Large Scale Data Engineering: Image Interpretation

Abhinav Shankar
abhinav.shankar@student.vu.nl

Corneliu Soficu
corneliu.soficu@gmail.com

Leonard Herold
leonard.herold@gmail.com

ABSTRACT

This work provides a detailed description of the six-week-long practical assignment that was done as part of the Large Scale Data Engineering (LSDE) course at Vrije Universiteit Amsterdam under the guidance of Peter Boncz. The assignment, as chosen by the authors, conducts an image interpretation task using the Yahoo Flickr Creative Commons 100 Million (YFCC100M) dataset, wherein the data was analyzed, classified, and visualized into a search engine that can be used to retrieve images using keyword search. Throughout this work, an approximate 90 million images with a size of over 5 TB were downloaded and classified using a Convolutional Neural Network (CNN), producing a search index of over 22.3 GB including 309,726,315 label associations for over 89 million images. Overall this work outlines how the images from the YFCC100M can be efficiently downloaded and classified using Databricks cloud-based Spark environment as well as how to implement a Vue.js-based search engine on top of an inverted list-based search structure.

1. INTRODUCTION

Transforming our human's visual reception into some form of representation is a deeply-rooted part of our culture, and artworks or pictures are arguably one of such forms. One of the oldest of these kinds were discovered in Europe and date back around 40,000 years [3]. Over this entire period, humankind has used visuals to reproduce the real world reception in the form of paintings, or other more recent forms such as pictures. Nowadays, sharing pictures has become a global phenomenon. Technical revolutions that are ubiquitous these days, such as digital cameras or smartphones, enable millions of people to capture any moment of their life at any time. As people also have become users of internet-based social media platforms such as Instagram, WhatsApp, or Pinterest, it has become natural to share such moments on these platforms. As a result, companies store a large corpus of data on behalf of their users. Depending on the kind of information stored, companies have a very varying understanding of what that data is actually representing.

This uncertainty poses a huge challenge for companies, as they have to comply with laws and regulations, preventing or resolving users' violations caused by copyright-infringements, or fight inappropriate content such as hate speech or illegal images. However, companies have already taken measurements against this. Facebook, for example, operates large content moderation centers across the globe, Tumblr scans uploaded pictures for pornographic images since 2018 and YouTube uses their ContentID system to detect copyright-protected content in users' videos [6, 10, 5]. Apart from that, knowing and extracting information about the user's uploaded content simply helps companies providing sophisticated searching capabilities within their applications as they can index information and make it searchable.

Concerning images, improvements in their classification technologies have provided companies with another method to determine what images captured. In the past, companies mostly relied on user-provided tags, metadata extraction (i.e., Exif), and manual content moderation. Due to the fact that in recent years machine learning and AI-based approaches have become much more feasible for large amounts of data, as a result of increasing computational power that is available on-demand from hyperscale cloud providers, open-sourced software development kits and publicly available pre-trained Artifical Neural Networks (ANNs). Using ANNs to classify images to text has been proven to be a viable method identifying specific attributes within an image [4, 11].

This work, being part of the LSDE course 2019 at Vrjie Universiteit Amsterdam, aims at practically applying state-of-the-art image-classification methodologies on a large amount of image data and evaluate the applicability of these methods. Given the symbiosis of research and practical focus, the YFCC100M dataset was chosen as a basis for this work. This dataset, being the most recent standard of present multimedia research [11], contains 100 million media objects shared on the Flickr media-sharing platform. Given the aim and the underlying data, this work's research question is formulated as follows:

How can all the images currently available on Flickr be downloaded and classified effectively, that is, the computational time required of each task, by using image-to-text and scene classification and made available via a fundamental search engine?

Regarding the limitations and duration of the course's assignment, this work focuses only on the still available data that is downloadable on Flickr as of October 2019. Further, to fulfill the hosting requirements, all processing will be done throughout the duration of the course; thus final results that can be searched are static only.

The remainder of this paper is as follows. First, the related literature will be presented. Further is shown on which scientific literature this project is built on. Subsequently, the project itself will be described based on the individual components and overall process. A brief overview of the project is given, including project organization and then general ar-

chitecture. This is followed by presenting insights and the underlying statistics obtained during the initial analysis of the YFCC100M. After that, each component of the whole architecture is explained in detail. Then the project timeline is discussed, as well as the project's total budget. Finally, a conclusion is made regarding the entire project.

2. RELATED WORK

In this chapter, related work from academia and practice is discussed, more specifically, the two research papers that are closely related to our project's objective. These two scientific papers did offer some substantial insights for our project and its implementation, as each paper did shed light on different aspects that were relevant to the project.

2.1 YFCC100M: The New Data in Multimedia Research

The first paper, "YFCC100M: The New Data in Multimedia Research" from Thomee et al. [11], presents the Yahoo Flickr Creative Commons 100 Million Dataset (YFCC100M), which arguably acts as the groundwork for this project. In fact, the published dataset is the basis for the entire project described in this work.

The authors claim that previously, no sufficient datasets for multimedia research was available for academic and commercial use-cases. They stress the relevance for publicly available datasets, that are easy to share and do not pose any licensing issues, as other datasets do. Furthermore, they criticize that previously published datasets were only collected to support a related paper. Therefore, they argue that their work "meets the call for scale, openness, and diversity in academic datasets" [11, p. 1] and aim at publishing a new reference dataset for multimedia research. The paper presents a thorough overview of the data included, touches on the strengths and limitations of the dataset and ends on drawing future research directions.

The data collected compromises exactly 100 million rows of images and videos that have been published under a CC commercial or non-commercial license. Regarding its distribution the paper states that there are 99.2 million images and only 0.8 million videos. While the media data on Flickr itself is a total 16.5 TB large, the dataset only contains the data required to locate the images and videos on the Flickr service enriched with some additional metadata, such as location, upload time and the camera model that took it (see Table 6 in the Appendix for more information on which data is included).

Given the fact, that this work aims at classifying only the images referenced by the dataset, most of the metadata was not relevant to this work's task. Therefore, it is not further discussed in detail, however, it should be mentioned that the authors discuss five types of metadata contained in the dataset, namely tags provided by users or some type of automatic labelling, the timespan that goes up to 2014, location, camera-related metadata and licences.

During the project initiation we also realized that the authors already have analyzed the images and videos contained in the dataset using a self-trained deep CNN, more specifically ImageNet [9]. They included in the paper the top 25 of 1570 classed detected in the dataset, while they also made the whole dataset of autotags accessible via their website [1]. In fact, this provided our work with valuable directions in how to proceed. They also highlight that in the future more data might be release, providing additional types of annotations.

Regarding the strengths and limitations of the dataset the authors, although they do not want to replace existing datasets, encourage other researchers to specifically built onto their work, due to the fact that the dataset was designed for exactly this purpose. Additionally, they argue that the data ensures some extent of research equality strengthening important aspects such as reproducibility, verifiability and extensibility of research experiments. Apart from that they also stress the size of their dataset, the multi modality, which means that it is including images and videos, existing metadata and suitable licensing for reusing the data present in the dataset. They also point out that their dataset lacks proper annotations. Regardless of that, they point out, that annotating images is seen as a good challenge for further research activities.

In the remainder of the paper, the authors give some recommendations for using the dataset, which this work followed in the initial data analysis by mentioning our sampling technique. They also address future directions of research. These include how to use the YFCC100M to further advance AI and computer vision, the opportunity to analyzes location and time information included in the dataset as well as using the dataset for studying the digital culture and preservation of media. In fact, the paper discussed in the following section built on the original dataset and allows researchers to explore the dataset without any other additional tools.

2.2 Real-time Analysis and Visualization of the YFCC100m Dataset

The paper published by researchers from the University of Kaiserslautern [7] was introduced as a documentation to a tool that can be used by the research community to utilize the full potential of the YFCC100M data. By providing the associated tool[1] the researchers aim at allowing the users to search for small subset within the entire dataset with the possibility to browse images and videos of the dataset. The paper mentions the target was to have a similar contribution to the industry as its predecessors such as ImageNet and MS Coco, by presenting a means of accessibility for the YFCC100M dataset [7]. The paper talks about the functionality of Flickr upload mechanism, where it allows users to upload an image without a title, an optional free-text description and giving the users the choice to select from an arbitrary number of tags or in some cases no tags at all. The following led to concluding that such data should be considered incomplete, and thus could be seen to be inhomogeneously distributed which therein has a major implications on the dataset. The paper then discussed how there was an average of 7.06 tags per individual item, which led to a total of 386,435,393 tags. According to the authors, removing duplicate tags resulted in 7,940,039 distinct tags. This was then further reduced by grouping tags into different categories such as app-generated, descriptions, etc.

The core of the paper was a description on the search interface that can be used by scholars as an initial analysis to find a subset for research. This was done by enabling easy and quick access to a given query, by filtering and exploring the entire dataset and retrieved by a keyword search that is said to be pretty straightforward. The paper mentions that

[1]Reachable via www.yfcc100m.org

for any given user query, the search engine would retrieve a set of images and videos in the form of thumbnails, where each given item redirects to Flickr page containing the image or the video. In addition to retrieval, the system also provides statistics which are generated dynamically based on the given subset that is retrieved. The statistics that the engine provides include features such as a tag cloud which is a visualization of the 100 most commonly used tags in the given subset, which are ranked by count, i.e., the most used to the least used. It also had a feature where the search could be kick-started by just clicking on a particular tag and retrieving a subset for that given tag as a keyword. To add multiple tags to search, the system provided a "+" which can be used to append another keyword which added an exploitative component to the browser. Our project adapted some of these ideas in respect to the limits given, for example, that the search index is required to be static.

The paper mentions briefly the technical side of how the whole search engine works and what it is built on. Since the dataset YFCC100M is pretty large, allowing high accessibility and scalability with respect to multiple queries or multiple users querying into the browser, they decided to use the Google Compute Engine. Google App Engine Environment was used to run the frontend, which in the papers description is a framework which allows scalable web applications to run smoothly on the Google Infrastructure. This is done by spreading the application across multiple servers and instances are spawned automatically based on the demand that is generated to be able to scale up smoothly with increasing application load. The backend system which was running the keyword search, which included retrieval, storage and aggregation of the search results feeding into the browser was built with the help of Google BigQuery. The paper emphasizes the main importance of choosing BigQuery, as it has a database-like query language and database schemas which allow the system to process very large quantities of data including duplicate and nested fields in a very distributed manner. The statistics that are also generated by the frontend were done on the client side using JavaScript. These were mostly visualizations in terms of charts.

The paper however also states the minor drawback of using BigQuery. The paper mentions although BigQuery is easily accessed similar to any SQL-like languages, querying on single static datasets with grouping, sorting and selection is mostly a very high performant, where joins especially take a lot of time. This is stated in the paper as one of the reasons why the generation of statistics takes a little longer than ideal compared to the result preview. Overall the paper does lay claim that the performance is high enough to be able to view results in a matter of seconds, making the whole experience of browsing very fluid.

Overall, the paper provided valuable insight for the subsequent project execution, and provided a vision to our group, how to implement our own solution.

3. DESIGN AND DEVELOPMENT

This chapter describes the overall solution that was used to answer the given research question. First, a general overview on the process steps required during the project as well as the whole architecture, consisting of a data pipeline and web application is given. Then, each individual component is described in consecutive sections. Finally, the project plan and the project budget are discussed in hindsight.

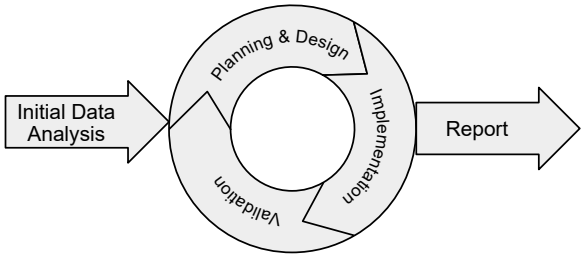


Figure 1: Process Steps of Project

3.1 Overview

During the entire project a simple and lean project process was followed. It can be divided into three different distinctive phases that are depicted in Figure 1. The first step was dedicated to the initial data analysis to obtain a rough understanding of the dataset that was given. This was followed by an iterative development phase that consists of three process steps, namely planning and design, implementation, and validation. The last step's purpose is to communicate the project results and thus consisted of writing this report.

Concerning the iterative development phase's main result, a high-level architecture was designed based on the understandings gained during the prior analysis. This architecture consists of a data processing part and a user friendly search interface. The data processing includes five data processing pipelines displayed in Figure 2, whose output is used to provide the search results and data required for providing additional user experience within the interface. The search interface, depicted in in Figure 5, consists of five different components that together provide the search experience for the user. Each component serves a dedicated purpose and is described in more detail in the search interface section.

3.2 Initial Data Analysis

The initial data analysis, being the first step in the project, was conducted to obtain a deep and thorough understanding of the dataset given. Therefore, it acted as a foundational basis for the subsequent iterative development and from the insights gained the first draft of the architecture was crafted.

The initial analysis included three main aspects, the results of which are displayed in Table 1. First, the overall structure of the dataset and the determination of which data is relevant to the project. Second, what implications can be drawn from the dataset that might affect the overall implementation of the project. Thirdly, and regardless of the fact that the underlying dataset was published in the communications of the ACM suggesting that the dataset should potentially be from high quality, tests for potential inconsistencies were carried out to confirm the suggest high quality.

The dataset itself originating from a scientific paper by Thomee et al. [11] consisting of 100 million rows with 23 columns (see Table 6 in the Appendix for a full description). Each row represents a single media object on the Flickr service. The dataset itself only includes the data that is required to locate the data on the Flickr service. This includes, for example, the Flickr user id of the person who uploaded the media object or data about Flickr's infrastructure indicating where it is stored,i.e., the server id. Apart

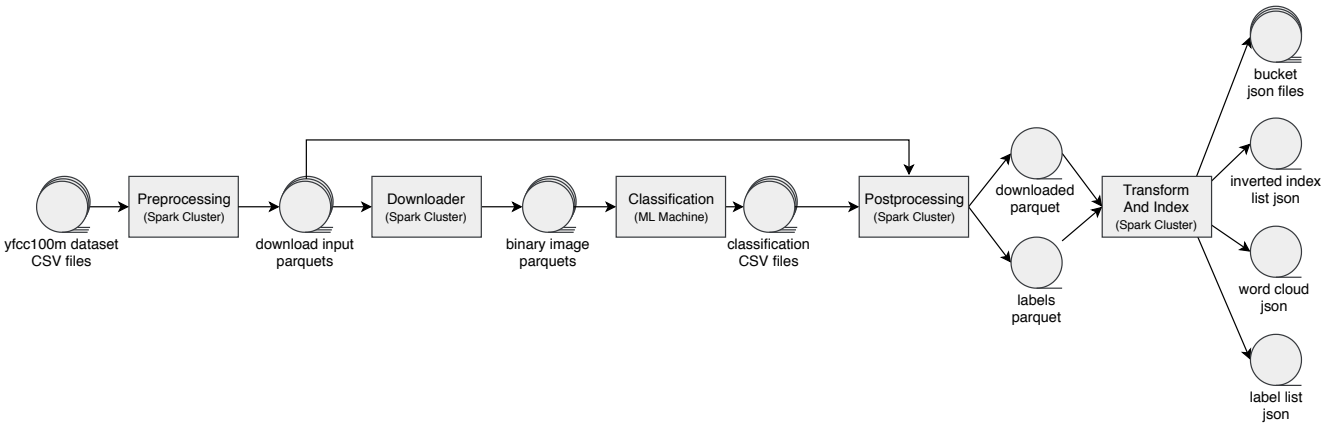


Figure 2: Data Processing Pipeline Overview

Property	Value
General Figures	
Total Dataset Size	100 million rows
Partitions	10 with 10 million rows each
Total File Size	12.8 GB bz2 compressed
Total Number of Columns	23
Random Sample Figures	
Ratio Videos / Images	10010 img, 84 videos
Random Sample Size	10094
Random Sample File Size	4.85 MB uncompressed
Random Sample Figures - Image Download	
HTTP 200	8984
HTTP 404	683
HTTP 410	343
Images Download Volume	539.20 MB
Image File Statistics	
Bytes Mean	60018.35
Std. Dev.	31906.52
Bytes Min	1018
Bytes Max	1907786
Predictions	
Total Download Volume	5.39 TB
Downloadable Images	88,852,698
Total Download Time	6,417 hours on single thread

Table 1: Initial Dataset Analysis

from that some meta data is also present, such as a title of the image, tags which the user or some form of automated system provided, and location information allowing to see where a image was taken (see also [11]. Overall only eight columns were relevant for the project. This includes the data required to download the data and display it later to the user.

In order to draw further conclusions and determine possible implications, the images associated to each row needed further evaluation. However, because of the size of the dataset which due to the 100 million rows makes up a total of 12.8 GB in compressed format (calculated from the files on the provided S3 bucket) and the fact, that the data of the associated images is even bigger (around 13.5 TB according to [11]) sampling the data was deemed a feasible alternative. Therefore, a random sample[2] of the entire dataset was taken and used for further analysis. The sample with a size of 10,094 rows, included 10,010 images which indicates that 99% of the dataset contain images. During downloading the images from Flickr three data points were recorded. First, the time each request took and, second, the HTTP return code for requesting each image, and third, if the request was successful the image's resolution was recorded. Also each image was stored on S3. As shown in Figure 4 a successful download, represented by HTTP return code 200, took in average 0.26 seconds. Unsuccessful requests only took 0.05 seconds for both HTTP 404 and HTTP 410. However, Figure 4 also shows that the request time has a significant variation from the median value of 0.21. Further investigation suggested that this is reasoned in the different file sizes stored on the Flickr services. For example, the minimum and maximum file sizes in bytes are 1,018 and 1,907,786 bytes respectively, while the mean is at 60018.35 bytes. According to our analysis, the variety in file sizes originates mostly from the different resolutions of the images. As shown in Figure 3 a variety of resolutions is present in the data. Further, all images from the sample are at most 500 pixels wide or high. Overall, the variety in resolutions was naturally expected, however, it was also evident that this must have been accounted for in the data processing architecture.

[2]Sampling was done with Spark's dataframe *sample* method and a sample size of 0.0001 based on the random seed of 108115100101 (representing *"lsde"* in ASCII decimal numbers)

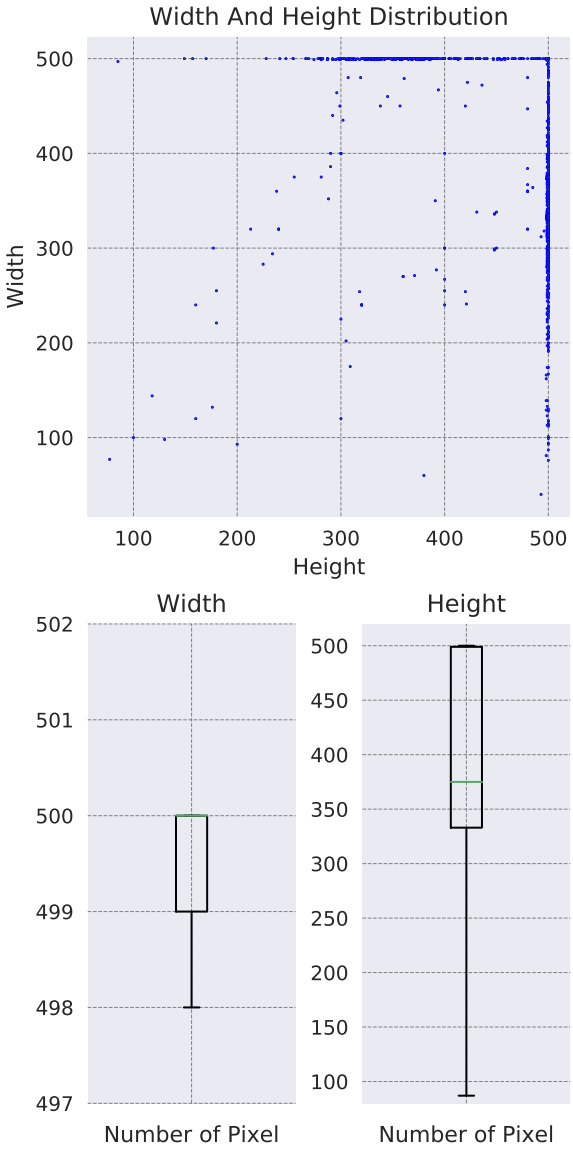


Figure 3: Image Resolutions in Initial Analysis

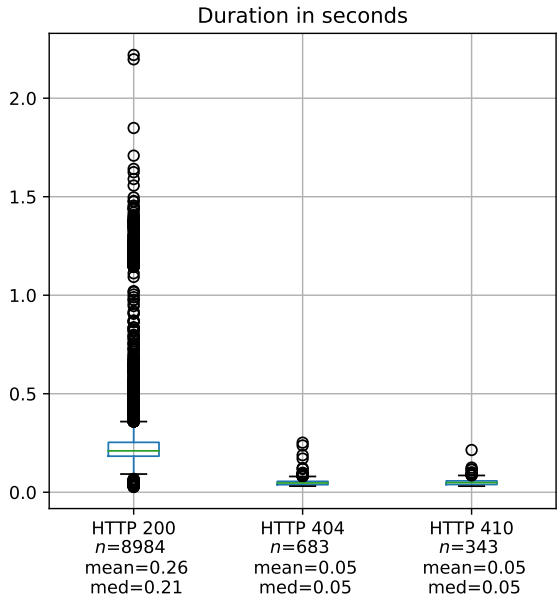


Figure 4: Downloading Statistics in Initial Analysis

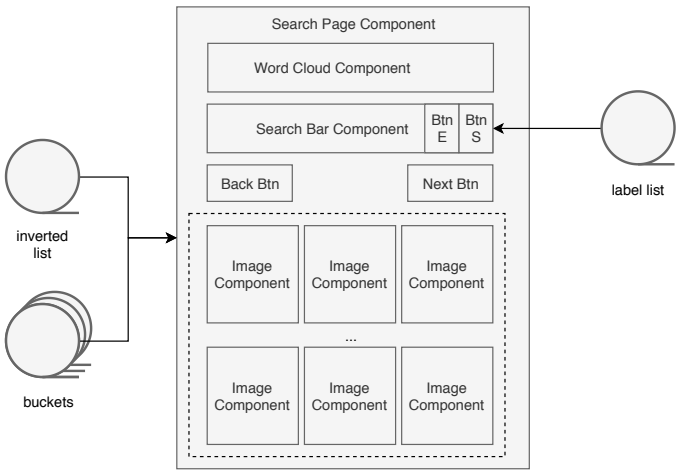


Figure 5: Components of the Search Interface

Regarding the quality of the dataset it became also apparent, that the sampled data did not have an issue in regards of inconsistencies. All URLs used to request images from Flickr were valid URLs, also all columns adhered to the specified schema (see Table 6) and the data they suggest to contain (i.e., the image extension contained jpg and png formats).

As a result, it was concluded that around 99% of the rows are associated to images, which is consistent with the claims in [11]. Furthermore, it assumed that currently (as of October 2019) around 90% of the images are currently available. Based on the other figures obtained during the analysis, it is projected that a total of 5.39 TB would have to be downloaded. This is in stark contrast, to the claimed approximately 13.5 TB of image data in the published paper. Initially, we assumed that as the standard deviation already indicates (see Table 1), that a simple linear forecast using the mean file size and the assumed amount of still available images might be highly inaccurate. In fact, based on the final download volume, we can conclude that our prediction was reasonably accurate (See Section 3.3.1). However, because of the uncertainty at that point in time, one of the main concerns after the initial data analysis was that it might turns out to be unpractical to download all images in a short time span. Therefore, the initial focus of the iterative development process was to download all images, which is discussed in the following section.

3.3 Iterative Development

3.3.1 Flickr Image Downloader

As mentioned before, the initial analysis did have an impact on architectural decisions for downloading the images. The main areas that were considered during drafting the downloading process included a suitable storage format and structure, a scalable and lightweight downloading implementation and some potentially necessary post- and preprocessing steps.

Concerning, the storage of images, first, we had to decide if storing images on disk or keeping them in memory and directly classifying them results in an overall higher processing speed. Based on classification experiments conducted during the initial analysis we decided that storing images on disk first and classifying them in a later step is a workable and preferable solution. The main reason for this is that the main Spark cluster used for this project did not allow GPU-based computation, which were deemed necessary to classify images in a reasonable time. In fact, the GPU-enabled computations were only possible on a dedicated instances, that did not provide any Spark-based distribution of work. This matter is discussed in more detail in the subsequent Section 3.3.2. Additionally, it was recognized that storing images for a couple of days is not costly at around 23 USD for 1 TB per month (according to the AWS Pricing Calculator [2]). Apart from our decision to store images on disk, a suitable way of storing images was also required. Given the previous experience from the initial data analysis that storing single images on the Databricks File System (DBFS), which is backed by a AWS S3 storage instance, comes with additional overhead, it was evident that some form of container file format was needed. Given the fact, that the Parquet format is well supported in Spark, and that it supports storing binary data in columnar storage, experimentation on a small subset

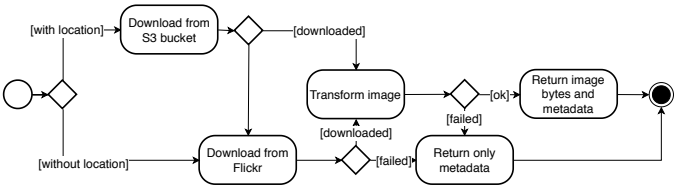


Figure 6: UML Activity Diagram for Downloader UDF

of the data was carried out. After reasonable performance a decision towards storing images in a binary column was made.

Aside the decision for a storage format, a scalable and lightweight downloading implementation needed to be found. After preliminary research on how to write scaleable programs on Spark, a decision for using User Defined Functions (UDFs) was made. UDFs provide a simple but efficient Application Programming Interface (API) injecting processing logic and applying it on the each row of a Spark Dataframe. The UDF written for downloading and preprocessing is depicted in Figure 6. As shown, the UDF first tries to download data from an S3 bucket, when location data is present. As the Flickr image data is not only used by this project, but also another project that is part of the LSDE course, it was decided to collaborate between groups and use the data the group already downloaded. However, if downloading from the S3 bucket fails the processing falls back to the Flickr service, as the other group stated some errors occurred during downloading in their process. If the image is successfully downloaded, the image is prepared for classification by resizing and cropping the image. This way, the variety of resolutions in the data is removed and images are ready straightaway for further processing. Additionally, we assumed that the transformation is also advantageous in comparison to storing the raw image, as storing the images at an unnecessary size is avoided. Based on the analysis, we assumed that resizing and cropping the images would lower overall file size, especially since the paper claimed to have a total of 13.5 TB of image data [11]. However, after downloading all images, we can conclude that, first, the download volume was significantly closer to our prediction based on the random sample (see Table 1 and 2) as the total amount of data claimed by Thomee et al. [11], and second, that storing cropped images actually takes up more space than the raw image data (7.07 TB compared to 5.33 TB). Despite the increase in storage required, preprocessing images did still turn out to be advantageous as the data was ready for classification straightaway.

Furthermore, our initial doubt about whether we can download all images in time, was not justified. Collaborating with the other group, especially Mathijs Maijer, which was processing people's faces from a subset of the Flickr dataset and the fact that Flickr did provide constant downloading speeds of 20 MB per seconds according to the Spark Cluster statistics, allowed us to download all images still available in approximately 100 hours (one partition of the dataset took around 10 hours processing time on the Spark cluster). It is evident that the sufficient speed originates from Spark's capabilities to distribute work well, as the UDF ran across the whole cluster with 16 worker nodes, each with 8 parallel threads. Overall, downloading and storing images on

Property	Value
General Figures	
Total Volume Image Data	5.34TB
Total Volume Stored Data	7.07 TB
Total Images Downloaded	89,205,355
Average Downloading Speed	20MB per second
Image File Size Figures	
Average File Size	59899.52 Bytes
Std. Dev. File Size	26038.14 Bytes
Min File Size	159 Bytes
Max File Size	3683448 Bytes
Image Resolution Figures	
Average Image Width	389.22 pixel
Std. Dev. Image Width	73.06 pixel
Min Image Width	1 pixel
Max Image Width	729 pixel
Average Image Height	463.99 pixel
Std. Dev. Image Height	65.36 pixel
Min Image Height	1 pixel
Max Image Height	729 pixel

Table 2: Downloading Figures

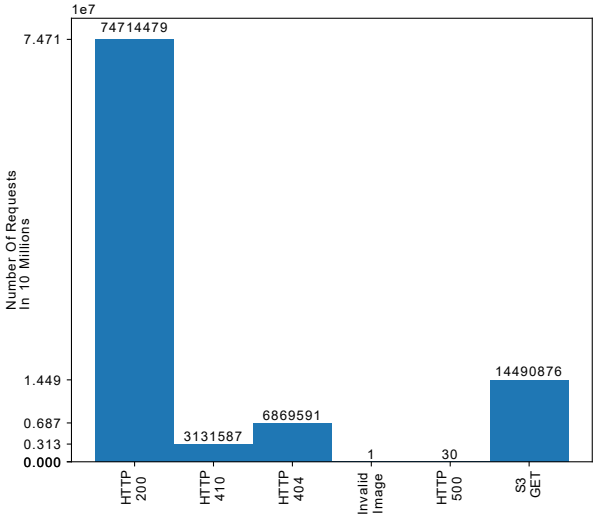


Figure 7: Overview on Request Responses

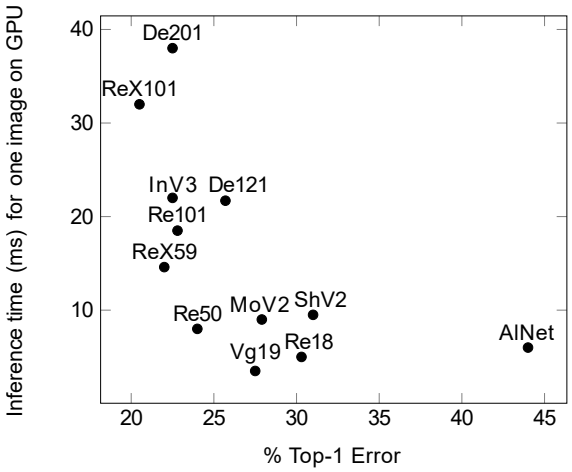


Figure 8: Pre-trained model comparison [12]

disk prior classification turned out to be viable, as CPU-instances and S3 storage is compared to the GPU-enabled instances reasonably cost-efficient. Furthermore, it also allowed us to scale the GPU-based instances easily if needed, which turned out to be useful during the classification phase of the project.

3.3.2 Image Classification

Another challenge of this project was to identify a solution that can process large amounts of photos and recognize the main characteristics that are present in each image in a timely fashion. Fortunately, image classification is a highly researched field that has produced various solutions which can be applied for this project. Currently, the best approach for image classification is to use a deep learning technique that makes use of CNN. These types of networks are primarily used in analysing visual imagery and give outstanding results due to the connectivity pattern of the neurons that resembles the organization of the animal visual cortex. These kinds of networks have already been developed by many research institutions such as The University of Oxford, Stanford University or Google and have been made publicly available to be used for solving challenges in the computer vision field.

In order to classify images that can contain any kind of features such as trees, cars, fruits or animals, a CNN model that was already trained on multiple categories of items can be applied. Such a pre-trained model must offer a fast inference speed so that it can easily scale on a large dataset and it must also offer a relatively small error rate. For this project, we decided to go with the InceptionV3 CNN model, developed by Google and pre-trained on 1000 classes of the ImageNet database. The ImageNet classes are very specific and contain many categories of items such as dog breeds or airplane types and can successfully be applied to identify a variety of features from the images. Because the InceptionV3 model was already trained on a general set of classes, there is no need for additional model training using techniques such as Transfer Learning as the model can directly be setup for the inference of images.

To make classification of images possible, we created a Python module that includes the PyTorch package. Using

PyTorch, one can easily build and make use of the various tools for Neural Network development, such as tensor computation or batch inference. This package also offers the InceptionV3 pre-trained model as an embedded component so that developing a classification pipeline can be done much easier. In order to correctly classify the dataset, an intermediate step was introduced that normalizes each image by converting it to a tensor format and making each element of the tensor fit within a range for faster inferencing and reducing skewness.

By classifying a subset of images from the YFCC100M dataset, we noticed that for all greyscale images, the classification was failing. This is caused by the model expecting a tensor that was calculated from an RGB image. These images contain three channels, one for red, one for green and one for blue. In comparison, the greyscale images have a single channel, called L which represents the luminance. To also enable the classification of these images, we implemented a converter function that makes the greyscale image "appear" as an RGB image, by repeating the image array for the luminance three times, for each RGB channel.

Due to the huge volume of images that must be processed, certain parallelization techniques must be employed to ensure that the entire dataset can be classified in a reasonable amount of time. The first attempt at parallelizing the classification of images was to distribute the classification work on multiple worker nodes that are part of a Spark cluster. Although a good solution for most parallelization problems, the classification of the entire dataset would take up to 120 days using 16 worker nodes with 4 cores each. The faster alternative came in the form of GPU computing, by storing the classification model and the input tensors in the GPU memory. This way, we can make use of the large amount of simple cores that are found in a GPU, which can allow us to effectively run the CNN model inference much faster. By classifying a sample of our dataset on a compute instance with an Nvidia V100 GPU, we noticed a speed of 1000 images per second which is significantly larger than the classification speed on multiple worker nodes without GPUs. Unfortunately, by also factoring in the download time of the images on the GPU instance, the actual speed was drastically reduced and to solve this, we decided to scale up to two GPU intensive compute instances. This way, we manged to find the optimal solution that can be applied to classify our entire dataset of nearly 90 million images in a reasonable amount of time. To benefit more from the performance of GPU computing, we decided to make use of the Batch Processing function of the PyTorch package, which adds the benefit of classifying multiple images in parallel instead of classifying one image at a time.

The result of the classification is stored in a CSV file using the following format: *id, label, confidence*. The reasoning behind choosing CSV as an output format for the classification was the lack of a Spark environment which could have allowed us to directly write the result in parquet format. The entire process of classification is displayed in Figure 9.

Based on the classification output, and the fact that the subsequent processing steps are computed on a dedicated Spark Cluster, a postprocessing step, as depicted in Figure 2 was required. This postprocessing collects all classification results into a labels parquet, and the downloaded image ids from the downloading step as the downloaded parquet. Additionally, the postprocessing collects additional statistics

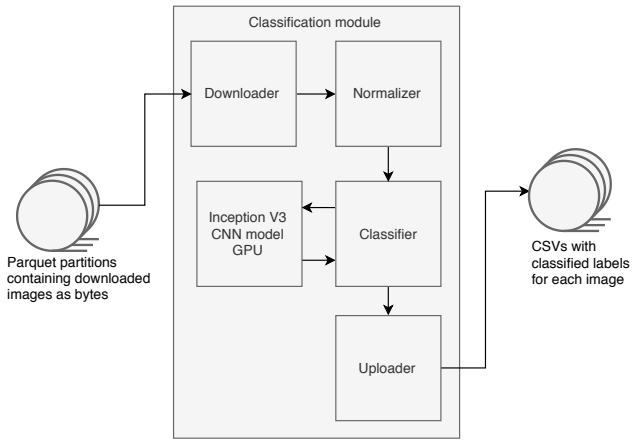


Figure 9: Component diagram of the classification module

for documentation purposes, such as the amount of bytes from the raw image and the resolution. The labels parquet and the transform and index are the input for the final transform and index step.

3.3.3 Transform and Index

The transform and index step prepares the raw pipeline output so that any application that can read JSONs would be able to handle the project's final output data. This way we were able to independently implement the visualization, while implementing the data pipeline at the same time. In fact, very early in the project it was agreed on how the data structures for the visualization have to look like. This reduced the amount of changes required to process the data in the visualization throughout the project.

The whole transform and index job is divided into four stages. The overall pipeline combines the Flickr metadata the labels for all images that were downloaded and provides an inverted list that references a partitioned search result for each label that was found during classification. Each image that that the pipeline was able to classify is present in one or multiple rows, depending on the confidence level.

Concerning the four stages of the job, first the input data is read and prepared for the main processing. This includes the extraction of all classified images, so that each image that was previously classified is associated with at least the label having the highest confidence, remaining labels must meet a minimum confidence of at least 1%. This confidence value was chosen to reduce the search index significantly, as some images only belong to one class at max and labels predicted with a very low confidence would not be present in the picture. We acknowledge that some correct labels are dropped during this process, especially when they are in the background. However, we believe that dropping images with very low confidence not just helps reducing the overall size of the search index, but also making the overall results better. We determined the minimum confidence value through various experimentation during implementation. In the first step, the other labels that were identified and associated with the image are also added to each labelled image row, which allows cross-referencing labels across single images. Finally we calculate partitions, so called buckets in the context of this work. For each label one to multiple buckets

make up the whole calculated search index. The buckets contain each a maximum of 30,000 images that are labelled with the associated label given a certain confidence. Each bucket has a unique identifier and is associated to a label. Each bucket is sorted by confidence and also all buckets of one label are sorted in the same manner. Essentially, the results for each label are divided into multiple buckets, and all buckets calculated provide the full search index of our processing.

In the second stage of the pipeline additional data required for the search interface is collected. For each image and its associated labels metadata that enables our web application to link to the matching Flickr URL is collected and transformed into a JSON representation via a UDF. Subsequently, the output of that UDF is partitioned into the previously calculated buckets, reducing the total amounts of rows to the total number of partitions. Finally, each bucket is written to disk totalling at around 2 MB per bucket and an overall bucket count of 10805.

In the third stage of the transform and index processing the final data product is created. First an inverted list is created. The inverted list is important to provide the search capabilities for our image retrieval search, as it provides a dictionary of labels mapped to the buckets, across which the calculated search results are spread. Furthermore, a list of all labels that were present in the dataset is generated, to allow the visualization providing some search functionalities to the enduser. Additionally, a list of all classes with the total count of images is generated so that a word cloud of the fully classified dataset can be generated. Initially, this word cloud was planned to be created within the pipeline. To compute the word cloud an existing python package was forked[3] and extended to generate vector-based graphics, but as it would have required Cython as a dependency on the Spark Cluster, local computation of the word cloud was a preferrable option. Especially, since the generated result needed some adjustments to be used in the final visualization.

In the final and fourth stage all results are collected and written to tar files to enable easier downloading. The final results consist of 10,805 partitions, containing 309,726,315 labels for all downloaded images. Of all downloaded images only 29,824 images could not be classified. According to analysis on a small subset, and looking at images via searching the keyword *unclassified*, these images seem to have mostly invalid metadata causing classification to fail. This might have occurred due to that fact that users upload large sets of images without knowing what images they are uploading.

3.3.4 Search Interface

This section describes the visualization and the inner workings of the search engine. The search engine is the frontend of our project, allowing users to search through the final classified dataset using keywords and performing information retrieval to return images that correspond to the given keyword. The description below has also been illustrated in Figure 10.

The frontend loads word cloud of all possible tags, below which is a search bar located through which the user can enter a keyword. Adjacent to the search bar are two action buttons, one for choosing a keyword at random, taking the choice away from the user and still being able to deliver a search function with valid results. The second one is a simple search button which can be used to trigger the search when the user enters a keyword. The user can enter a keyword in multiple different ways. The user is allowed to type in a keyword, to select a keyword from the word cloud and to choose from an auto-complete function drop-down that appears when the user is typing any keyword. On the backend to perform these functions, when the homepage is loaded, both the list of all possible keywords and the inverted list are loaded on into the webbrowser's memory to make the process of searching smoother and more seamless.

Once the user enters a keyword, the system checks if the keyword exists in the inverted list. If the keyword does not exist, the user is returned with a "no results" page and asked to search again. If the user chooses a keyword from the word cloud return no results is not possible since every keyword in the word cloud exists in our system. Similarly if the user chooses the "choose a random" action, the search bar is populated with a keyword term that exists in our list. Once the keyword is validated, if it exists, the total number of images the keyword is linked to and the unique identifiers of all buckets, making up the search result are fetched from the inverted list. A bucket, as previously mentioned, refers to a subset of results. Since the frontend deals with a large amount of data, these buckets help the browser to better process a query without requiring too much computational power. Each bucket contains at most 30,000 results. To provide a smooth and fast user experience the bucket is divided into pages, that contain a maximum of 100 images. The system then sets the bucket index and page index for the keyword so that the first 100 elements are shown.

To allow the user to navigate through the results, up to four navigational buttons are offered. Two of these are the next button which takes the user to the next page and the last button which takes the user to the last page. The other two buttons allow the user to navigate back one page or jump back to the start of the results. During navigation, additional buckets are loaded on demand. Thus, if a user reached the end of a bucket, and more results are present in the inverted list, the subsequent bucket is loaded, when the user navigates further through the results. The user is allowed to switch the keyword in the middle of looking at the results, the search resets and the validation checker for if the keyword exists is run again and the whole process in repeated.

Concerning the results, all images are shown in equal sized squers to provide a uniform presentation of each image. Upon hovering a result, or click on it on a mobile device, the whole image is shown. Additionally, for the current image selected, the classification confidence is displayed upon hovering/selecting. In general, results are sorted on high to low confidence. When a user clicks on an image or, when using a mobile device,s a user has clicked the open button, redirection to Flickr is initiated.

The static-only limitation does however impose some restrictions on the functionality of the search engine. There are no results for keywords not in our system. Multiple keyword search is not possible. Although keywords that have multiple words can be searched, two independent keywords searched together can not. However, this could be a possible future task continuing on from this as a foundation. The performance of the search engine is smooth, even though the

[3]See `https://github.com/amueller/word_cloud`

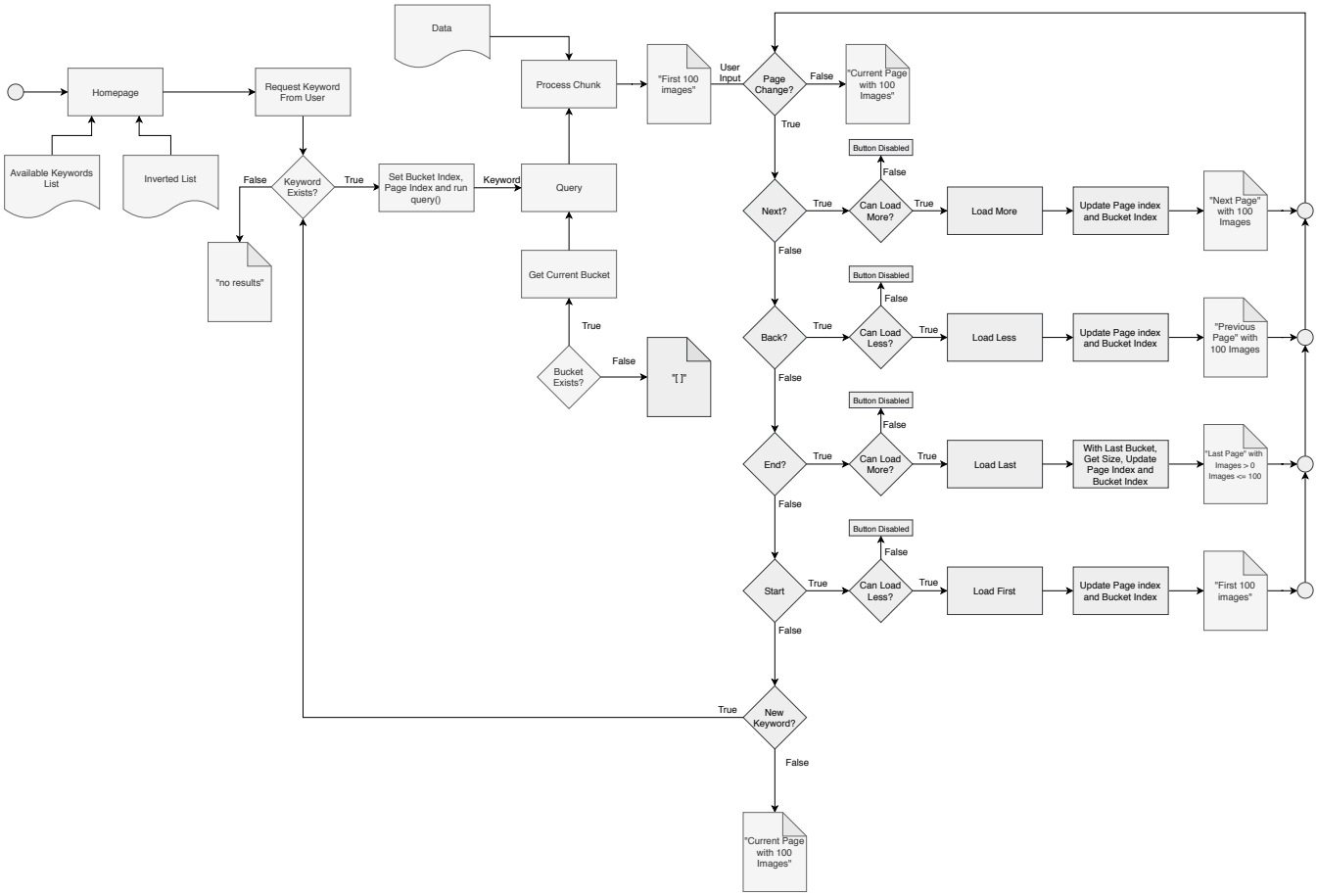


Figure 10: Visualization of Search Sequences

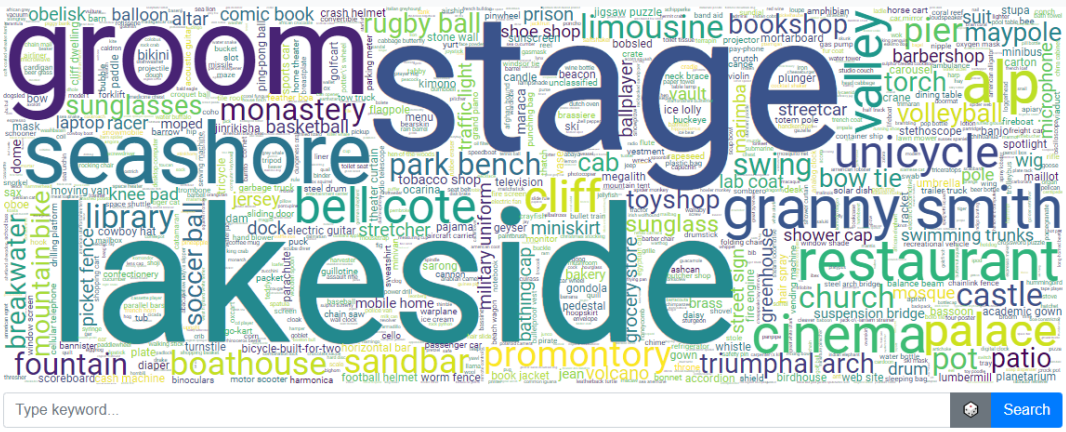


Figure 11: Homepage of Search Engine

Table 3: Validation Test-case

Feature	Optional?	
Visualization can be opened	required	✓
User can enter some input	required	✓
User can auto-complete input	optional	✓
User can select a keyword from word cloud	optional	✓
User explorer search index with random keyword	optional	✓
User gets feedback for keyword without results	required	✓
User gets first 100 Results for the keyword on first page	required	✓
User can go to next page	required	✓
User can go to previous page	required	✓
User can go jump to last page	required	✓
User can go jump to first page	required	✓
Image Confidence and Resolution Transformation	required	✓
Redirect to Flickr	required	✓
Mobile Ready	optional	✓
Multiple Keyword Search	optional	✗

dataset of results is pretty huge, the concept of pagination and bucketing helped to provide a fast and easy to use search engine without slowing down the user's device. In general, one could argue that the provided search engine is pretty sophisticated given the limitations.

3.3.5 Validation and Testing

Throughout the project, validating the insights from the data and testing the provided visualization was required. In general, this was done mostly directly after implementing certain components or obtaining certain statistics. Validation of the data processing pipelines was done by checking plausibility and cross-checking information on the overall dataset that we obtained for predicted values on our obtained subset. We also used the information provided by the dataset publishing paper by Thomee et al. [11]. However, is was decided that a general user interaction flow should be tested for the final submission. Therefore, we defined a list of features that were planned to be implement within the visualization. This list was used to define a final testcase to provide measure for final sign off prior submitting the results. The final checks for this list are displayed in the Appendix in Figures 13 and 14.

3.4 Project Plan

Throughout the whole six weeks a project plan was followed, which was composed during the project initialisation. Essentially, we planned on how to achieve our objectives in the time that is given and with the highest quality possible. However, throughout the project some changes to our initial plan occurred as shown in Figure 12. The first half of the project plan remained unchanged, as during the first two weeks tasks were well defined and therefore easier to judge in regards of time required.

The project began with a kick off stage wherein we received the details of our project, which then led us to the planning and work distribution phase. During this the project was setup and the work was distributed amongst the team. Work was divided across the different modules which had to be worked on independently in parallel and later combined.

The next phase was the architecture design phase which was a prerequisite to the three main modules. Once the initial design for the architecture was finalized, the focus was then shifted to doing some initial data analysis, which also involved cleaning the data and running some preliminary tests to understand exactly what kind of data we are dealing with. This step allowed to further refine the planned architecture and to come up with a more specific project budget.

After that, all three team members worked to varying degrees on the downloading/data scraper, classification and visualization. Based on the sample generated during the data exploration step, the entire pipeline was implemented until the 14th of October. Subsequently, based on the first working version of the entire pipeline the large datasets could be tested. Initially, it was assumed that downloading and classification of the images could be done during implementing the final visualization. However, running the classification only started on the 20th of October, due to various changes required. Once all the three stages were finalized, it was possible to process the entire dataset. This posed a challenge in regards of classification, since it was the most time consuming part of the entire pipeline. Further, it took also significantly longer to implement the visualization, as providing a good user experience was deemed advantageous.

Testing was carried out through all the stages as various libraries were tested, different techniques to find what best fit the project's objectives and could help to achieve the goal in time. Due to the changes in time and scope of other activities writing the report was delayed, but finished in time.

Table 4: Initial Cost Estimations

Calculation	Estimated Total Costs
AWS Egress	0.90 USD
Download with CPU	89.59 USD
Image Storage	101.48 USD
Classification time in GPU	196.39 USD
Total Estimated Cost	**388.36 USD**

3.5 Project Budget

Apart from handling the limited time for the project, cost also had to be considered. At the beginning of the project one aspect was to estimate the project costs. The importance of having a closer to an actual estimation was necessary, in order to avoid going over the given budget of 1500 USD. Although the budget was not an entirely strict line, by estimating the project costs it could be ensured that the creation process of the data product was not only complete in its technical aspects but it also was economical within the given budget. In the following sections, the initial and final estimated cost are described.

3.5.1 Initial Estimation

Based on the initial data analysis and the AWS Pricing calculator[4], the overall cost for completing the project were

[4]Based on the Northern Ireland Pricing Region

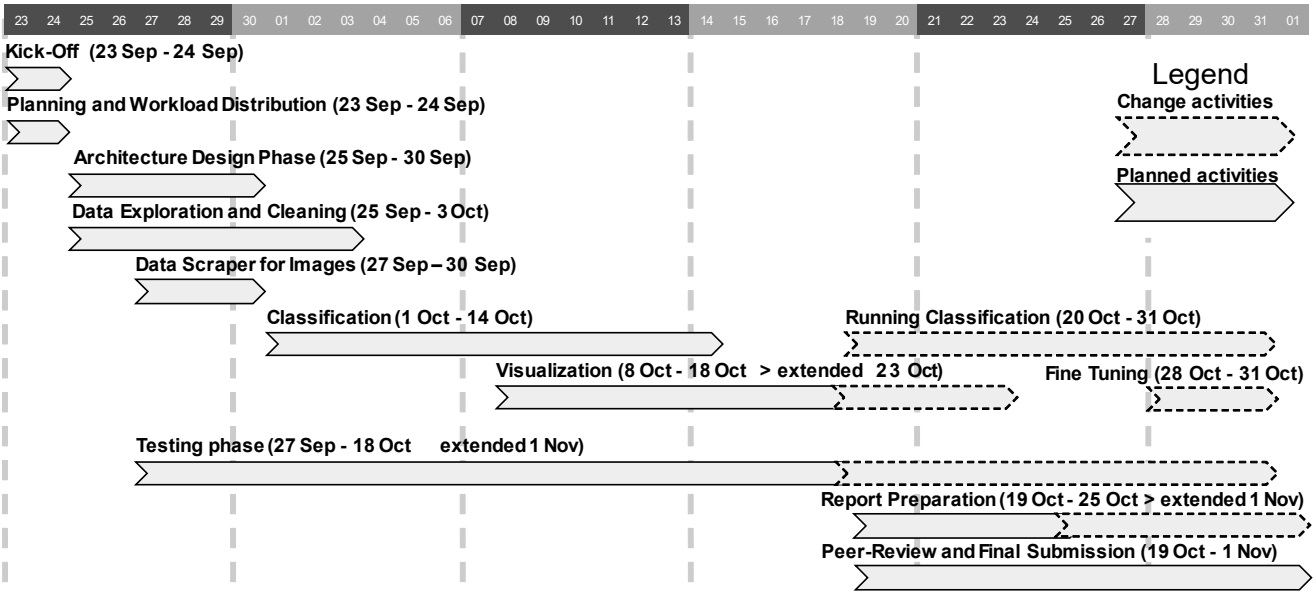


Figure 12: Project Plan

estimated to total 388.36 USD. These total cost are composed of various aspects, as listed in Table 4. The most significant position in that listing are the required GPU-based instances that were deemed required for classification, totalling at around 196.39 USD based on the latest Spot market prices for p3.2xlarge instances. Alternatively, we estimated that cost could be marginally reduced if the classification was to be done on a CPU, which would have cost 165 dollars. We argued that this is a insignificant difference in price for a much faster execution. A design decision had to be made, and after consultation on the availability and the freedom for such a choice, considering we were still well under the actual budget with our estimation, we decided to include classification using the GPU in the actual cost, instead of using the CPU. Given the initial worry, that download might be a bottleneck during the project execution, storing images on S3 was also included in the cost estimation. The cost to store the images on S3 was estimated at about 102 USD, while downloading the images in regards of the total download time was estimated to cost about 90 USD. Other components of the price estimation were AWS-Egress, which was estimated at about 0.90 dollars. However, it needs to be stated that this estimation was presented as part of our project plan, taking into consideration the total loadable images, the average file size of the images in bytes, the time it took to classify each image using a GPU and using a CPU. Given the fact, that a quite a few changes were required throughout the project, estimated cost had limited applicability in hindsight.

3.5.2 Actual Costs

After completing the majority of computational work, it was considered to be good practice reflecting on the total project costs. However, no exact measurements were possible due to a lack of monitoring data present. Due to this, only a worst cast scenario based on the continuously manually tracked running times of the Spark cluster, the dedicated classification instances and the used storage could be estimated as of Table 5. Regardless of that, it is argued that the estimated worst case final cost still indicate that initial cost were underestimated. First, the actual development and download of temporary data as well as the final results (totalling at 22.3 GB) increases the AWS egress position by a multitude. Second, performance of classification was slower than expected, although managing around 120 images per second, the classification took almost nine days in total. However, the amount of storage required was lower than previously expected, also we initially did not account that using the data from S3 with EC2 instances is free too. Expecting worst-case the regular price for i3.xlarge and r4.4xlarge instances at 0.344 USD per hour and 1.1856 USD per hour respectively, running the Spark cluster would have been significantly more expensive than anticipated. This is simply due to the fact, the for the initial cost estimation mostly Spot pricing was assumed and that later during execution, at least for the p3.2xlarge the regular price was relevant due to the configuration given. Furthermore, it needs to be mentioned that the Spot prices are quite volatile. During the project we noticed changing cluster configurations,

Table 5: Worst-Case Final Cost

Calculation	Total Costs
100 GB AWS Egress (0.15/GB for the first 10 TB p/m)	15 USD
16 x i3.xlarge for Spark Worker (approx. 9 days at 0.344 USD p/h)	1188.86 USD
1 x r4.4xlarge for Spark Driver (approx. 9 days at 1.1856 USD p/h)	256.09 USD
1 x p3.2xlarge for Spark Driver (approx. 9 days at 3.305 USD p/h)	713.88 USD
6 TB AWS S3 for files (approx. 11 days at 141.32 USD p/m)	51.82 USD
Total Worst Case Cost	**2,225.65 USD**
Total Worst Case with Spot Prices	**1,393.45 USD**

as prices and therefore the amount of Spot instances used, changed over time quite often.

Regardless, it is believed that cost are likely to be significantly lower. As the given calculation is estimated based on a worst-case scenario, which is that all listed resources were used fully over the whole period. In fact, due to dynamic resizing and shutdowns on idle, resources were not fully utilized over the whole period. Furthermore, as the most recent market prices for Spot instances, according to AWS's Spot Instance Advisor[5], were significantly lower. All instance types showed (as of the 01.11.2019) 70% of savings over the On-Demand prices in the last 30 days. Based on this, total cost would drop by 832.20 euros[6]. This both highlights the advantages and disadvantages using cloud-based computations, as prices do vary a lot and reliable predictions about project cost are difficult without the appropriate data.

4. CONCLUSION

Gary King, social science researcher at Harvard University, argues in a presentation that "Big Data is Not About the Data" [8]. He argues that Big Data is more about what statistical and other methods allow to extract from that large amount of data. According to him, the novel insights made possible by Big Data are what makes data so important these days. Upon reflecting this statement and the work done document in this paper, we, the authors, truly believe that this is true.

Looking back over the last six weeks it is not necessarily the total amount of data processed that is most inspiring, but that by classifying the huge corpus of images, we had the opportunity to discover what type of images are actually uploaded on the Flickr platform. We realized that these images cover a huge bandwidth of scenes and and objects that were photographed. Images on Flickr seem mostly related to free time activities or social events, given that the keywords *lakeside*, *stage*, *groom* or *seashore* are predominately identified entities in the data. Overall, this is not surprising because people possibly are more eager to share such moments in their life publicly than others, however, that such insights can be easily extract from just images and nothing else, is marvellous.

From a technical perspective, it can be argued, taking into account the entire course contents of LSDE, that the technology used during the project really helped the reach our objective. Spark's scalability helped in tackling a lot of problems, that would have required much more work otherwise. The usage of Spark did not remove the necessity to carefully think about how to implement programs that operate on large data, but we argue that the entire pipeline scaled very well from the initial small sample to scaling it up to the entire dataset. However, some minor challenges were still present in the development phase. While the downloading of images did work and scale very well, implementing the image classification for the processing of large parquet files without Spark required some thought, especially since the GPU-based instances should be fully utilized for maximized cost-efficiency. More specifically, implementing the classification was straightforward on smaller datasets, but had to be adapted after applying it to the bigger dataset, reading single row groups from the parquet. Overall, it can be concluded that Spark's scaling does work very well when done right, but non-Spark-based processing always should be kept in mind.

Concerning, the implementation of the static search engine, it can be said that Vue.js turned out to be the right tool of choice. Due to its lightweight nature both JavaScript and Vue.js-based implementations can easily be combined. Using available frameworks such as Bootstrap and axios boosted implementation speed drastically. Furthermore, it was very straightforward to quickly get the search interface mobile ready. The search mechanism implementation, which was based on an inverted list, mapping each keyword to its precalculated search index scaled also very well. Due to the partitioning and the inverted list, the search is fast and reliable. However, due to the large search index and denormalization a lot of storage is required.

Regarding the project's cost and timeline, it is also evident, that most activities were planned too optimistic and took longer than expected. Also costs can be very hard to predict, especially, when prices are not constant but influenced by the current market demand. Therefore, careful analysis of the data at hand is crucial and also having some additional information on the history of market prices for cloud computing is benefical.

Reflecting upon the overall personal effect of the LSDE project, it is evident that this project did not only taught us how to handle large amount of data with Spark, but involved searching and experimenting with sophisticated data structures and algorithms. Additionally, we learned more about how image classification with CNNs is done. Therefore, LSDE was a huge step towards a broad and also deeper knowledge of Big Data processing fundamentals, while making each one of us acquainted with relevant side areas, such as web development, machine learning and distributed data processing.

Therefore, we truly believe that this work gives an appropriate answer to our given research question. We downloaded all images available in the given time using Spark's UDFs to scale easily, we classified over 89 million images during the course of this project, and made the search index available through a static search interface.

Finally, the implemented solutions still showed some shortcoming which could be improved in the future. First, classification is occasionally not accurate. Maybe cross-model classification might yield better results. Also, the keyword search could be improved to support more than one keyword at a time. The main challenge would be, how to move processing away from the web-based client to precalculating results while also maintaining a reasonable size of the search index. Also, a Word2Vec model could be trained on the used ImageNet classes, to provide a more sophisticated search experience. All necessary frameworks ane methods to do so exist already, such as TensorFlow.js to run pretrained model in the browser or approaches using data from Wikipedia to train the Word2Vec model. We believe this would have even further improved our search, so that users would not even realize that the search index was precalculated.

[5] `https://aws.amazon.com/ec2/spot/instance-advisor/`

[6] Savings only applied to Spark worker instances

5. REFERENCES

[1] The multimedia commons initiative, 2015.

[2] Amazon Web Services. Amazon web services simple monthly calculator.

[3] M. Aubert, A. Brumm, M. Ramli, T. Sutikna, E. W. Saptomo, B. Hakim, M. J. Morwood, G. D. van den Bergh, L. Kinsley, and A. Dosseto. Pleistocene cave art from sulawesi, indonesia. *Nature*, 514(7521):223, 2014.

[4] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[5] Google. How content id works - youtube help, 2019.

[6] A. Hern. Revealed: catastrophic effects of working as a facebook moderator, 2019.

[7] S. Kalkowski, C. Schulze, A. Dengel, and D. Borth. Real-time analysis and visualization of the yfcc100m dataset. In *Proceedings of the 2015 Workshop on Community-Organized Multimodal Mining: Opportunities for Novel Solutions*, pages 25–30. ACM, 2015.

[8] G. King. Big data is not about the data! *Presentation (Harvard University USA, 19 November 2013)*, 2013.

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[10] L. Shannon. Tumblr will ban all adult content on december 17th, 2018.

[11] B. Thomee, D. A. Shamma, G. Friedland, B. Elizalde, K. Ni, D. Poland, D. Borth, and L.-J. Li. YFCC100M: The new data in multimedia research. *Communications of the ACM*, 59(2):64–73, 2016.

[12] Vishwesh Shrimali. Pytorch for beginners: Image classification using pre-trained models.

APPENDIX

A. Glossary

ACM The Association for Computing Machinery (ACM) is a educational and scientific computing society that delivers resources that advance computing as a science and a profession.

ContentID YoutTube ContentID is a system that automatically detects videos that are protected by copyright.

Exif The Exchangeable image file format specifies a standard how metadata from cameras, scanners and other media devices are handled and stored within a media file.

B. Acronyms

ANN Artifical Neural Network

API Application Programming Interface

CNN Convolutional Neural Network

DBFS Databricks File System

LSDE Large Scale Data Engineering

UDF User Defined Function

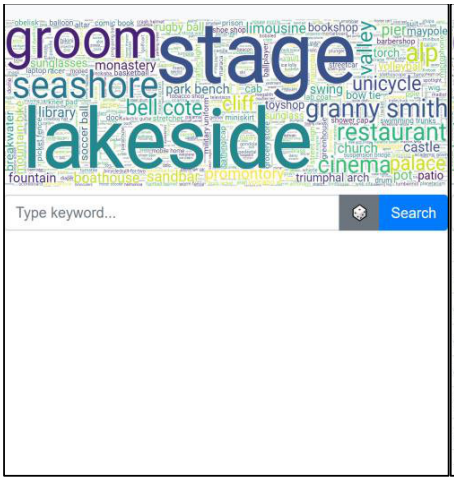
YFCC100M Yahoo Flickr Creative Commons 100 Million

Table 6: Raw Data File Schema

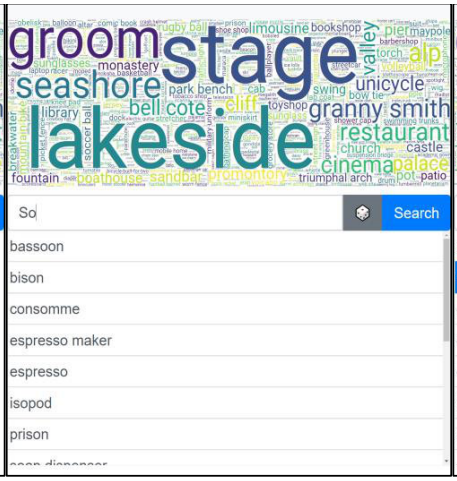
Column Name	Data Type
id	long
user nsid	string
user nickname	string
date taken	date
date uploaded	integer
capture device	string
title	string
description	string
user tags	string
machine tags	string
longitude	string
latitude	integer
accuracy	integer
photo video page url	integer
photo video download url	string
license name	string
license url	string
photo video server id	string
photo video farm id	integer
photo video secret	integer
photo video server id	string
photo video secret	string
photo video extension	string
photo video marker	short

C. RAW DATA FILE SCHEMA

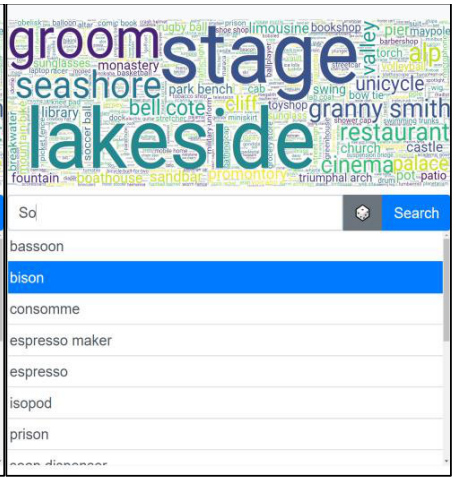
In Table 6 the raw table schema is listed. Only eight columns, namely id, user nsid, photo video server id, photo video farm id, photo video secret, photo video server id, photo video secret and photo video extension were relevant for the visualization. Based on these eight columns all required URLs could be constructed and each image uniquely identified.



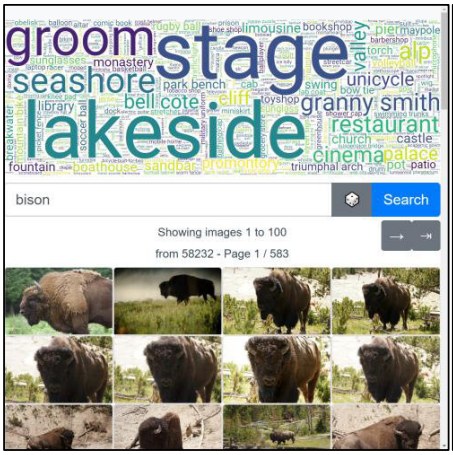
1. Open Visualization



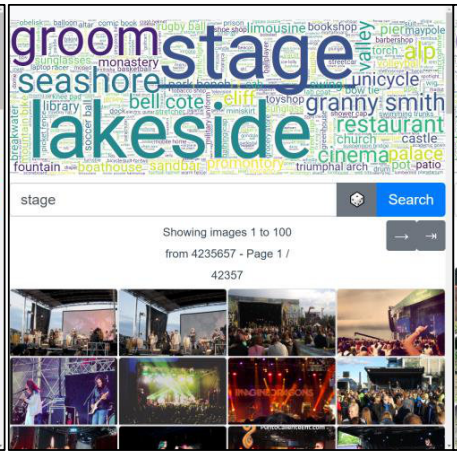
2. Enter Input



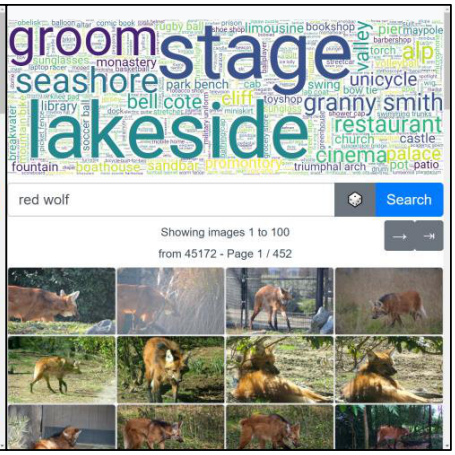
3. Autocomplete



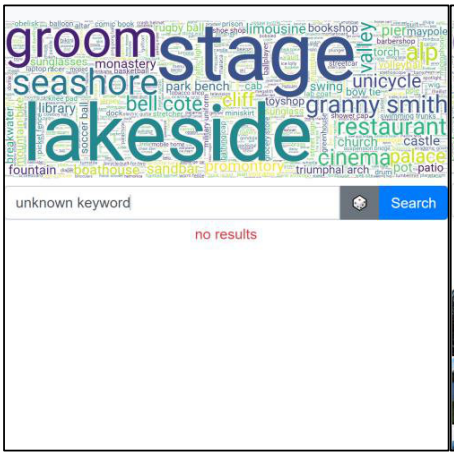
4. View Results of Autocomplete



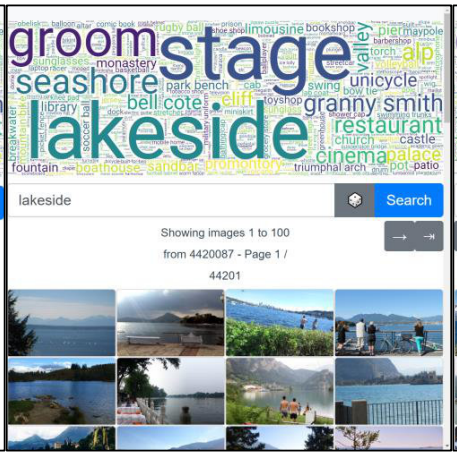
5. Select Label from Label Cloud



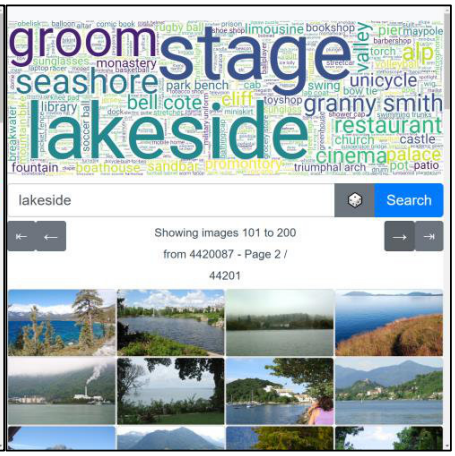
6. Select Random Label with Dice Button



7. Enter keyword not in search space

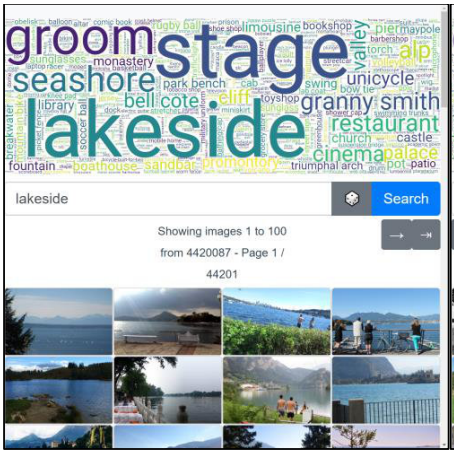


8. First Page displays 100 items max

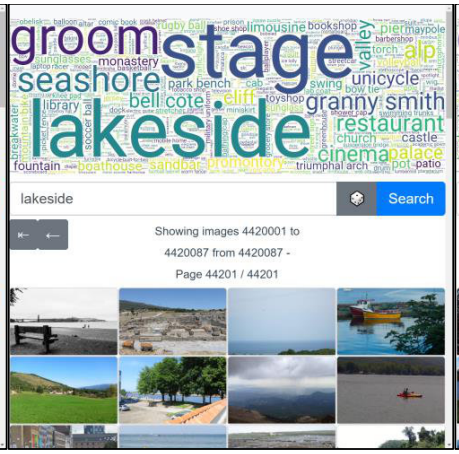


9. Go to next page

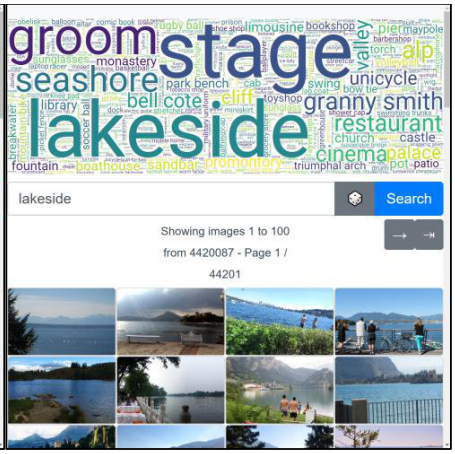
Figure 13: Testing Screens 1 to 9



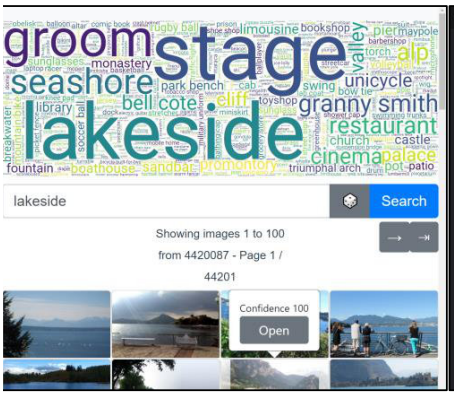
10. Go to previous page



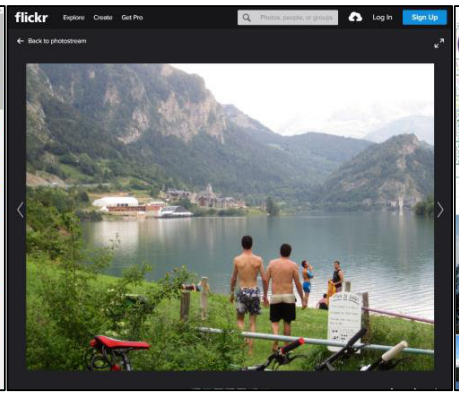
11. Jump to last page



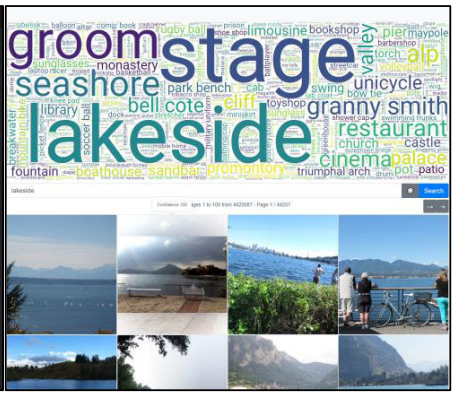
12. Jump to first page



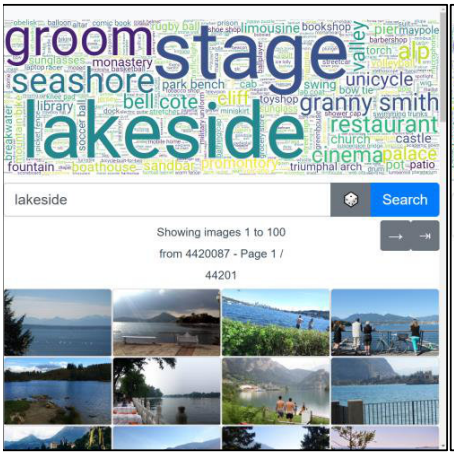
13. Confidence And Image Resolution Change



14. Redirect to Flickr



15. Desktop View



16. Mobile View

17. Multi Keyword Search

Figure 14: Testing Screens 10 to 17

Table 7: Contribution Overview 1/2

Part	Person	Percentage
Paper		
Abstract	Abhinav Shankar	50%
	Corneliu Soficu	0%
	Leonard Herold	50%
Introduction	Abhinav Shankar	0%
	Corneliu Soficu	0%
	Leonard Herold	100%
Related Work	Abhinav Shankar	50%
	Corneliu Soficu	0%
	Leonard Herold	50%
Design And Development - Overview	Abhinav Shankar	0%
	Corneliu Soficu	0%
	Leonard Herold	100%
Design And Development - Initial Data Analysis	Abhinav Shankar	0%
	Corneliu Soficu	0%
	Leonard Herold	100%
Design And Development - Flickr Image Downloader	Abhinav Shankar	0%
	Corneliu Soficu	0%
	Leonard Herold	100%
Design And Development - Image Classification	Abhinav Shankar	0%
	Corneliu Soficu	90%
	Leonard Herold	10%
Design And Development - Transform And Index	Abhinav Shankar	0%
	Corneliu Soficu	0%
	Leonard Herold	100%
Design And Development - Search Interface	Abhinav Shankar	70%
	Corneliu Soficu	0%
	Leonard Herold	30%
Design And Development - Validation And Testing	Abhinav Shankar	30%
	Corneliu Soficu	0%
	Leonard Herold	70%
Design And Development - Project Plan	Abhinav Shankar	40%
	Corneliu Soficu	0%
	Leonard Herold	60%
Design And Development - Project Budget	Abhinav Shankar	30%
	Corneliu Soficu	0%
	Leonard Herold	70%
Conclusion	Abhinav Shankar	0%
	Corneliu Soficu	0%
	Leonard Herold	100%
Total	Abhinav Shankar	**23%**
	Corneliu Soficu	**8%**
	Leonard Herold	**69%**

Table 8: Contribution Overview 2/2

Part	**Person**	**Percentage**
Code Parts		
Benchmarks (Effort Factor 0.9)	Abhinav Shankar	0%
	Corneliu Soficu	100%
	Leonard Herold	0%
Initial Data Analysis (Effort Factor 0.9)	Abhinav Shankar	10%
	Corneliu Soficu	0%
	Leonard Herold	90%
Downloader	Abhinav Shankar	0%
	Corneliu Soficu	0%
	Leonard Herold	100%
Classification	Abhinav Shankar	0%
	Corneliu Soficu	80%
	Leonard Herold	20%
Transform And Index	Abhinav Shankar	5%
	Corneliu Soficu	5%
	Leonard Herold	90%
Total	Abhinav Shankar	**3%**
	Corneliu Soficu	**37%**
	Leonard Herold	**60%**
Visualization		
Coding (Effort Factor 0.9)	Abhinav Shankar	12%
	Corneliu Soficu	3%
	Leonard Herold	85%
Deployment (Effort Factor 0.1)	Abhinav Shankar	0%
	Corneliu Soficu	0%
	Leonard Herold	100%
Weighed Total	Abhinav Shankar	**10.8%**
	Corneliu Soficu	**2.7%**
	Leonard Herold	**86.5%**