TECHNICAL UNIVERSITY OF MUNICH

TUM Data Innovation Lab

Let me show you! Developing an image-based document search

Authors	Anika Apel, Piotr Chodyko, Kyle Hiroyasu, Festina Ismali, Hyein Koo
Mentor(s)	Dr. Robert Pesch, Julia Kronburger (M.Sc.), Dr. Andre Ebert inovex GmbH
Co-Mentor	Olga Graf (M.Sc.)
Project Lead	Dr. Ricardo Acevedo Cabra (Department of Mathematics)
Supervisor	Prof. Dr. Massimo Fornasier (Department of Mathematics)

Feb 2021

Abstract

With the growth of retrieval and information systems a new type of search engines has emerged: image-based search engines. Instead of searching textual documents by text a user can search for a specific document by uploading a picture of the resource. Such a search engine can be especially useful to improve user experience of different expert tools. An exemplary application is in maintenance to find documentation and manuals given the photo of a defective component. These use cases motivated the development of an image-based document search application in this project.

The underlying task of this application is finding relevant textual descriptions for a given image and is referred to in the literature as image-to-text retrieval. To realize this task, we implement and evaluate in our project different state-of-the-art deep learning techniques to embed text and images in a joint visual-semantic vector space. We explore VSE++, a pairwise learning method, as well as Visual Semantic Reasoning, an attribute learning method. With a joint-embedding approach, the retrieval can be formulated as a similarity search in the retrieval corpus using an efficient and scalable vector-based search engine.

The main goal of this project is to develop a production-ready end-to-end system. Throughout the whole project, design decisions were carefully considered to achieve scalability and reliability of the application as well as good user experience including retrieval quality and fast response time. We present a cloud-based microservice application running in a Kubernetes cluster ensuring these desiderata. As we are presenting an end-to-end-application the cluster includes a web application for the user interface, a REST-based backend, a service for training and evaluating deep learning retrieval models as well as the distributed search engine Elasticsearch. To overcome the challenges of deploying machine learning models we also include MLflow as a service component to manage the machine learning lifecycle.

We show in this report how to shift machine learning based software into production, assuring extensibility of the system as well as scalability and ease of maintenance. Overall, we achieved a good retrieval performance on the Flickr30k and COCO dataset as well as fast response time to user requests. We also see that methods like attributes learning significantly improve the retrieval quality but also lead to a longer inference time due to additional extraction of specific attributes. The system we are presenting can easily be adapted to any kind of retrieval corpus by training the presented deep learning models on provided data.

Contents

Acronyms

AMT Amazon's Mechanical Turk. 12

API application programming interface. 19

BLEU Bilingual Evaluation Understudy. 13, 14, 23

CD Continuous Deployment. 5

CI Continuous Integration. 5, 27

COCO Common Objects in Context. 12, 13, 15, 23, 24

GCN Graph Convolutional Network. 8, 9

GCP Google Cloud Platform. 17

GRU Gated Recurrent Unit. 8, 9, 14

IaaS Infrastructure as a Service. 17

MVP Minimum Viable Product. 5

NLTK Natural Language Toolkit. 14, 15

OSCAR Object-Semantics Aligned Pre-training for Vision-Language Tasks. 7, 10, 14, 27

VSE++ Visual-Semantic Embeddings with Hard Negatives. 7–9, 14, 15, 23, 24, 27

VSRN Visual Semantic Reasoning Network. 7–9, 14, 15, 23, 24, 27, 28

1 Introduction

With the increasing adoption of smart devices and the explosion of available data, methods to search and retrieve information have grown in importance. In the most familiar settings text-based retrieval systems such as modern day search engines, allow us to search large corpora of documents to find relevant information. This application is not only valuable on its own but also plays a fundamental role in even more advanced systems such as question answering and machine understanding.

In addition to such applications, research has also focused on cross-modal retrieval, which is the task of retrieving information from different forms of media, i.e. searching using not just text, but also images, sound, etc. In this project we focused specifically on the image-to-text retrieval problem which can enable an image-based document search. In this domain we would like to use an image as our query and retrieve text documents with a high relevance. The intended use case is to allow technicians to search for detailed information about the parts of a machine which are getting serviced. This can simplify the process of searching for information instead of spending time to find obscure names and details to describe these parts.

The focus of our project aims to leverage state-of-the-art image-to-text retrieval methods and deploy it to a production-ready system which can scale and evolve as demand increases. This means that our focus was not only on retrieval quality but also on practical considerations. Furthermore, our system enables a retrieval process to be done in real time which in our case implies fast response time of the retrieval.

The following list describes the necessary requirements for our end-to-end system:

- Implement a service for retrieving text documents based on images using state-of-the-art deep learning models for multi-modal embeddings.
- Models can be replaced and extended.
- Develop a standardized workflow for evaluating and deploying models to production.
- Load data and train models in a consistent workflow.
- Scalable database / search strategy is used to retrieve documents based on the vector embeddings.
- Continuous Integration (CI)/Continuous Deployment (CD) Pipelines are available for all components to run testing and deployment.
- Services are deployed in cloud environment.

For developing our application, we followed the agile framework SCRUM [26] which makes inherent complexity of new software development manageable by starting with a Minimum Viable Product (MVP) and improving it incrementally. Therefore, our first step was to explore academic research for evaluating promising approaches for the image-to-text retrieval task and design the initial architecture of our end-to-end system. We continued

working in two-week sprints where we developed further features and refined our application.

To present our project, we first provide an overview about related work in Section 2. In Section 3 and 4 we introduce the methodology of the project by first explaining the image-to-text retrieval methods and second, explaining the system architecture of the developed application. We present the results of our end-to-end application in Section 5 before we conclude with a summary in Section 6 and provide an outlook for further enhancements of the application in Section 7.

2 Related Work

In this section, we first give an overview about deep learning methods for image-text-retrieval and then focus on three promising approaches for our use case.

2.1 Review of Image-Text-Retrieval

Image-to-text retrieval has been extensively studied by the researchers, therefore there are many different approaches for solving this problem. Following the categorization introduced by [4], one can classify the methods based on the way the final cross-modal embeddings are obtained:

- pairwise learning,
- interaction learning,
- attributes learning.

Pairwise learning methods aim at embedding texts and images in a joint vector space. The goal is to obtain semantically similar vector embeddings for images and texts. These approaches consist commonly of two branches, one for encoding of the images and another one for encoding of texts. The outcomes of both branches are vectors and their similarity is calculated. The measure of their similarity provides then the supervisory signal that is used to improve the encoders in the training phase. Prominent examples of such methods are Visual-Semantic Embeddings with Hard Negatives (VSE++) and DCMP [8, 36].

Interaction methods like Object-Semantics Aligned Pre-training for Vision-Language Tasks (OSCAR) or CAMP [20, 32] tend to achieve the highest accuracy among all types of methods. This group of approaches assumes that images and texts are not encoded separately. Information flows between both branches before obtaining the embeddings in the joint vector space, which makes it possible to learn correspondences between particular parts of images and texts. However, such interactions make these methods prohibitively expensive in terms of computation.

Attributes learning methods such as Visual Semantic Reasoning Network (VSRN) and ACMM [19, 13] attempt to obtain high-level features from both texts and images and then compute correlation between them. The difference to interaction methods is that there is no flow of information between text and image branches when creating their vector representations. The correlation between features is calculated after the feature extraction, which leads however to increased inference time.

In the next sections we provide a broader introduction to the methods that we decided to focus on for our image-to-text retrieval system. As stated in Section 1, our system should provide fast retrieval, that is why, the accuracy scores achieved by the methods could not be our only selection criterion. Hence, we will introduce VSE++ and VSRN, which we find to be suitable approaches for fulfilling the pre-defined goals. We provide also a comparison of them with OSCAR, which outperformed all other approaches in terms of accuracy but is not suitable for an efficient image-to-text retrieval system due to its computational complexity.

2.2 VSE++: Improving Visual-Semantic Embeddings with Hard Negatives

VSE++ embeds text and images in a joint vector space. The image i is passed through an image encoder $\phi(i, \Theta_\phi)$, whereas a caption c through a text encoder $\Psi(c, \Theta_\psi)$ to obtain their vector embeddings. Θ_ϕ and Θ_ψ express the model parameters of image and text encoders respectively. These embeddings are then mapped into the joint vector space:

$$f(i, W_f, \Theta_\phi) = W_f^T \phi(i, \theta_\phi) \tag{1}$$

$$g(c, W_f, \Theta_\psi) = W_f^T \psi(i, \theta_\psi) \tag{2}$$

Similarity in the joint vector space denoted as $s(i, c)$ is measured by cosine similarity.

The model is trained using a loss function focused on hard negatives. Hence, for an image text pair (i, c) one wants to maximize its dissimilarity with the most similar incorrect image and caption in the batch. Most similiar incorrect image can be denoted as: $i' = \text{argmax}_{j \neq i} s(j, c)$ and most similar incorrect caption as $c' = \text{argmax}_{d \neq c} s(i, d)$.
Thus, the loss is defined as:

$$L(i, c) = \max_{c'}[\alpha + s(i, c') - s(i, c)]_+ + \max_{i'}[\alpha + s(i', c) - s(i, c)]_+ \tag{3}$$

where α denotes a margin, which is a hyper-parameter of the model [8].

The authors used in the experiments VGG19 as an image encoder and Gated Recurrent Unit (GRU) as a text encoder [27, 6].

2.3 VSRN: Visual Semantic Reasoning for Image-Text Matching

In past work, the image embeddings had no semantic features that represent relationships between objects or salient regions in the image. However, when humans perceive an image, we not only see the individual objects but also recognize the relationships between them. VSRN [19] tries to build a system similar to human vision and reasoning by introducing an image representation that captures the relationships between objects.

Inspired by the success of bottom-up attention mechanism in image captioning [1], recent image-text retrieval models make use of object detection models such as Faster R-CNN [25]. One of previous works, SCAN [18], uses a Faster R-CNN model to extract image region features and then applies attention to the regions and text tokens to generate the embeddings. It achieved remarkable performance on image-text retrieval tasks. VSRN introduced a more advanced architecture by including a Graph Convolutional Network (GCN) to capture semantic features of images and outperformed SCAN.

VSRN is built on top of VSE++, and therefore they are similar in the structure. It also maps images and text into a joint embedding space of dimension D and computes cosine similarity between them. However, VSRN has a unique image encoder architecture, which consists of GCN and bottom up attention using Faster R-CNN.

Bottom-Up Attention The goal of bottom-up attention is to generate a set of image region features $V = \{v_1, ..., v_k\}, v_i \in \mathbb{R}^D$ using regional features $F = \{f_1, \ldots, f_k\}, f_i \in \mathbb{R}^{2048}$ extracted by a Faster R-CNN. Each feature vector v_i represents an object or salient region in this image. The amount of regions to represent an image is restricted to 36 selected by the highest detection confidence scores. To obtain v_i, we first extract regions features $f_i \in \mathbb{R}^{2048}$ using the Faster R-CNN and forward it through a fully-connected layer with the parameters W_f and b_f to transform them to a D-dimensional vector:

$$v_i = W_f f_i + b_f \tag{4}$$

Image Encoder with GCN and GRU Using the region features, a fully-connected graph $G_r = (V, E)$ with weighted edges is constructed where V is the set of region features obtained by bottom-up attention and the adjacency matrix E contains the affinity scores for each edge connecting two regions. An edge with high affinity score indicates a high correlation between the two regions. The pairwise affinity of two regions v_i, v_j is computed by

$$\begin{aligned} (v_i, v_j) &= \varphi(v_i)^T \phi(v_j) \\ \varphi(v_i) &= W_\varphi v_i \\ \phi(v_j) &= W_\phi v_j \end{aligned} \tag{5}$$

where W_φ and W_ϕ are learnable weight matrices. GCN with residual connections is applied to the graph G_r to generate a set $V^* = \{v_1^*, ..., v_k^*\}, v_i^* \in \mathbb{R}^D$ containing region features with relational properties. Mathematically, this can be expressed as

$$V^* = W_r(EVW_g) + V \tag{6}$$

where $W_g \in \mathbb{R}^{D \times D}$ is the weight matrix of the GCN layer, W_r is the weight matrix of residual structure and $E \in \mathbb{R}^{k \times k}$ is the row-wise normalized affinity matrix.
To perform reasoning on these region features, GRU takes the region features in V^* and updates the hidden representation which describes the whole image. The final embedding I of the image is the last memory cell.

Text Encoder and Loss Functions VSRN uses the same GRU text encoder as VSE++. To train the model, a loss function consisting of two objective functions L_M and L_G is used.

$$L = L_M + L_G \tag{7}$$

The matching loss L_M also inherits the loss of VSE++, a hinge-based triplet ranking loss with emphasis on hard negatives. Additionally there is a caption generation loss L_G to optimize the reasoning capabilities of the GCN. Sentences similar to ground-truth captions can be generated by applying a sequence-to-sequence model with attention mechanism [31] to GCN image embeddings. The caption generation loss is defined as the negative log-likelihood of the predicted sentence:

$$L_G = -\sum_{t=1}^{l} \log p(y_t | y_{t-1}, V^*; \theta) \tag{8}$$

where l is the length of output word sequence $Y = (y_1, ..., y_l)$ and θ is the parameter of the sequence-to-sequence model.

2.4 OSCAR: Object-Semantics Aligned Pre-training for Vision-Language Tasks

OSCAR [20] is the state-of-the-art vision-language model. It can be widely used in many vision-language tasks such as image-text retrieval, visual question answering or image captioning. For image-to-text retrieval task, OSCAR is first pre-trained on image-text pairs to generate image and text embeddings and performs a binary classification to find the most similar image-text pair.

Pre-training Input of OSCAR is a triple (W, Q, V) where W is the sequence of word embeddings of the text, Q is the word embedding sequence of the object tags detected from the image, and V is the set of image region features. Q and V are generated using Faster R-CNN. Each region feature $v \in V$ is a position-sensitive region feature vector. It is generated by concatenating region feature $v' \in \mathbb{R}^{2048}$ and region position $z \in \mathbb{R}^4$ or $\mathbb{R}^6$ extracted from an image by Faster R-CNN. A set of object tags with high precision is also detected by Faster R-CNN. Q is the sequence of word embeddings of these object tags. The pre-training objective of OSCAR consists of two loss functions, masked token loss and constrastive loss. The masked token loss is defined as the negative log-likelihood:

$$L_{\text{MTL}} = -\mathbb{E}_{(V,H)\sim D} \log p(h_i | H_{\setminus i}, V) \tag{9}$$

where $H \triangleq [W, Q]$ is the discrete token sequence. At each iteration, each input token in H is masked with probability 15% and replaced by a special token `[MASK]`. The masked tokens should be predicted by their surrounding tokens $H_{\setminus i}$ and all image features V.

The constrastive loss is defined as:

$$L_{\text{C}} = -\mathbb{E}_{(H',W)\sim D} \log p(y | f(H', W)) \tag{10}$$

where $H' \triangleq [Q, V]$. 50% of tokens in Q are replaced by random tokens from the dataset. The loss L_C aims at punishing the examples that contained the replaced tokens, but were predicted to be correct.

The full pre-training objective is defined as the sum of masked token loss and contrastive loss:

$$L_{\text{Pre-training}} = L_{\text{MTL}} + L_{\text{C}} \tag{11}$$

Image-Text Retrieval To solve the actual image-text retrieval task OSCAR casts the retrieval into a binary classification problem for each image-text pair where a higher classification score indicates semantic similarity between image and text. Therefore, the model is also optimized using a binary classification loss. During test time, each image-text pair is forwarded through the network to obtain classification scores. By selecting the top K image text pairs with highest classification scores the results are retrieved.

The authors of OSCAR argue that using tags of objects from images and learning correspondence between image regions and words of captions with multi-head attention are crucial for achieving the state-of-the-art results on benchmarks. However, due to the

complexity of pre-training procedure as well as foreseen long inference time because of computational complexity of this approach, we find it not suitable for an efficient image-to-text retrieval system.

3 Model Selection

In the following section we will introduce the methodology to develop the image-to-text retrieval component of our service. Therefore, we will first introduce the datasets which serve as our retrieval corpus. Due to a lack of real-world data we use publicly available datasets which are used in academic research. Second, we explain the evaluation metrics used to compare different approaches. Finally, we will introduce a baseline followed by the actual image-to-text retrieval methods we implemented.

3.1 Datasets

To evaluate our implementations and benchmark additional experiments we chose to use Microsoft Common Objects in Context (COCO) [21] and Flickr30k [35]. These datasets are the de facto standard for evaluating research in this area. In general the images contained in both datasets can be characterised as diverse, with objects in context, making them challenging for computer vision models to understand. They also include short one sentence descriptions of what the picture contains. The captions were generated by following the same guidelines as originally outlined by [12] using Amazon's Mechanical Turk (AMT). These instructions included focusing on visible objects in the image and describing the scene from an objective third person perspective. Captions were also checked for basic spelling correctness and relevance to the reference image.

3.1.1 Flickr30k

The Flickr30k dataset was originally published with the intent of spurring more advanced research in the domain of natural language processing [35]. It consists of 31,783 images collected from Flickr each annotated with five captions. The dataset sets aside 1,000 images and related captions for validation and testing respectively. The original purpose of such data was to measure the linguistic similarity between the five related captions written by different annotators. This means that in its original inception, the dataset was not explicitly focused on the actual content of the images and the authors didn't balance the images for different classes of objects leading to a bias towards images of humans and animals. Despite the shortcomings in the image data, because of it's high prevalence in the research we deemed it worthwhile of using in our experiments.

3.1.2 COCO

In contrast, the COCO dataset was conceived with the explicit purpose of advancing the state-of-the-art object recognition and scene understanding [21, 5]. While previous computer vision datasets included pictures of single objects in "iconic" poses, the purpose of COCO was to collect more complex images where objects are contextualized, meaning objects can be found in scenes and in all manner of poses. Exploiting lessons learned from preexisting datasets like ImageNet [7] the authors also carefully considered things like:

- How often do categories of objects appear in the dataset.
- The number of copies of an object in a single image (i.e. the number of people in a photo).

- The size of objects varies so that computer vision models must consider additional items, aside from the largest object in the foreground.

In total COCO contains 123,287 images, from which we randomly select 113,287 for training and 5,000 for validation as well as testing [15]. Similar to Flickr30k each image is annotated with five captions.

3.2 Evaluation Metrics

We have chosen the following metrics to evaluate our models. For most of our comparisons we use recall and precision as they are commonly used to evaluate information retrieval tasks. These metrics allow us to evaluate if our model is able to retrieve the relevant text given that the text exists in our database.

In addition, we are evaluating our retrieval system using Bilingual Evaluation Understudy (BLEU), a metric to measure similarity of texts. In this way, we introduce a continuous text relevance instead of a binary relevance and measure how the retrieval is performing on unseen data. This simulates the situation where we want to retrieve the most appropriate caption for a new image from a fixed corpus containing only imperfect captions. Additionally, it allows us to compare our baseline approach which we introduce in Section 3.3 with the image-to-text models.

3.2.1 Recall and Precision

In order to evaluate our retrieval performance we use the ranking metrics recall and precision at position K (R@K and P@K respectively). Generally, R@K measures how often the correct response appears in the top K results whereas P@K measures how many of the top K results are relevant.

To calculate R@K for a given image query, we sort the results of an image query by their cosine similarity, and then considering the resulting top K captions we indicate in a binary fashion whether or not one of the relevant captions is present. While at the instance level this is binary, averaged over the entire test set it provides an intuitive measure of retrieval performance. Similarly P@K is calculated by considering what percentage of the top K ranked results from an image query are relevant. P@K is especially useful if the underlying dataset contains multiple relevant documents for a given input image. In most of the literature K is set to 1, 5, and 10. However, for our use case we expect a user will want to look through fewer than ten documents and chose to measure performance at K set to 1, 3, 5.

3.2.2 Bilingual Evaluation Understudy

In addition to precision and recall, we also utilize BLEU [23] to provide a multi-level ranking between the captions. Originally devised as an automatic means of evaluating machine translations, BLEU considers a set of reference sentences and a candidate translation and scores it based on n-gram precision. This means in the uni-gram case for a reference sentence `"the cat on the mat"` and candidate `"the fat cat"`, the candidate

would score $\frac{2}{5} = 0.4$. While there are many possible ways to configure the BLEU score, in particular defining how many n-grams one wants to consider, we use the default Natural Language Toolkit (NLTK) implementation which considers up to quad-grams [3]. We do not further experiment with other configurations because we are not focused on translation and the BLEU score serves simply as a metric for evaluating retrieval quality.

3.3 Baseline

Since our data contains explicit image-caption pairs, a naive approach to achieve image-to-text retrieval is to first apply image-to-image retrieval and then, for the best matching image result, return its corresponding captions from the dataset.
Using this approach we avoid the task of learning a mapping into a joint embedding space and instead assume that an image encoder can create embeddings which will help us retrieve similar images with similar captions. While this is simpler in terms of modeling, it is also more restricted because our model cannot use additional semantic information that is provided by the captions. Due to this limitation, we anticipate that the retrieval quality will suffer.

To evaluate this baseline we use our test set as query images, the training data as a pool of candidates and then we compared if image-to-image search yielded more similar captions than the primary image-to-text retrieval process. Since P@K and R@K are not well defined for the task of image-to-image retrieval, we adopt the BLEU metric to compare retrieval methods. Concretely, for a given image and its set of captions, (i_q, C_q) we generate an embedding for the image and then search for the image embedding with the highest cosine similarity from our set of candidate images and return the resulting image and its captions (i_r, C_r). In order to generate these embeddings a pretrained ResNet152 [11] with the final classification layer removed is used to generate image embeddings of dimension 1024. This model was chosen in order to maintain comparability with our VSE++ implementation which we will outline next.

3.4 Implementation

In the following section we will describe the final implementations of the image-to-text retrieval models. We chose VSE++ and VSRN because, despite lower accuracy than OSCAR, it provides faster inference time as mentioned in Section 2.1.

3.4.1 VSE++

The dimensionality of the joint embedding space in which VSE++ aims to embed images and text is set to 1024 in our experiments. We use a Resnet152 for encoding the images and a GRU for encoding text. Before feeding the images in the encoder, the images are resized to the shape 224x224 and for training additionally centered and randomly cropped.

In order to preprocess our captions a vocabulary was determined using available training data. The vocabulary contains all tokens that occur more often than a defined threshold

in all captions. On the combined flickr30k and COCO dataset this results in a vocabulary with 16461 words.
The captions are tokenized using NLTK [3], one-hot encoded respecting our vocabulary and then passed through a embedding layer to obtain word embeddings of dimension 300 which serves as the input to the text encoder.

3.4.2 VSRN

As described in Section 2.3 VSRN uses semantic features in the images to generate visual-semantic embeddings. The original paper relied on pre-computed image regions from a Faster R-CNN model as the semantic features [25]. Since we, however, need to build an end-to-end system which can generate embeddings for new, unseen images, we need to integrate Faster R-CNN into our model. Because training a Faster R-CNN is computationally expensive and resource consuming we use Detectron2 for that task - an open-source framework which implements state-of-the-art detection algorithms and region extractors [34].

Due to the fact that VSRN is built on top of VSE++ and to ensure comparability many configurations remain identical. We set the dimension of the joint embedding space again to 1024 and applied the same transformations to the text including tokenization, one-hot encoding using pre-defined vocabulary and learnable embedding layer resulting in word embeddings of dimension 300.

4 Service Architecture

Having introduced the deep learning component of our system we will now describe the overall service architecture required to build our image-based document search. In order to manage the complexity of this system while implementing the project goals as stated above in Section 1 we made use of agile software deployment methodologies, DevOps frameworks and careful user experience design.

Our application is decomposed into a set of manageable services which in our case are faster to develop, easier to understand and updated independently. This architecture design is also known as microservice architecture.
Figure 1 gives an overview of our system architecture by visualizing components of our application, interactions between individual components as well as hosted infrastructure. Most of our services are deployed in a Kubernetes cluster which uses the Google cloud infrastructure but when it comes to model development and embedding generator which are computationally expensive we run the services in the inovex GPU cluster. The frontend is the only service the user interacts with, where the request for text-retrieval for a given image is handled. That request is sent from the frontend to our backend service where we manage the business logic of our application. To enable a separation of concerns, many of our applications rely on the REST [1] standard to communicate.

The following sections will discuss in further detail the technology tool and process for deploying our cloud application.

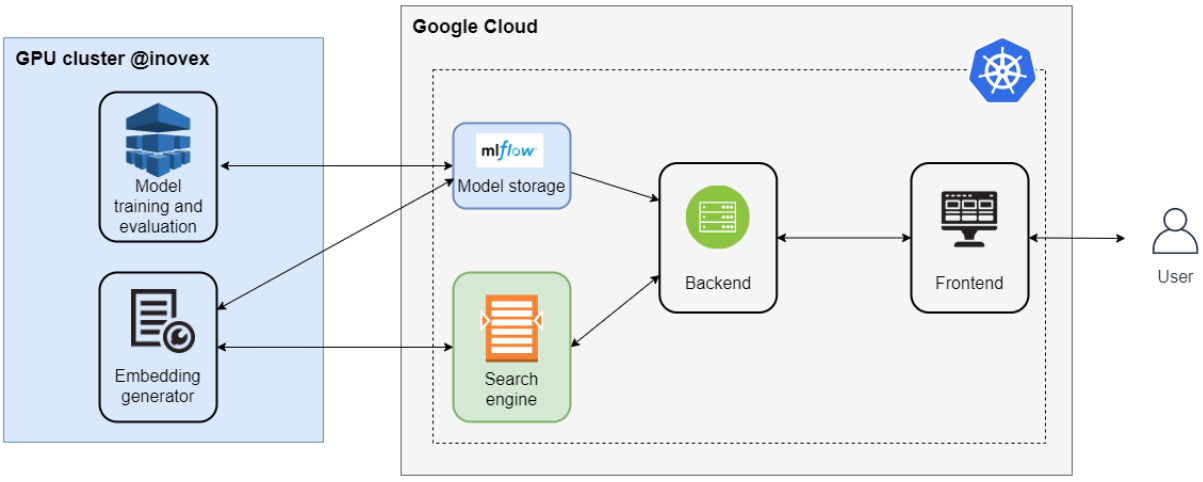

Figure 1: System architecture overview

[1] restfulapi.net

4.1 Cloud Computing

Cloud computing is the on-demand delivery of IT resources via the internet with pay-as-you-go pricing. Instead of owning physical data centers and servers, one can access technology services such as computing power, storage and databases.
The essential characteristics of cloud computing are on-demand self service, broad network access, resource pooling, rapid elasticity and measured service [24]. Cloud service providers such as Google Cloud, Microsoft Azure and Amazon Web Services offer different service models, where the three most common models are Software as a service, Platform as a Service and Infrastructure as a Service (IaaS).

4.1.1 Google Cloud

Google Cloud Platform (GCP)[2] is a group of Google's computing resources, made available as a public cloud offering. There are many services offered by GCP in terms of networking, big data, compute and storage, which make it possible to construct custom cloud-based infrastructure. By leveraging the IaaS capabilities of GCP our application benefits from running technology agnostic frameworks thereby giving us the flexibility to deploy any application server and web framework that we deem fit. Some of the IaaS products that are offered are: Compute Engine, Cloud Storage, Virtual Private Cloud, Kubernetes Engine and Persistent Disk.

For deploying the application we considered Kubernetes Engine and Compute Engine. While the Compute Engine offers virtual machines, storage options and configuration options, we decided to use the Kubernetes Engine. Kubernetes is a step up from Compute Engine and offers many convenient features that suit our application. In the next section we will further illustrate Kubernetes and its benefits.

4.1.2 Kubernetes

Kubernetes[3] is a platform for automating the deployment and scaling of containerized applications across a cluster [30]. By using container-based virtualization for our microservices, we leverage the container clustering solutions to accelerate the development and operations process. [14]. The leading container platform is Docker[4] which we use to package our services into "containers", allowing them to be portable among any system running the Linux operating system.

In Kubernetes we can not run containers directly, instead one or more containers are wrapped into a higher-level structure called a *pod* which is the smallest unit of Kubernetes. Furthermore, in Kubernetes we have *master* and *worker nodes* where each worker node is managed by the master node. A node can contain multiple pods, and the Kubernetes master node automatically handles scheduling the pods across the nodes in the cluster [17].

[2] cloud.google.com
[3] kubernetes.io
[4] docker.com

Achieving scalability, high-availability and fault tolerant capabilities is possible and easier with Kubernetes because some of its features include service discovery, load balancing, horizontal scaling and self-healing.

The most popular and flexible method for load balancing in Kubernetes is Ingress. *Ingress* is a resource for routing external HTTP(S) traffic to internal services [17]. With load balancing we distribute network traffic across multiple servers such that no single server carries too much demand. Moreover, load balancing sends requests only to servers that are online and also offers the flexibility to add or remove server based on the demand.

If a worker node in a Kubernetes cluster fails, the self-healing feature guarantees that it is replaced automatically and this way provides high availability of our service. This means that we don't have to deal with checking health status of every component in our application (see Figure 1).

4.2 Continuous Integration

One of the key challenges in software development is the detection of errors introduced while developing new code, before its deployment. One idea is that each application upgrade should have incremental code changes between deployments so that fewer unintended consequences are introduced. By following this practice we manage to improve the maintenance of our components.

To implement this upgrade we apply the practice of *Continuous Integration(CI)* where every code change is automatically and continuously built and tested, ensuring the introduced changes pass all tests, and the application can be containerized.

We use GitLab as our Git-repository manager where the continuous integration pipeline is also provided. After each code push in our repository the continuous integration pipeline is executed.

4.3 Components

Since our application is based on a microservice architecture, we will explain in this section the basic functionality of each of the components and the tools we used.

4.3.1 Model Training and Evaluation

The selection of a proper model for image-to-text retrieval is very important for the quality of the implemented system. In order to ensure a reliable process of model selection, the models shall be trained on the same data and evaluated using the same metrics. In this way, a fair comparison of the models is performed. Automation of this process has been achieved by implementation of a software module for training and evaluation of the models in a unified way.

This module is responsible for defining the following aspects:

- unified processing of datasets
- implementation of the image-to-text retrieval models
- implementation of the training procedure
- implementation of evaluation metrics
- logging the metrics to the model management service
- storing the trained model in the model management service

One of the project goals mentioned in Section 1 was ensuring that models can be easily extendable and replaceable. For that purpose, the implementation of the models as well as training and evaluation procedures was based on Pytorch Lightning[5]. The evaluation is performed using the metrics described in section 3.2. Results of the experiments and trained model are stored in the model management service introduced in the next section.

4.3.2 Model Management

Developing a deep learning based software poses several new challenges to the software development process. It requires handling the complex and iterative process of the machine learning lifecycle including data collection, model training, verification, deployment and monitoring [2]. In contrast to a software development process this lifecycle is "empirical, combinatorial and data driven" [9] requiring extensive experiments to find the optimal model in the vast space of possibilities. To ensure reproducibility, documentation and responsible handling of compute resources, tracking and organizing the experiments is important. Further, model versioning allows tracking and monitoring which model is in production.

To overcome these challenges and manage the machine learning lifecycle we use the open-source platform MLflow [22]. In particular we rely on the following three features: MLflow Tracking, MLflow Models and Model Registry. The open-source platform also comes with a Python application programming interface (API), user interface as well as built-in integrations for Pytorch which simplifies the integration into our system.

The first component of MLflow we are using is the tracking tool. As the name suggests the component MLflow Tracking is used to track metadata of experiments in a centralized storage. It enables the developer to log metadata about training, dataset and the model itself to organize experiments and compare and reference metadata at any time. For packaging the trained models we use the second component, MLflow Model. For later deployment of the model we serialize the trained models in the environment-independent format torchscript. Torchscript is a unified framework to deploy models for inference in production without requiring source code or pytorch installed. In order to deploy these

[5] github.com/PyTorchLightning/pytorch-lightning

packaged and serialized models the third component Model Registry is used. It is a centralized model storage organized by model categories which facilitates model versioning. This enables simple tagging and deployment of models to to the end-user inference system. We tag the models manually to add a checkpoint before shifting a model into production.

4.3.3 Search Engine

Following the project goals stated in Section 1, our system should be able to perform an efficient and fast retrieval of the textual results based on the similarity of vector embeddings. By relying on two-branch methods with a joint embedding space for image and text it is possible to pre-compute all vector representations of our underlying retrieval database and store them. The retrieval step can be therefore formulated as:

1. Receive an embedding of an image
2. Find the most similar text embedding among all the pre-computed text embeddings

Such formulation allows us to reduce the retrieval time in the production system as the texts do not need to be processed by a text encoder for each user request. This makes it possible to reduce the operation costs of our system and increase its performance. Another benefit is the possibility of leveraging the existing solutions, that are designed to perform an efficient similarity search of dense embeddings. In the next sections of this chapter we will introduce the chosen search engine as well as the process of indexing the vector embeddings and their retrieval.

Elasticsearch

The choice of a search engine for our system was conditioned on a few criteria:

- storage and indexing that supports efficient retrieval of dense vectors
- scalability to large amounts of data
- simple integration with the chosen technology stack

Elasticsearch is a distributed storage that fulfills all the defined criteria. Additionally it is a well-established, mature solution with good documentation and support. In the following we describe how Elasticsearch was incorporated into our system.

Elasticsearch Data Schema

The structure of the data stored in Elasticsearch is presented in Listing 1. Every image-caption pair (i, c) in the dataset has its corresponding entry $(\phi(i), c, \psi(c))$ stored in Elasticsearch, where $\phi(i)$ is the vector embedding of an image i and $\psi(c)$ is a vector embedding of a caption c. Both image and caption embeddings are of dimensionality 1024, which correspond to the dimensionality of the joint vector embedding space described in section 3.4.1.

Listing 1: The data schema stored in Elasticsearch

```
{
    "image_embedding":{
        "type":"dense_vector",
        "dims":1024
    },
    "text":{
        "type":"keyword"
    },
    "text_embedding":{
        "type":"dense_vector",
        "dims":1024
    }
}
```

Embedding generation

The procedure of supplying our instance of Elasticsearch with data consists of the following steps:

1. fetch the trained model from model registry (training procedure and model management described in the sections 4.3.2 and 4.3.1)
2. calculate the vector embeddings of all texts and images in the dataset
3. save the calculated vector embeddings in Elasticsearch

Text retrieval

In order to retrieve a text, Elasticsearch is queried with an embedding of an image. Later, a search process based on cosine similarity is performed with the aim of finding the entry containing the most similar text embedding. As the entry contains also the original text corresponding to the embedding, it can be directly returned as the result of the query.

4.3.4 Backend

One of the core components of our end-to-end system is the backend of our web application which coordinates the retrieval task and serves user requests. Using the Python framework FastAPI[6], our backend server provides an endpoint which receives an image and returns the most similar captions.
The workflow to retrieve the text for the requested image is as follows:

- The backend is always provided with the latest version of our model by using the model registry explained in 4.3.2.
- The endpoint processes the received image and generates an embedding using the loaded model.
- With the generated embedding we find the most similar text embedding by querying in our search engine.
- The response in the end contains ten descriptions which best represent the given image.

4.3.5 Frontend

The frontend service offers a simple and responsive user interface that allows users to upload an image and retrieve texts that best describe the uploaded image. For building the user interface we use Vue.js[7] which is a frontend JavaScript framework. Vue.js has many benefits when it comes to building light-weight web apps. Some of the benefits include:

1. Vue.js itself as a framework has a very small size compared to other JavaScript frameworks.
2. It offers reactive two-way data binding that makes it easier to update related components and track changes to the data.
3. It is easy to use and simple to integrate into other applications built on JavaScript.

[6] fastapi.tiangolo.com
[7] vuejs.org

5 Results

The following section presents the results of our full end-to-end application. Therefore, we first discuss the achieved retrieval performance of the implemented models. Then, we present some scalability benchmarks for our search engine and end this section with the actual price estimates of our application and how the user interface looks.

5.1 Models

5.1.1 Baseline vs. VSE++

First, we focus on the relative performance of our baseline method using ResNet152 with our multi-modal VSE++ model by comparing their BLEU scores. The BLEU scores for the baseline are calculated taking the maximum achievable score with C_q as our reference sentences, and treating the sentences of C_r as candidate captions. As explained in Section 3.3 C_q refers to the ground-truth caption and C_r corresponds to the set of captions from the most similar image. To compare this to image-to-text retrieval we can again use our query image and captions (i_q, C_q) and retrieve the five most similar captions C_{ms}, note here, the captions could be associated with different images, and again calculate the maximum achievable BLEU score. As shown in Table 1 we see that that our VSE++ model achieves a comparatively higher score than the baseline implementation on both datasets. Although these scores are generally low, this only serves as a proxy to measure how well our image to image model can translate an image into one of the predefined captions of our dataset. These results also provide confirmation that our image-text retrieval can outperform our naive image-image retrieval approach.

Model	**Flickr30k**	**COCO**
VSE++	0.182	0.251
Baseline	0.132	0.229

Table 1: BLEU scores comparing VSE++ with Baseline

5.1.2 Cross-Modal Retrieval

In the following, we will present and discuss the results we obtained for image-to-text retrieval using VSE++ and VSRN. In contrast to academic research, we took the additional step of training the models on the combined datasets, Flickr30k and COCO, since the focus of our project is on creating a compelling system for the end user. This enhances retrieval quality by providing model with as much training data as well as using the broadest possible corpus of captions available. To first validate the correctness of our implementations we confirmed that the models performance are inline with the published results.

The models were trained for 25 epochs using Adam optimizer [16] with a learning rate of $2 \cdot 10^{-4}$ and a batch size of 128. The learning rate was reduced every 15 epochs by a factor of 0.1. To avoid vanishing gradients gradient clipping for a norm of 2 was applied. The margin for the contrastive loss was set to 0.2. To decrease the training time loading

Model	R@1	R@3	R@5	P@1	P@3	P@5
VSE++	0.343	0.542	0.634	0.343	0.388	0.244
VSRN with pre-computed features	0.461	0.667	0.759	0.461	0.392	0.339
VSRN with Detectron2	0.315	0.516	0.607	0.315	0.275	0.241

Table 2: Results of image-to-text retrieval models evaluated trained on Flickr30k and COCO. We refer to Recall@K as R@K and Precision@K as P@K.

data was parallelized among 10 workers.

As shown in Table 2, the obtained results are mixed. While VSE++ achieves a relatively strong performance of R@1 0.343 and R@5 of 0.634, the VSRN implementations yield inconsistent results. We can see that while the VSRN with pre-computed regions clearly outperforms VSE++ on every metric (improving the recall by 12%) our end-to-end implementation of VSRN using Detectron2 scores worse than VSE++ on every metric.

Although discouraging, we expect that it must be due to subtle differences in the training of the Faster R-CNN. While Detectron2 was developed for segmentation tasks and trained on COCO, the originally used bottom-up attention [1] was developed for image captioning which is more similar to our intended use case than image segmentation. To achieve the same results and have a model that can properly be used for inference, implementing the bottom-up attention introduced by [1] in Pytorch is required. Due to a lack of time and computational resources we leave this as subject of further research.

5.2 Search Engine

Next we turn our attention to some of the other practical aspects of the project. In particular we chose Elasticsearch as our search engine because we wanted to use a stable system which was performant, simple to integrate and supported search using embeddings. While relatively simple to query and use from the client perspective, managing this system was more challenging than anticipated. In particular, we were surprised to find that in our current system we needed to allocate a large amount of memory to our elasticsearch cluster. As shown in Figure 2, for indexes with relatively few documents, the retrieval step is acceptably fast. The problem comes as scale your index to hundred of thousands or millions of documents that you would like to rank using cosine similarity. In such a situation, it is critical that one considers using compute resources which are memory optimized. This will allow the application to benefit from the scalability and speed of Elasticsearch.

5.3 Cloud Pricing

In Table 3 we present a brief overview of the monthly costs we accrued from running all the services running to support entire application during the month of January. Unsurprisingly, we can see that the major cost components are associated with Google's Compute Engine which are the fixed compute resources we've allocated to the cluster. If we wanted to consider cost saving measures we could certainly think about automatically

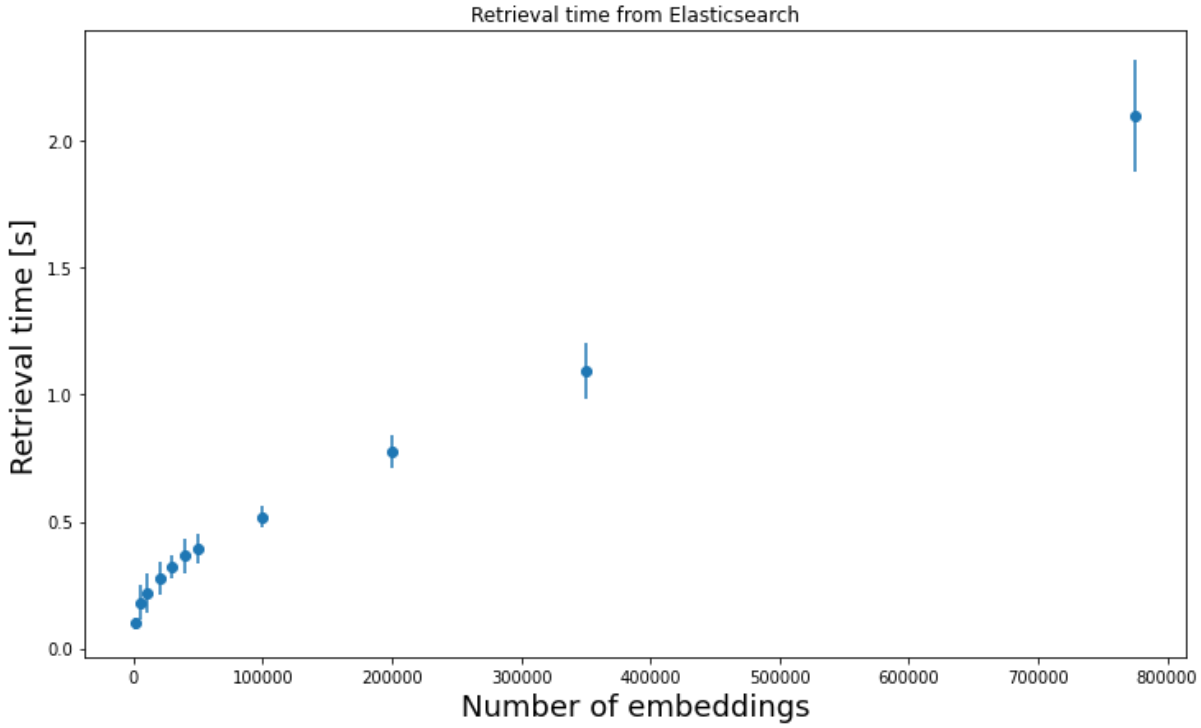

Figure 2: A plot comparing the retrieval time in seconds for a single image embedding query as the size of the index i.e. the number of embeddings increases.

shrinking the cluster during low usage hours. However, cost savings was not one of the core goals of our project, and since prices were in an acceptable range, we left things as they were. That being said, the system in it's current state is a first iteration. In a production environment with high demand we would certainly benefit even more from features like Kubernetes auto-scaling which allow the available resources to both grow and shrink with changes in resource utilization.

5.4 Web Application

Finally, we present a brief overview of our web application in Figure 3. Here the user just needs to select and upload their favorite picture. Upon submission they will be presented with a selection of ten possible captions ranked by similarity as shown on the right side. This example show how effective our retrieval method is, from our database of over 500,000 captions we see that all results match thematically, and within our top five results we have relevant results. For the final use-case where we would like to retrieve long technical documentation instead of brief captions it is likely that the user interface would need to be restructured. As previously mentioned, a simple step would be reducing the number of results returned to avoid overwhelming the user.

Service	Resource Type	Subtotal
Compute Engine	E2 Instance Core	€153.36
Compute Engine	E2 Instance Ram	€82.20
Compute Engine	Storage PD Capacity	€26.47
Compute Engine	Network Load Balancing	€17.82
Compute Engine	Misc. Networking Services	€14.58
Cloud Storage	Download Worldwide Destinations	€18.88
Cloud Storage	Standard Storage	€0.71
Cloud Storage	Regional Standard Class B Operations	€0.09
	Total	**€314.11**

Table 3: A high-level overview of the operational costs for the month of January, note, we manually aggregated a group of "Misc. Networking Services" which on average cost less than 1 euro.

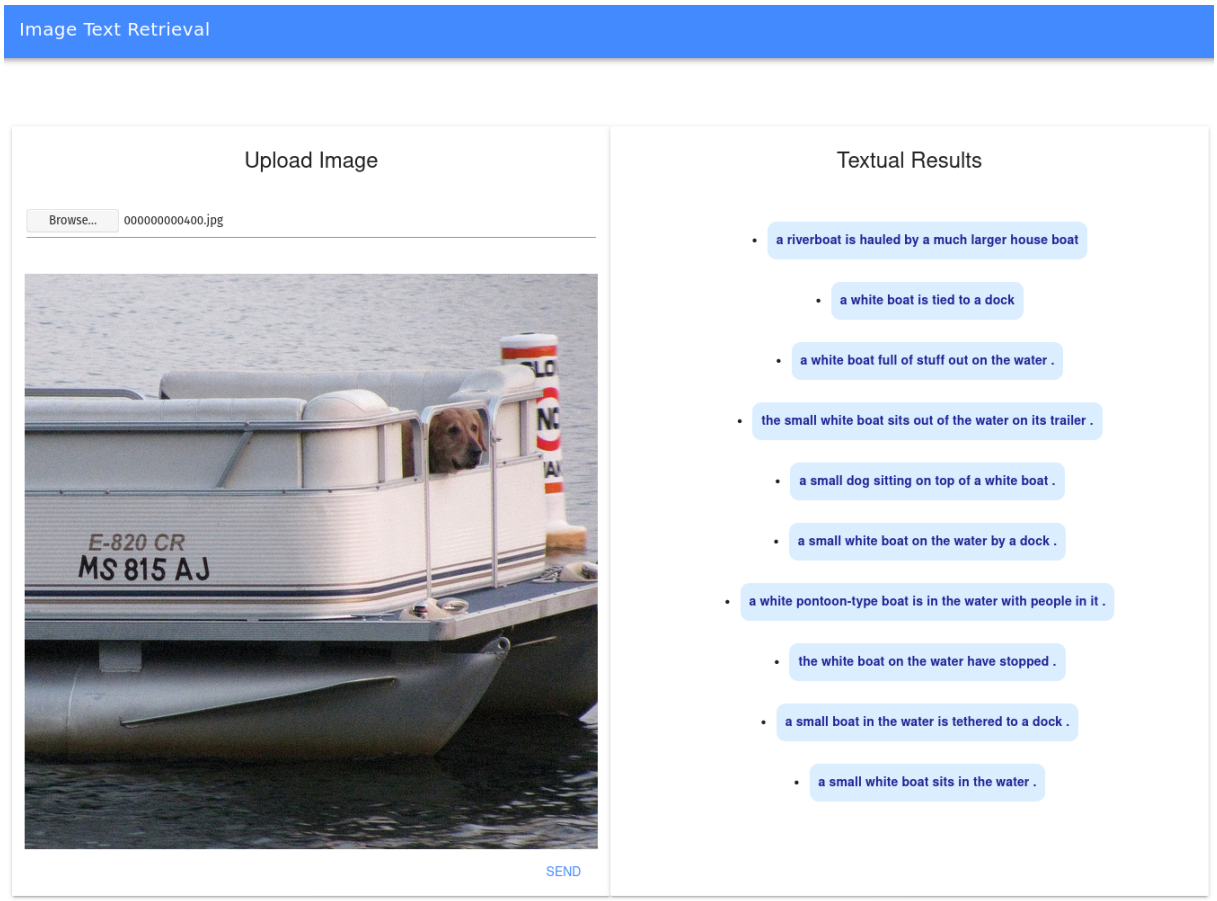

Figure 3: An example screenshot of the user interface returning relevant captions for the uploaded picture.

6 Summary

This project focused on building a scalable, modular image-based document search. Building such a system required that we not only understand and implement state-of-the-art research, but also consider what impacts it would have on the architecture and speed of our system. This goal guided us to prioritize simpler approaches VSE++ and VSRN over state-of-the-art methodologies such as OSCAR.

Furthermore, in contrast to the typical one-off approach of academic code, our deep learning models needed to be trained, evaluated and deployed in a consistent manner in order to manage complexity of the overall system. This allowed us to not only train multiple models in a relatively brief period of time, but also provides a path to maintain and seamlessly replace models as the system develops.

Finally, in order to serve as a framework which can benefit inovex in the future we leveraged modern infrastructure best practices to provide the foundation for a stable system. Through the use of technologies like unit tests, CI pipelines, and infrastructure as code we can be confident that our code and infrastructure are running in a stable and reproducible manner.

Execution of this project also presented many challenges for our team. Fundamentally, the academic topic is challenging because it draws knowledge from natural language processing, computer vision, and information retrieval. Even with the challenges we had training VSRN it still showed significant performance improvements over VSE++ showing the power of attributes learning methods.

In addition, this project lays out a strong roadmap how to build a production-ready image-to-text retrieval system. While the tech stack we employed is complex with many layers of abstraction, we observed that many of the technologies we have adopted are in fact very mature and help manage the complexity of an end-to-end system. Further, as previously mentioned there are also many open topics which could be investigated to improve aspects such as scalability and performance.

7 Outlook

To further enhance the performance of the image-to-text retrieval system, there are many improvements to the modeling which could be done. First, training an end-to-end VSRN which includes a Faster R-CNN inspired by the bottom-up attention mechanism and fine-tuned on the retrieval corpus should yield better performance. Especially for the intended real-world use case this can provide a huge benefit regarding user experience since the retrieval quality increases.

As stated in our report we focused on approaches with a joint visual-semantic embedding space to achieve efficient and fast retrieval. However, interaction methods have shown that removing this constraint and modeling interactions between features of images and text leads to a significant improvement in performance [20]. To circumvent the high inference time of $\mathcal{O}(n)$ with n being the total number of text entities in the retrieval database we propose a two-step retrieval to balance the trade-off between retrieval quality and response time [29]. In the first step responsible for pre-selection the top l similar image-text pairs are retrieved using a pairwise learning method with low inference time. In the second step, we then apply the interaction method which operates on each of the l image-text pairs instead of the whole corpus to retrieve the final result of top K most similar text entities. With $K \ll l \ll n$ the required response time can be significantly reduced compared to relying only on an interaction method. It is subject of further research to verify the effectiveness of this approach and evaluate if it leads to the expected increase in retrieval performance while keeping the inference time low.

Further, we want to propose adding uncertainty quantification to our retrieval model to assess the reliability of our system and to identify out-of-distribution samples to reject queries for which we have no fitting response in the underlying retrieval corpus. Besides improving the reliability uncertainty quantification can be useful to identify weaknesses in the model.

Uncertainty quantification is a novel upcoming research topic and has mostly been studied for regression and classification tasks. For retrieval, however, research is still in its infancy. First approaches cast the retrieval task to a regression or classification task using Monte-Carlo Dropout [28], Deep Bayesian Neural Networks [10] as well as stochastic embeddings to estimate uncertainty [33]. In all three approaches, uncertainty of the mapping into the visual-semantic space is estimated instead of representing the uncertainty in the similarity score of two embeddings. The presented approaches seem to be promising for our application and we would like to encourage further work on that to improve the trustworthiness and reliability of our system.

Bibliography

[1] Peter Anderson et al. "Bottom-up and top-down attention for image captioning and visual question answering". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018, pp. 6077–6086.

[2] Rob Ashmore, Radu Calinescu, and Colin Paterson. "Assuring the machine learning lifecycle: Desiderata, methods, and challenges". In: *arXiv preprint arXiv:1905.04223* (2019).

[3] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.", 2009.

[4] Jianan Chen et al. "Review of Recent Deep Learning Based Methods for Image-Text Retrieval". In: *2020 IEEE Conference on Multimedia Information Processing and Retrieval (MIPR)*. IEEE. 2020, pp. 167–172.

[5] Xinlei Chen et al. "Microsoft COCO Captions: Data Collection and Evaluation Server". In: *CoRR* abs/1504.00325 (2015).

[6] Kyunghyun Cho et al. "Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, Oct. 2014, pp. 1724–1734.

[7] J. Deng et al. "ImageNet: A Large-Scale Hierarchical Image Database". In: *CVPR09*. 2009.

[8] Fartash Faghri et al. "VSE++: Improving Visual-Semantic Embeddings with Hard Negatives". In: *Proceedings of the British Machine Vision Conference (BMVC)*. 2018.

[9] Rolando Garcia et al. "Context: The missing piece in the machine learning lifecycle". In: *KDD CMI Workshop*. Vol. 114. 2018.

[10] Kenta Hama et al. "Exploring Uncertainty Measures for Image-Caption Embedding-and-Retrieval Task". In: *arXiv:1904.08504 [cs, stat]* (Apr. 9, 2019).

[11] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[12] Micah Hodosh, Peter Young, and Julia Hockenmaier. "Framing Image Description as a Ranking Task: Data, Models and Evaluation Metrics". In: *J. Artif. Int. Res.* 47.1 (May 2013), pp. 853–899.

[13] Yan Huang and Liang Wang. "Acmm: Aligned cross-modal memory for few-shot image and sentence matching". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 5774–5783.

[14] A. Kanso, H. Huang, and A. Gherbi. "Can linux containers clustering solutions offer high availability". In: *IEEE Third International Workshop on Container Technologies and Container Clouds*. 2016.

[15] Andrej Karpathy and Li Fei-Fei. "Deep visual-semantic alignments for generating image descriptions". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2015, pp. 3128–3137.

[16] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *arXiv:1412.6980 [cs]* (Jan. 29, 2017).

[17] *Kubernetes*. Kubernetes. URL: `https://kubernetes.io/` (visited on 02/02/2021).

[18] Kuang-Huei Lee et al. "Stacked cross attention for image-text matching". In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 201–216.

[19] Kunpeng Li et al. "Visual Semantic Reasoning for Image-Text Matching". In: Proceedings of the IEEE/CVF International Conference on Computer Vision. 2019, pp. 4654–4662.

[20] Xiujun Li et al. "Oscar: Object-Semantics Aligned Pre-training for vision-language Tasks". In: *European Conference on Computer Vision*. Springer. 2020, pp. 121–137.

[21] Tsung-Yi Lin et al. *Microsoft COCO: Common Objects in Context*. 2015.

[22] *MLflow - A platform for the machine learning lifecycle*. MLflow. URL: `https://mlflow.org/` (visited on 02/01/2021).

[23] Kishore Papineni et al. "Bleu: a Method for Automatic Evaluation of Machine Translation". In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, July 2002, pp. 311–318.

[24] D. Puthal et al. "Cloud Computing Features, Issues, and Challenges: A Big Picture". In: *2015 International Conference on Computational Intelligence and Networks*. 2015, pp. 116–123.

[25] Shaoqing Ren et al. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. 2016.

[26] Ken Schwaber and Jeff Sutherland. "The scrum guide". In: *Scrum Alliance* 21 (2011), p. 19.

[27] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *International Conference on Learning Representations*. 2015.

[28] Ahmed Taha et al. "Exploring Uncertainty in Conditional Multi-Modal Retrieval Systems". In: *arXiv:1901.07702 [cs]* (Jan. 22, 2019).

[29] Stefanie Tellex et al. "Quantitative evaluation of passage retrieval algorithms for question answering". In: *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*. 2003, pp. 41–47.

[30] Leila Abdollahi Vayghan et al. "Kubernetes as an Availability Manager for Microservice Applications". In: *ArXiv* abs/1901.04946 (2019).

[31] Subhashini Venugopalan et al. "Sequence to sequence-video to text". In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 4534–4542.

[32] Zihao Wang et al. “Camp: Cross-modal adaptive message passing for text-image retrieval”. In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 5764–5773.

[33] Frederik Warburg et al. “Bayesian Triplet Loss: Uncertainty Quantification in Image Retrieval”. In: *arXiv:2011.12663 [cs]* (Nov. 25, 2020).

[34] Yuxin Wu et al. *Detectron2*. `https://github.com/facebookresearch/detectron2`. 2019.

[35] Peter Young et al. “From image descriptions to visual denotations: New similarity metrics for semantic inference over event descriptions”. In: *Transactions of the Association for Computational Linguistics* 2 (2014), pp. 67–78.

[36] Ying Zhang and Huchuan Lu. “Deep cross-modal projection learning for image-text matching”. In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 686–701.