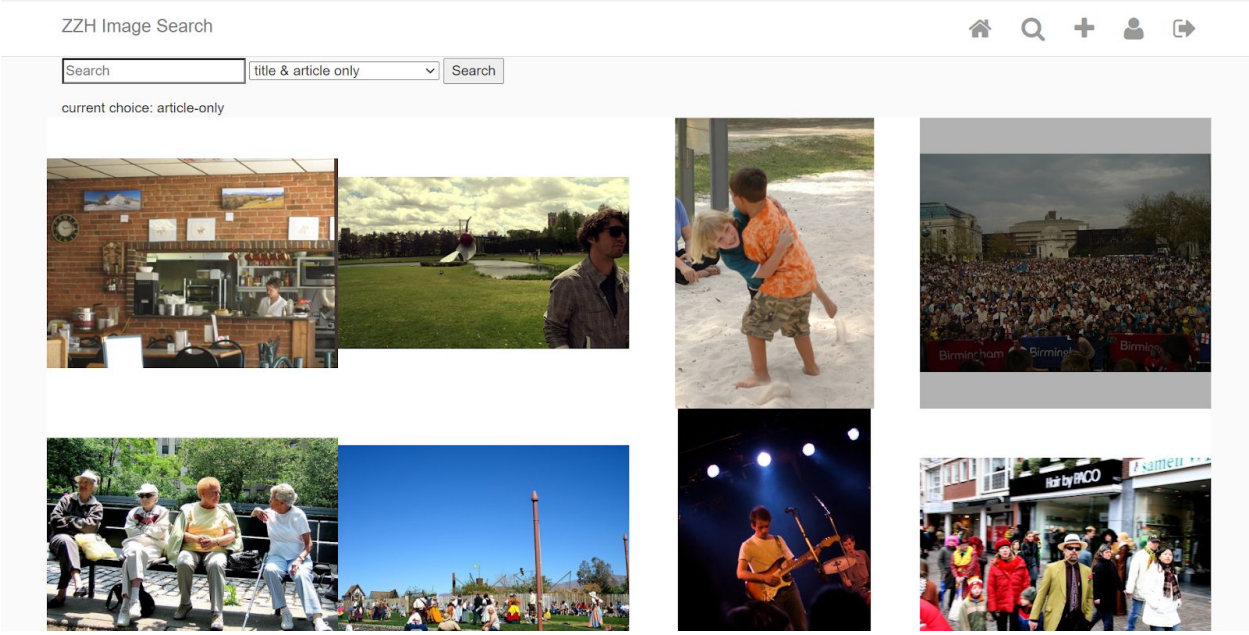




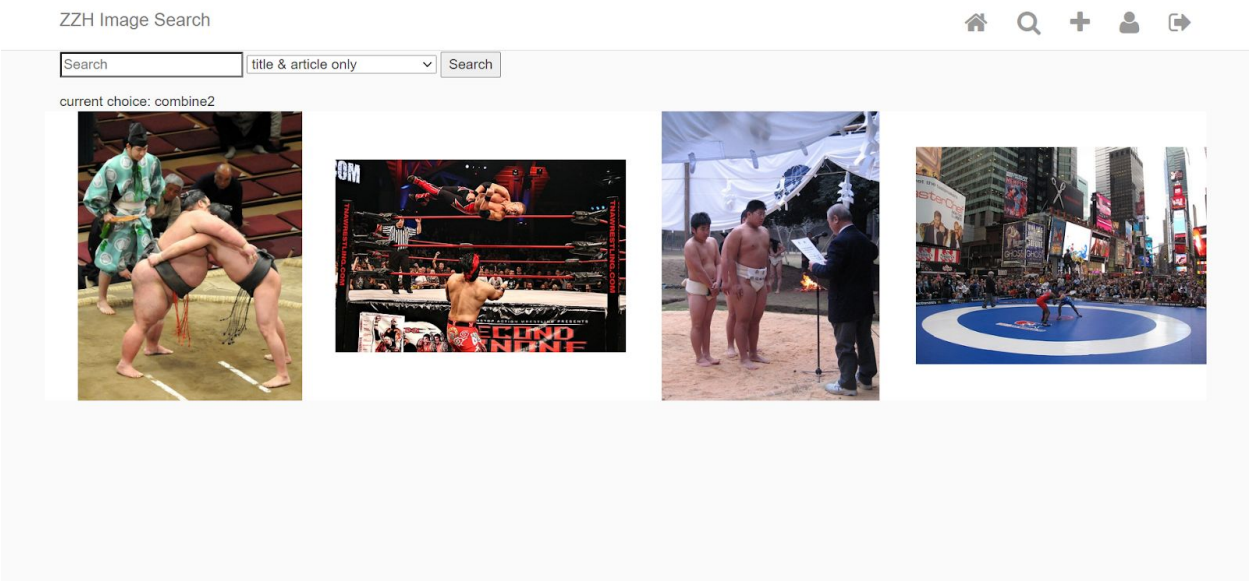
7.Conclusion & Results

Take **“man wrestle”** as a test case on this project:

Result for “title & article only”:



Result for combined with description”



As shown above, adding the weight of description, we can handle queries based on image contents.We tried several other test cases on the lecture and the result is acceptable.