

Using Self-Organizing Maps to Explore and Query Multimedia Collections

Bachelor Thesis

Natural Science Faculty of the University of Basel
Department of Mathematics and Computer Science
Databases and Information Systems Group

Examiner: Prof. Dr. Heiko Schuldt
Supervisor: Silvan Heller, MSc.

Maurizio Pasquinelli
maurizio.pasquinelli@unibas.ch
17-050-790

July 8, 2020

Acknowledgments

First and foremost, I am using this opportunity to thank Prof. Dr. Schuldt for giving me the possibility to write my Bachelor's Thesis in his research group. I would also like to express my gratitude to Silvan Heller, being a great supervisor who has guided me with his constructive criticism and friendly advice throughout the project work. Furthermore, a huge thank you goes to Mahnaz Amiri Parian for helping out with the deep feature selection. Additionally, all participants of the evaluation as well as Sandro Lutz for proofreading this thesis shall receive a warm thank you.

Lastly, a part of this thesis is currently under review at the Similarity Search and Applications Conference [3].

Abstract

With the ever-growing amount of devices that are capable of recording videos, the task of handling multimedia collections by hand has continuously become harder. Since this also means that such collections are rapidly growing in size, the use of multimedia retrieval systems, like vitivr [9], that are capable of finding textual and visual similar items became increasingly important. Though vitivr provides great support in finding items, it has rather limited support in exploration, so a new way to discover the content of a collection shall be implemented during this Bachelor's Thesis. We approach this challenge by taking advantage of self-organizing maps [6] that will help us sort and group items of a given collection together. Moreover, an opportunity is given to the end user to mark retrieved results as either positive or negative what will then be taken into account during further iterations. Consequently, the introduction of relevance feedback allows to set a focus during such exploration and query tasks improving the overall experience. When the sole implementation of the map functionality is utilized, the allotment of a right spot has turned out to be rather difficult as an easy way to find some related area is missing during initialization. However, evaluation results suggest that it can be seen as a powerful platform when it is used in combination with the existing toolset.

Table of Contents

Acknowledgments	ii
Abstract	iii
1 Introduction	1
1.1 Goals	1
1.2 Contributions	1
1.3 Outline	1
2 Related Work	3
2.1 Multimedia Retrieval	3
2.2 Multimedia Retrieval System Evaluation	4
2.3 Self-Organizing Maps	5
3 Components	10
3.1 vitrivr	10
3.1.1 Cottontail DB	11
3.1.2 Cineast	11
3.1.3 vitrivr-ng	12
3.2 kohonen4j	12
3.3 Content-Based Features	13
4 Implementation	14
4.1 Visualization	15
4.1.1 Relevance Feedback	15
4.1.2 SOM Toolbar	16
4.2 API: Queries and Results	17
4.3 Training	18
5 Evaluation	20
5.1 Setup	20
5.2 Tasks	20
5.3 Contest Realization	21
5.4 Analysis	21
5.5 DRES	22

6 Conclusion and Future Work	23
6.1 Conclusion	23
6.2 Future Work	23
 Bibliography	 25

1

Introduction

The usefulness of some given multimedia collection is not only defined by its searchability, as users should also have the opportunity to gain an idea about what the content is about. For that reason, providing guided exploration should be seen as a necessary feature inside multimedia retrieval systems. This thesis addresses this issue in vitrivr by extending it with different views and algorithms based on self-organizing maps.

1.1 Goals

The ambition of this Bachelor's Thesis is to allow users of vitrivr to explore some given multimedia collection. Guided by relevance feedback they should also be able to steer the exploration and query tasks into the desired content region of their collections.

1.2 Contributions

Allowing users to give feedback on the currently presented items is a first contribution. The second one summarizes the introduction of self-organizing maps inside the vitrivr stack. Whereas the third addition consists of designing and implementing new views, so displaying and navigating through such maps is easily doable. The performed evaluation and its analysis is a fourth contribution. Further enhancements to the vitrivr stack and kohonen4j¹ have been made by reporting bugs and fixing issues.

1.3 Outline

Chapter 2 begins by laying out a motivation for the need of an exploration functionality inside multimedia retrieval systems. Moreover, an outline on the concept of self-organizing maps is given. With Chapter 3, the vitrivr retrieval system is introduced and the required internal changes explained. In Chapter 4, we describe all changes in more detail, focusing on the implementation and also clarifying the interaction of the components. Chapter 5 will

¹ <https://github.com/dashaub/kohonen4j>

analyze the evaluation, followed by the final Chapter 6 giving a conclusion as well as an outlook on future work.

2

Related Work

We first discuss the idea and necessity of multimedia retrieval (systems). For this purpose Section 2.1 follows the introduction given by [8] and [2]. Whereas Section 2.3 is based on [6] and [1] to provide the required theoretical knowledge about self-organizing maps.

2.1 Multimedia Retrieval

Professionals, bloggers or individuals generate daily a rich diversity of media, for instance audios or videos, that is intended to be stored somewhere for future use. As such sets of data are rapidly increasing in size, the active management and handling of those accumulated information sources are suffering from a promptly decreasing usability.

Multimedia collections can only be useful, i.e., provide an increase of knowledge, if we are also able to find relevant items throughout a media store. In other words, if nothing can be found, adding new recordings does not provide any further insights, especially as every new item gets directly lost in the shuffle. Moreover, as a librarian knows where he can find a specific book, i.e., the data we are looking for, we should know what is inside a specific collection, where to find related objects and how to retrieve the desired ones.

This describes exactly the multimedia retrieval problem that consists of three parts: First, finding ways of how to *extract* all relevant information from the source types, second, how to efficiently *store* these insights and lastly, how the objects can be quickly accessed again, i.e., how the look up process should be designed such that any user can easily *retrieve* any desired content out of the system.

The possible areas of application are almost limitless. An individual probably likes to have an option to maintain personal photo galleries, reporters may like to search for relevant newspaper content or doctors would like to find earlier transcripts for better diagnoses as well as sport teams coaches like to provide a more efficient training as specific scene analysis can be provided. This is a little excerpt of the much wider range of application possibilities.

In summary, a full stack multimedia retrieval system provides the abilities to extract, store and retrieve data. Therein, the process of annotating segments needs to be automated as too many inconsistencies would be introduced doing this immense work by hand. However, this is rather tricky, as the context, like time dependencies as well as emotions, can not be recognized that easily. Working on unstructured data, normal queries where exact matches are returned do not work in that context either. We would already need to know the whole data to query for it which is obviously useless. So approaches like k -nearest-neighbors search are used in newly designed database systems to find similar items for some input. Therefore retrieval is becoming more an explorative search among relevant content than a direct search task. Allowing the user at retrieval time to, for instance, input a sketch of a scene and then return the most promising results on which he can add relevance feedback are promising approaches to follow.

2.2 Multimedia Retrieval System Evaluation

Currently, a Video Browser Showdown (VBS)² and a Lifelog Search Challenge (LSC)³ are organized every year.

At VBS a distinction must be made between Ad-hoc Video Search (AVS) and Known-Item Search (KIS) tasks. The first describes the mission where the candidates have to query a multimedia collection based on some generic description given, like *show all cars in front of a house*. The idea is to find as many possible candidates matching the description.

KIS tasks, on the other hand, consist of filtering a single but known item (e.g., because it was previously seen) out of the whole collection. There are also textual KIS tasks where specific descriptions are given which only apply to a specific entry [7].

An other field of research is covered by the Lifelog Search Challenge where Lifeloggers, people with cameras on their bodies, provide long series of images that are combined with divers sensor information. With this change, a whole new query range on metadata of wearable devices can be added to the multimedia retrieval problem. For instance, *show me the taxi I took after having run and been nervous* can be asked as information about position, heart rates, calories consumed and the number of steps done are part of the dataset [4].

² <https://videobrowsershowdown.org/>

³ <http://lsc.dcu.ie/>

2.3 Self-Organizing Maps

A self-organizing map, SOM for short, is a kind of artificial neuronal network. As it organizes some n -dimensional data onto a previously defined map with a lower dimensional topology, it is known for the ability of dimensionality reduction.

Such maps consist of regularly interconnected nodes, either arranged in a line, in a rectangular or hexagonal lattice.

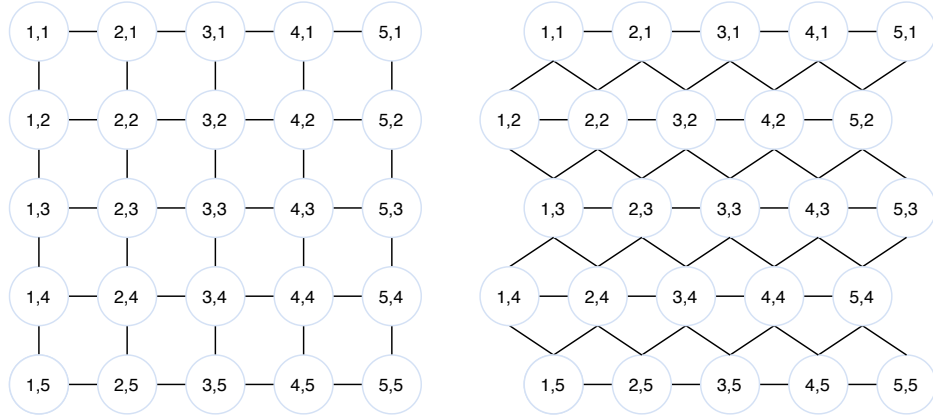


Figure 2.1: Rectangular and hexagonal lattice.

An example of those shapes can be found in Figure 2.1. These are the most commonly used ones, but there is no restriction on them. Therefore, for instance, also cubes can be used beside such 1D or 2D models. The form has to be chosen per purpose, also according to the factor of dimensionality reduction that is desired to be achieved. So let us call the nodes $N_{x,y}$, where x and y define their positions inside a grid G . If G has some size s , then it will consist of $|G| = s^2$ nodes with $G = \{N_{x,y} \mid 1 \leq x, y \leq s\}$.

Each node has exactly one weight:

$$N_{x,y} \mapsto w_{x,y} \quad (2.1)$$

Next, some information about the algorithm should be given. First, all neurons of the chosen map are getting randomly assigned weights, this step is called *initialization*. After having done this, all weights are pointing somewhere in feature space. This means, if we work on some color vectors, the node weights will point somewhere in that color space. Such an example initialization can be found in Figure 2.2, where the node is directly colored by its weight value simplifying the graph.

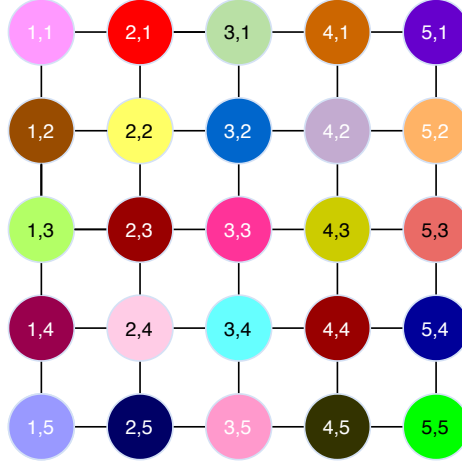


Figure 2.2: Self-organizing map being randomly initialized on training begin.

Having done this, the rest of the process will iteratively be repeated until a fixed number of iterations has been reached. So in a next step, also known as *competition*, for each input vector the best matching unit (BMU) is computed. This is simply the nearest neighbor neuron, more precisely the one with minimal euclidian distance between its node's weight and the input vector.

Formally, we search the node N with a weight being closest to input vector v :

$$\hat{N}_{i,j} = \arg \min_{N_{i,j} \in G} ||v - w_{i,j}|| \quad (2.2)$$

This can be done, as the vectors in the training set have per definition the same dimension as the weights:

$$\dim(v) = \dim(w) \quad (2.3)$$

After having determined to which node our input sample belongs to, the *cooperation* phase takes places. Therein the weights of the BMU as well as of the nodes being in its neighborhood are getting adjusted towards the input vector.

The topological distance between two grid neurons $N_{x,y}$, $N_{i,j}$ is by definition:

$$d(N_{x,y}, N_{i,j}) = |x - i| + |y - j| \quad (2.4)$$

Now, we can compute the neighborhood H :

$$H = \{N_{x,y} \in G \mid d(N_{x,y}, \hat{N}_{i,j}) \leq \sigma\} \quad (2.5)$$

Where σ is a variable, called *neighborhood*, that decreases exponentially over time. Often σ starts with around 3/4 of the longest distance between nodes.

The adjusted weights w' will replace the current weights w in the next iteration:

$$w'_{x,y} = w_{x,y} + \gamma \cdot (v - w_{x,y}) \quad \forall N_{x,y} \in H \quad (2.6)$$

Whereas all nodes outside the neighborhood remain untouched:

$$w'_{x,y} = w_{x,y} \quad \forall N_{x,y} \notin H \quad (2.7)$$

Furthermore, γ is a factor, known as *learning rate*, which defines the impact of the single weight corrections. Typically, γ starts with 0.5 and decreases linearly in time. Would the *neighborhood* and *learning rate* not be decreased during the training process, the map would never settle down as almost all neuron weights would always be moved towards the current input vector again.

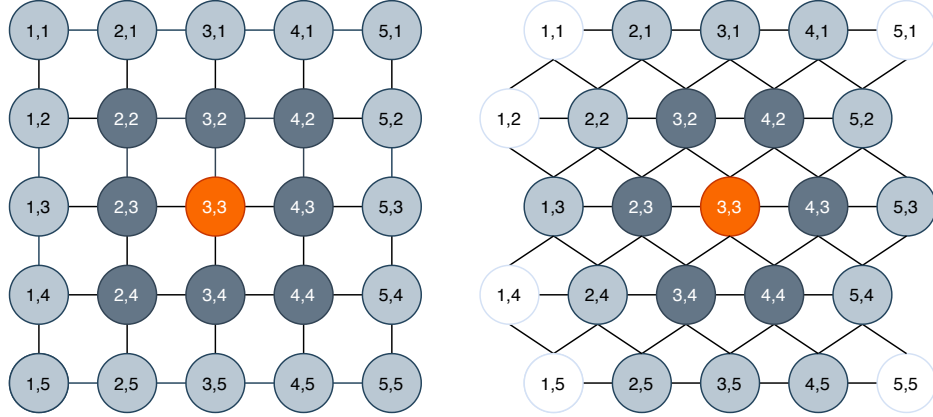


Figure 2.3: BMU and its neighborhood.

As can be seen in Figure 2.3, the *neighborhood* determines the area size around the BMU. Being equal to 1, only the direct neighbors (dark gray) are part of it. Would it be equal to two, also the light gray nodes are part of the node's neighborhood.

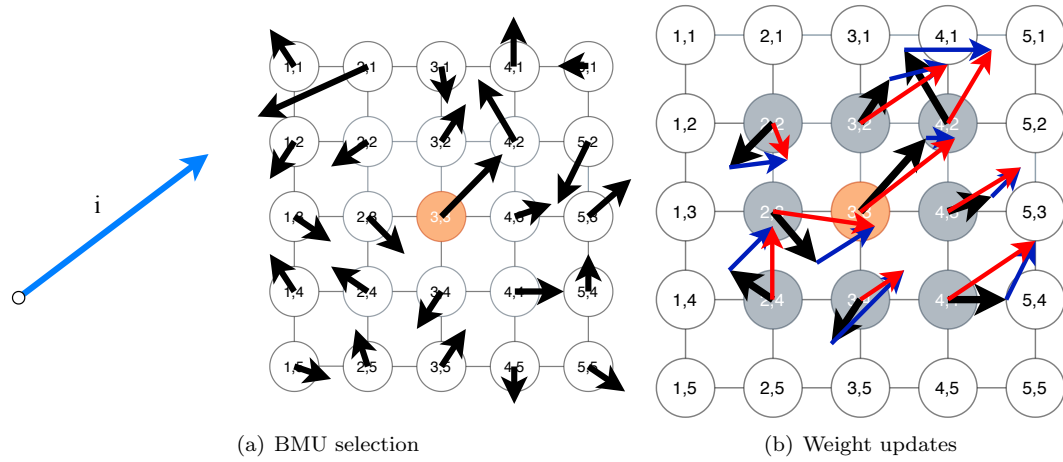


Figure 2.4: Update step (vector version).

In Figure 2.4(a) the BMU (in orange) was chosen as its vector was the closest among all node weights (in black) to the input vector (in blue). Next, in Figure 2.4(b) we see that all black vectors in the neighborhood will be moved towards the input vector i . The applied differences, scaled by the factor *learning rate*, are visible in blue. The red vectors represent the updated weights w' that will be used in the next iteration.

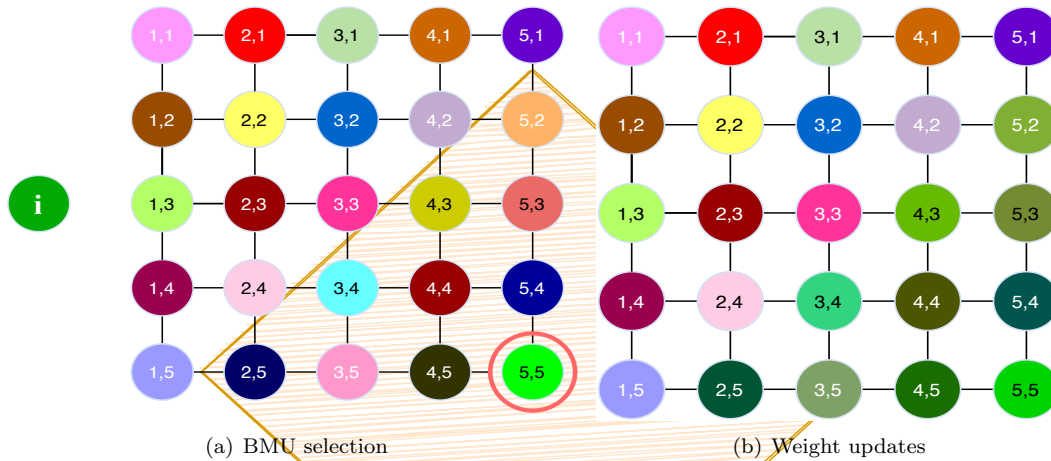


Figure 2.5: Update step (color version).

We can visualize this update procedure in a simplified version, like it was done in Figure 2.2 with colors. So Figure 2.5(a) shows the BMU selection with its neighborhood, whereas Figure 2.5(b) shows the weights after the first iteration.

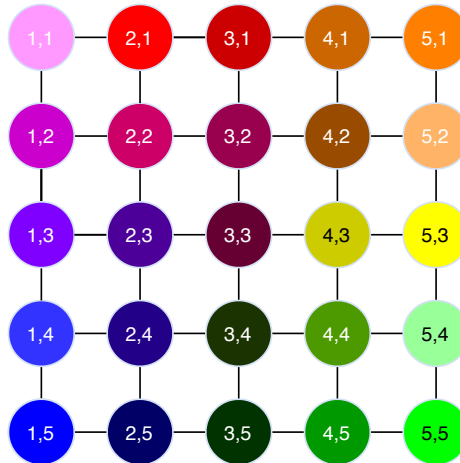


Figure 2.6: A self-organizing map results sorted after training.

Through such a training process, the neuron weights are getting ordered along the axes. Although the nodes would not change their positions inside the grid, if the inputs would be some color vectors, then we would, for instance, have red on one map's end and blue on the other. Figure 2.6 shows an example result.

Such an update formula is the reason for its competitive self-organization, compared to other learning algorithms that, for example, rely on backpropagation with gradient descent. Moreover, as the grid size and therefore the number of nodes is related to the amount of identified clusters, this parameter can be used for fine tuning the algorithm. A larger number will result in a more fine grained map, where a lower one will do some more abstraction and generalization.

Important to mention and what sometimes can lead to confusion is that the selected topological map layout, i.e., the positions of the nodes, as well as the number of neurons remain fixed throughout the entire process. The only thing that changes is the neuron's weight, but not their positions nor interconnections inside the map.

In the context of Machine Learning, self-organizing maps belong to competitive and unsupervised training mechanisms. It is unsupervised as the network learns to classify training data without external help, i.e., the input data is not labeled. It is competitive as in each round from all neurons only one wins over all others, more exactly gets activated.

3

Components

This chapter presents the multimedia retrieval system vitrivr, based on [9]. So the SOM related changes that were introduced as part of this thesis can be described in context to their corresponding modules. Further, we explain the integration of a library as well as the addition of deep features.

3.1 vitrivr

First, a brief overview of all components is given. Figure 3.1 shows three main units, i.e., database, retrieval engine and an user interface, including their interconnections forming the whole stack.

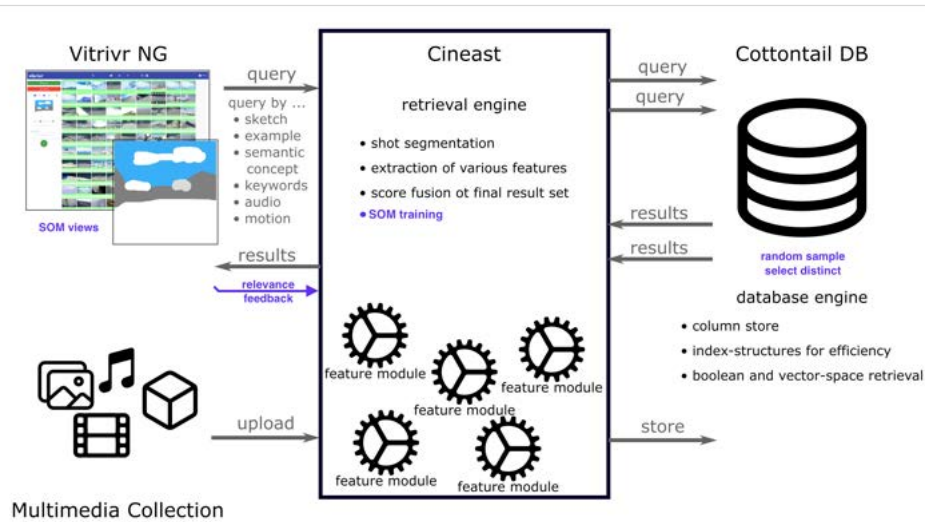


Figure 3.1: vitrivr stack, adopted and amended from [4].

In the following subsections the functions of these units will be explained and related additions, written in violet inside Figure 3.1, discussed.

An overview of the modifications only can be seen in Figure 3.2.

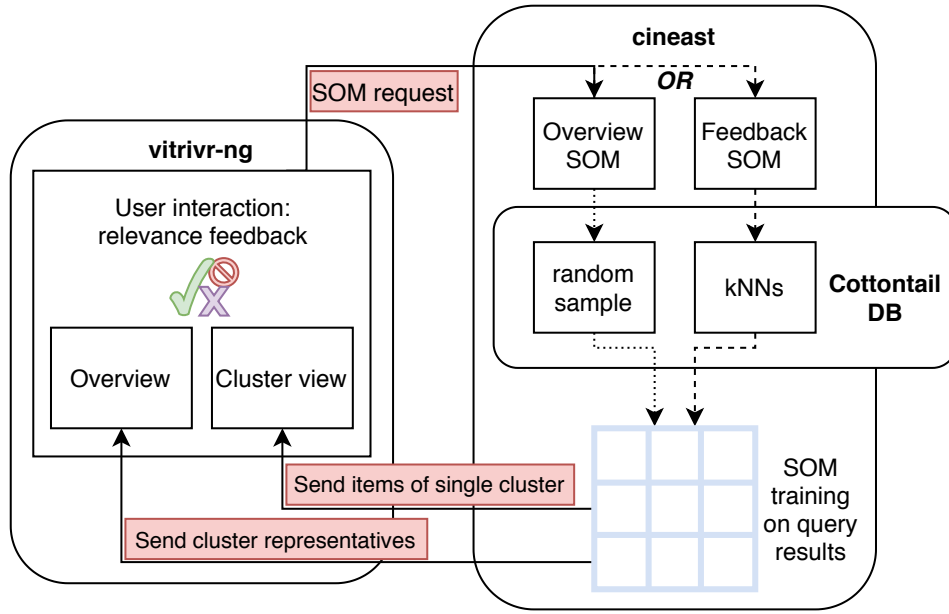


Figure 3.2: Enhanced vitivr modules.

3.1.1 Cottontail DB

Cottontail DB⁴ is the storage layer of the vitivr stack. It serves as the place where all data of media extraction are being stored. It is a column store that offers essential features like classical boolean and text retrieval as well as more complex operations inside a vector space, for instance, the k-nearest-neighbours lookups used for queries requesting similar items.

As the SOM addition should provide some exploration functionality, a way was required to get a random sample out of a given collection that in turn can be visualized with the help of self-organizing maps. After having a discussion with the maintainer of the database project, an *EntitySampleTask* was introduced inside Cottontail DB allowing to query for random entries. This execution task was therefore programmed to return a record set of given size where the tuples have been selected based on randomly generated identifiers.

3.1.2 Cineast

In the middle of all components lays the core part of vitivr, Cineast⁵. Written in Java, it can fill the underlying database (Cottontail DB) by extracting features from a given multimedia collection consisting of audio, videos or images. Also 3D models are supported. During such an extraction phase, as can be seen at the bottom side of Figure 3.1, a variety of descriptors, i.e., feature vectors, are being generated and stored in the database for different future retrieval purposes. The possibilities listed on the upper side of Figure 3.1 range from example or sketch-based to motion queries. Therefore information about edges and colors, for instance, are used to find similar items.

⁴ <https://github.com/vitrivr/cottontaildb>

⁵ <https://github.com/vitrivr/cineast>

As this central unit has access to the database, i.e., is responsible for doing kNN queries, the SOM training - a computation process generating a self-organizing map - is placed here. Once we have calculated the map, representative items, e.g., for the cluster view, can be passed a layer up towards the user interface.

Furthermore, we would like to point out that we first experienced low performance doing sequential kNN requests, so we now make use of batched kNN queries where all related requests are sent together to the database.

3.1.3 vitrivr-ng

vitrivr-ng⁶ is the last and uppermost part of the vitrivr multimedia retrieval stack. Browsing a multimedia collection to finding similar items can either be done by sketch, example or keywords. There are also further options like tag search. When such a task is initiated, vitrivr-ng will send queries towards Cineast, continuously processing the results so that even if the query involves multiple categories of feature sets, the user sees results as they come in. These partial results will induce an update of the total score for each element, such that the current best findings are always at the beginning of the result list.

Concerning self-organizing maps, two views have been added. One to display the whole grid, i.e., the *overview* with one segment per cluster (the cluster representatives), and one for listing all segments belonging to a specific cluster, called *cluster view* respectively.

3.2 kohonen4j

To achieve the goal of content exploration and queries by similar segments, we need a way to group related items together, i.e., find clusters throughout the whole feature space. Having such clusters, a user can navigate therein, more specifically, inspect a set of similar items. Moreover, this also creates the possibility to look on all clusters simultaneously by just considering one representative item per cluster and displaying these next to each other.

As multimedia content is highly complex, the features extracted from the sources are often high dimensional vectors. These can represent things like color distributions, occurring shapes in the scene, objects with their orientation and much more details.

But as such high-dimensional content, i.e., the computed clusters, can not simply be arranged on a display, it has first to be projected onto a smaller dimension like a two dimensional grid, so that we are able to visualize all clusters in a simple and ordered way. This is where the concept of self-organizing maps comes in.

For this thesis, we chose to go with a simple two dimensional grid approach as it fits well in the context of displaying segments on a screen and as it provides enough abstraction

⁶ <https://github.com/vitrivr/vitrivr-ng>

from the complexity of any feature space. Looking for a Java implementation, doing the SOM training as described in Section 2.3, we found `kohonen4j` and decided to use it. But after having played around with some vectors out of our collection, quickly revealed that something is going wrong in this tool as almost always only one map node got all items assigned. As we still believed that it must be a good library but just have a bug somewhere, we started to debug until the cause was found. The *learning rate* as well as the *neighborhood* were reduced to quickly so they become negative quite soon, so the training process resulted in some undefined behavior. Furthermore, we were able to reduce the required training time drastically by optimizing some memory accesses as well as merging some for loops over all segments together into one. These and further improvements are now pending in the origin repository as pull request.

3.3 Content-Based Features

In fact, end users would primarily like to get comparable items based on their content similarity (for instance, get some more dogs or cars), not only on their visual resemblance, e.g., through features describing their average color. To have the possibility to do the former type of query, some further extraction had to be performed on the V3C1 video dataset [10]. For such a deep feature extraction, the choice has fallen onto the MobileNet V1 model⁷, which is an implementation of Efficient Convolutional Neural Networks for Mobile Vision Applications [5]. As the input images to this neuronal network are required to be small squares, such rectangles have been cut out of the middle of the sampled segments first, for then being resized to 192x192 pixels. Each of these input frames could thereby be converted to a 512 dimensional vector. After this whole process, the gained vector set can provide the ability for content based exploration and retrieval inside vitivr.

⁷ https://tfhub.dev/google/imagenet/mobilenet_v1_050_192/quantops/feature_vector/3

4

Implementation

In any kind of interaction with the central part of this thesis, namely the SOM training process, a set of segments has to be handed over to the SOM training engine so that a new map can be generated on them.

As can be seen in Figure 4.1, there are two ways for contributing such a set. First, by providing a random sample of segments, preferably good representatives for the whole feature space, so that the resulting map can give an overview of what is inside the whole collection as the ranges become visible. Second, by submitting segments from a sub-selection of the feature space only, so the map will result in a closer look on this area. To get such a focused map, a user has to give feedback on segments that are currently displayed first. Such relevance feedback can be added to any segment, i.e., it does not matter if the segments were found inside a previously generated SOM or by any other type of executed query (like by text, tags or semantic sketches).

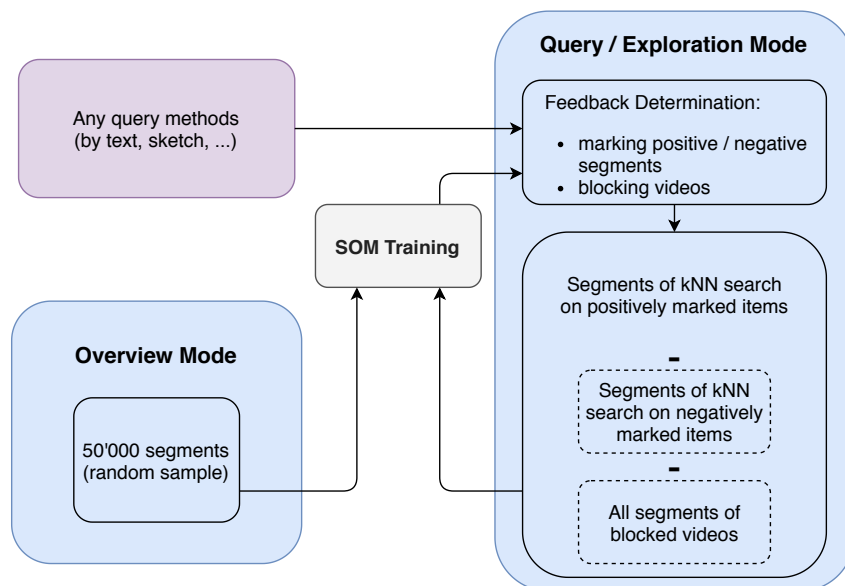


Figure 4.1: Possible interaction modes with the SOM training engine.

4.1 Visualization

As the visible front end of a retrieval system is the major part a normal user, i.e., not a developer, will interact with, it was important to make the new feedback and exploration functionalities easily accessible and controllable therein. For that reason, many different views like toolbars and labels have been introduced.

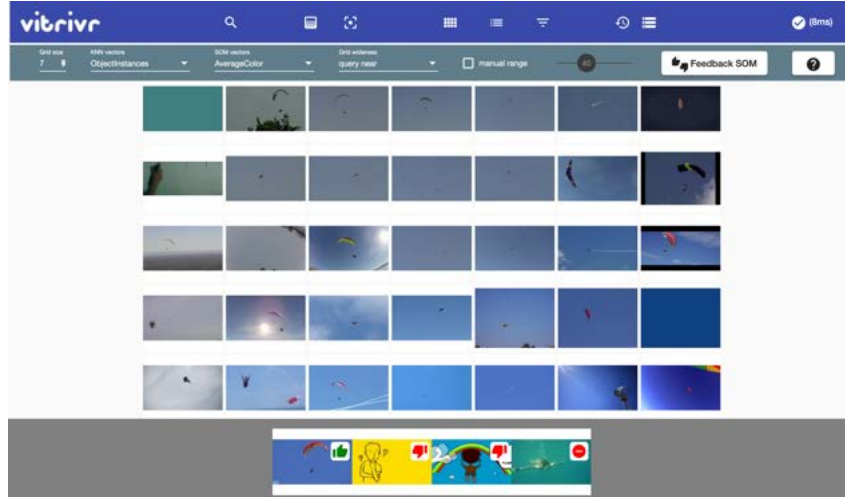


Figure 4.2: New feedback and SOM interface in vitivr-ng.

4.1.1 Relevance Feedback

Marking segments should be fast and simple to accomplish and the current feedback should be easily readable from a segment. The user can give his opinion about the relevance of a segment by hovering it, see Figure 4.3(a). Both buttons for *positive* (green, thumb up) as well as for *negative* feedback (red, thumb down) can simply be clicked with the cursor already being above the current segment. A video can also be completely *blocked* from further trainings by right clicking the negative feedback button. After the choice was made, a corresponding icon is visible on the top right of the image. All variants of such labeled segments can be seen in Figure 4.3(b). Changing feedback is simple too, hover and select the new choice. If a feedback should be removed completely, a user can press the currently active button, e.g., if a segment is positively marked, press the green thumb again, and the feedback will be removed. To always have the overview of all items being currently marked, a bottom toolbar, visible in Figure 4.2, containing all those segments has been added.



Figure 4.3: Feedback selection.

4.1.2 SOM Toolbar

Additionally, a new toolbar at the top is added containing a variety of SOM settings, i.e., its size, a selection for the feature set the nearest neighbors should be queried from such as for the feature defining on which vector set the SOM should be train on. Furthermore, the query wideness (also called *range*) can be defined by choosing one of the available modes, i.e., query close/near, exploration little/wide. Most importantly, also a train button that initiates the process of a new map generation and visualization is included there. Examples demonstrating this component's behavior can be found in Figure 4.4.

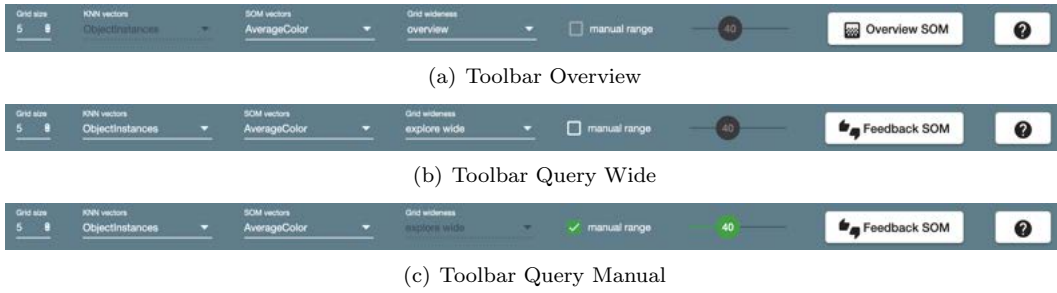


Figure 4.4: SOM toolbar in vitrivr-ng.

Having chosen the overview mode, the selection of a kNN feature set is disabled, as only a random sample of vectors from the SOM feature set will be used for training. See Figure 4.4(a). If any other mode is selected, both selections are activated, see Figure 4.4(b). In order to start a training iteration with such a setting, at least one segment has to be marked positive first. Otherwise the training set of vectors is empty, preventing a new map generation. The range can also be set manually, as done in Figure 4.4(c), which defines how many segments should be fetched totally through the positive kNN queries (the slider number has to be multiplied by a thousand).

The last button in the toolbar will open a help dialog (visible in Figure 4.5), showing some shortcuts to facilitate the interaction. For example, switching between modes, clearing all given feedback or simply start a new training can be achieved by pressing a single key.

Shortcuts

Fundamental	Grid size	Wideness
<i>T</i> → <i>train</i>	<i>S</i> → <i>small</i> (5)	<i>E</i> → <i>query exact</i>
<i>Q</i> → <i>quit cluster</i>	<i>D</i> → <i>medium</i> (10)	<i>N</i> → <i>query near</i>
<i>arrow keys</i> → <i>cluster walk</i>	<i>X</i> → <i>extra large</i> (20)	<i>L</i> → <i>explore little</i>
		<i>W</i> → <i>explore wide</i>
		<i>O</i> → <i>overview</i>
Feedback <i>C</i> → <i>clear / reset</i>		

Figure 4.5: Help dialog.

4.2 API: Queries and Results

As the frontend needs to communicate with the underlying part of the retrieval stack, i.e., Cineast, new query types have been introduced. The most important new query type, namely the *SomTrainQuery*, contains all information about the requested grid size, the name of feature set it should train on and how big the considered neighborhood of the positive selected frames should be. Complementing the last parameter, the query also contains lists of all positively and negatively marked as well as blocked segments, which Cineast will then use as a basis for training the new self-organizing map. An example request can be found in Listing 4.1.

Listing 4.1: Snapshot of a *SomTrainQuery* request.

```

1 {
2   "size": 5,
3   "retriever_knn": "ObjectInstances",
4   "retriever_som": "AverageColor",
5   "deepness": 4,
6   "positives": ["v_04358_273", "v_05937_115"],
7   "negatives": ["v_04303_31"],
8   "blacklist": ["v_00049_242"],
9   "config": {
10    "queryId": null,
11    "hints": []
12  },
13   "messageType": "Q_SOM_TRAIN"
14 }
```

Using preset query ranges, i.e., not using the manual range feature, the considered neighborhood will be defined based on the selected mode, here *QueryMode*, as well as the cardinality of the positive segment set P . More precisely, the range r will be defined as follows:

$$r(P, QueryMode) = \begin{cases} \min(|P|, 6), & \text{if } QueryMode = QueryClose \\ \min(2 \cdot |P|, 14), & \text{if } QueryMode = QueryNear \\ \min(5 \cdot |P|, 20), & \text{if } QueryMode = ExplorationLittle \\ \min(10 \cdot |P|, 40), & \text{if } QueryMode = ExplorationWide \\ -1, & \text{if } QueryMode = Overview \end{cases}$$

Using $r = -1$ when being in overview mode, allows us to tell the back end to create a random sample instead of doing a kNN query. More precisely, just passing $r = 50$, as r is a factor thousand smaller, would not have been a solution, as the back end could not distinguish between a kNN request or the random sample of size 50'000.

The neighborhood wideness has to be passed along with the request, so the resulting map will either stick around the positive elements providing a better query possibility or it will cover a wider section around these to provide an exploration opportunity to the user.

When it comes to send the resulting segments back to the front end, *MediaSegmentQueryResult* messages are used. As all already existing query types use these in responses too, we are not going to explain that result type in more detail here. But until now, all incoming segments were treated the same way. They were just appended to all views, e.g. list and media gallery. Without changing this, any resulting segments would have been added to the normal and to the SOM related views as they could not have been distinguished from each other. To correct this behavior, some information about segment routing had to be provided. For this, a *type* field was added to these segment result messages. Either it is set to *default* which means that it is appended to all original views or it has the value of *som_overview* respectively *som_cluster* where it will be appended to that corresponding view only. As all frames are transferred in the same order as the nodes of the map are arranged, the SOM overview automatically shows the map in the correct orientation.

4.3 Training

The requests reach Cineast through a web socket. Cineast is the service between the front end and the database. The received request contains:

- grid size s
- kNN retriever R_{knn}
- SOM retriever R_{som}
- query range r
- list of positively marked segments P
- list of negatively marked segments N

- list of blocked segments B_s

A query function Q with retriever R_{knn} maps to a set S containing the k nearest neighbors for some segments s :

$$Q_{R_{knn}} : s, k \mapsto S \text{ where } k \in \mathbb{N} \text{ and } |S| = k \quad (4.1)$$

In a first step, P is processed. In other words, for each $p \in P$ a nearest neighbor search using $Q_{R_{knn}}$ with parameter $k = \frac{r \cdot \alpha}{|P|}$ is performed, where the default range α is set to 1'000. Let the union of all result sets be called R , as it contains relevant segments only.

$$R = \bigcup_{p \in P} Q_{R_{knn}} \left(p, \frac{r \cdot \alpha}{|P|} \right) \quad (4.2)$$

Afterwards, the same is done for every $n \in N$ but with fixed $k = \alpha$. Analogously, let the union of all irrelevant result sets be called I .

$$I = \bigcup_{n \in N} Q_{R_{knn}}(n, \alpha) \quad (4.3)$$

The function V maps a segment s to the video v it was extracted from:

$$V : s \mapsto v \quad (4.4)$$

As not only the single segments of B_s should be blocked but the whole videos they are in, B_v is created:

$$B_v = \{V(s) \mid s \in B_s\} \quad (4.5)$$

The final batch of frames used for training is now a subtraction of the result sets, i.e., the positive minus the negative nearest neighbors, filtered against the blocked video set which is more formally:

$$T = \{s \in R \setminus I \mid V(s) \notin B_v\} \quad (4.6)$$

But if the parsed message is an overview request, the training set is defined differently. As already mentioned, in this case a random sample of the whole video collection C , the overall set of extracted segments, is queried from Cottontail DB:

$$T \subseteq C \text{ and } |T| = 50'000 \quad (4.7)$$

For all the segments in T we now need to query their vectors, on which the self-organizing map can be trained next. The second query function Q with retriever R_{som} maps a segment s to a feature vector v :

$$Q_{R_{som}} : s \mapsto v \quad (4.8)$$

So the final set of training vectors that will be passed to the SOM training engine is V :

$$V = \bigcup_{s \in T} Q_{R_{som}}(s) \quad (4.9)$$

5

Evaluation

Lastly, a VBS-like contest has been organized to get an impression on how this new functionality performs in contrast to the already existing retrieval methods.

5.1 Setup

To be able to do so, six full stack vitivr instances have been set up. Four of them being checked out at the state of this thesis progress and the two remaining on the state of vitivr/dev, more specifically, without having all the SOM related additions. Moreover, a Distributed Retrieval Evaluation Server⁸, for short DRES, had to be set up such that during tasks the participants can submit all promising findings whereas the judges can approve or reject these items. Through DRES also all tasks as well as the participants such as their groupings are being managed.

5.2 Tasks

For this evaluation four textual KIS and five visual KIS tasks are used. The former mentioned have to be solved within 8 minutes whereas for the latter type of task only 5 minutes are intended. Furthermore, there are four Ad-hoc Video Search (AVS) tasks during 5 minutes too. The clips provided in the visual Known-Item Search tasks have a duration of approximately 20 seconds. All these tasks were picked out of this year's VBS (2020) event task set. One of these KIS task views can be found in Figure 5.1(a), further Figure 5.1(b) shows a running AVS task. For KIS tasks the segments submitted to DRES are evaluated automatically. In AVS context, on the other hand, each submission has to be rated by judges either as *correct* or *incorrect*. As can be seen in Figure 5.1(c), there is also an option *unsure*, for the case they could not conclusively state the presence of a frame around the submitted time that fully matches the description. This last type of verdict doesn't affect the score at all.

⁸ <https://github.com/lucaro/DRES>

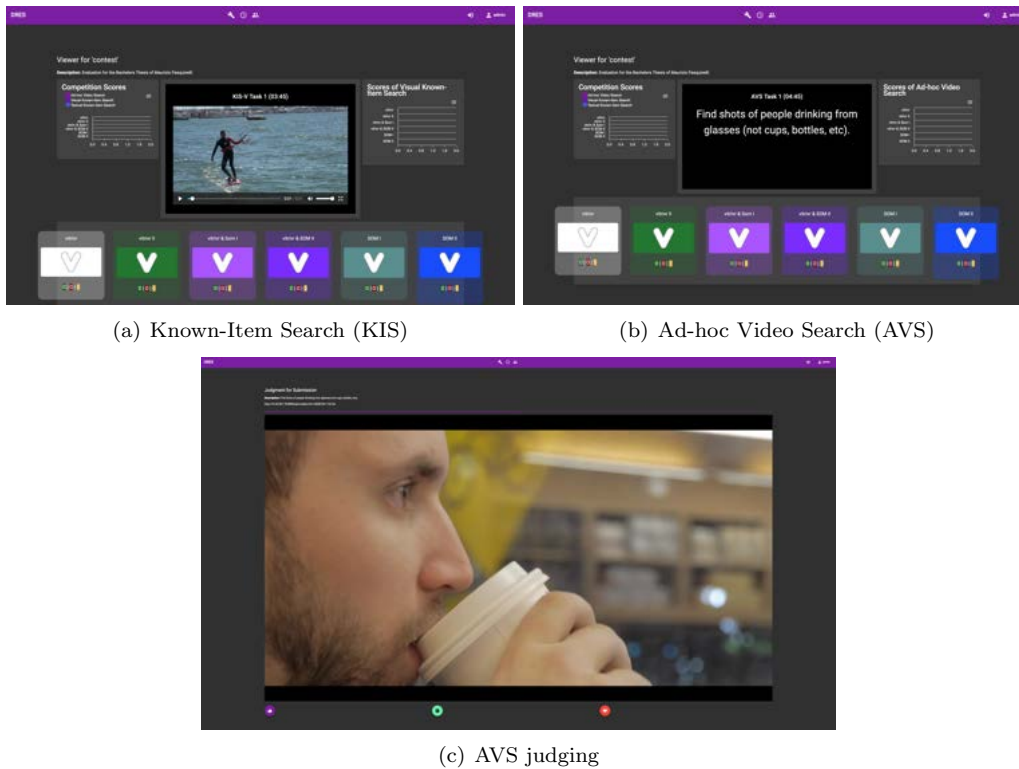


Figure 5.1: DRES interface.

5.3 Contest Realization

On a day where enough test participants could join, the contest was held. As social distancing has become very important during these days, we could not hold the session together in a room as planned. Due to these circumstances, everything had to be moved online and coordinated through a video conference call. Apart of the coordinator, two judges as well as 2 SOM only users, 2 combo players and 2 vitivr only users joined the contest.

Though the judges have attended the VBS20 iteration, the other participants have not. Moreover, some of them were already experienced in using vitivr. For those who did not know, as they have never come into contact with it, an introduction was given beforehand.

5.4 Analysis

The SOM variant seems to be quite successful once it's getting trained around the right context, i.e., the region in feature space. So one main problem seems to be the missing opportunity for initialization. E.g., in tasks where a specific text is mentioned, users with text search can quickly find it whereas for the SOM it's almost an unsolvable task. But again, once items being similar to the desired one are appearing on the map, it's likely to find more related content by further filtering, looking through and perhaps some further trainings with closer items, i.e., with new and improved feedback. Fig. 5.2 shows AVS scores of the contest. We see that SOM only users were either not able to find anything or they had competitive or even better results than the vitivr only and combo users.

Task	Metric	Teams		
		vitivr	combination	vitivr-explore
AVS 1	precision	0.5	0.85	-
	correct	9	12	0
	unique	7	10	0
	score	35	45	0
AVS 2	precision	0.82	0.73	0.8
	correct	6	6	6
	unique	3	3	4
	score	45	45	42
AVS 3	precision	0.86	0.85	0.9
	correct	5	6	7
	unique	3	4	6
	score	47	48	49
AVS 4	precision	0.71	0.86	0.5
	correct	15	9	2
	unique	13	6	1
	score	41	47	33

Figure 5.2: AVS scores, adopted from [3] (vitivr-explore corresponds to SOM only users).

Further I want to mention my observations about users that were not familiar with the use of self-organizing maps. For instance, they often did not even leave the overview screen and started the training, mostly without big changes in the given feedback, again. But most segments will only be found through the inspection of the single clusters as they contain many items. Also worthy to mention, is that it is quite useful to know *what* the selected vectors really represent. So by which shapes or parameters of a picture the similarity of the vectors is influenced the most. For instance, in a task where the participants had to find seven people from a wedding in a shot, they only marked pictures containing one or two people as positive. Throughout the iterations, they then tried to restrict to people wearing white clothes only. During implementation and testing, I got used to these vectors, so that I would say they mainly take care of the number of visible people and their orientation (from profile or side, for instance) if they are describing pictures containing humans at all. For that reason, I would have tried to first select pictures with more people (~7) and then, for instance, request a map sorted on the colors for faster elimination. But again, finding some text will remain a hard problem as the available vectors are not trained for that.

5.5 DRES

The logging functionality of DRES collects some basic information like submitted frames and their scores. To get an idea on, for instance, how many unique videos were found, manual computation and filtering had to be performed. Furthermore, it would be really helpful to know if some submitted segments were wrong due to the submitted time frame only. So which user submitted the right video but didn't find the exact timed shot. As often in a video, there are very similar occurring scenes but only one of them being the correct one. It could also be nice to know about users that got the correct frames in their query results but didn't see them.

6

Conclusion and Future Work

6.1 Conclusion

As some sort of exploration functionality was not part of vitrivr until now, the introduced additions can be seen as a good complement for the whole retrieval stack. Finding similar items with the help of self-organizing maps works quite well. Though looking for a specific item has emerged to be quite tricky to impossible without having a good initialization. A direction is needed, otherwise we probably end up lost.

6.2 Future Work

The realized work can be seen as a base for a wide range of improvements.

Cluster Sort A possible extension could consist of allowing to sort the single clusters again. If a cluster contains like a thousand items, this would be quite useful. And just marking one of the cluster segments as positive and then request a new SOM based on that will not result in the same set of segments. Extending this strategy, we could end up in something like a hierarchical tree of self-organizing maps.

Vector Merge As sometimes a trained map shows totally empty clusters or such containing just a few items, we could think about a strategy to fill these gaps, e.g., by doing a kNN on an averaged vector of all the neighboring cells as most probably some content is being in between there. Thinking about some sort of vector merge could be promising too. Because giving positive feedback on, for instance, a dog as well as a human picture will probably not result in showing pictures where both items (dog and human) are present, although that is most probably what the user wants to achieve with this selection. As the nearest neighbors of a human are still humans and analogically for the dog, the map will again get trained on segments containing only one of the desired items.

Any Query SOM Something that could really improve the experience and especially the usefulness, would be to allow normal query results being displayed as a self-organizing map. For example, after having done some tag based search, the results are not just

listed as normally but given as an input to the SOM training engine. As the SOM needs some vectors to train on and, for instance, tags are not vectors per se, we could simply fetch like the color vectors or some deep features on the query result set and train on these. Moreover, we could try produce a new vector on the fly where each value is a module score, given that multiple modules were queried.

Temporal Context View Some further improvement that is not necessarily restricted to the SOM views would be to introduce some temporal context view when the user hovers over an item. So there is no need to open a video, check the past and the future of the shot and to close it again. Often this is just too much work, i.e., too time consuming, so that if it's not exactly what we are looking for, we omit it although it would have been close.

Bibliography

- [1] John A. Bullinaria. Self Organizing Maps: Fundamentals. <https://www.cs.bham.ac.uk/~jxb/NN/116.pdf>, 2004. [Online; accessed 06-July-2020].
- [2] Ivan Giangreco. Database support for large-scale multimedia retrieval. <https://edoc.unibas.ch/64751/>, 2018. [Online; accessed 06-July-2020].
- [3] Silvan Heller, Mahnaz Amiri Parian, Maurizio Pasquinelli, and Heiko Schuldt. vitrivr-explore: Guided multimedia collection exploration for ad-hoc video search. SISAP '20. Under Review.
- [4] Silvan Heller, Mahnaz Parian, Ralph Gasser, Loris Sauter, and Heiko Schuldt. Interactive lifelog retrieval with vitrivr. pages 1–6, 06 2020. doi: 10.1145/3379172.3391715.
- [5] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. URL <https://arxiv.org/abs/1704.04861>.
- [6] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.
- [7] J. Lokoč, W. Bailer, K. Schoeffmann, B. Muenzer, and G. Awad. On influential trends in interactive video retrieval: Video browser showdown 2015–2017. *IEEE Transactions on Multimedia*, 20(12):3361–3376, 2018.
- [8] Luca Rossetto. Multi-modal video retrieval. <https://edoc.unibas.ch/66289/>, 2018. [Online; accessed 06-July-2020].
- [9] Luca Rossetto, Ivan Giangreco, Claudiu Tanase, and Heiko Schuldt. Vitrivr: A flexible retrieval stack supporting multiple query modes for searching in multimedia collections. In *Proceedings of the 24th ACM International Conference on Multimedia*, MM '16, page 1183–1186, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450336031. doi: 10.1145/2964284.2973797. URL <https://doi.org/10.1145/2964284.2973797>.
- [10] Luca Rossetto, Heiko Schuldt, George Awad, and Asad A. Butt. V3C – A Research Video Collection. In Ioannis Kompatsiaris, Benoit Huet, Vasileios Mezaris, Cathal Gurrin, Wen-Huang Cheng, and Stefanos Vrochidis, editors, *MultiMedia Modeling*, pages 349–360, Cham, 2019. Springer International Publishing. ISBN 978-3-030-05710-7.

Declaration on Scientific Integrity

Erklärung zur wissenschaftlichen Redlichkeit

includes Declaration on Plagiarism and Fraud
beinhaltet Erklärung zu Plagiat und Betrug

Author — Autor

Maurizio Pasquinelli

Matriculation number — Matrikelnummer

17-050-790

Title of work — Titel der Arbeit

Using Self-Organizing Maps to Explore and Query Multimedia Collections

Type of work — Typ der Arbeit

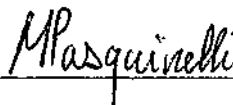
Bachelor Thesis

Declaration — Erklärung

I hereby declare that this submission is my own work and that I have fully acknowledged the assistance received in completing this work and that it contains no material that has not been formally acknowledged. I have mentioned all source materials used and have cited these in accordance with recognized scientific rules.

Hiermit erkläre ich, dass mir bei der Abfassung dieser Arbeit nur die darin angegebene Hilfe zuteil wurde und dass ich sie nur mit den in der Arbeit angegebenen Hilfsmitteln verfasst habe. Ich habe sämtliche verwendeten Quellen erwähnt und gemäss anerkannten wissenschaftlichen Regeln zitiert.

Basel, July 8, 2020

A handwritten signature in black ink, appearing to read 'M Pasquinelli', is written over a horizontal line.

Signature — Unterschrift