Final Report

Optional Semester Project
College of Humanities - DHLAB
Supervised by M. Ehrmann, S. Ares Oliveira and Prof. F. Kaplan
8 ECTS

Newspaper Image Classification

Automatic Labeling of Images in the Impresso Database

Edoardo Tarek Hölzl (Data Science)
Lucas Antoine Gauchoux (Data Science)

Spring Semester 2019

Summary

1 Introduction

1.1 Impresso project

"Impresso: Media Monitoring of the Past" [7] is an ambitious project which aims at processing 200 years of newspapers. Historical newspapers are a massive source of information. Traditional newspapers cover the actuality from an international to a local level, and on every aspect of society. Moreover their periodic publications offer the opportunity to keep track of the evolution of events, political current, urban planning, etc. This temporal aspect of the project could be very valuable for historians.

The goal of the project is therefore to make this data accessible, understandable and exploitable. The first step was to process the data. Traditional newspapers are gradually disappearing for their digital versions. The latter makes preservation and information access easier. How to make the information in the printed version accessible as well? Digitization. Technology nowadays allows us to create digital documents from books, newspapers and even handwriting documents.
Having a digital version of newspaper archive makes keyword search possible over multiple decades, however looking for one word in 200 years of archive won't help you understanding context at a given time and will most likely not give you the information you were looking for.
The goal of the Impresso project is to enable critical text mining of newspaper archives.

Its main objectives to achieve this goal are to develop:

- A Historical media monitoring tool suite
  - multilingual and time-specific text mining techniques to transform noisy and unstructured textual content into semantically indexed, structured and linked data
  - fully traceable and inter-operable historical semantic knowledge base
- Visualisation interfaces and visual analytic for content exploration
  - visualisation interface to enable the seamless exploration of vast amounts of complex historical data
  - novel functionalities to accommodate text analysis research tools and empower users to approach the system in a reflective way
- Digital History
  - study the usage and impact of developed tools
  - gives rise to questions on biases introduced by digitization

1.2 Goals & objectives

As explained before, all the newspapers archives were digitized and pages were segmented, namely extracting text block, tables, images, etc...

There are numerous images in the 200 years archives, around 200'000 for one newspaper.

During the processing, image metadata have been created. The OCR process recognized pictures in scans of facsimiles and when possible , linked them to articles and caption, however this is usually not the case and image search is not possible. If you are looking for a specific image of an event present in the archive you could look for an article that references this event and then look for the references of this article, or a similar article inside an image metadata. Another tool has also been developed, that allows for searching images by visual similarities. This work, called Replica [**replica**], uses state-of-the-art machine learning to retrieve images with analogous appearance and therefor allow users to make image-based image retrieval.

The goal of our project was to experiment automatic image classification/tagging and to investigate if we could augment this metadata to allow text-based image retrieval. Impresso users would thus be able to query directly the images by their content, e.g. getting all maps images between 1993 and 1995. This method has already been experimented by the Bibliothèque Nationale de France (BNF) in their project "Image Retrieval in Digital Libraries - A Multicollection Experimentation of Machine Learning techniques"**??**.

The work was separated in several milestones:

- Evaluation of existing pre-trained models and API
- Pipeline prototyping
- Effectively run the enrichment process on the whole Impresso collection
- Evaluation of the enrichment process

The main concern at the beginning was to evaluate how existing pre-trained models performed with scans of historical images that are, moreover, distributed over 200 years, resulting in a considerable variability in terms of image quality and content. Indeed available labeled data-sets mainly contain high resolution images, which makes sense because almost all applications of object detection and image classification are used on those kind of images and are thus trained accordingly. The goal of this first milestone was to determine if it would be necessary to retrain a model, which would require to create our own labeled data, or if current state of the art model would be sufficient.

The performance evaluation for this prototyping part is done by visually checking a random batch of images in the data-set, i.e. it's a qualitative evaluation.

Once the model has been chosen we run the classification on the Impresso image collection and enrich its metadata with the extracted labels.

The final part of the work consists in evaluating the confidence level of this labeling.

2 Data overview

Before diving into the classification and architecture of our pipeline, we first need to understand the origin and visualize the images, in order to determine what possible labels we could have, and hence guide us into choosing an appropriate model. As mentioned above, the images were extracted from a daily newspaper, over a period of 200 years. With that in mind, it gives us a broad idea on the wide variety of images and photographs present in the database, with also different levels of quality (since they are scans of physical paper). Examples of such images are shown in Figure 1. As we can see, all of them are in black and white, with different levels of granulation and quality.
Also, the scrapped images are not only photographs, but consist also of doodles, weather maps, weekly board games (sudoku, mots croisés, etc.) and all types of other segments that are not extracted as "Text" by the OCR, illustrated in Figure 2. Such images might not play an important role for historians and researchers, but classifying them correctly can help with search queries, in order to narrow down searches and filter them out because it is considered as noise for some applications.

Figure 1: Example images from the database

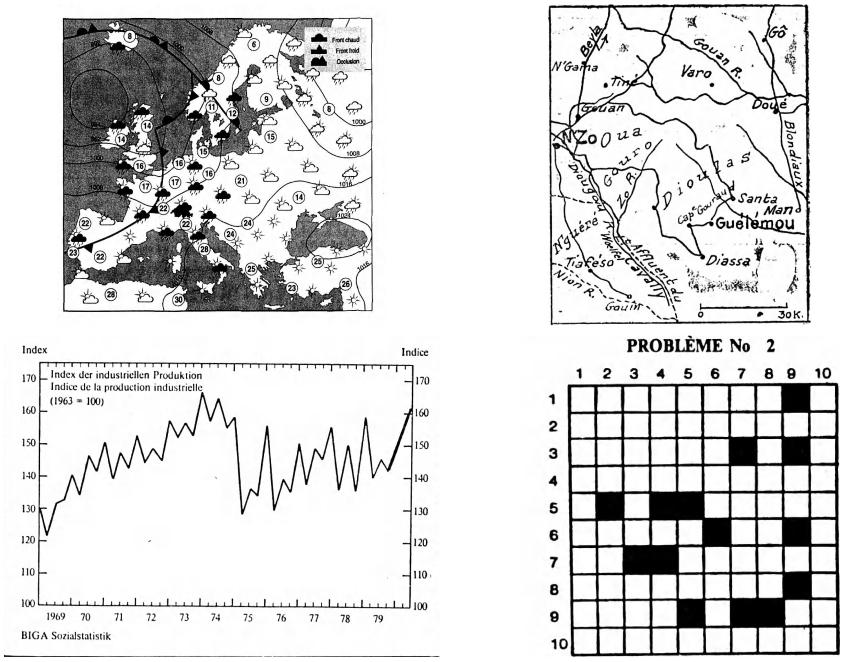


Figure 2: Example images from the database

Another interesting aspect of this database is its time span. The 80,000 images we classified cover a time period ranging from 1809 to 1998, which gives rise to a gap of quality, and needs to be considered when choosing the right algorithm.

3 Model Comparison

As mentioned in the objectives, the first milestone of this project consisted of evaluating the performance of existing models. For that, several popular object detection models were tested on a batch of archive images. This batch was taken pseudo-randomly, meaning a random batch of images was extracted, filtered and some of them re-sampled others. This had to be done as for this part, evaluation was to be done visually to assess the performance of each model and thus needed some identifiable features to assess the performances. We tried to not be too specific but to have images with distinguishable objects.

3.1 Pre-Trained Models

We started by evaluating the available pre-trained model. Some of them are known to be very efficient and considered to be state-of-the-art. Finding models that are already trained is almost a requirement. A well known problem in machine learning is that it is difficult to find labeled data to perform supervised learning and that it is extremely expensive/time consuming to label your own data. The goal of this project being to experiment and assess the value of automatic labeling of images in one semester, labeling

our own data was not affordable.

3.1.1 OpenCV

The OpenCV [3] is a popular computer vision library that offers an embedded object detection module. However it is very basic and allow us to classify only faces, person and vehicles which is clearly not sufficient for our application.

3.1.2 Faster-RCNN

Faster-RCNN [10] is a state-of-the-art model from 2016 and the available pre-trained one is trained on the OpenImages data-set [8] with 600 labels. This greater number of categories is very interesting for us since it offers a potentially wider enrichment of our images metadata. Unfortunately Faster-RCNN performs very poorly on low quality images and detected only few instances of items, most of them being persons. Since the model was not able to take advantage of the 600 categories it could not be directly used for our application as well.

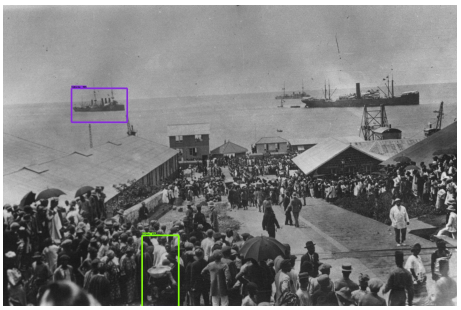


Figure 3: Faster-RCNN purple: vehicule, green: person

Figure 4: Faster-RCNN green: airplane, blue: man, salmon: woman

3.1.3 Mask RCNN

The Mask RCNN [1] model is an extension of the Faster-RCNN. Its main difference with the previous model is that it is capable of outputting a mask for the object. In terms of visual evaluation it is very desirable to be able to see how the model segmented the image into different objects. Performances of the model were great, as we can see on Figure 5 , however it was trained on COCO data-set [9] with only 80 categories. By looking at the categories most of them had little to no interest for our application. Retraining this model with a larger data-set such as OpenImages [8] with 600 categories was close to impossible since it requires object segmentation for the training set. This model was thus not suitable as well.

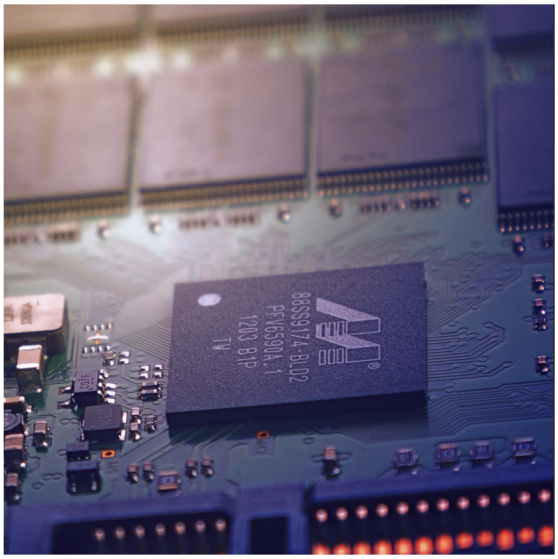
Figure 5: Example of Mask-RCNN performance

3.2 Image classification APIs

The performances of the existing pre-trained models seem to be non sufficient to reach the goals of this project. Available labeled image data-sets also did not meet the requirements in terms of time varying images.
Some of the biggest tech company such as Amazon, Google, IBM, invest a lot of time and money into image classification. One can easily deduce that their training data-sets are huge and spread across long periods of time and cover a lot of labels. When thinking about Amazon and object detection, the labeling is already done by the name of the products they sell and the image associated with it, and the same intuition goes for Google. As a side remark, one of the solutions considered was to re-train one the models presented above with pre-defined interesting labels some random images appearing in the first pages of Google Image.
Their services are accessible through API calls. Once again, and the evaluation was done on some pseudo random batch of images to get an intuition of their performances. Given the use case, one feature that is really interesting for this project, in addition to object classification, is scene classification, which is offered by the APIs proposed. Scene classification is the act of finding labels for the whole scene of the image, instead of classifying objects within the image. For example, an image of a mountain range could be classified as "Scenery", which is rather a scene than an object. Classifying scenes might be even more useful than objects for this application, as the whole image is taken into account and the model tries to put a name on what the picture is about, not what is on it, e.g, map, landscape, demolition, etc.

3.2.1 IBM Watson

IBM Watson [6] is the name under which each IBM AI product is developed. One of these products is Visual Recognition. Their service "uses deep learning algorithms to analyze images for scenes, objects, faces, and other content". In Fig. 6 we can see one of the images [reference to image] that is put forward by IBM on its website. We can see that you can have general descriptive labels such as "ultramarine color" and also some very specific ones related to the image with their associated confidence.



ultramarine color	0.91
electrical device	0.69
apparatus	0.66
sequencer (apparatus)	0.66
equipment	0.60
conductor	0.60
electronic equipment	0.60
memory device	0.59
computer circuit	0.58
electrical circuit	0.55
microchip	0.51

Figure 6: IBM demo image

Despite this very promising example, IBM Visual Recognition also suffers from the fact that it was trained on very high resolution modern images. For that, it is clear that it doesn't perform as good on sample images from the Impresso [BNF] database, as we can see in Figure 7.

black color	0.95
school	0.94
building	0.94
gray color	0.83
person	0.79

Figure 7: IBM Watson classification with image from Impresso

3.2.2 AWS Rekognition

Rekognition [2] is the deep-learning framework developed by Amazon Web Services, for which we do not know the architecture, and hence we can only use as a black-box. However, the performances were beyond expectation, especially with older, less clear images. The improvement compared to IBM is considerable, as shown in the comparative Figure 8: It is much more efficient at detecting classes in older images, which is exactly our use case. The labels can be scene-wide labels, meaning they do not have a specific instance on the image, or they can be labels with instances, meaning bounding boxes for each detected object are returned by the API. For both kinds of labels, we are also given a confidence score, as well as the parents of that label, which would allow us to reconstruct the hierarchy, as it is not available anywhere online.

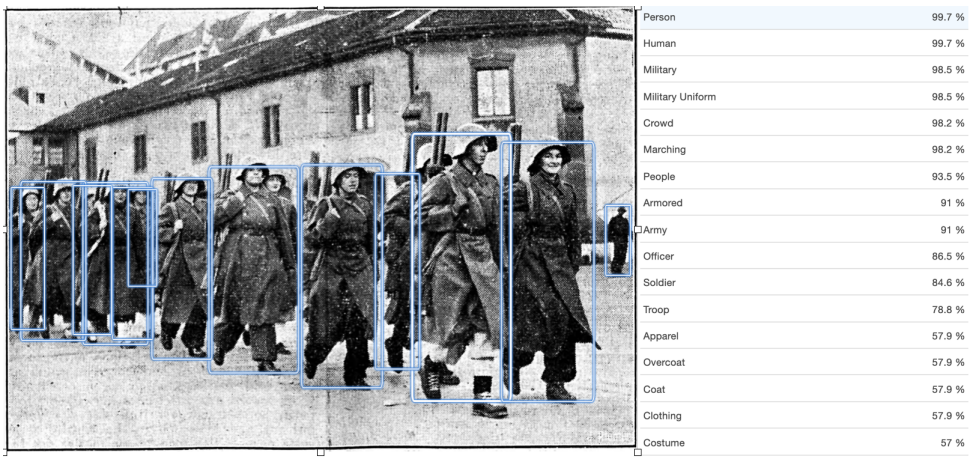


Figure 8: AWS Rekognition classification with image from Impresso

Another important aspect of Rekognition, is that labels are constantly updated, and new ones are added. Also, we do not now how many total labels can be recognized, but we can have a good approximation of what kinds of labels are detected from our data-set.

3.2.3 Google Cloud Vision

GC Vision [5] is essentially very similar to Rekognition, in the sense that both are black-boxes, and output the labels with no further information on how the detection is done. The difference however, is that GC Vision does not provide a hierarchy of labels, but only outputs the scores, and no instances of labels, meaning no bounding boxes. This gives us less information about the model than AWS, but gives us a wider range of more detailed labels. Performance-wise, Vision is quite similar to Rekognition, and we figured that both models can be complementary: often when one model is mistaken, the other gives us better results, as we can see in Figure 10. GC Vision also offers a *Web Entity* search for images, where it would output similar images found on the web. This service seemed to work quite well, but it was too expensive and evaluating it would have been impossible.

One of the requirements of the project is the ability to detect maps and diagrams, which would allow historians to search and compare them across multiple points of history. None of the pre-trained models that were tested before have the ability to detect the label *Map*, but GC Vision and AWS can detect them quite easily, as shown in Figure 9.

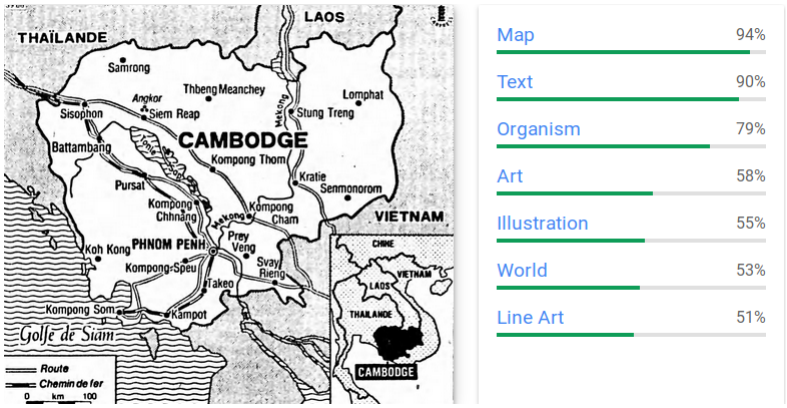


Figure 9: GC Vision classification with image from Impresso

Figure 10: AWS classified as "Chain mail", GC Vision as "Apparel"

4 Pipeline architecture

After having experimented with multiple available APIs and pre-trained models, we came to the conclusion that the best and most effective option was to combine the results of the last two APIs.

The section describes in detail the architecture of the classification pipeline, as well as the techniques used to merge the results of both models.
The pipeline is coded in *Python*, and has been packaged (the name of the package is *newspaperclassifier* [4]) with available scripts that will be clarified in the following subsections.

4.1 Classifying the images

Initially, the image data (date, page number, coordinates, metadata), were stored in *JSON* format on an S3 bucket, and fetching them was made possible through the IIIF framework The first step of the pipeline is to read the image data from the Impresso S3 Bucket, and feed them to Vision and Rekognition APIs. Since the queries are very similar, we have created an *AbstractBaseClass* that takes care of common methods, and 2 children of this class, that implement the specifics of each API. The labels are then stored in JSON format, using the keys *id* for the image id, and *labels* for the list of labels.

Vision Classifying images with this API costs $1.5 per 1000 images, is limited to 50 requests per second, and are done via the *google-cloud-vision* python library. We have extracted the labels from the response, and stored them in JSON files as a temporary result of the pipeline. The labels are stored in the following form.

```
{'id': 'GDL-1968-10-16-a-i0113',
 'labels': [{'mid': '/m/030jx3',
   'description': 'Square',
   'score': 0.6448917388916016,
   'topicality': 0.6448917388916016}]}
```

The only important here are *description*, storing the name of the label, and *score* which gives the confidence level. The topicality value is always the same value than the scores in all the JSONs, which is why it is not worth considering. Google API defines the topicality as: "The relevancy of the ICA (Image Content Annotation) label to the image. For example, the relevancy of "tower" is likely higher to an image containing the detected "Eiffel Tower" than to an image containing a detected distant towering building, even though the confidence that there is a tower in each image may be the same. Range [0, 1]." [5]

Rekognition Is a bit cheaper than Vision, at $1 per 1000 images, and is limited to 30 requests per seconds. The calls are made through the libraries *boto3* and *botocore*. A similar procedure has been done to save intermediate results. The labels are stored in the following form.

```
{'id': 'GDL-1981-05-27-a-i0088',
 'labels': [{'Name': 'Art',
   'Confidence': 81.97529602050781,
   'Instances': [],
   'Parents': []},
  {'Name': 'Drawing',
   'Confidence': 56.234920501708984,
   'Instances': [],
   'Parents': [{'Name': 'Art'}]}]}
```

The field *name* gives the name of the label, *Confidence* gives the confidence level, *parents* gives the list of parents for this given label and *instances* gives the bounding boxes for the found instances of this label. The list of labels contains all the parents of the label at the lowest level of the tree: we can see here that the detected label is "Drawing", and that "Art" is a parent of it.

Running the script The script which performs the classification and saves the labels as intermediate results can be found under *scripts/* and has the following signature

```
impresso_classify --client [gc | aws] --destination <dir> -b [batch_mode]
```

Running this script requires the hosting machine to be configured to work with Google Cloud and AWS, meaning it needs to have set up the correct credentials. For this, one would need to create both AWS and GC accounts, and store the access keys and secret keys using a file (in /.aws/ for AWS and downloading the credentials for GC). Beware that it might take some time. It took around 20 hours for 80,000 images.

4.2 Merging labels

After having classified the images and stored them into folders, the next step would be to quantitatively evaluate the given labels. In order for that evaluation to be unbiased, the labels of both sources needed to be merged, not by just concatenating the lists of labels, but by also finding those common ones. With that idea in mind, one could start thinking about a voting mechanism between the two APIs, that would allow for an increased confidence in the given labels.

Therefore, the merging part consists of augmenting the Vision labels with their respective parents of Rekognition hierarchy, and then finding the common labels between the two APIs. Adding the parents of the AWS hierarchy to the GC labels will help increase the number of common labels for the same images, and give us a better, unbiased evaluation score.

Hence, before merging, we had to reconstruct the tree of Rekognition and add them.

Tree Reconstruction The reconstruction of the AWS tree is quite straight forward. First the parents of each label need to put in a list, and then the reconstruction has

to be done top-down in the hierarchy, starting with labels having no parents, and incrementally adding children. The code relative to this part can be found under *evaluation/tree_aggregation*, and is automatically reconstructed when the merging script is called.

Running the merging script The script can also be found under *scripts/*, and has the following signature

```
impresso_merge_labels --aws-dir <dir> --gc-dir <dir> --destination <file>
```

Result The resulting DataFrame has the following columns

	Image	gc_label	gc_score	aws_score	aws_label	label	is_common
586033	GDL-1988-08-22-a-i0072	NaN	NaN	88.694038	photo	photo	False
30674	GDL-1950-07-28-a-i0049	stock photography	67.624575	NaN	NaN	stock photography	False
490213	GDL-1994-07-14-a-i0190	leaf	75.946116	NaN	NaN	leaf	False
718429	GDL-1987-01-06-a-i0167	NaN	NaN	80.564781	apparel	apparel	False
55695	GDL-1959-01-13-a-i0008	building	67.391312	55.887299	building	building	True

Figure 11: Merged DataFrame of labels

For each image and label there is a row, and labels that are common between GC and AWS are on the same row. From this DataFrame, we can already extract a few numbers: in total, we have 4692 distinct labels for both APIs, Vision having 3555 and Rekognition 2150, with 1013 common labels after having added the parents. Of those common labels, 805 have been given for the same image at least once. These results will be explored more in the next section.

4.3 Dumping new data

Finally, now that we have our merged labels, we need to dump them onto the S3 bucket for further indexing and use by the Impresso platform. This part is fairly straight forward, and simply extends the JSON of each image by adding three fields: *aws_labels* and *gc_labels*, both storing their respective label names and scores, and another *common_labels*, which stores the common labels for this image, and the mean score between AWS and GC

Running the script Like all other scripts, it can be found under *scripts/*, and has the following signature:

```
dump_labels --merged <dir> --endpoint <url> --bucket <url>
```

By default, the dump will be done on the *s3://impresso-image-labels/* bucket, but this can be changed using the above mentioned syntax.

5 Evaluation & Results

The raw responses from the two APIs having been processed and stored into DataFrames, we are now able to get more insights on what are the available labels, and how they can be used. Some basic statistics such as label distribution or score distribution are the first we looked at, considering one or the other API as well as the union of labels from both models. The most difficult part is to quantify the efficiency of the classifier. As stated before, those APIs are complete black boxes for the users. Even though confidence scores are also returned, one cannot blindly trust them given that those scores strongly depend on the training set. The fact that, by visual evaluation it seems to work very well, is an indicator that images similar to the ones in Impresso, are probably present in their training set. However it is still a supposition.

We will present the different methods used to evaluate our pipeline and the results.

5.1 Statistics

Label Distribution The first thing to look for is the distribution of the labels. For example plots of the most common labels for AWS and for GC, at the exception of the label person (for AWS) that is detected way more times than the others, 44142 instances, and that has no interest for us:

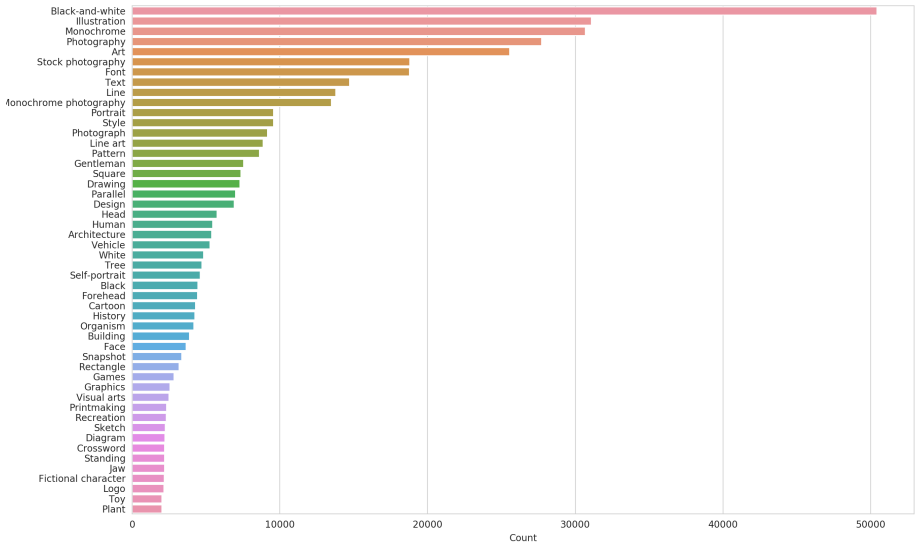


Figure 12: Most common labels given by Google Cloud Vision

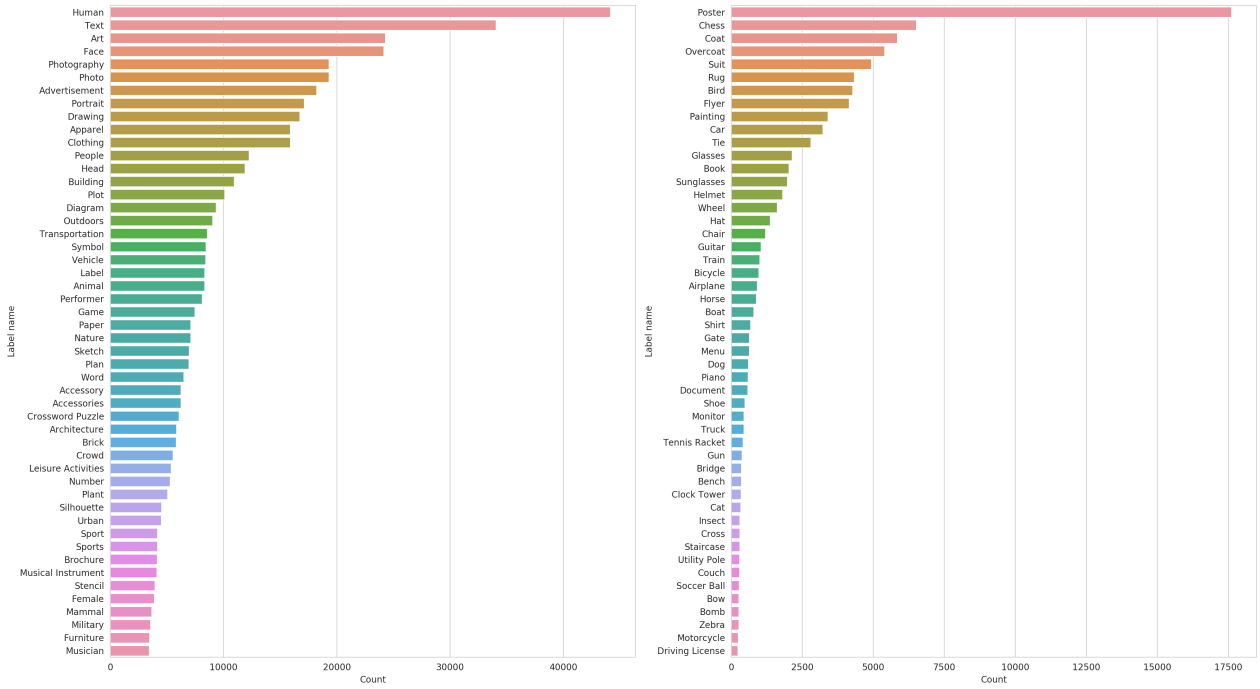


Figure 13: Most common labels given by Rekognition (scene labels on left, instanced labels on right)

Recalling that AWS does a distinction with the object instantiated, namely distinguishable object in the image and scene label (non-instantiated). The most common output for GC and non-instanciated labels of AWS are very general, like photography, black and white, human, etc. This might not be of direct interest for historians, however it can still be used to filter some results as already mentioned.

Label distribution in time We also investigate the distribution of both images and labels over time as seen in Figure 14. As expected, we can see an exponential increase, with a dip around the 70s-80s, in both the number of images and number of labels (since the are very correlated). Unfortunately we are not able to extract meaningful information from this analysis.

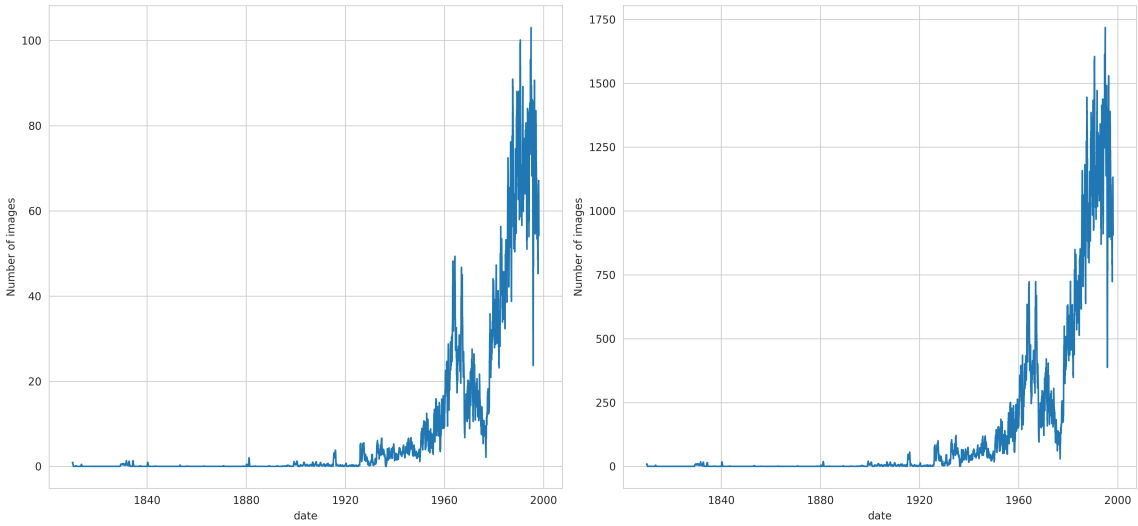


Figure 14: Distribution over time of the number of images and labels

5.2 Evaluation Method Hypothesis

A way to quantify the results and present them is needed, as it would allow to build confidence in the labelling of the APIs.

A first idea was to label manually a batch of the data-set and use it as a test set by comparing our labeling with the output of APIs. There are multiple problems with this method. First of all, there are 4692 distinct labels between the two APIs, which means that if a random batch of images is taken, even a small one, for each object or adjective we think could describe the scene, we should check in the 4692 labels to see if it exists. Or doing it the other way around, looking at each possible label and check if it would fit the image. But how can confidence be evaluated? This is not feasible.

Another idea would be to select a subset of labels, ideally the ones that fit best the uses cases, and label a random batch of images only using those labels. Once again this would not work. As said before, the labels of interest are not among the most present, therefore, by taking a random batch of images there is a great probability that none of the selected label would suit the random image. Therefore it is needed to isolate images that would contain certain objects, but then it is totally biased.

The best solutions we find are by coupling algorithms, we presents two evaluation methods that seemed to be less biased than the ones mentionned above.

5.3 Dual Voting

The first plausible evaluation method we thought of was to take advantage of the fact that we were using two APIs to create labels. In machine learning, voting systems between different implementations is a widely used method. By using the responses from

both models and merging them, we noticed that more than 1000 labels are common, and therefore decided to try to evaluate those common labels. The main idea of this evaluation is that if both APIs gave an identical label for an image we can be more confident than if only one of them gave it, knowing that both APIs have this label.

We thus computed a ratio per label. This ratio is given by:

$$\text{label x}_{ratio} = \frac{\text{Number of images labeled x by both GC and AWS}}{\text{Number of images label x by either GC or AWS}}$$

By computing this for all labels we can extract some labels in which we can be more confident. We show in Figure 15 the labels detected in at least 10 images with the highest ratio. The filter on the minimum number of images, gives more meaningful data when looking at the highest ratio. For example, if there is only one image of an opossum in the database and that both APIs label it, it does not give information on how frequently it will by detected by both, and it's ratio would still be 1.

label	common	total	ratio
dice	15.0	21	0.714286
fireworks	10.0	17	0.588235
map	1497.0	2733	0.547750
sailboat	76.0	149	0.510067
wine bottle	21.0	43	0.488372
portrait	8359.0	18339	0.455805
sedan	276.0	608	0.453947
glasses	1184.0	2637	0.448995
vehicle	4199.0	9470	0.443400
digital camera	7.0	16	0.437500
racket	187.0	481	0.388773
wheelchair	12.0	31	0.387097
motocross	8.0	21	0.380952
boat	373.0	1024	0.364258
postage stamp	380.0	1044	0.363985

Figure 15: Labels with the highest ratio, for labels found in more than 10 images

As we can see map is the 3 highest, which delights us!

5.4 Replica "cross-validation"

In parallel to our project, another research project, called "Replica", is under integration in the Impresso search engine. It initially aimed at building "the first search engine designed specifically for the search and exploration of artistic collections". [**replica˙epfl**] This model makes it possible to retrieve images that are visually similar to the query image. Similarly to the dual voting methods explained above, we would like to confront

those two models to gain confidence in our automatic labelling.

Our method was thus to use Replica and create scores for each label. The algorithm only works when images are already labeled, and labels are merged. Also, keep in mind that querying similar images for 80,000 images takes around 40 hours, so this step should be done beforehand.

The algorithm works in 3 steps. The first step is to use Replica and query the n similar images above a certain score, and store them. The second step is described in Algorithm 1 and counts for each image and each label given for that image, the number of similar images with that same label.

```
Data: merged, similar_images
Result: similar_label_count
for image i in images do
    labels = get_labels(i);
    similar = get_similar_images(i);
    count = {};
    for label l in labels do
        for image j in similar do
            if l in get_labels(j) then
                count[l] += 1;
        end
    end
end
```

Algorithm 1: Counting occurrences of labels in similar images

The *merged* represents the merged labels of AWS and GC, and *similar_images* represents the pre-computer similar images of each image. We can assume the function *get_similar_images* returns the similar images of a given image. Figure 16 shows an example output of this algorithm.

	Image	counts	num_similar
0	GDL-1809-08-11-a-i0003	{'art': 18, 'black-and-white': 29, 'bow': 6, '...	37
1	GDL-1809-08-22-a-i0003	{'art': 12, 'black-and-white': 30, 'bow': 4, '...	40
2	GDL-1809-09-01-a-i0003	{'bow': 4, 'clip art': 6, 'font': 41, 'line': ...	43
3	GDL-1809-09-12-a-i0003	{'font': 41, 'number': 23, 'symbol': 32, 'text...	42
4	GDL-1809-09-26-a-i0003	{'black-and-white': 33, 'font': 41, 'line': 36...	42
5	GDL-1809-10-06-a-i0003	{'black-and-white': 30, 'font': 38, 'label': 1...	40

Figure 16: Result of Algorithm 1, using 50 similar images

After that, the counts are transformed to ratios, by dividing them with *num_similar*,

and then the ratio of each label is computed by "transposing" the DataFrame: instead of counting per image, we count by label. The final result computes the mean, standard deviation and total count for each label, giving us an idea of the performance. Figure 17 shows the final result for the top labels.

	label	score_mean	score_std	count
759	crossword puzzle	0.968776	0.144432	6059
554	chess	0.949496	0.149748	6442
1156	game	0.929283	0.202502	7351
2507	square	0.913933	0.150066	7305
1385	human	0.849351	0.208480	44633
1971	person	0.843008	0.208878	44088

Figure 17: Final result example, score of each label, using 50 similar images

As we can see, "crossword puzzle" is the highest rated, meaning that if an image has this label, then there is a very high chance that a visually similar image will have this same label. The same goes for "chess", "game" and "human".

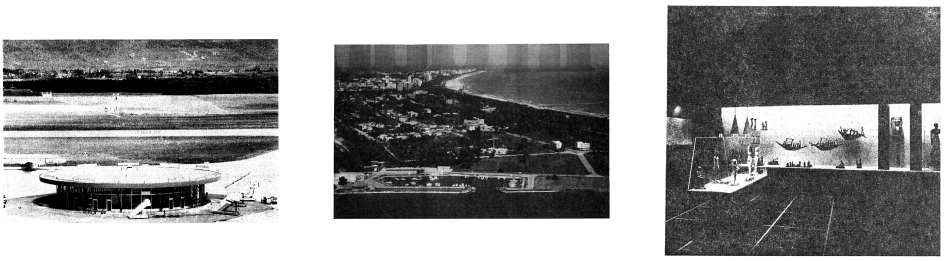


Figure 18: Example of combining Replica and Newspaperclassifier

Figure 18 shows an example of combining Replica and the classifier output. The leftmost picture was given the label "Building", and the two other pictures were returned by Replica as being very visually similar to the first one. However, only the first and second were labeled "Building". As we can see in that case, the labeling as more accurate than Replica, and it shows that coupling both systems can be beneficial.

6 Conclusion & further ideas

The images in the Impresso database present a high degree of variability, in terms of the content of the image, the quality and even within the objects themselves: images that were taken in the 1900s are mostly low quality compared to later images in the 2000s. Also, a car, for example, has changed a lot in appearance over 100 years. After having tested multiple models and APIs, it turned out that using Google Cloud Vision and AWS Rekognition was the best choice for this use case.
These APIs gave us a wide range of labels, which represent quite well the variability of image content, and have a considerable degree of confidence. The strong advantage of using them is their ability to function on low quality and old images, in a much better way that other pre-trained open-source models. However, there might still be discrepancies and wrongfully labeled images. To help with that, the voting system comes into play, by merging the labels and increasing the confidence level of labels found by both APIs.

In the view of a search engine on this database of images, we believe using all the labels returned can be useful, and can provide a way for researchers and historians to perform keyword searches, either by selecting or eliminating labels. Furthermore, we believe a search based on drop-down selection or auto-complete is usable, but another more complicated search engine can provide better accuracy for queries: Using NLP techniques, we could possibly reduce the search query to the corresponding labels, by searching for synonyms, translating words etc.

References

[1] Waleed Abdulla. *Mask R-CNN for object detection and instance segmentation on Keras and TensorFlow*. https://github.com/matterport/Mask_RCNN. 2017.

[2] *Amazon Web Services Rekognition*. https://aws.amazon.com/rekognition/.

[3] G. Bradski. *The OpenCV Library*. 2000.

[4] Lucas Gauchoux Edoardo Hölzl. *Impresso Newspaper image classifier*. https://github.com/lggoch/Newsp

[5] *Google Cloud Vision*. https://cloud.google.com/vision/.

[6] *IBM Watson Visual recognition*. https://www.ibm.com/cloud/watson-visual-recognition.

[7] *Impresso, Media Monitoring of the Past*. https://impresso-project.ch/.

[8] Ivan Krasin et al. *OpenImages: A public dataset for large-scale multi-label and multi-class image classification*. 2016.

[9] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *Computer Vision – ECCV 2014*. Ed. by David Fleet et al. Cham: Springer International Publishing, 2014, pp. 740–755. ISBN: 978-3-319-10602-1.

[10] *Tensorflow Modelzoo*. https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md.