Master Thesis

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Database of personal web pages from Wikipedia

Bc. Lukáš Kotrbatý

Supervisor: Vojtech Franc, Ph.D
Field of study: Open Informatics
Subfield: Software Engineering
September 2022

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Kotrbatý** Jméno: **Lukáš** Osobní číslo: **457184**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Otevřená informatika**

Specializace: **Softwarové inženýrství**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Databáze osobních stránek z Wikipedie

Název diplomové práce anglicky:

Database of personal web pages from Wikipedia

Pokyny pro vypracování:

The goal will be to develop a program for automated downloading and processing of personal web pages from Wikipedia. The processing will involve the extraction of a representative facial image and attributes characterizing the captured person like his/her age, birthday, gender, occupation, and so on. The output will be a database of facial images annotated by the extracted attributes and a statistical evaluation of the accuracy of the annotation via using the database to solve a face recognition problem.
Instructions:
Survey methods for efficient downloading and processing personal web pages from Wikipedia.
Design and implement the automated software which produces the database.
Design and implement statistically sound experiments to evaluate the correctness of the created database.

Seznam doporučené literatury:

- https://www.mediawiki.org/wiki/API:Main_page
- Rothe, R.; Timofte, R.; et al. Dex: Deep expectation of apparent age from a single image. In Proceedings of the IEEE International Conference on Computer Vision Workshops, 2015, pp. 10–15.
- V. Franc, J. Cech. Learning CNNs from Weakly Annotated Facial Images. Image and Vision Computing, 2018.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Vojtěch Franc, Ph.D. Strojové učení FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **12.04.2022** Termín odevzdání diplomové práce: **15.08.2022**

Platnost zadání diplomové práce: **19.02.2024**

Ing. Vojtěch Franc, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací.
Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.	
Datum převzetí zadání	Podpis studenta

Acknowledgements

I would like to thank primarily my thesis supervisor Vojtěch Franc for his guidance, feedback, and patience during the development of this thesis. Furthermore, I would like to thank the Center for Machine Perception at CTU in Prague for using their hardware, without which this work would hardly have been completed.

At last, my gratitude belongs to my beloved family and friends for their support and care.

Declaration

I declare that I worked out the presented thesis independently and I cited all references according to Methodical instruction about ethical principles for academic thesis writing.

In Prague, 1. September 2022

Abstract

In this thesis a script, which can automatically create annotated facial images dataset from data available on Wikipedia was developed. The script can be used repeatedly, creating an even larger datasets as Wikipedia grows. Two new facial datasets were created as a part of the thesis – Wikipeople and Wikifaces. The uniqueness of these datasets lies in the number of faces that are annotated with attributes related to the subject's age (age in the picture with year precision, date of birth and death). Other annotated attributes include name, gender, occupation, nationality and so on. Wikipeople contains all the collected data. Wikifaces is a subset of Wikipeople and contains only people with images in which exactly one face was detected and at the same time the age of the person in an image was found. Wikifaces dataset is composed of 604,391 people and contains 617,418 images. Because the Wikifaces dataset contains images which have been captured under completely uncontrolled conditions it is considered as 'in-the-wild' assembled. Quality of the obtained annotation was statistically evaluated, with focus on the most challenging part – age of the person in an image. Furthermore, the age and gender estimation tasks were formulated as classification problems and a set of convolutional neural networks were trained using the Wikifaces dataset. AgeDB was used as a benchmark and the results suggest that even though the Wikifaces dataset contains some noise it could be used in the computer vision field as well as for further research.

Keywords: Wikipedia, convolutional neural networks, big data, gender classification, age classification, image dataset of faces, in-the-wild dataset

Supervisor: Vojtech Franc, Ph.D
xfrancv@cmp.felk.cvut.cz

Abstrakt

V této práci byl vyvinut skript, který dokáže automaticky vytvořit anotovaný dataset obrázků s lidskými tvářemi z dat dostupných na Wikipedii. Skript lze používat opakovaně a vytvořit tak ještě větší datasety, jak se bude Wikipedie zvětšovat. V rámci práce byly vytvořeny dva nové datasety obrázků s lidskými tvářemi – Wikipeople a Wikifaces. Unikátnost těchto datasetů spočívá v počtu tváří, které jsou označeny atributy souvisejícími s věkem subjektu (věk na obrázku s přesností na rok, datum narození a úmrtí). Mezi další atributy patří jméno, pohlaví, povolání, národnost apod. Wikipeople obsahuje všechna sesbíraná data. Wikifaces je podmnožinou Wikipeople a obsahuje pouze osoby s obrázky, na kterých byl detekován právě jeden obličej a zároveň byl nalezen věk osoby na obrázku. Datová sada Wikifaces se skládá z 604 391 lidí a obsahuje 617 418 obrázků. Vzhledem k tomu, že datová sada Wikifaces obsahuje obrázky, které byly zachyceny za zcela rozdílných podmínek, je považována za posbíranou „in-the-wild“. Kvalita získané anotace byla statisticky vyhodnocena se zaměřením na nejnáročnější část — věk osoby na snímku. Dále byly úlohy odhadu věku a pohlaví formulovány jako klasifikační úlohy a pomocí Wikifaces datasetu byla natrénována sada konvolučních neuronových sítí. Databáze AgeDB byla použita jako benchmark a výsledky naznačují, že vytvořený dataset by mohl být, s ohledem na možnou nepřesnost některých atributů, využit v poli počítačového vidění, stejně jako k dalšímu zkoumání.

Klíčová slova: Wikipedie, konvoluční neuronové sítě, big data, klasifikace pohlaví, klasifikace věku, databáze obličejů, in-the-wild dataset

Překlad názvu: Databáze osobních stránek na Wikipedii

Contents

Figures

Tables

Chapter 1

Introduction

Estimation of human characteristics (age, gender, nationality, and so on) from a picture is a very fascinating problem from both machine learning and computer vision point of view. Not only because it directly exhibit the range of skills artificial intelligence is capable of nowadays, but also because of its wide use in many applications such as social understanding, biometrics, identity verification, video surveillance, human-computer interaction, electronic customer, crowd behavior analysis, on-line advertisement or item recommendation. With the fields of machine learning and computer vision constantly developing, the demand for annotated datasets of facial images arises.

With the recent rise of semantic web the process of creation of semi-automatically collected datasets is easier than ever. The Internet is full of data that can be used, the only task is to extract said data using a method that will not only be time- and energy-efficient, but also legal.

1.1 The aim of the work

The goal of this thesis is to develop a script for automated downloading and processing of personal web pages from Wikipedia. The processing involves the extraction of a representative facial image and attributes characterizing the captured person. The most important attribute is the age of a person in an image, but the final datasets come with other attributes related to the person such as birth date, death date, gender, occupation, and so on. The output of this thesis are two datasets (Wikipeople and Wikifaces) and a statistical evaluation of the accuracy of the annotation. Wikipeople contains all the collected data (people with related facial images and annotated by the extracted attributes). This dataset is rather sparse as some people might not have all the attributes or a facial image. Wikifaces is a subset of Wikipeople and contains only data suitable for attribute prediction from an image. Data suitable for attribute prediction from an image are defined as people with facial images in which exactly one face was detected and at the same time the age of the person in an image was found. Accuracy of the annotation is evaluated both directly and indirectly. As a direct method manual evaluation is used. Due to time and labour intensive nature of such task only age of the

person in an image is manually evaluated. As an indirect method dataset is used to solve a face recognition problem and accuracy is compared to a benchmark dataset.

There are datasets created in a similar manner, this was demonstrated for example in this paper [2], where IMDB database as well as Wikipedia were used as a source. Datasets created in this way, meaning that they contain images which have been taken under completely uncontrolled conditions (e.g., different photo capture angles, distinct lighting settings, varying backgrounds, presence of 'noise' in the pictures, multiple nonidentical kinds of cameras used to capture the images, etc.) are referred to as in-the-wild dataset [1].

Created datasets (Wikipeople and Wikifaces) are unique in multiple ways. Firstly, a source that has not been fully exploited before – Wikipedia is used to create them. Secondly, both datasets are still growing, by simply running the script again one can obtain an up-to-date datasets. Thirdly, created datasets contain original attributes (birth date, death date, occupation, etc.), which brings new research options. Lastly, most of the state-of-the-art datasets are somehow biased. For example they can contain people born close to present time or only celebrities and so on. Created datasets are very diverse as they contain people of various nationalities, races, occupations and birth dates of people in them spans from 1840 to 2015.

Figure 1.1: An example of Wikipedia infobox for a person with some filled data. This is exemplary input for the created script. As discussed further multiple data sources were used, not only the infobox. An example output for this person should contain his name (František Křižík), gender (male), birth date (1847-07-08), death date (1941-01-22), occupation (engineer, entrepreneur), nationality (Czech), age in picture (55) and so on. This picture was taken from: `https://en.wikipedia.org/wiki/Franti%C5%A1ek_K%C5%99i%C5%BE%C3%ADk`.

1.2 Outline of the Thesis

In order to meet the goals mentioned in Section 1.1, there are several things to discuss before diving into uncharted waters. Firstly, the state of the art in this field with focus on existing datasets is discussed in the Chapter 2.

The resources (Wikimedia Foundation projects) from which data are collected as well as semantic web are introduced in Chapter 3.

Brief overview of created datasets presented in Chapter 4 introduces final data structure of created datasets, brief description of the process of creation and statistics of both the Wikipeople and the Wikifaces datasets.

Experiments such as manual evaluation and indirect statistical evaluation of obtained annotation in the Wikifaces dataset are introduced in Chapter 5.

The Chapter 6 is dedicated to the proposal of future work with the Wikipeople and the Wikifaces datasets and created script. Possible improvements of created program as well as new ideas and what can be done to enhance the accuracy of the models are discussed.

All examples of requests used in the script are presented in Appendix B.

Parts of the work, which ultimately led to creation of the final script, but are not necessary for understanding of the development process were placed into Appendix. In Appendix C extensive list of different possible methods for gathering the data from Wikipedia is described. Detailed summary of how the script for generating the dataset works is presented in Appendix D. Both processes (data collection and face detection) are explained in detail with process diagrams in Appendix D.1 and Appendix D.2, respectively. Furthermore challenges as well as possible improvements are discussed in Appendix D.3. The Appendix E is dedicated to the full documentation of created program.

Chapter 2

State of the art

Finding a sufficiently large dataset that will be suitable for training neural networks to predict attributes from human faces is not particularly easy as datasets are hard to create. In this chapter state of the art in the field of face datasets is discussed.

The most important prerequisite for using any face image dataset is its size, quality and also the purpose for which it was created. There is a big difference between dataset of faces, which can be used on tasks such as face detection, landmark localization, etc. and those, that can be used for predicting age, gender or another personal attribute. This thesis focus on the latter one.

There are a lot of available datasets, all the progress in this area was very well documented in [1]. Table 2.1 presents the most broadly used annotated facial datasets in the Computer Vision community from 2002 to 2020.

Dataset	Year	# Imgs	# Subjs	Age labels	NF	ITW
FG-NET	2002	1,002	82	By year	Yes	No
MORPH	2006	1,724	464	By year	Yes	No
IFP-Y	2008	8,000	1,600	By year	Yes	No
Gallagher	2009	28,231	5,080	7 age groups	No	Yes
VADANA	2011	2,298	43	4 age groups	Yes	Yes
AdienceFaces	2014	26,580	2,984	8 age groups	Yes	Yes
CACD	2014	163,446	2,000	By year	No	Yes
IMDB-WIKI	2015	523,051	20,284	By year	No	Yes
AgeDB	2017	16488	568	By year	Yes	Yes
UTKFace	2017	20,000	-	By year	Yes	Yes
MAADFace	2020	3.3M	9K	3 age groups	No	Yes

Table 2.1: Concise overview of the most broadly used age datasets in the Computer Vision community since 2002 onwards. **# Imgs** refers to number of images in the dataset. **# Subjs** refers to number of subjects in the dataset. **NF** refers to presence of noise-free labels. If the dataset was collected 'in-the-wild' can be checked in the **ITW** column. The table was completely taken from [1].

The Google datasearch tool [3] can be used for exploring over 25 million datasets. They are not posted directly on the website, but the service provides direct links to searched datasets.

Chapter 3

Overview of resources used to create the dataset

In this chapter terms that are used frequently throughout the entire thesis as well as related applied technology are introduced. Section 3.1 covers Wikimedia Foundation, as the prime and only source of data gathered as a part of this thesis. Section 3.2 covers the technology of semantic web and its standard SPARQL.

3.1 Wikimedia Foundations and its projects

Wikimedia Foundation or WMF, is a foundation that revolves around all of the wiki-like projects, including but not limited to:

- Wikipedia – online encyclopedia
- Wiktionary – online dictionary and thesaurus
- Wikibooks – collection of textbooks
- Wikiquote – collection of quotations
- Wikivoyage – travel guide
- Wikisource – digital library
- Wikimedia Commons – repository of images, sounds, videos, and general media
- Wikispecies – taxonomic catalog of species
- Wikinews – online newspaper
- Wikiversity – collection of tutorials and courses, while also serving as a hosting point to coordinate research
- Wikidata – knowledge base

The only projects used in this thesis are Wikipedia, Wikidata, Wikimedia Commons and MediaWiki software, which is not a Wikimedia foundation project per se, but it is crucial to introduce it for better understanding of the totality of the topic [4].

3.1.1 Wikipedia

The author is well aware that theses should refrain from using Wikipedia as a main source, but in this case it probably is the most reliable source. The definition of Wikipedia as presented on [5]: *„Wikipedia is a multilingual free online encyclopedia written and maintained by a community of volunteers through open collaboration and a wiki-based editing system. Individual contributors, also called editors, are known as Wikipedians. Wikipedia is the largest and most-read reference work in history.“*.

3.1.2 Wikidata

Wikidata is a project of a collectively edited database of information available under a free license. It is operated by the Wikimedia Foundation and is closely linked to Wikipedia and other projects of the Foundation. The project was originally initiated by the German branch of the Wikimedia Foundation. Wikidata is supposed to implement a common repository of database data, for example birth dates (similar to Wikimedia Commons which implements a common repository for photos or audio files), that can then be used by other projects (for example, Wikipedia, Wikisource, Wikinews, etc.). Wikidata is the newest project of Wikimedia Foundation.

Wikidata database stores structured and machine-readable data unlike Wikipedia, which stores just any data. Even though it is a great source of information for both humans and machines, its prime goal is to make Wikipedia data available for machines, which is the reason why most Wikipedia pages have their own Wikidata page equivalents. It is important to note that Wikidata only serves as a secondary database for supporting Wikipedia and other projects. One should therefore keep in mind that information presented on Wikipedia and Wikidata might differ, agree, complement each other or be absent altogether [6] [7] [8].

Accuracy of Wikidata is also questionable. There are control mechanisms to prevent vandalism and inaccuracy, usually in a form of some artificial intelligence crawling newly added or edited data. But in comparison to Wikipedia itself there are still quite some weak points. Wikipedia is checked manually, which takes a lot of man power, but on the other hand Wikipedia became reliable source people are turning to most of the time. Wikidata project was created with the intention that every piece of information has to be related to a source, but creators stepped away from this requirement and as you can see in figure Figure 3.1 it backfired. Most of the data in Wikidata does not include its source and if they do it is usually reference to Wikipedia. So one is left with circular reference, when Wikipedia is referencing Wikidata and vice versa.

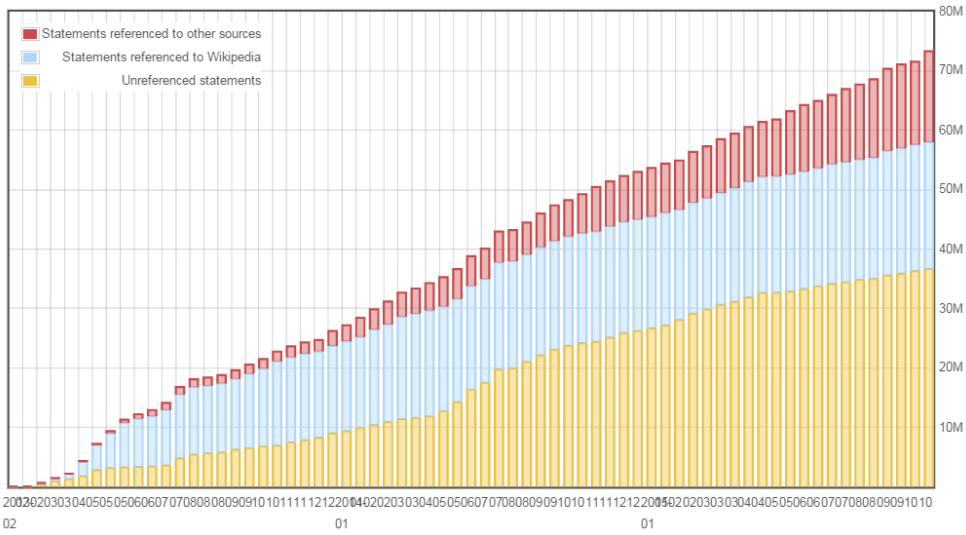


Figure 3.1: Amount of data on Wikidata referenced to different sources or not referenced at all. The horizontal axis indicates the month when the data were obtained. This picture was taken from: `https://en.wikipedia.org/wiki/Wikipedia:Wikipedia_Signpost/2015-12-02/Op-ed`.

3.1.3 Wikimedia Commons

All Wikipedia articles use a large number of images that take up a lot of disk space on their servers. That is why Wikimedia Commons was founded. This project saves significant disk space for all sub-projects that the Foundation manages. Wikimedia Commons aim is to provide a media file repository that: *„makes public domain and freely-licensed educational media content available to all“* and *„that acts as a common repository for the various projects of the Wikimedia Foundation“* [9]. Files available on Wikimedia Commons can be accessed from all Wikimedia Foundation projects in all language mutations.

The note about a freely-licensed educational media content is very important. Wikimedia Commons has its rules for files that can be uploaded to the repository [10]. But that does not mean that all files presented on Wikimedia Commons are free to use. Some of them can only be used for educational purposes, which fit perfectly on projects from Wikimedia Foundations. Some uploaded files violate the Wikimedia Commons rules and are very likely to be removed soon. This 'grace period' is very important factor for this project as some of those images might end up in the created datasets.

3.1.4 MediaWiki software

MediaWiki software is the software Wikipedia and other wiki-like projects are build upon. This is how MediaWiki site presents itself: *„The MediaWiki software is used by tens of thousands of websites and thousands of companies and organizations. MediaWiki helps you collect and organize knowledge and make it available to people. It's powerful, multilingual, free and open,*

extensible, customizable, reliable, and free of charge.“ [11].

Amongst its advantages one can find – scalability, feature-richness, durability, general use and speed. MediaWiki uses wiki implementation written in PHP to process and display data stored in a database such as MySQL. One of the most important features of MediaWiki software is its ability to create wiki-like pages. This is achieved through the use of so-called wikitext, sometimes called wikicode. It is similar to the Markdown language with some additional features. Wikitext is used to render finished HTML pages, data from different projects, pages or websites can be loaded in the process. Pages use MediaWiki's wikitext format, so users without any HTML or CSS knowledge can edit them easily. When a user submits a change to a page, MediaWiki writes it to the database, while keeping track of previous versions of the page, allowing for easy back-editing in the event of vandalism or spam. MediaWiki can also manage image and multimedia files stored in the file system. For large wikis with many users, MediaWiki supports caching and can be easily coupled with proxy server software. With dedicated extensions, MediaWiki can also handle structured data [12].

MediaWiki comes also with an API, which can be used for accessing pages automatically. More on this can be found in Appendix C.2.

3.2 Semantic web

The idea of semantic web was first introduced to the world in 1999. Tim Berners-Lee, creator of the web and director of the W3C Consortium said: *„I have a dream for the Web in which computers become capable of analyzing all the data on the Web – the content, links, and transactions between people and computers. A 'Semantic Web', which makes this possible, has yet to emerge, but when it does, the day-to-day mechanisms of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The 'intelligent agents' people have touted for ages will finally materialize.“* [13].

Formal definition then could be that the semantic web allows machines to understand the meaning of information on the World Wide Web. The technology is mainly based on Resource Description Framework (RDF) and Ontology Web Language (OWL). The World Wide Web Consortium (W3C) is working on specifications for the Semantic Web. To put it in simple terms semantic web is therefore about making data presented on the Internet have a precisely defined meaning and allowing automated (machine) processing to a large extent [14].

The field of semantic web is very complex, to keep this thesis short and focused, only the most important parts are introduced.

3.2.1 Resource Description Framework

Resource Description Framework is a simple language (set of specifications or a model) for describing objects and their relations in a graph of semantic web. In general, the data source can be anything that is clearly identifiable

(documents, people, physical objects as well as abstract concepts, numbers, strings, etc.). The Framework is developed by the W3C.

RDF is based on an atomic element denoted by a triple called RDF triple(t) or RDF statement. The structure of such element is:

$$< subject >< predicate >< object >$$

which expresses the oriented relationship between two nodes of the RDF graph. An RDF graph is defined as a set of RDF triples. Each triplet makes one valid statement. A statement is valid when there is a relationship (*predicate*) between an *object* and a *subject*. A set of statements is valid if all statements in it are valid. Valid statements can be used to infer the validity of other statements or to derive new statements [15].

En example of RDF triplet could look like:

$$< Michelle\ Obama > \ < is\ married\ to > \ < Barack\ Obama >$$

$$< Barack\ Obama > \ < is\ a > \ < politician >$$

3.2.2 SPARQL

Simple Protocol and RDF Query Language (SPARQL) is a semantic query language and protocol for manipulating RDF data. It is a language for querying graph (RDF) data just like SQL is used for querying data in relational databases. The syntax is very similar to SQL as SPARQL also uses keywords such as $SELECT$, $WHERE$, $ORDER$, $FILTER$, etc. SPARQL is used to access all data presented in Wikidata project [16].

Chapter 4

Overview of created datasets

In this chapter overview of created datasets is presented. Firstly, for better understanding of the whole process, the final structure of the created datasets is discussed with specific example in Section 4.1. Then, the process of how datasets are created together with how all the extracted attributes are collected is shortly described in Section 4.2. Statistics of both the Wikipeople and the Wikifaces datasets are presented in Section 4.3. More detailed description of how the process work as well as observations that were made during the implementation of this project can be found in Appendix C and Appendix D.

4.1 Data structure

Only the text data contained in the created dataset takes up approximately 1.97 GB, which expresses a choice that an efficient data structure needs to be used. The size of all data is not the only concern, others include:

- Re-usability
- Low complexity and clarity
- Requirements found during development

Created dataset is represented by two directories. First one contains all images and the second one contains information about people in a form of multiple JSON files. Each JSON file is named after one year (e.g. 1840.json) and contains all extracted annotations about the people born is that year.

The final data structure found in JSON files, which contain all the data is a dictionary with a person's Wikidata identifier as a key and the person object as a value. Person, image and face attributes definitions as well as their sources can be seen in Tables 4.1, 4.2 and 4.3, respectively. Not all people have all the attributes, most of the attributes are optional. Only name, birth date, Wikidata ID and images are always present. See the Section 4.3 for more details.

This structure might look too complex and unusual in the area of machine learning, but flattening it down (to CSV or similar format) would create too many duplicates and even more space would be required. The author also

believes, that JSON is universal and simple to work with, so a final dataset can be easily turned into any other preferred data structure.

For more details one can look at Listing 4.1, where all the information gathered about one person in JSON format are presented. The image related to that person, with the face bounding box, can be seen in Figure 4.1.

Attribute name	Data type	Source	Note
`name`	string	WD (P31)	Name or nickname of the person. Birth names were not collected, only the tag labels, which sometimes contained a nickname instead of a name. This can be seen for example with the youtuber PewDiePie [17].
`description`	string	WD	Short description of the person, which usually includes person's nationality and occupation.
`gender`	list	WD (P21)	Gender is defined as a list, because there might be people with multiple values.
`birthDate`	date	WD (P569)	Dates are in YYYY-MM-DD format.
`deathDate`	date	WD (P570)	Dates are in YYYY-MM-DD format.
`nationality`	list	WD (P27)	
`occupation`	list	WD (P106)	
`images`	dictionary	WD (P18), WP, WC	Attribute images is defined as a dictionary with the name of the file as presented on Wikipedia Commons as a key and image object as a value. See Table 4.2 for more details.
`wikipediaTitle`	string	WD	Name of the page on Wikipedia if one exists.
`wikidataID`	string	WD (P31)	This attribute is also used as a key in the dataset dictionary.

Table 4.1: Overview of person attributes in the dataset with data types and sources. In the column 'sources' **WD**, **WP** and **WC** refers to Wikidata, Wikipedia and Wikimedia Commons, respectively. Information in the parantheses behind **WD** refers to specific property used to obtain the attribute.

Attribute name	Data type	Source	Note
`caption`	list	WD (P2096), WC	Short description of what is in the picture or its origin.
`date`	list	WD (P585), WC	The date the image was taken.
`exifDate`	datetime	WC	EXIF dates are in YYYY-MM-DD HH:mm:SS format.
`url`	string	WC	The source URL from which the image can be downloaded.
`fileNameLocal`	string	-	Name of the local file representing the image. The name is created by hashing the image content by SHA-256 and shortening the result by the hexdigest() method.
`fileNameWiki`	string	WC	Name of the file as presented on Wikipedia Commons.
`age`	int	-	Estimated age of the person in the picture. Heuristic algorithm presented in Section 4.2.5 is used for the estimation.
`faces`	list	-	Attribute faces is defined as a list of face objects. See Table 4.3 for more details.

Table 4.2: Overview of image attributes in the dataset with data types and sources. In the column 'sources' **WD**, **WP** and **WC** refers to Wikidata, Wikipedia and Wikimedia Commons, respectively. Information in the parantheses behind **WD** refers to specific property used to obtain the attribute. Missing source means, that the information is a direct result of the created script, for example `age` is calculated by the heuristic algorithm described in Section 4.2.5.

Attribute name	Data type	Note
`score`	float	Confidence score of face detection process.
`box`	list	Bounding box of the detected face. `[x1, y1, x2, y2]`

Table 4.3: Overview of face attributes. The source column is missing in this table, because all the data come from face detection phase of the script.

```
"Q132524": {
    "name": "Ayn Rand",
    "description": "Russian-American novelist and philosopher",
    "gender": [
      "female"
    ],
    "birthDate": "1905-02-02",
    "deathDate": "1982-03-06",
    "nationality": [
      "Russian Empire",
      "Soviet Union",
      "United States of America"
    ],
    "occupation": [
      "essayist",
      "journalist",
      "literary critic",
      "philosopher",
      "writer"
    ],
    "images": {
      "Ayn_Rand_(1943_Talbot_portrait).jpg": {
        "caption": [
          "Photo portrait of Russian-American writer Ayn Rand used
              for the first-edition back cover of her novel The
              Fountainhead (1943)."
        ],
        "date": [
          "1943-01-01",
          "Published 1943"
        ],
        "exifDate": "2020-05-17 20:24:21",
        "url": "https://upload.wikimedia.org/wikipedia/
               commons/b/b1/Ayn_Rand_%281943_Talbot_portrait%29.jpg",
        "fileNameLocal": "1ca173463f30cf4c160d6ce7ab81
                         ae6b29e28ad95747b5674629f505e1a0f9b6.jpg",
        "fileNameWiki": "Ayn_Rand_(1943_Talbot_portrait).jpg",
        "extension": ".jpg",
        "age": 38,
        "faces": [
          {
            "score": 0.9996246099472046,
            "box": [96, 64, 753, 721]
          }
        ]
      }
    },
    "wikipediaTitle": "Ayn_Rand",
    "wikidataID": "Q132524"
  }
```


Listing 4.1: Example of structure of one person data from dataset.

Figure 4.1: Example of image from dataset with bounding box. This picture was taken from `https://en.wikipedia.org/wiki/Ayn_Rand`.

4.2 How is the dataset created

This section contains a brief description of how the Wikipeople and the Wikifaces datasets are created. For more detailed explanation with process diagrams one should look into Appendix C and Appendix D. This section describe only the creation of the Wikipeople dataset, because Wikifaces is a subset of Wikipeople and therefore, it was created by filtering the Wikipeople dataset. The word dataset used in this section always refers to the Wikipeople dataset.

All data presented in the dataset come from one of Wikimedia Foundation projects, more specifically from Wikidata, Wikipedia or Wikimedia Commons. It is important to note that Wikipedia and Wikimedia Commons share the same API, generally called MediaWiki API. The script begins with downloading data from Wikidata API then completing it with data from Wikipedia and Wikimedia Commons. After all the data are collected image files are downloaded. In the final stage, age of person in all images is calculated and faces are detected in all images. Next sections follow described structure of the process.

All data are collected year by year, meaning that firstly all the following steps are done for year 1840, then 1841, then 1842, and so on. The reason behind iteration representing exactly one year is that trying to request multiple years at once usually resulted in time-out. These requests are very heavy as in the most dense years they contain around 60,000 people. One has to take into account the structure of the data returned by the service as mentioned in Appendix D.1.6.

4.2.1 Use of Wikidata

The first necessary step is to obtain all the people presented on Wikipedia. This is done via Wikidata SPARQL API. Wikidata SPARQL endpoint is repetitively called requesting all people born in specific year `Y` together with a wide variety of attributes about them. These attributes are: name, description, image, image caption, image date, birth date, death date, gender, nationality, occupation, wikipedia title, wikidata ID. Each attribute refers to one property of a person and each property has its specific Wikidata tag. For example property P21 refers to a gender of a person. All attributes with their sources and specific Wikidata tags can be found in Table 4.1 and Table 4.2. An example call to SPARQL endpoint requesting all the important information about all the people on Wikidata born between years 1840 and 1841 can be found in Appendix B.3. One can view the call in user interface of Wikidata Query Service. The query used in the script was put together with the help of [18].

Some of the attributes are collected from multiple sources, namely image, image date and image caption are also collected in steps which are discussed in Section 4.2.2 and Section 4.2.3. Data presented on Wikidata API are also presented in human readable form directly on specific Wikidata page of the person. Property P18 (image) sometimes contains property P585 (point in time) or property P2096 (media legend). See Figure 4.2 for more detail.

President Barack Obama.jpg
2,687 × 3,356; 1.22 MB

point in time	6 December 2012
media legend	Official photograph of President Barack Obama (English)

Figure 4.2: Observable information about image available on Wikidata page of a person. This picture was taken from `https://en.wikipedia.org/wiki/Barack_Obama`.

4.2.2 Use of Wikipedia

Not all records of people in Wikidata come with a representative facial picture. Therefore, more pictures need to be collected from Wikipedia. As aforementioned not all people presented on Wikidata have a corresponding

page on Wikipedia. MediaWiki API is used for getting the thumbnail pictures for people with Wikipedia pages. Result of the query is a name of an image as presented on Wikimedia Commons, this name can be further used for getting more information about the picture (caption, date, EXIF, etc.) as described in Section 4.2.3. An example call can be seen in Appendix B.4.

There are usually more pictures on a personal Wikipedia page, but some of the pictures contains multiple people, some of them do not contain a person at all and some of them might have completely different person in them. That is the reason, why only thumbnails are taken into consideration. This is further discussed in Section 6.2.

4.2.3 Use of Wikimedia Commons

Before this stage begins the dataset contains all the people with their attributes and names of the images as presented on Wikimedia Commons. This stage uses MediaWiki API for gathering additional information about those pictures, such as dates of creation, captions and URL to download the picture from. Some of those attributes were already collected during the initial Wikidata SPARQL call, but different information can be found in this step. Therefore, all of them are stored in a list and later processed in Section 4.2.5. An example call can be seen in Appendix B.5. Data presented on MediaWiki API are also presented in human readable form directly on Wikimedia Commons. More specifically Wikipedia file page – summary table under the image usually contains a caption of the image as well as the date of the creation. See Figure 4.3 for more detail.

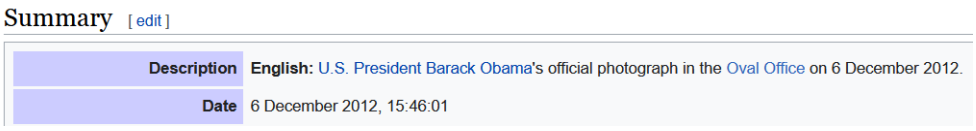


Figure 4.3: Observable information available on Wikipedia image file page. This picture was taken from `https://www.wikidata.org/wiki/Q76`.

4.2.4 Downloading pictures

URLs of images were collected in previous stage, so in this one all pictures in a dataset are downloaded. The local image file name is derived from SHA-256 hash of the image content shortened by the hexdigest() method. The reason behind this decision is that files can be removed from Wikimedia Commons and different file can be uploaded under the same name. This could create some inconsistencies in face detection. Suggest naming should be robust enough to work even if that happens. This is the most time-consuming process of all in the script.

4.2.5 Finding age of a person in an image

Once all the information is collected, the script can continue with finding the age of a person in an image. For this step only two pieces of information are needed, namely the date of image creation (image year) and the birth date of a person (birth year). For simplicity, only the years are used to calculate the age of a person in an image. Therefore, the following description uses 'image year' to refer to the date of an image creation and 'birth year' to refer to the birth date of a person.

The image is annotated with the age of the person found in it. This process is based on a very simple heuristic algorithm. Firstly, the script finds the year which is most likely the image year. This is done by gathering all years (four digits in a range from 1800 to 2500) found in date and caption attributes as well as in the image file name. All found years are stored in a list. Then those that do not fit into the possible range are removed. The possible range is defined as a range between the birth and death date of a person the image belongs to and between 0 and 110 exclusively. In the end, the most common value is found and chosen as the image year. Finally, this birth year is subtracted from the image year and found value is saved in the dataset as the age of a person in an image.

4.2.6 Face detection

This stage detects faces in all images in the created dataset and save them back to the dataset. Numerous face detectors were tested out, but because face detection is a time-costly process the speed was the deciding factor. The fastest one turned out to be Retina-face [19]. After putting it through a test on GPU, the decision for speeding up this process with `processedImages.json` was made.

The stage starts with looking for file called `processedImages.json`, which contains faces for all the images that have already been processed by face detector in previous runs. This file contains a dictionary where keys are local file names of images and values are faces detected in said image. If the file is found then it is read and used. As aforementioned face detection is a time-costly process, therefore, multiple conditions were implemented, so no image is processed more than once. First, if an image is annotated with a face, then the face detection step is skipped altogether. Second, if the image is found in `processedImages.json`, then the saved values are added to the image object in JSON data file. If none of those conditions are true, then the face detection is run with the image and results are saved in both the JSON data file as well as to `processedImages.json`.

4.3 Dataset statistics

Statistics were calculated over two different datasets. First dataset is called Wikipeople and refers to the created dataset as a whole. The second one is called Wikifaces and it is a subset of the Wikipeople dataset. Wikifaces contains only data suitable for attribute prediction from image. Data suitable for attribute prediction from image are defined as people with images in which exactly one face was detected and at the same time the age of the person in an image was found.

Most of the statistics are in a form of graphs, some statistics were unsuitable for display in a graph and therefore, are displayed in a table. This only applies to the overview of all properties in a dataset (occupation, nationality, gender, etc.).

4.3.1 Wikipeople

This subsection presents statistics calculated over the Wikipeople dataset. This dataset consists of all the collected data. Because of the sparse nature of this dataset, it can be used for further research as well as for completing missing data.

This dataset is composed of 4,421,816 people and contains 849,066 images. The numbers for all attributes are shown in the table Table 4.8. The first column should be self-explanatory, the second column represents the number of people with that attribute. Attribute can be a string, integer or a list, but at least one value must be present. In case there are multiple values present the third column is used. Specifically for properties that are lists (gender, nationality, occupation) or dictionaries (images). The number in the third column then refers to all of the values in the data structure. For example with gender there are 4,271,048 people with at least one gender, but 4,271,723 gender values were found in the dataset, which means some of the people in the dataset have more genders assigned to them. The last column refers to number of unique values in the Wikipeople dataset.

1,541 nationalities, 10,745 occupations and 37 genders were found in the dataset. Overviews of the ten most common countries, occupations and genders in the dataset can be found in Table 4.5, Table 4.6 and Table 4.7, respectively. Some of the tables contains values which might look redundant, for example actor and film actor. This is only due to Wikidata database containing a lot general values as well as specific ones. In the end, usually both of them are used, for example Angelina Jolie is labeled as actor as well as film actor and fifteen other occupations [20].

Attribute name	# ppl w/ attr.	# of vals	# unique vals
`name`	4,421,816	-	-
`description`	3,368,673	-	-
`gender`	4,271,048	4,271,723	37
`birthDate`	4,421,816	-	-
`deathDate`	1,761,955	-	-
`nationality`	2,947,009	3,223,750	1,541
`occupation`	3,585,401	5,382,328	10,745
`images`	825,744	849,066	-
`wikipediaTitle`	1,522,861	-	-

Table 4.4: Overview of number of values of individual properties in the Wikipeople dataset. The second column represents the number of people with specific attribute. Attribute can be a string, integer or a list, but at least one value must be present. In case there are multiple values present the third column is used. Specifically for properties that are lists (gender, nationality, occupation) or dictionaries (images). The number in the third column then refers to all of the values in the data structure. For example with gender there are 4,271,048 people with at least one gender, but 4,271,723 gender values were found in the dataset, which means some of the people in the dataset have more genders assigned to them. The last column refers to number of unique values in the Wikipeople dataset.

Males		Females		All	
Country	**#**	**Country**	**#**	**Country**	**#**
USA	329,502	USA	89,937	USA	421,235
France	176,880	Germany	52,340	France	223,113
Germany	167,648	France	45,635	Germany	220,229
UK	135,521	Japan	38,623	UK	167,382
Japan	113,654	UK	30,652	Japan	156,022
Italy	84,238	Spain	20,709	Italy	104,016
Spain	75,812	Sweden	19,619	Spain	97,090
Soviet Union	62,219	Italy	19,598	Soviet Union	75,206
Canada	53,843	Canada	16,965	Canada	71,114
Poland	53,760	Poland	16,909	Poland	70,723

Table 4.5: Overview of ten most common nationalities in the Wikipeople dataset by gender.

Males		Females		All	
Occupation	**#**	**Occupation**	**#**	**Occupation**	**#**
politician	368,786	actor	90,059	politician	433,656
soccer player	262,688	writer	64,406	soccer player	308,821
writer	151,945	politician	62,720	writer	218,893
actor	113,499	soccer player	44,858	actor	204,606
painter	98,984	singer	39,123	painter	129,707
univ. teacher	94,701	film actor	29,456	journalist	115,513
journalist	88,678	painter	28,650	univ. teacher	113,027
composer	60,425	journalist	26,083	singer	86,364
lawyer	52,307	model	17,425	composer	69,840
historian	48,570	univ. teacher	16,556	film actor	63,432

Table 4.6: Overview of ten most common occupations in the Wikipeople dataset by gender.

Gender	# of occurrences
male	3,228,374
female	1,041,424
transgender female	1,011
non-binary	393
transgender male	226
intersex	77
genderfluid	34
genderqueer	32
cisgender female	27
transgender person	24

Table 4.7: Overview of ten most common genders in the Wikipeople dataset.

Distribution of the estimated age of people in images can be found in Figure 4.4, age is shown separately for males, females and all genders. Distribution of birth years in the dataset can be found in Figure 4.5. Distribution of images extensions in the dataset can be found in Figure 4.6. Distribution of gender in the dataset can be found in Figure 4.7, most of the people in the dataset are males and females, the rest of the genders were put into one category labeled as 'other'. Distribution of detected attributes in images from the dataset can be found in Figure 4.8. Detected attributes in images are age of person and number of faces found.

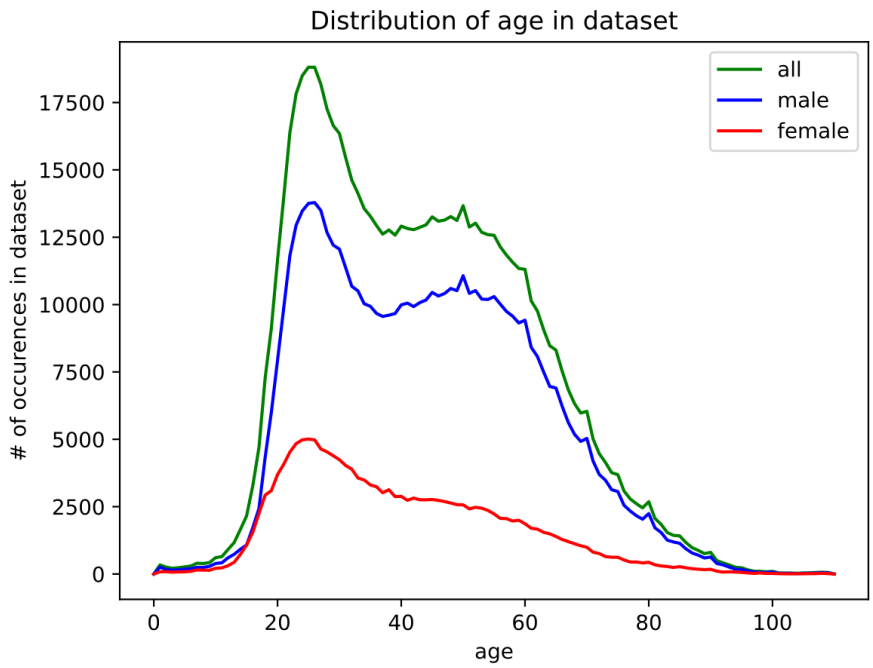


Figure 4.4: Distribution of age in the Wikipeople dataset.

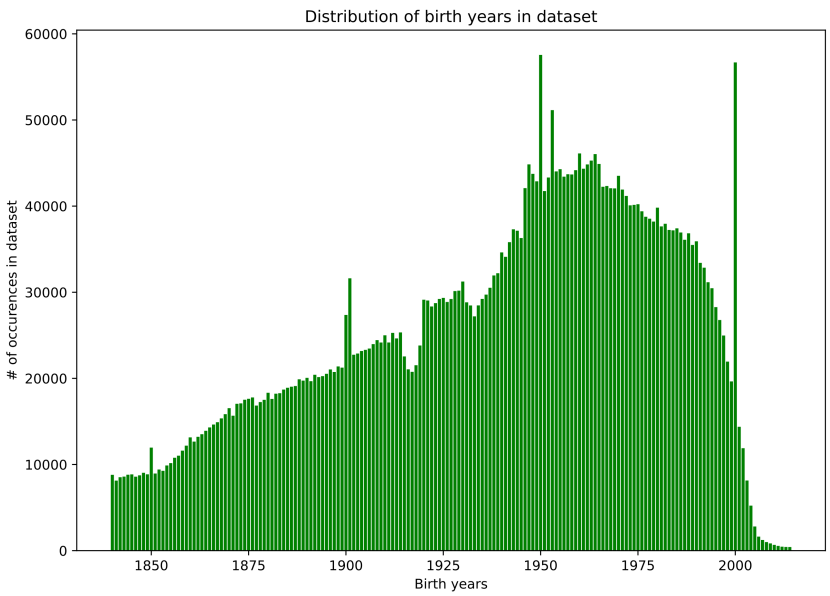


Figure 4.5: Distribution of birth years in the Wikipeople dataset.

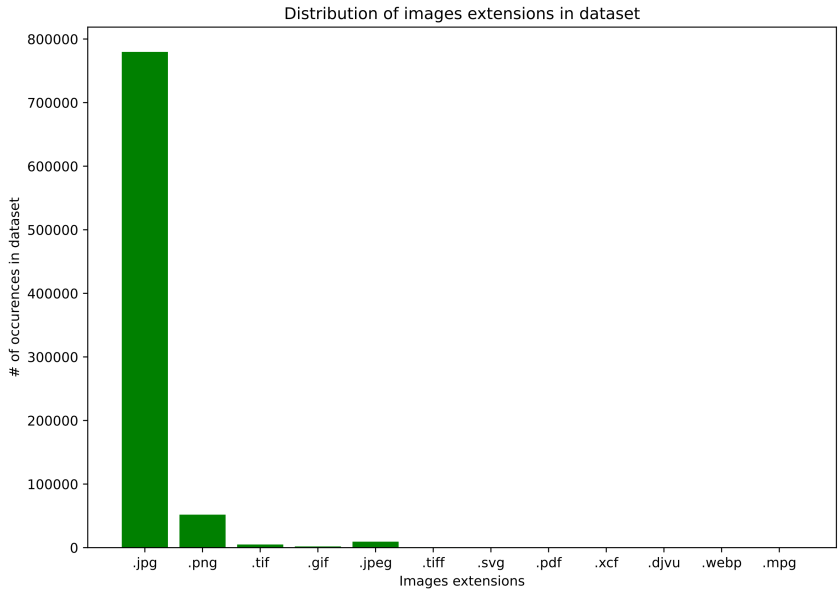


Figure 4.6: Distribution of images extensions in the Wikipeople dataset.

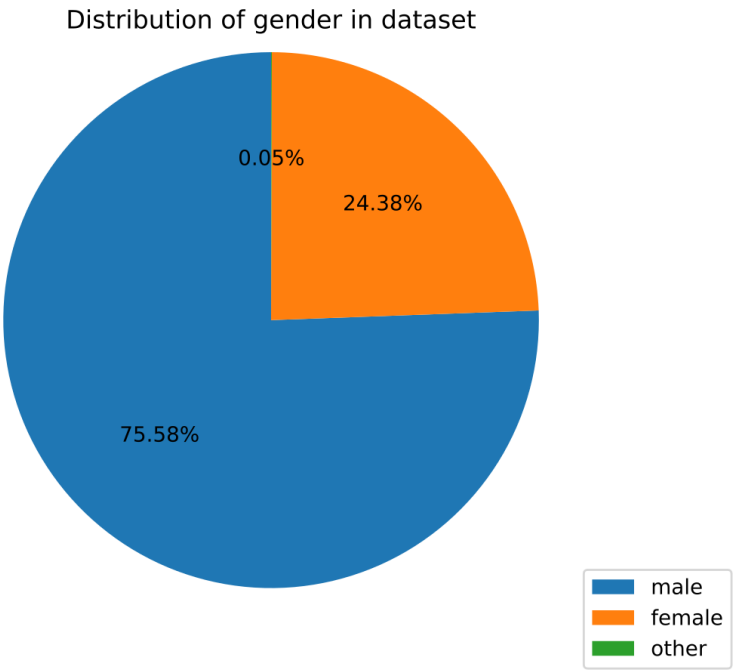


Figure 4.7: Distribution of gender in the Wikipeople dataset.

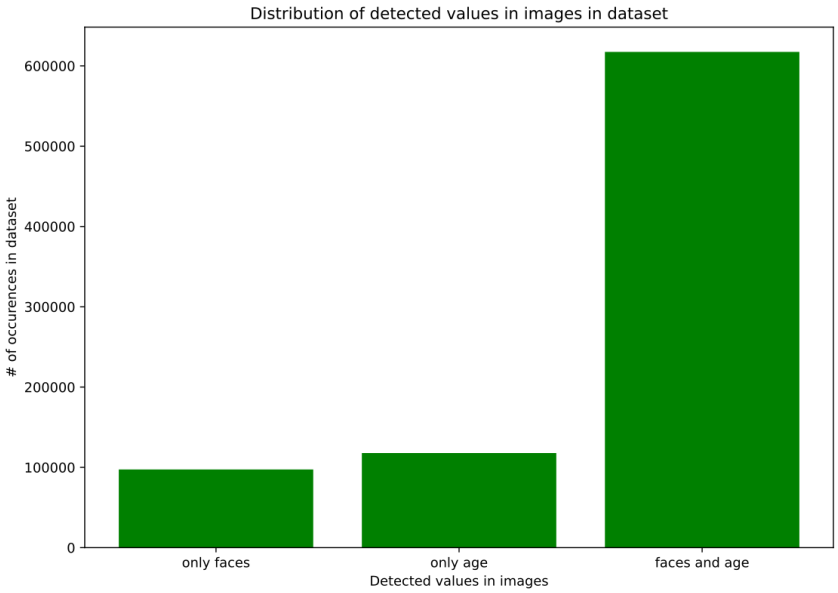


Figure 4.8: Distribution of detected attributes in images from the Wikipeople dataset.

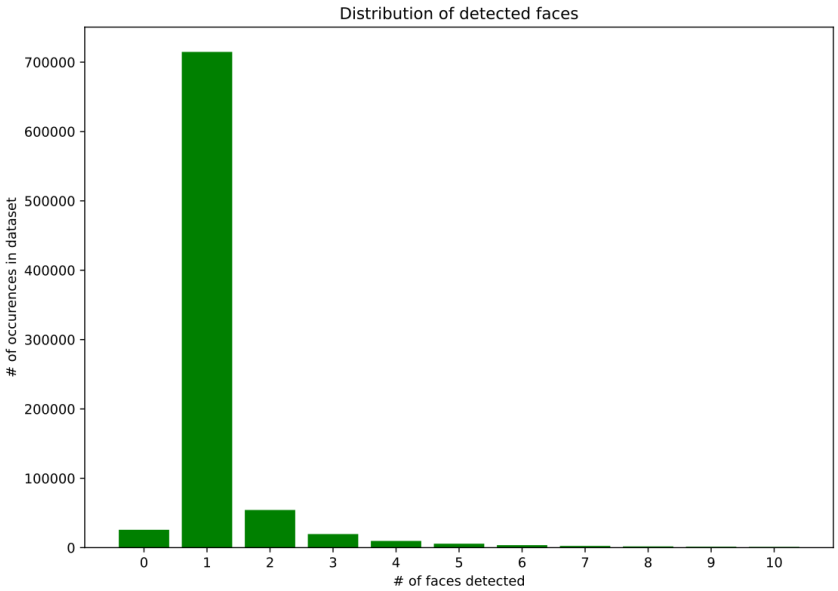


Figure 4.9: Distribution of detected faces in the Wikipeople dataset.

The dataset is growing every day, dataset growth measured between 17th August 2022 and 23rd August 2022 suggest that around 550 people and 250 images can be added to the dataset every day. The number for images will probably be a little lower, because some of them will get removed.

4.3.2 Wikifaces

This subsection presents statistics calculated over the smaller dataset. Wikifaces is a subset of the Wikipeople dataset and contains only data suitable for attribute prediction from an image task. Data suitable for this task are defined as people with images in which exactly one face was detected and at the same time the age of the person in an image was found. This dataset is composed of 604,391 people and contains 617,418 images. Explanation of the same table for the whole dataset can be found in Section 4.3.1.

Attribute name	# ppl w/ attr.	# of vals	# unique vals
`name`	604,391	-	-
`description`	548,405	-	-
`gender`	603,329	603,448	27
`birthDate`	604,391	-	-
`deathDate`	202,939	-	-
`nationality`	550,395	610,700	848
`occupation`	584,716	1,074,837	6,080
`images`	604,391	617,418	-
`wikipediaTitle`	379,024	-	-

Table 4.8: Overview of number of values of individual properties in the Wikifaces dataset. The second column represents the number of people with specific attribute. Attribute can be a string, integer or a list, but at least one value must be present. In case there are multiple values present the third column is used. Specifically for properties that are lists (gender, nationality, occupation) or dictionaries (images). The number in the third column then refers to all of the values in the data structure. For example with gender there are 603,329 people with at least one gender, but 603,448 gender values were found in the dataset, which means some of the people in the dataset have more genders assigned to them. The last column refers to number of unique values in the Wikifaces dataset.

848 nationalities, 6,080 occupations and 27 genders were found in this dataset. Overviews of the ten most common countries, occupations and genders in the dataset can be found in Table 4.9, Table 4.10 and Table 4.11, respectively.

Males		Females		All	
Country	**#**	**Country**	**#**	**Country**	**#**
USA	87,088	USA	25,399	USA	112,771
Germany	29,296	Germany	9,228	Germany	38,543
France	25,363	France	7,077	France	32,468
UK	19,771	UK	6,034	UK	25,854
Italy	16,061	Spain	5,431	Spain	19,941
Spain	14,482	Sweden	4,078	Italy	19,820
Japan	12,127	Italy	3,749	Poland	15,451
Poland	12,029	Netherlands	3,542	Netherlands	15,444
Netherlands	11,878	Poland	3,420	Japan	15,359
Soviet Union	11,778	Russia	3,349	Soviet Union	14,432

Table 4.9: Overview of ten most common nationalities in the Wikifaces dataset by gender.

Males		Females		All	
Occupation	**#**	**Occupation**	**#**	**Occupation**	**#**
politician	87,914	actor	24,809	politician	105,200
soccer player	33,967	politician	17,099	actor	52,583
writer	33,572	writer	14,726	writer	48,422
actor	27,612	singer	13,668	soccer player	36,901
journalist	21,044	film actor	11,426	singer	29,670
univ. teacher	18,430	journalist	7,401	journalist	28,489
singer	15,886	model	6,203	film actor	22,993
lawyer	15,480	TV actor	5,539	univ. teacher	21,477
composer	14,053	stage actor	5,381	lawyer	17,908
painter	12,624	painter	3,482	composer	16,757

Table 4.10: Overview of ten most common occupations in the Wikifaces dataset by gender.

Gender	# of occurrences
male	463,061
female	139,652
transgender female	373
non-binary	162
transgender male	77
intersex	21
genderfluid	20
genderqueer	14
travesti	13
cisgender female	12

Table 4.11: Overview of ten most common genders in the Wikifaces dataset.

Distribution of the estimated age of people in images can be found in Figure 4.10, age is shown separately for males, females and all genders. Distribution of birth years in the dataset can be found in Figure 4.11. Distribution of images extensions in the dataset can be found in Figure 4.12. Distribution of gender in the dataset can be found in Figure 4.13, most of the people in the dataset are males and females, the rest of the genders were put into one category labeled as 'other'. Distribution of images resolution in the dataset can be found in Figure 4.14.

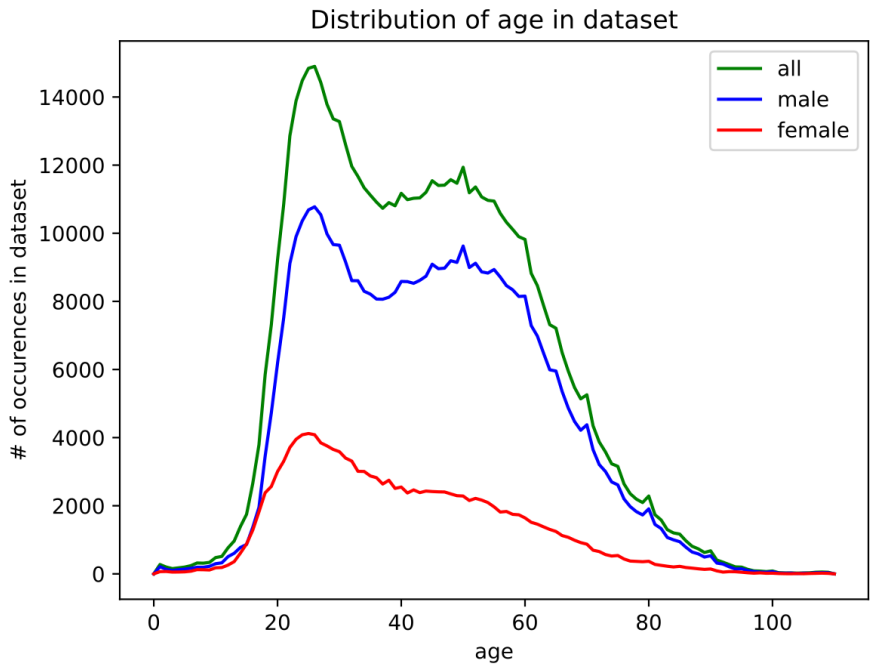


Figure 4.10: Distribution of age in the Wikifaces dataset.

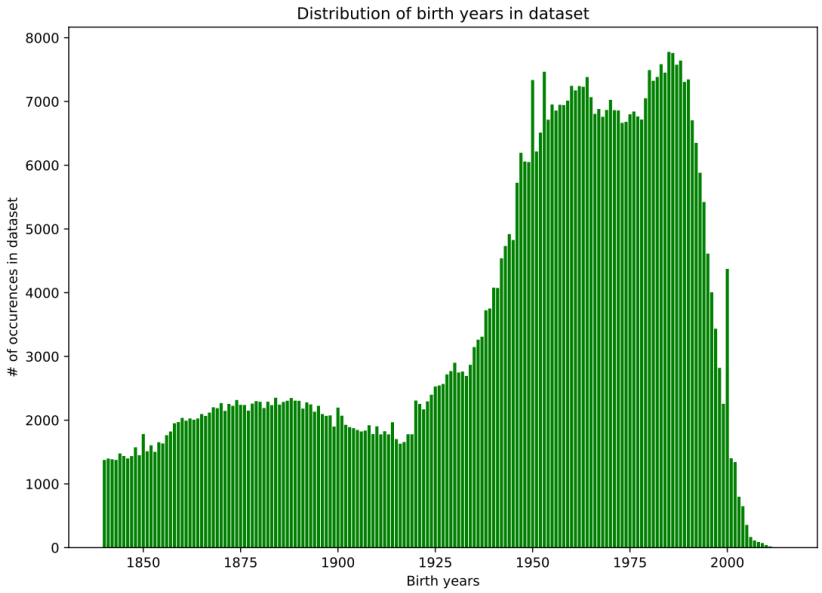


Figure 4.11: Distribution of birth years in the Wikifaces dataset.

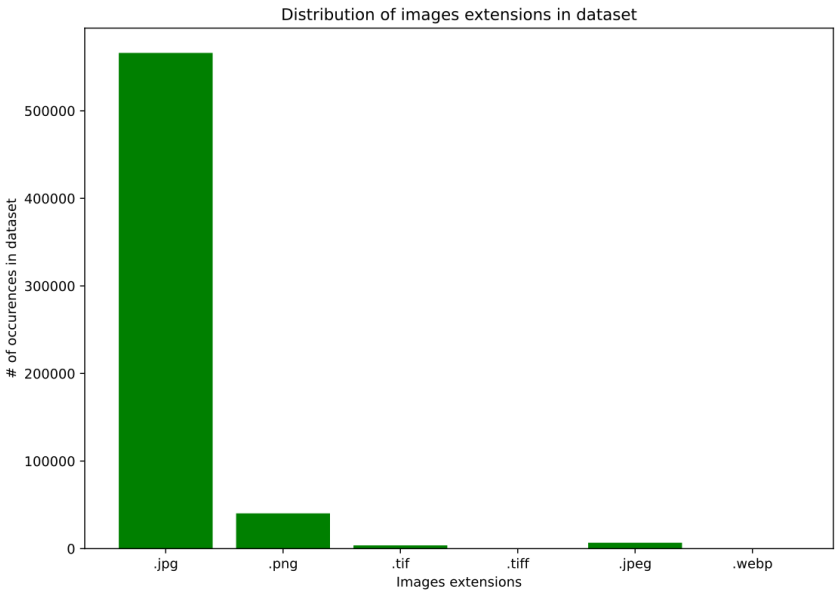


Figure 4.12: Distribution of images extensions in the Wikifaces dataset.

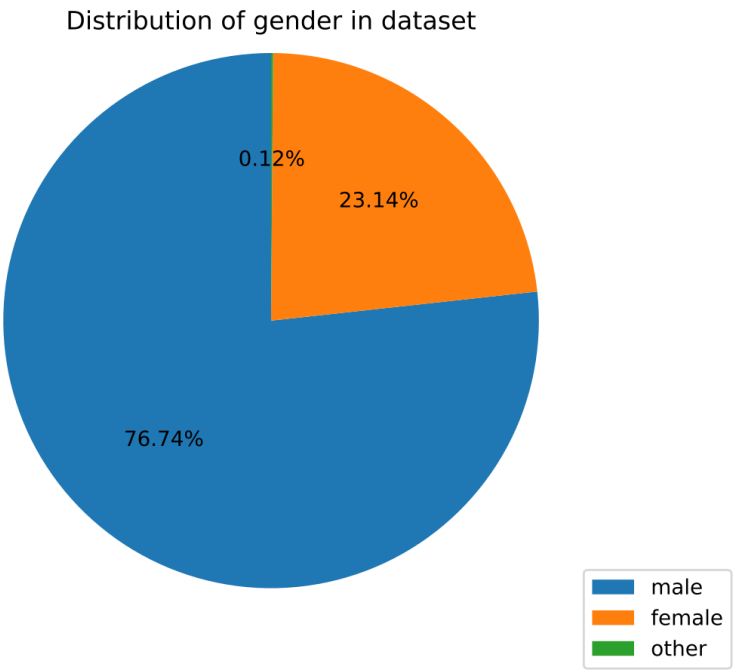


Figure 4.13: Distribution of gender in the Wikifaces dataset.

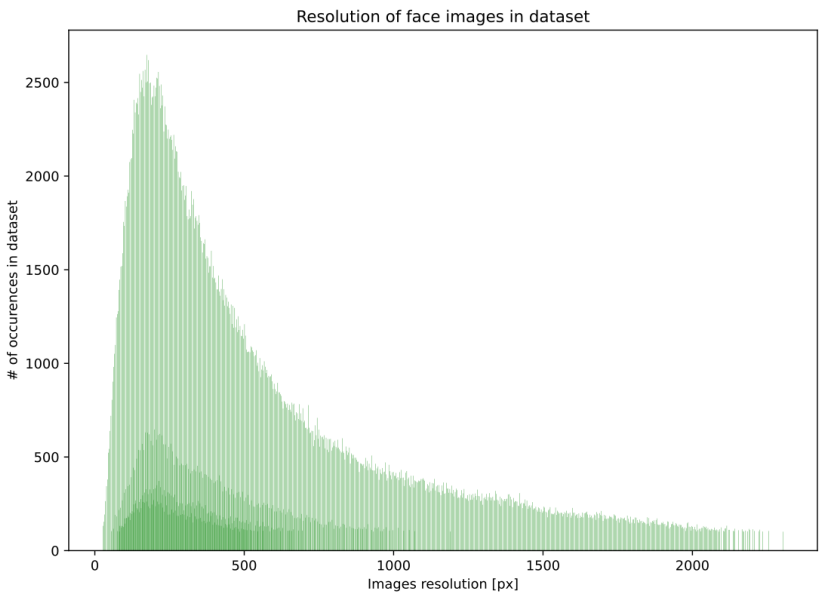


Figure 4.14: Distribution of images resolution in the Wikifaces dataset.

Figure 4.15: Random sample of images from the Wikifaces dataset.

Chapter 5

Experiments

It is one thing to create the dataset; it is quite another to put it into practice. Before doing that the dataset should be statistically evaluated. The goal of this section is to evaluate the reliability of the annotation in dataset. Namely, the age annotation of facial images is evaluated by two methods – direct and indirect. The direct manual method is presented in Section 5.1 and the indirect in Section 5.2.

5.1 Manual evaluation

All attributes were collected automatically, which means that the data come with some noise. Sources of all data were discussed in Chapter 4. The best possible evaluation would be manually annotating a subset of final dataset on all of the attributes (name, gender, description, date of birth, date of death, occupation, nationality and age in picture), but this options is not only time and labour intensive, but also quite cumbersome. Mostly because some of the data presented on Wikidata are not presented on Wikipedia and one can have a hard time deciding what is and is not the right information.

Because of the complications discussed above the author decided to believe in verity of data presented on Wikidata and evaluate only his own algorithm. This heuristic algorithm focuses on finding age of people in the pictures from provided related data (captions, file name, date tag, etc.). For more details see Section 4.2.5.

5.1.1 Proposed method

This experiment was carried on the Wikifaces dataset. Images from this dataset were further filtered for related Wikipedia page. The Wikipedia page is needed, because the infobox presented on the page can be used as a source of date of image creation. All images without a related Wikipedia page were filtered out. The full evaluation dataset has exactly 388,360 images and from those 500 were randomly chosen.

A short script to automate the manual evaluation was created. For every image in the evaluation dataset there were three source, where requested information (image date and caption) could potentially be presented. As

aforementioned in Section 4.2 the final script only collects image captions and dates from two sources (Wikidata and Wikimedia Commons). The infobox from Wikipedia page is not used. Because of this (manual annotation – 3 sources, script – 2 sources) the manual evaluation should be conclusive. Those sources were:

- Wikipedia page of the person in the image – infobox under the thumbnail usually contains a caption of the image with the year of creation. See Figure 5.1 for more detail.

Figure 5.1: Observable information for evaluation available on Wikipedia page of a person. This picture was taken from `https://en.wikipedia.org/wiki/Barack_Obama`.

- Wikidata page of the person in the image – property P18 (image) sometimes contains property P585 (point in time) or property P2096 (media legend). See Figure 4.2 for more detail.
- Wikipedia file page – a summary table under the image usually contains a caption of the image as well as the date of the creation. See Figure 4.3 for more detail.

Those three sources were examined for every single image in the evaluation dataset and the most likely true value was chosen.

As evaluation protocol for this experiment the standard mean absolute error (MAE) is used. MAE is defined as the average of the absolute errors between the estimated age x and the ground truth age y.

$$MAE = \frac{\sum_{i=1}^{n} |x_i - y_i|}{n}$$

5.1.2 Challenges of manual annotation process

Most of the dates of creation of the images were easily identifiable from aforementioned sources. Among the most common obstacles in the annotation process were:

- Thumbnail images presented on Wikipedia page and Wikidata page were different, so one of the sources could not be used.
- Caption or date tag contained decade or range of years instead of a single year. Median of such values was chosen instead.
- Sometimes the year the images were created was difficult or impossible to determine. In that case the best guess from provided data, as well as from further search, was made. Further search included Google image search.

5.1.3 Results

500 images were manually evaluated on the year of their creation. Mean absolute error of heuristic algorithm was 0.63 years and 91.4 % of all the dates were completely correct. The results are also presented in Table 5.1. As presented in the Section 5.1.2 this method has its drawbacks, because it was not always possible to annotate images with a year accuracy. The next method of statistical evaluation presented in Section 5.2 should be more error proof.

Mean absolute error	% of completely correct age annotation
0.63	91.4 %

Table 5.1: Results of the manual annotation experiment.

5.2 Indirect statistical evaluation

The second evaluation method is based on comparing accuracy of models trained on different datasets. Rather simple method was proposed due to author's limited knowledge in machine learning area, but method presented in Section 5.2.1 should be statistically sound to evaluate the benefits of the Wikifaces dataset.

AgeDB [1] presented in Section 5.2.2 is utilized as a benchmark dataset for comparing accuracy.

5.2.1 Proposed method

The method is pretty straightforward. Because two different, but very similar experiments (age and gender prediction model training) were carried out, some of the description which follows might appear duplicate. 'Age prediction task' refers to the experiment where age prediction models were trained and 'gender prediction task' refers to the experiment where gender prediction models were trained.

As aforementioned AgeDB [1] was chosen as ground truth dataset and split into training, validation and testing samples. The split ratio was 0.6 : 0.2 : 0.2. Training sample was used for model training. Validation sample was used

for choosing the best version of a model and testing sample was used for calculating the final results (accuracy for gender prediction task and MAE for age prediction task) of a model.

Task	Train	Validate	Test
Gender prediction task	9,892	3,297	3,297
Age prediction task	9,455	3,151	3,151

Table 5.2: Overview of train/val/test split on datasets derived from AgeDB. For gender prediction task AgeDB dataset was filtered by gender of people in images. Only images annotated with male or female gender were kept in. For age prediction task AgeDB dataset was filtered by age of people in images, only images with annotated age between 17 and 80 inclusive were kept in.

Five different face datasets were created for age prediction task and five different face datasets were created for gender prediction task. The reason why both age and gender prediction tasks were not performed on the same datasets is that the maximal possible number of images, that could be used for the experiments differed. For gender prediction task the Wikifaces dataset was filtered by gender of people in images. Only images annotated with male or female gender were kept in. For age prediction task the Wikifaces dataset was filtered by age of people in images, only images with annotated age between 17 and 80 inclusive were kept in. The same was done for AgeDB, which is the reason why the numbers in Table 5.2 differ for both tasks.

All of the datasets contained train sample from AgeDB database and a specific amount of data from the Wikifaces dataset. Amounts of data added were $[0, 1000, 10000, 100000, all]$. The added data from the Wikifaces dataset were chosen randomly, so stratified sampling was not used. The age distribution for AgeDB and the Wikifaces dataset is somewhat similar as one can observe in Figure 5.2 and Figure 4.10.

These datasets were then used to train two models:

- Gender estimator – this model was trained to guess gender from a facial image. Gender was limited only to male and female due to lack of data.
- Age estimator – this model was trained to guess age with year precision from a face picture. The range of possible estimated years was from 17 to 80 inclusive. The reason for choosing this range was the lack of data for younger and older ages.

The convolutional neural networks created for this experiment were heavily inspired by [21]. A CNN for image classification was trained using transfer learning. Pytorch was chosen as a ML framework and ResNet-50 as a base for transfer learning. As for data transformation each image was resized to 256x256 px, center cropped to 224x224 px and normalized using predefined numbers from the example in [21]. The loss function used for fitting the models was cross-entropy and the optimization method used is an SGD

(stochastic gradient descent) with $learningrate = 0.001$ and $momentum = 0.9$. Generally the models were defined as:

$$p(gender|image)$$

$$p(age|image)$$

The model function then looks like:

$$\hat{y} \in \underset{y \in Y}{\operatorname{argmin}} \sum_{y'} p(y'|x) l(y', y)$$

The loss function used in predictions was different for both models:

- Gender prediction model – Result selection was based on 0-1 loss function, which was done by using `torch.max()` function.

$$l(y', y) = \begin{cases} 0 & y = y' \\ 1 & y \neq y' \end{cases} \qquad y, y' \in Y$$

- Age prediction model – Result selection was based on mean absolute error loss function in order to minimize the expected decision loss.

$$l(y', y) = |y - y'| \qquad y, y' \in Y$$

In the end, all trained models were compared. For gender estimation task, percentage accuracy of a model was chosen as an evaluation protocol. Percentage accuracy is defined as number of correctly estimated genders N_C divided by number of all estimations N_A:

$$ACC = \frac{N_C}{N_A} * 100$$

And for age estimation task, mean absolute error was chosen, which was defined in Section 5.1.

5.2.2 AgeDB dataset

Description of this dataset as presented in the introductory article: *„AgeDB is manually collected "in-the-wild" age database, containing images annotated with accurate to the year, noise-free labels. AgeDB contains 16,488 images of various famous people, such as actors/actresses, writers, scientists, politicians, etc. Every image is annotated with respect to the identity, age and gender attribute. There exist a total of 568 distinct subjects. The average number of images per subject is 29. The minimum and maximum age is 1 and 101, respectively. The average age range for each subject is 50.3 years.“* [1]. A line chart depicting the age distribution of the AgeDB database is presented in Figure 5.2.

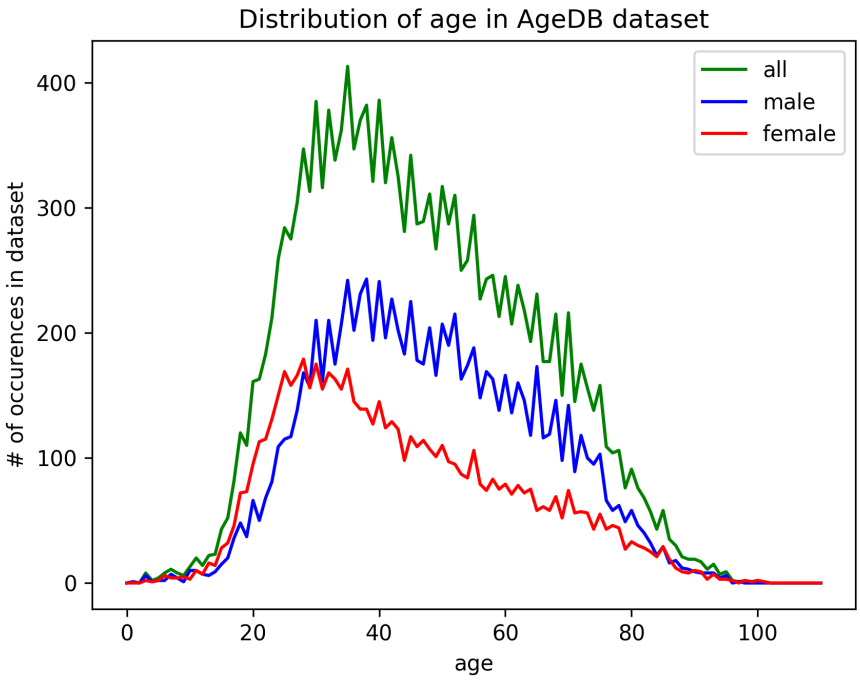


Figure 5.2: Age distribution of AgeDB database.

5.2.3 Challenges of indirect statistical evaluation

Multiple errors were found in AgeDB during the testing of created models. Short script was created to return all people from AgeDB with multiple genders assigned. During this process five people with wrong annotated gender were found and corrected before models' training. Those include:

- Frances Dee – marked 22 times as a female and 22 times as a male
- Joan Lorring – marked 15 times as a female and 15 times as a male
- Roy Scheider – marked 27 times as a female and 27 times as a male
- Gladys Cooper – marked 29 times as a female and 29 times as a male
- Tom Jones – marked 39 times as a female and 39 times as a male

After correcting those, short script was created and one image from every person in AgeDB was compared to an annotated gender. During this process three new errors were found:

- Gena Rowlands – marked as a male in all pictures
- Leslie Caron – marked as a male in all pictures
- Marisa Pavan – marked as a male in all pictures

Even with this effort some of the errors still slipped through as one can observe in Figure 5.3 and Figure 5.4. Removing these errors probably increased the accuracy of the gender prediction model.

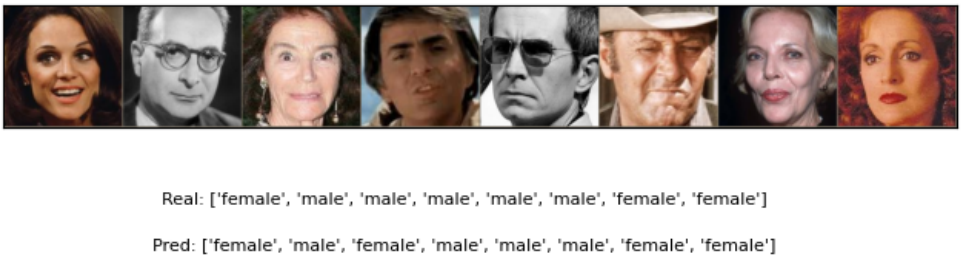


Figure 5.3: Example of gender error in AgeDB which did not get fixed. Pictures come from [1].

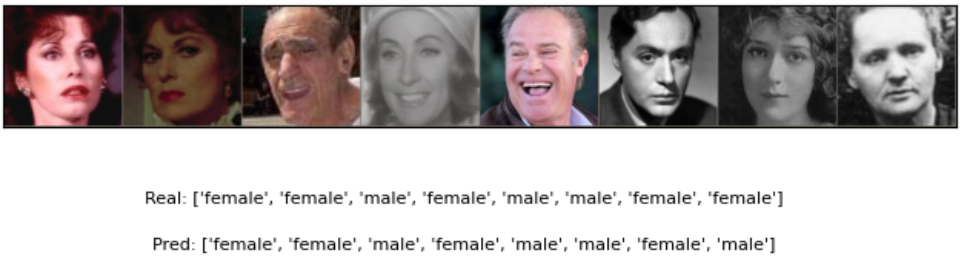


Figure 5.4: Example of gender error in AgeDB which did not get fixed. Pictures come from [1].

5.2.4 Results

Models were trained on grid provided by Center for Machine Perception at FEE CTU. The training of all gender estimation models took around 4 days. Accuracies of all models can be seen in Figure 5.5 and sample estimation can be seen in Figure 5.6.

Dataset	Accuracy
Pure AgeDB = 9,892	98.0285 %
Pure AgeDB + 1k = 10,892	98.0891 %
Pure AgeDB + 10k = 19,890	98.5744 %
Pure AgeDB + 100k = 109,764	98.8474 %
Pure AgeDB + all = 616,958	99.1507 %

Table 5.3: Results of the gender prediction task of indirect statistical evaluation.

The training of all age estimation models took around 5 days. Mean absolute errors of all models can be seen in Figure 5.8 and Table 5.4. Difference in mean absolute errors by years between the first model (pure AgeDB) and last model (pure AgeDB + all possible images from the Wikifaces dataset) can be observed in Figure 5.9. Sample estimation can be seen in Figure 5.7.

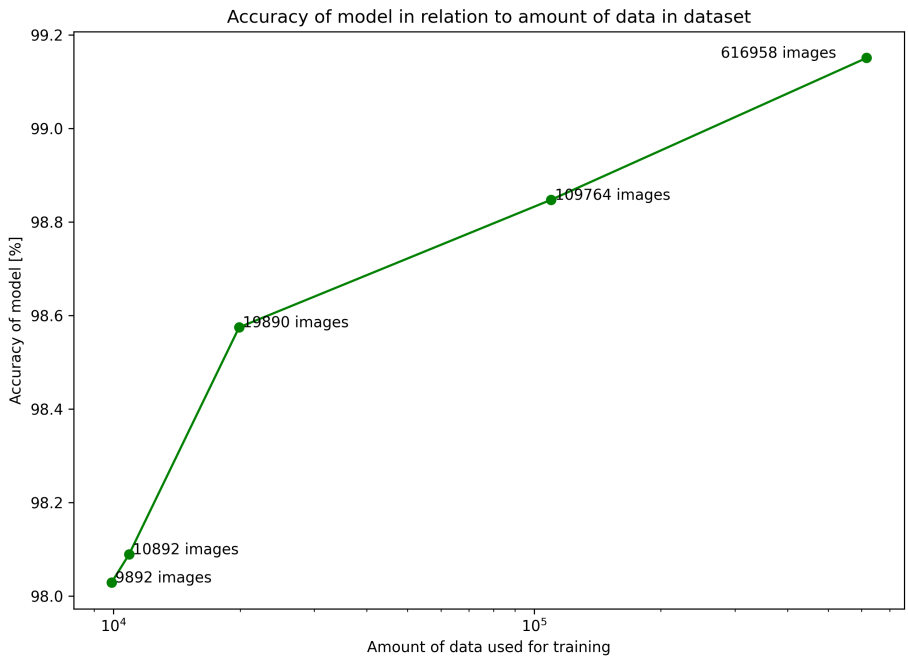


Figure 5.5: Accuracies of all gender prediction models on test data.

Figure 5.6: Random sample of test data annotated with gender prediction.

As the accuracy of gender estimation model increased and mean absolute error of age estimation model decreased it is safe to say that even though the Wikifaces dataset contains some noise, its data can be used for attribute prediction from an image.

Real: ['27', '28', '40', '56', '65', '72', '40', '49']

Pred: ['29', '27', '38', '51', '54', '69', '41', '49']

Figure 5.7: Random sample of test data annotated with age prediction.

Dataset	Mean absolute error
Pure AgeDB = 9,455	7.3941
Pure AgeDB + 1k = 10,455	7.2991
Pure AgeDB + 10k = 19,455	6.5415
Pure AgeDB + 100k = 109,449	5.7933
Pure AgeDB + all = 598,901	5.2652

Table 5.4: Results of the age prediction task of indirect statistical evaluation.

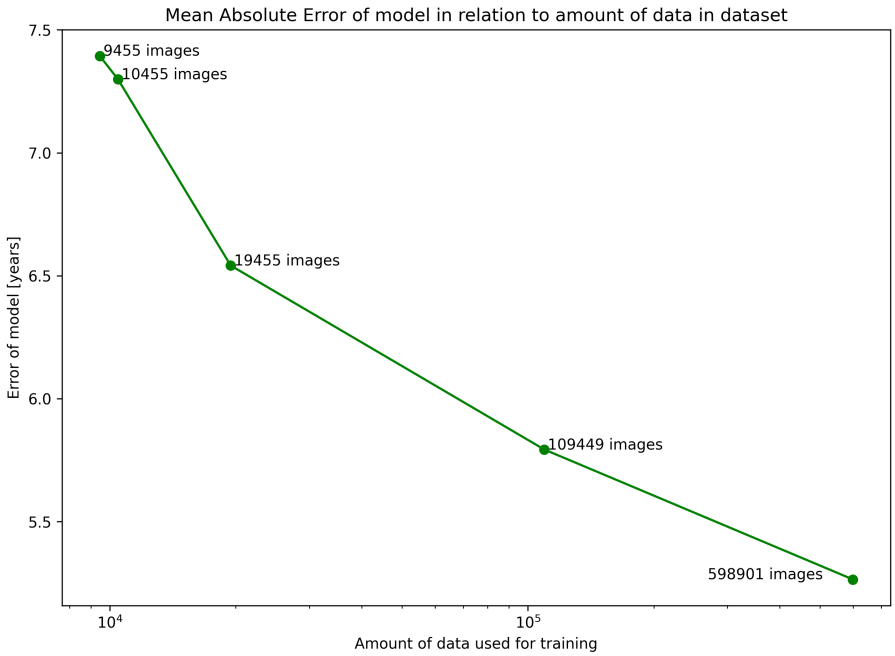


Figure 5.8: Mean absolute errors of all age estimation models on test data.

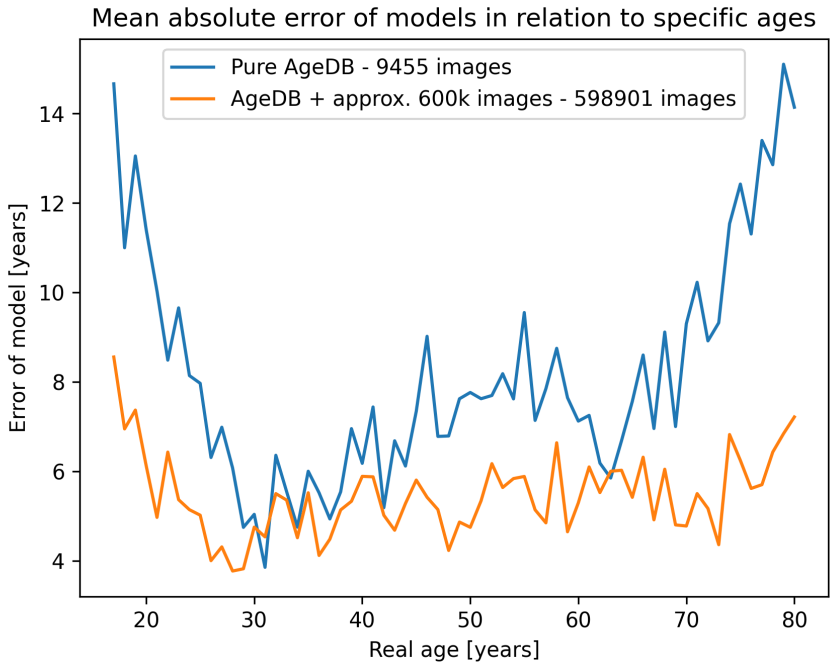


Figure 5.9: Difference in mean absolute errors by years between the first model (pure AgeDB) and last model (pure AgeDB + all possible images from the Wikifaces dataset).

Chapter 6

Future Work

During the development phase a lot more ideas were discovered, yet the time was limited to implement them all. The following ideas could be an inspiration for anyone who would like to continue what has been started and work closely with this thesis. Author believes that just with few more changes, a face dataset of completely different size could be created. All suggested changes should be statistically verified before potential use.

6.1 Using Wiki Projects in Other Languages

Different language variations of Wikipedia have APIs on different addresses. If one wants to search for information about a page on German Wikipedia, one has to go to the proper link. An example of this behaviour can be seen in Appendix B.6. These calls are requesting a thumbnail pictures of Ayn Rand's pages on German and Czech Wikipedia, respectively. At the time of writing this thesis, two different pictures were returned.

The necessity of asking on different APIs does not apply to pictures as they are all stored on Wikipedia Commons which uses its own API. Current version of the script only uses the MediaWiki API on English Wikipedia. Therefore, suggested improvements might include:

- Data obtained from Wikidata SPARQL service contains a lot of people without a Wikipedia page. That is due to a filter in the SPARQL call. Getting rid of this filter increases the number of Wikipedia pages that can be searched for thumbnails. With this change, methods `getThumbnails` and `getThumbnailsForChunk` need to be adjusted too. For more details see the Appendix D.
- As one could observe in Appendix B.6, two different pictures were returned. This suggests trying to call all the possible APIs for thumbnails and saving all unique ones.
- Retrieving metadata information and links for images is done in SPARQL call to Wikidata service as well as in calls to the Wikipedia Commons API. Both services returns data annotated with language tag, so all languages but English get filtered out. Getting rid of this filter can

add more captions and dates to images. At the same time there is a possibility of negatively affecting the heuristic algorithm for adding age to images.

6.2 Downloading all the pictures presented on Wikipedia page

Many personal pages on Wikipedia present multiple images of the same person. Those pictures can be added to the dataset, too, and expand it once again. The only disadvantage is that some of the pictures contain multiple people, some of them do not contain a person at all, and some of them might have a completely different person in them. For example, Russian-born American writer and philosopher Ayn Rand's page contains her own photo as a thumbnail followed by a lot of pictures of her books, and at the very end of the page also an image of Victor Hugo who she admired immensely.

The problem with pictures containing multiple people was already discussed and solved on similar IMDB database. Algorithm developed by Vojtěch Franc and Jan Čech in [22] could be a potential solution after adjusting it to a different task.

6.3 Using other knowledge bases for adding or verifying information

Wikidata was the only knowledge base used in development, but there are other similar projects, which can be used to add more data or verify those that are already in the dataset. Amongst the biggest ones of them which are still alive are DBpedia, WolframAlpha or Google Knowledge Graph. Especially DBpedia is seen as very complementary to Wikidata. The only problem with this approach is that a lot of data in these knowledge bases comes from Wikipedia itself, thus the problem of circular referencing persists. Example requests to those service can be found in Appendix B.7 [23] [24].

6.4 Implementing different strategy for age finder

Algorithm for finding age of people in picture is purely heuristic and therefore, can be improved. Possible adjustments include:

- Using more sources for image caption. Current version is not using the caption from Wikipedia page infobox, because most of the time this caption comes directly from the summary on Wikipedia file page. But there might be exceptions as well as more sources should mean more accurate age determination.

- Implementing more complex heuristic. Used heuristic is fairly simple, but has its caveats. Possible change could be putting years that are close in a group and then using a median of the largest group.
- Using machine learning for estimating the year the picture was taken from related text. The input could be all the possible captions, dates or basically any data related to a certain image. The output then could be a year when picture was taken.

6.5 Using different setting of face detector

Retina-face face detector [25] [19] was used in implementation. Suggestions that could potentially further improve the performance of implemented scripts are:

- Trying different threshold for face detector. Most of the face detectors' output comes with a value of how sure the detector is that the face is actually in the picture. This threshold can be set higher as well as lower for better results.
- Ignoring smaller faces. There are a lot of pictures from public spaces where multiple faces were detected and therefore, those pictures cannot be used for training. For example, take a look at Figure 6.1 in which Michael Jordan is the main person, yet a lot of other faces would be detected. It might still be possible to decide which face matches the person to which the Wikipedia page belongs. For example simple heuristic could be implemented by ignoring all faces smaller then certain size (half of the size of the biggest face found, etc.).
- Using alignment during face detection. According to [26] and [25] alignment can improve the performance of gender and age estimation.
- Using completely different face detector. Significant improvements in this research area are made every year, soon the chosen Retina-face face detector might be outperformed by a better one.

6.6 Creating a downloadable dataset of free images

Current script downloads all the pictures despite their set permissions. This strategy prevents creating a downloadable dataset, which could be shared easily. Anyone with the script can create the dataset, but the dataset should not be shared as images presented on Wikipedia can fall into three categories:

- Public domain - those are images that have had their copyright waived or expired, so no copyright limitations apply. On the other hand it's still good practice to indicate one's source.

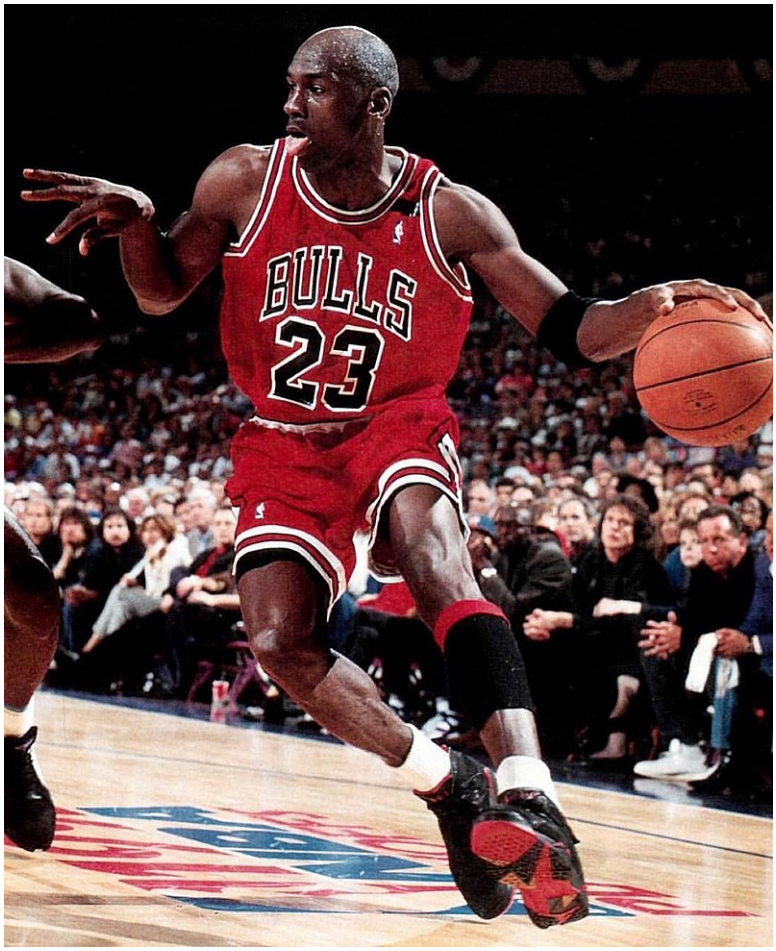


Figure 6.1: Example of image in which multiple faces would be detected, but the dominant one is easily recognizable. This picture was taken from `https://en.wikipedia.org/wiki/Michael_Jordan#/media/File:Jordan_elgrafico_1992.jpg`.

- Freely-licensed images - those are available under a specific license which makes them freely available to reuse in any way, including commercial use. However, images with this license usually require credit to be given to the original author(s).

- Fair use images - those are not free to use. English Wikipedia uses them without license, under an exception built into U.S. copyright law. Wikipedia's non-profit, educational, limited use of images is an ideal example of applied fair use.

Modifying the script in a way so it only uses the free images might be a good starting point. MediaWiki API can return information about the license of file listed on Wikipedia Commons. Example of such a call can be found in Appendix B.8.

6.7 Creating a dataset with only noise-free labels

The dataset contains images whose age annotation can be considered noise-free because the age determination was absolutely unambiguous. The process of obtaining the age of the person in an image is somewhat similar to how AgeDB was created. If only pictures with unambiguous age determination are kept in the dataset, the age accuracy would be even higher.

Chapter 7

Conclusion

In this thesis two new facial datasets were introduced – Wikipeople and Wikifaces. These datasets are annotated with multiple attributes. The main one is the age of a person in a picture with year precision. Created datasets are to the author's best of knowledge the biggest semi-automatically in-the-wild collected facial image datasets annotated with the age of a person in a picture with year precision.

The first created dataset is called Wikipeople and refers to the entire created dataset with all of the data. Therefore, the Wikipeople dataset is of a sparse nature. The second one is called Wikifaces and it is a subset of the Wikipeople dataset. Wikifaces contains only data suitable for attribute prediction from an image, which are defined as people with images in which exactly one face was detected and at the same time the age of the person in an image was found.

The Wikifaces dataset is composed of 604,391 people and contains 617,418 images. This dataset is further annotated with related personal data such as name, description, gender, birth date, death date, nationality, occupation as well as related image data such as captions, dates of creation, EXIF dates, source URLs, age of person in an image and detected faces in an image. Amongst the benefits of data put together in this thesis are unique attributes, variability of data (nationality, races, genders), size, potential, because these data are growing every day and also accuracy, which could be improved as proposed with other suggestions in Chapter 6.

This work does not come with the sharable dataset, but anyone can use provided script to create the dataset for oneself. The average time for creating the dataset should be under a week, but of course this depends on available network and used hardware. Amongst the advantages of provided script are simplicity, maintainability and possible parallelization.

This dataset can be used not only as image dataset for the purposes of label (age, gender, etc.) estimation research but can also be used as a general dataset containing biographies of a large number of people and therefore, used for other research.

Quality of the obtained annotation was statistically evaluated, with focus on the most challenging part – age of the person in an image. Furthermore, the age and gender estimation tasks were formulated as classification problems

and a set of convolutional neural networks were trained using the Wikifaces dataset. AgeDB was used as a benchmark and the results suggest that even though the Wikifaces dataset contains some noise in the age of a person in an image attribute it could be used in the computer vision field as well as for further research. This noise was determined to be around 0.63 years.

Link to the repository with the source code and explanation of how to use the script can be found in Appendix F.

Bibliography

[1] Stylianos Moschoglou, Athanasios Papaioannou, Christos Sagonas, Jiankang Deng, Irene Kotsia, and Stefanos Zafeiriou. Agedb: The first manually collected, in-the-wild age database. *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 1997–2005, 2017.

[2] Rasmus Rothe, Radu Timofte, and Luc Van Gool. Dex: Deep expectation of apparent age from a single image. *2015 IEEE International Conference on Computer Vision Workshop (ICCVW)*, pages 252–257, 2015.

[3] Google datasearch tool. `https://datasetsearch.research.google.com`. Accessed: 19.07.2022.

[4] Wikimedia Foundation. Definition of wikimedia foundation [online], 2022 [Accessed: 17.07.2022].
Available from: `https://en.wikipedia.org/wiki/Wikimedia_Foundation`.

[5] Wikimedia Foundation. Definition of wikipedia [online], 2022 [Accessed: 17.07.2022].
Available from: `https://en.wikipedia.org/wiki/Wikipedia`.

[6] Wikimedia Foundation. Wikidata - main page [online]. 2022 [Accessed: 11.05.2022]
Available from: https://www.wikidata.org/wiki/Wikidata:Main Page.

[7] Milan Welser. *Wikipeople*. 2021.

[8] Wikimedia Foundation. Wikidata - wikipedia page [online]. 2022 [Accessed: 11.05.2022]
Available from: https://en.wikipedia.org/wiki/Wikidata.

[9] Wikimedia Foundation. Scope of wikimedia commons project [online], 2022 [Accessed: 17.07.2022].
Available from: `https://commons.wikimedia.org/wiki/Commons:Project_scope`.

[10] Wikimedia Foundation. Wikimedia commons - licensing [online], 2022 [Accessed: 17.07.2022].
Available from: https://commons.wikimedia.org/wiki/Commons:Licensing.

[11] Wikimedia Foundation. Mediawiki – main page [online]. 2021 [Accessed: 17.07.2022].
Available from: https://www.mediawiki.org/wiki/MediaWiki.

[12] Wikimedia Foundation. Manual – what is mediawiki [online]. 2021 [Accessed: 17.07.2022].
Available from: https://www.mediawiki.org/wiki/Manual:What_is_MediaWiki%3F/en.

[13] Tim Berners-Lee and Mark Fischetti. *Weaving the Web*. HarperSanFrancisco, 1999. ISBN: 978-0-06-251587-2.

[14] G. Kuck. Tim berners-lee's semantic web. *SA Journal of Information Management*, 2004.

[15] Richard Cyganiak, David Wood, and Markus Lanthaler. Rdf 1.1 concepts and abstract syntax [online]. 2014 [Accessed: 17.07.2022].
Available from: https://www.w3.org/TR/rdf11-concepts/.

[16] Eric Gordon Prud'hommeaux and Andy Seabornne. Sparql query language for rdf [online]. 2008 [Accessed: 17.07.2022].
Available from: http://www.w3.org/TR/rdf-sparql-query.

[17] Pewdiepie record on wikidata. https://www.wikidata.org/wiki/Q13423853. Accessed: 08.07.2022.

[18] Wikidata request a query service. https://www.wikidata.org/wiki/Wikidata:Request_a_query. Accessed: 08.07.2022.

[19] Retina-face. https://github.com/serengil/retinaface. Accessed: 08.07.2022.

[20] Wikimedia Foundation. Angelina jolie record on wikidata, 2022 [Accessed: 17.07.2022].
Available from: https://www.wikidata.org/wiki/Q13909.

[21] Pytorch example of transfer learning. https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html. Accessed: 23.06.2022.

[22] Vojtech Franc and Jan Cech. Learning cnns from weakly annotated facial images. *Image and Vision Computing*, 2018.

[23] Andrea Wei-Ching Huang. A preliminary study on wikipedia, dbpedia and wikidata [online]. 2015 [Accessed: 08.07.2022].
Available from: http://andrea-index.blogspot.com/2015/06/wikipedia-dbpedia-wikidata.html.

[24] Michael Färber, Basil Ell, Carsten Menne, and Achim Rettinger. A comparative survey of dbpedia, freebase, opencyc, wikidata, and yago. 2015.

[25] Sefik Ilkin Serengil and Alper Ozpinar. *HyperExtended LightFace: A Facial Attribute Analysis Framework*. 2021.

[26] Olatunbosun Agbo-Ajala and Serestina Viriri. Deeply learned classifiers for age and gender predictions of unfiltered faces. *The Scientific World Journal*, 2020.

[27] Wikimedia Foundation. Wikipedia purpose [online]. 2022 [Accessed: 10.05.2022]
Available from: https://en.wikipedia.org/wiki/Wikipedia:Purpose.

[28] Petscan – querying tool. `petscan.wmflabs.org`. Accessed: 13.05.2022.

[29] Wikimedia Foundation. Api:client code [online]. 2021 [Accessed: 10.05.2022]
Available from: https://www.mediawiki.org/wiki/API:Client code#Python.

[30] Wikimedia Foundation. Manual:pywikibot [online]. 2022 [Accessed: 10.05.2022]
Available from: https://www.mediawiki.org/wiki/Manual:Pywikibot.

[31] Wikimedia Foundation. Pywikibot documentation [online]. 2022 [Accessed: 10.05.2022]
Available from: https://doc.wikimedia.org/pywikibot/master/.

[32] Wikimedia Foundation. Wikidata sparql query service examples [online]. 2022 [Accessed: 08.07.2022].
Available from: `https://www.wikidata.org/wiki/Wikidata:SPARQL_query_service/queries/examples/human`.

[33] Wikipedia dumps. `https://dumps.wikimedia.org/`. Accessed: 08.07.2022.

[34] Wikipedia export tool. `https://en.wikipedia.org/wiki/Special:Export`. Accessed: 08.07.2022.

[35] Wikimedia Foundation. Wikipedia - size of wikipedia [online]. 2022 [Accessed: 11.05.2022]
Available from: https://en.wikipedia.org/wiki/Wikipedia:Size of Wikipedia.

[36] Wikimedia Foundation. Mediawiki - hashed upload directory [online]. 2022 [Accessed: 08.07.2022].
Available from: `https://www.mediawiki.org/wiki/Manual:\protect\T1\textdollarwgHashedUploadDirectory`.

[37] Wikimedia Foundation. Wikidata – autobiographies [online]. 2022 [Accessed: 17.07.2022].
Available from: https://www.wikidata.org/wiki/Wikidata:Autobiography.

[38] Wikimedia Foundation. Wikidata – sparql endpoint, 2022 [Accessed: 17.07.2022].
Available from: https://query.wikidata.org/.

[39] Mediawiki – api:etiquette. https://www.mediawiki.org/wiki/API:Etiquette. Accessed: 27.07.2022.

[40] Ijson project on github and pypi. https://github.com/ICRAR/ijson. Accessed: 23.07.2022.

[41] Tool for visualisating live changes made to wikimedia foundation projects with sounds. http://listen.hatnote.com/. Accessed: 23.07.2022.

Appendix A

List of abbreviations and terms used

WMF Wikimedia Foundation

AI Artificial Intelligence

ML Machine Learning

API Application Programming Interface

W3C World Wide Web Consortium

RDF Resource Description Framework

OWL Web Ontology Language

DL Deep Learning

KB Knowledge base

RDF Resource Description Framework

MAE Mean Absolute Error

JSON JavaScript Object Notation

PHP General-purpose scripting language geared toward web development

MySQL Open-source relational database management system

HTML HyperText Markup Language

CSS Cascading Style Sheets

wiki Hypertext publication collaboratively edited and managed by its own audience directly.

Wikitext The markup language used by Wikimedia Projects. It is also known as wiki markup or wikicode, consists of the syntax and keywords used by the MediaWiki software to format a page.

Appendix B

API requests examples

B.1 MediaWiki API Request Example

```
https://en.wikipedia.org/w/api.php?
    action=query&
    prop=revisions&
    rvprop=content&
    format=json&
    titles=Ayn Rand&
    rvsection=0
```

Listing B.1: MediaWiki API Request Example

Results can be seen here.

B.2 Wikidata REST API Request Example

```
https://www.wikidata.org/w/api.php?
    action=wbgetentities&
    sites=enwiki&
    titles=Ayn\%20Rand&props=claims&
    format=json
```

Listing B.2: Wikidata REST API Request Example

Results can be seen here.

B.3 Wikidata SPARQL API Request Example

Wikidata comes with a user friendly console also called Wikidata Query Service, that can be used for writing queries and testing them. This console is available on https://query.wikidata.org/. An example SPARQL call requesting all the important information about all the people on Wikidata born between years 1840 and 1841 follows.

```
SELECT DISTINCT ?wikidataID ?name ?description ?image ?caption
    ?imageDate ?birthdate ?deathdate ?gender ?nationality
    ?occupation ?wikipediaTitle WITH {
  SELECT ?wikidataID ?name ?birthdate WHERE {
  ?wikidataID wdt:P31 wd:Q5;
    rdfs:label ?name;
    wdt:P569 ?birthdate.
  hint:Prior hint:rangeSafe "true"^^xsd:boolean.
  FILTER(("1840-00-00"^^xsd:dateTime <= ?birthdate) && (?birthdate <
      "1841-00-00"^^xsd:dateTime))
  FILTER((LANG(?name)) = "en")
} } as %i
WHERE {
  INCLUDE %i
   OPTIONAL { ?wikidataID schema:description ?description.
       FILTER((LANG(?description)) = "en") }
  OPTIONAL { ?wikidataID wdt:P734 ?lastname. }
  OPTIONAL { ?wikidataID wdt:P570 ?deathdate. }
  OPTIONAL { ?wikidataID wdt:P27 ?nationality . }
  optional { ?wikidataID wdt:P106 ?occupation . }
  OPTIONAL { ?wikidataID wdt:P21 ?gender. }
  OPTIONAL {
    ?wikidataID p:P18 ?stat.
    ?stat ps:P18 ?image.
    OPTIONAL { ?stat pq:P2096 ?caption. FILTER
        (langmatches(lang(?caption), "en")) }
    OPTIONAL { ?stat pq:P585 ?imageDate. }
  }
}
```

Listing B.3: Wikidata SPARQL API Request Example

Results can be observed after copying the code into the Wikidata Query Service.

B.4 Example request for getting thumbnail of a Wikipedia page

Following calls are requesting thumbnail picture of the Ayn Rand's English page on Wikipedia.

```
https://en.wikipedia.org/w/api.php?
    action=query&
    titles=Ayn Rand&
    prop=pageprops&
    format=json&
    formatversion=2
```

Listing B.4: Example request for getting thumbnail of a Wikipedia page

Results can be seen here.

B.5 Example request for getting metadata and link for pictures

Following calls are requesting dates of creation, captions and URL of a picture.

```
https://commons.wikimedia.org/w/api.php?
    action=query&
    prop=pageimages\|imageinfo&
    iiprop=extmetadata&
    piprop=original&
    titles=File:Ayn\_Rand\_(1943\_Talbot\_portrait).jpg&
    format=json&
    formatversion=2
```

Listing B.5: Example request for getting metadata and link for pictures

Results can be seen here.

B.6 Difference in response between german and czech API

Following calls are requesting thumbnail picture of the Karel Gott's page. An example of such call to the german API.

```
https://de.wikipedia.org/w/api.php?
    action=query&
    titles=Karel Gott&
    prop=pageprops&
    format=json&
    formatversion=2
```

Listing B.6: Example request to the german API

Results can be seen here.

An example of the same call to the czech API.

```
https://cs.wikipedia.org/w/api.php?
    action=query&
    titles=Karel Gott&
    prop=pageprops&
    format=json&
    formatversion=2
```

Listing B.7: Example request to the czech API

Results can be seen here.

B.7 Other knowledge bases requests examples

Following calls are requesting information about Ayn Rand. An example of such call to DBpedia API.

```
https://dbpedia.org/page/Ayn_Rand
```

Listing B.8: DBpedia API request example

Results can be seen here.

An example of such call to WolframAlpha API.

```
https://www.wolframalpha.com/input?i=ayn rand
```

Listing B.9: WolframAlpha API request example

Results can be seen here.

B.8 MediaWiki API File License Request Example

```
https://commons.wikimedia.org/w/api.php?
    action=query&
    prop=imageinfo&
    iiprop=extmetadata&
    titles=File:Ayn_Rand_(1943_Talbot_portrait).jpg&
    formatversion=2&
    format=json
```

Listing B.10: MediaWiki API File License Request Example

Results can be seen here. The license can be found under the `imageinfo -> extmetadata -> LicenseShortName`.

Appendix C

Methods of gathering data from Wikipedia

Gathering data from Wikipedia is not an easy task, mainly due to unstructured nature of the presented information – human readable text, full sentences, HTML or another markup language. This is understandable considering Wikipedia's main purpose: *„Wikipedia's purpose is to benefit readers by acting as a widely accessible and free encyclopedia; a comprehensive written compendium that contains information on all branches of knowledge.“* [27]. And that is exactly the reason why one has to focus on machine readable data. An infobox is a fixed-format table usually added to the top right-hand corner of articles to present a summary of some unifying aspect that the articles share. An example of infobox can be seen in Figure 1.1. An infobox is the most machine readable part of Wikipedia page and one should take it into consideration.

There are number of ways to approach this task. In order to choose the right one to use in further development, one must evaluate its aspects. The most important aspects to look for are:

- Speed - According to [28] there are currently 1,452,276 pages of people born between 1840 and 2015 available just on English Wikipedia. Processing this number of pages will take some time.
- Complexity of the solution - method should have no or little overhead and should be easy enough to use and customize in case different attributes are needed.

C.1 Web Scraping

Web scraping is a term for getting information from a website or multiple websites using special tools (ParseHub, WebScraper) or using a scripting language (Python, JavaScript). Special tools are self-executing programs, that users can run and obtain desired data in desired format. Those tools usually cannot be plugged into the script pipeline.

Using scripting languages give user more room for maneuver. One can either use parsing libraries (Beautiful Soup, Xmltodict) or ready-made frameworks designed for scraping (Scrapy). The procedure of this method looks like this:

- Download HTML or XML file.

- File parsing. Converting text to an HTML tree structure that can be processed more easily.
- Element selection.

As aforementioned, Wikipedia pages are created from so-called wikitext, which means that parsing HTML page would bring more burden. Parsing XML is a little better, but it still contains a lot of semi-structured and unstructured data. Looking at Figure C.1 and Figure C.2 one can see the difference between structure of infobox in HTML and XML.

```
"font-size:125%;"><div class="fn" style="display:inline">František Křižík</div></th></tr><tr><td colspan="2" class="infobox-image"><a href="
https://en.wikipedia.org/wiki/File:Krizik.jpg" class="image"><img alt="Krizik.jpg" src="
Franti%C5%A1ek%20K%C5%99i%C5%BE%C3%ADk%20-%20Wikipedia_soubory/220px-Krizik.jpg" decoding="async" srcset="
Franti%C5%A1ek%20K%C5%99i%C5%BE%C3%ADk%20-%20Wikipedia_soubory/330px-Krizik.jpg 1.5x,
Franti%C5%A1ek%20K%C5%99i%C5%BE%C3%ADk%20-%20Wikipedia_soubory/440px-Krizik.jpg 2x" data-file-width="601" data-file-height="800" width="220"
 height="293"></a><div class="infobox-caption">František Křižík (most likely in 1902)</div></td></tr><tr><th scope="row" class="
infobox-label">Born</th><td class="infobox-data"><span style="display:none">(<span class="bday">1847-07-08</span>)</span>July 8, 1847<br><
div style="display:inline" class="birthplace"><a href="https://en.wikipedia.org/wiki/Pl%C3%A1nice" title="Plánice">Plánice</a>, <a href="
https://en.wikipedia.org/wiki/Habsburg_Bohemia" class="mw-redirect" title="Habsburg Bohemia">Bohemia</a>,  <a href="https://en.wikipedia.org
/wiki/Austrian_Empire" title="Austrian Empire">Austrian Empire</a></div></td></tr><tr><th scope="row" class="infobox-label">Died</th><td
class="infobox-data">January 22, 1941<span style="display:none">(1941-01-22)</span> (aged 93)<br><div style="display:inline" class="
deathplace"><a href="https://en.wikipedia.org/wiki/St%C3%A1dlec" title="Stádlec">Stádlec</a>, <a href="https://en.wikipedia.org/wiki/
Protectorate_of_Bohemia_and_Moravia" title="Protectorate of Bohemia and Moravia">Protectorate of Bohemia and Moravia</a></div></td></tr><tr>
<th scope="row" class="infobox-label">Resting place</th><td class="infobox-data label"><a href="https://en.wikipedia.org/wiki/
Vy%C5%A1ehrad_cemetery" class="mw-redirect" title="Vyšehrad cemetery">Vyšehrad cemetery</a></td></tr><tr><th scope="row" class="
infobox-label">Nationality</th><td class="infobox-data category"><a href="https://en.wikipedia.org/wiki/Czechs" title="Czechs">Czech</a></td
></tr><tr><th scope="row" class="infobox-label">Occupation</th><td class="infobox-data role"><a href="https://en.wikipedia.org/wiki/Engineer
" title="Engineer">Engineer</a>, <a href="https://en.wikipedia.org/wiki/Entrepreneur" class="mw-redirect" title="Entrepreneur">entrepreneur
</a></td></tr><tr><th scope="row" class="infobox-label">Known for</th><td class="infobox-data"><a href="https://en.wikipedia.org/wiki/
Arc_lamp" title="Arc lamp">Arc lamp</a></td></tr></tbody></table>
<p><b>František Křižík</b> (<small>Czech: </small><span title="Representation in the International Phonetic Alphabet (IPA)" class="IPA"><a
href="https://en.wikipedia.org/wiki/Help:IPA/Czech" title="Help:IPA/Czech">['fɾɑnʃɛsɟɛk 'kɾɪʒiːk]</a></span>; July 8, 1847 – January 22,
1941) was a <a href="https://en.wikipedia.org/wiki/Czech_people" class="mw-redirect" title="Czech people">Czech</a> inventor, electrical
engineer, and entrepreneur.
</p>
```

Figure C.1: Structure of Wikipedia infobox in HTML.

```
  <page>
    <title>František Křižík</title>
    <ns>0</ns>
    <id>1303977</id>
    <revision>
      <id>1043173949</id>
      <parentid>1043173898</parentid>
      <timestamp>2021-09-08T18:44:38Z</timestamp>
      <contributor>
        <username>FromCzech</username>
        <id>36336112</id>
      </contributor>
      <minor/>
      <comment>/* Biography */</comment>
      <model>wikitext</model>
      <format>text/x-wiki</format>
      <text bytes="4903" xml:space="preserve">{{More citations needed|date=June 2020}}
{{Infobox person
| name                      = František Křižík
| image                     = Krizik.jpg
| caption                   = František Křižík (most likely in 1902)
| birth_name                =
| birth_date                = {{birth date|1847|7|8}}
| birth_place               = [[Plánice]], [[Habsburg Bohemia|Bohemia]],  [[Austrian Empire]]
| death_date                = {{death date and age|1941|1|22|1847|7|8}}
| death_place               = [[Stádlec]], [[Protectorate of Bohemia and Moravia]]
| death_cause               =
| resting_place             = [[Vyšehrad cemetery]]
| resting_place_coordinates =
| nationality               = [[Czechs|Czech]]
| other_names               =
| known_for                 = [[Arc lamp]]
| education                 =
| employer                  =
| occupation                = [[Engineer]], [[entrepreneur]]
```

Figure C.2: Structure of Wikipedia infobox in XML.

Another common issue is users placing information in the wrong section of the page or using the wrong template. Every page belongs to a certain category

(person, animal, book, movie, etc.) and its infobox uses the corresponding template. This does not necessarily cause any trouble for human users, but much trouble for a machine extracting information from said page. Not to mention that HTML and XML are both prone to change, which makes it hard to maintain and use repeatedly.

It might seem like this method should be rather fast as the amount of downloaded data is not too big. The real problem lies in the overhead caused by parsing every single file, which can be slow.

This method seems to be quite hard to use in our case and even harder to maintain. The speed is not ideal either. This method is also usually advised against for another reasons. Wikipedia already offers other, easier methods for retrieving desired data, the programmer intending to scrape Wikipedia might reach the limit for requests allowed by Wikipedia. This can potentially affect the performance of Wikipedia servers and can result in connection limitation or IP ban [7].

C.2 Using MediaWiki API

MediaWiki Action API is a RESTful web service that allows users to perform all important actions that can be automated. These include such items as page creation, verification, analysis and search, but it can also provide meta information about the wiki or the logged-in user.

There are multiple different APIs that belong to a Wikipedia project. All of them are built similarly, but the resulting structure can be slightly different. They can also overlap in functionality.

The biggest advantage of using the MediaWiki API is in absence of parsing the HTML or XML files. Another format such as JSON can be used instead, which makes easier access to individual information. Most of the requests can be sent in chunks (using the symbol | to separate page titles), which can speed-up anything a user is trying to do. Another advantage can be, that users only get what they asked for. The API also includes a well-specified limit on the number of queries one can send to specific endpoints per second or minute.

The disadvantage is again speed. Overall there will be more requests than when using web scraping, but the ability of sending them in chunks can outweigh this disadvantage. Using API always comes with its downside as scripts are basically dependent on services of others. Because Wikipedia is a huge project, let's assume, that its API structure would not change. The documentation is also not ideal. It exists, but because Wikipedia is developed and maintained by a lot of people there is literally no structure. There are not many examples and if there are, they might be outdated.

See an example call to MediaWiki API in Appendix B.1. This call retrieves only infobox for Ayn Rand's page, the returned data are in form of wikitext, which still needs to be parsed.

```
▼ query:
  ▼ normalized:
    ▼ 0:
        from:                "František_Křižík"
        to:                  "František Křižík"
  ▼ pages:
    ▼ 1303977:
        pageid:              1303977
        ns:                  0
        title:               "František Křižík"
      ▼ revisions:
        ▼ 0:
            contentformat:   "text/x-wiki"
            contentmodel:    "wikitext"
          ▼ *:               "{{More citations needed|date=June 2020}}\n{{Infobox person\n| name                    = František Křižík\n| image
= Krizik.jpg\n| caption                 = František Křižík (most likely in 1902)\n| birth_name              =\n|
birth_date              = {{birth date|1847|7|8}}\n| birth_place             = [[Plánice]], [[Habsburg
Bohemia|Bohemia]], [[Austrian Empire]]\n| death_date              = {{death date and age|1941|1|22|1847|7|8}}\n|
death_place             = [[Stádlec]], [[Protectorate of Bohemia and Moravia]]\n| death_cause            =\n|
resting_place           = [[Vyšehrad cemetery]]\n| resting_place_coordinates =\n| nationality            =
[[Czechs|Czech]]\n| other_names          =\n| known_for               = [[Arc lamp]]\n| education
=\n| employer                =\n| occupation              = [[Engineer]], [[entrepreneur]]\n| home_town
=\n| title                   =\n| networth                =\n| height                  =\n| term
=\n| predecessor             =\n| successor               =\n| party                   =\n| boards
=\n| religion                =\n| spouse                  =\n| partner                 =\n| children
=\n| parents                 =\n| relatives               =\n| signature               =\n| website
=\n| footnotes               = \n}}\n\n'''František Křižík''' ({{IPA-cs|ˈfraɲcɪʃek ˈkr̝ɪʒiːk|lang}}); July 8, 1847 –
January 22, 1941) was a [[Czech people|Czech]] inventor, electrical engineer, and entrepreneur."
```


Figure C.3: Structure of MediaWiki API response for infobox in JSON.

C.3 Using libraries - Pywikibot

There are multiple libraries one can use for accessing the MediaWiki API. Their purpose is usually to simplify the process of writing a complex API call. Most of those libraries were created by individuals and were left after some time. As I planned on developing the script in Python I only focused on Python libraries. Currently there are just two, which are still maintained - Pywikibot and Pwiki. I assumed they are quite similar in what they can do and decided to only test Pywikibot, because its documentation was richer. You can check all libraries on [29].

Pywikibot framework is a Python library and collection of tools that helps automate work on MediaWiki sites. It was originally created only for Wikipedia, but it is now used throughout the Wikimedia Foundation's projects and on many other MediaWiki wikis.

It is written in Python and built on the API published by the Wikipedia itself (MediaWiki APIs). Its main goal is to make access easier for users to work with these APIs, which can be quite hard to understand at times. Anything that can be found or retrieved via the MediaWiki APIs can also be found or retrieved with Pywikibot. It is usually used for:

- Accessing and editing specific Wikidata items.
- Accessing and editing specific Wikipedia pages.
- Copying data from different wikis.
- Automating any of above tasks.

This method can be seen as very similar to web scraping itself, because as a result it does the same. The only difference is that the page items are accessed through the API, so the HTML page itself is not downloaded. Access to the

individual items of the wiki page will be easier, because one does not have to parse the text in any way, everything is done by the Pywikibot framework.

Speed will probably be an issue again. To gain every single piece of information one request has to be made, because HTML pages are not downloaded. Compared to scraping, we will therefore, have several times more requests to obtain information from one site. As a result, this method will be even slower than web scraping due to the framework overhead. The second problem is the documentation itself, which is not sufficiently informative [7] [30] [31].

C.4 Using available knowledge bases

If Wikipedia was created with the emphasize on human readability, there are also multiple knowledge bases which aims to provide machine readable data about our world. Amongst the biggest are Wikidata and DBpedia. Both are quite similar in their usage and both use SPARQL as its main tool to access data.

DBpedia is an active project which deals with structured data. Most of its data are extracted from the infoboxes in Wikipedia and published in RDF and other formats.

Wikidata project provides a secondary and a tertiary databases of structured data that anyone can edit. This approach aims to allow infoboxes to be created from structured data.

So they are indeed similar, the only difference is between the flow of data. Wikidata aims to provide data for infoboxes and DBpedia aims at making data from infoboxes available in a structured format. I decided to continue with Wikidata as I found its documentations to be easier to comprehend [23] [24].

Wikidata project was more deeply introduced in Section 3.1.2.

There are two options to obtain related data from Wikidata service:

- REST API – which is very similar to all MediaWiki API as it runs on the same backend. This API is again not very well documented as one can see; however, it can serve a purpose. For example, one can search all the information available on Wikidata about a specific Wikipedia page. An example, a call requesting all the available information about Ayn Rand can be found in Appendix B.2.

- SPARQL API – as stated before SPARQL is an RDF query language, that is, a semantic query language for graph databases. An example call to SPARQL endpoint requesting all the important information about all the people on Wikidata born between years 1840 and 1841 can be found in Appendix B.3. One can view the call in user interface of Wikidata Query Service.

A comparison between those two APIs is quite one-sided. SPARQL API is not only faster and easier to process, but also better documented. REST API

might be used in cases when dealing with small amounts of data. Working with SPARQL API requires knowledge of said language, which can be hard at the beginning. On the other hand, there are a lot of great examples [32] as well as the 'request a query' service [18], which can help anyone to get started with even quite complex queries.

This option looks very promising as it is fast as well as quite simple, once one gets over the basics of SPARQL. This steep learning curve can be seen as the one of its disadvantages. Another one could be relying on another source of information. Data might not be accurate as anyone can edit Wikidata as mentioned previously in Section 3.1.2.

C.5 Downloading a snapshot of Wikipedia

The last possible option is to download the entire Wikipedia and process everything offline. There are several Wikipedia dumps done every year, which are free to download. Those downloads contain only text in XML format which also includes the wikitext. There are two possible options:

- Downloading the whole snapshot which will take-up more memory and offline processing can be more complex [33].
- Using export tool on API to only export those pages one is interested in which will be smaller in size and processing can be less complex. At the same time another processing is needed to obtain all the titles of pages we are interested in [34].

Images can also be downloaded in bulk separately from mirror servers, but it does not make much sense, as according to [35] all the media files on commons were around 23TB at the end of 2014. The suggested option is to use mirror servers for downloading a single picture at a time. Media files on mirror servers tend to be outdated. So a lot of pictures which are currently used in Wikipedia pages cannot be found on mirror servers, but only on the main one.

At the beginning this method looked very promising as the speed should not be a problem here. The processing will be similar to using web scraping or the MediaWiki API as the text still contains wikitext. The real problem is data not being kept up to date.

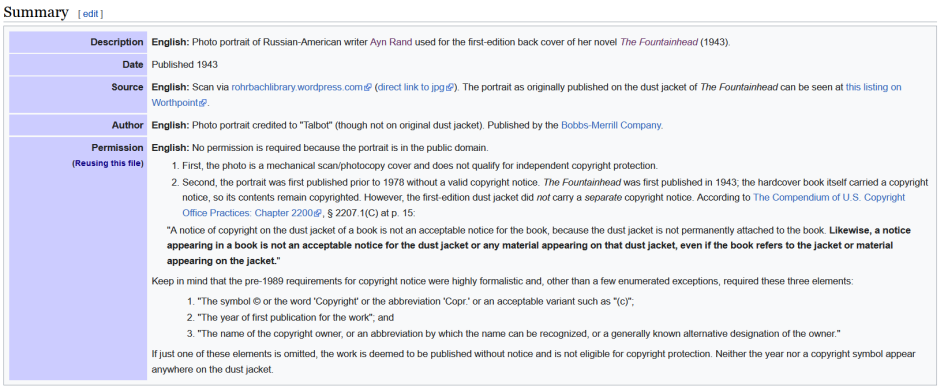
C.6 Retrieving pictures

As mentioned in previous sections – When downloading a snapshot of Wikipedia, one should ideally download pictures from mirror servers, but they tend to be quite outdated. Then the only option is using the main server. One can further decompose this task into downloading the picture itself and getting the related data about the picture (date or year of creation, etc.).

To be able to download the picture itself one needs to obtain an URL of said picture. There are two possible ways of doing that:

- Calculate the MD5 hash of the image file name as presented on Wikimedia Commons and using the first two characters of the hash to construct the URL as described in this article [36].
- Use the MediaWiki API to access such property. Files are listed as pages on Wikimedia Commons, so they can be accessed via the API. An example call can be found in Appendix B.5.

The first option looks a little faster as there is no overhead in waiting for the response from the server. On the other hand, the API needs to be used in order to get an information about the picture, so the overhead is already there. API calls can be merged together, one can therefore, ask for all the information about a picture at once. The data can also be sent in chunks of 50 for common users or 500 for a user with additional rights. See Figure C.4 for example of all the data about images that can be accessed via API. The URL is not listed in the table, but it is on the page as well.



Summary [edit]

Description	**English:** Photo portrait of Russian-American writer Ayn Rand used for the first-edition back cover of her novel *The Fountainhead* (1943).
Date	Published 1943
Source	**English:** Scan via rohrbachlibrary.wordpress.com (direct link to jpg). The portrait as originally published on the dust jacket of *The Fountainhead* can be seen at this listing on Worthpoint.
Author	**English:** Photo portrait credited to "Talbot" (though not on original dust jacket). Published by the Bobbs-Merrill Company.
Permission (Reusing this file)	**English:** No permission is required because the portrait is in the public domain. 1. First, the photo is a mechanical scan/photocopy cover and does not qualify for independent copyright protection. 2. Second, the portrait was first published prior to 1978 without a valid copyright notice. *The Fountainhead* was first published in 1943; the hardcover book itself carried a copyright notice, so its contents remain copyrighted. However, the first-edition dust jacket did *not* carry a *separate* copyright notice. According to The Compendium of U.S. Copyright Office Practices: Chapter 2200, § 2207.1(C) at p. 15: "A notice of copyright on the dust jacket of a book is not an acceptable notice for the book, because the dust jacket is not permanently attached to the book. **Likewise, a notice appearing in a book is not an acceptable notice for the dust jacket or any material appearing on that dust jacket, even if the book refers to the jacket or material appearing on the jacket.**" Keep in mind that the pre-1989 requirements for copyright notice were highly formalistic and, other than a few enumerated exceptions, required these three elements: 1. "The symbol © or the word 'Copyright' or the abbreviation 'Copr.' or an acceptable variant such as "(c)"; 2. "The year of first publication for the work"; and 3. "The name of the copyright owner, or an abbreviation by which the name can be recognized, or a generally known alternative designation of the owner." If just one of these elements is omitted, the work is deemed to be published without notice and is not eligible for copyright protection. Neither the year nor a copyright symbol appear anywhere on the dust jacket.

Figure C.4: Example of image summary on image page. This picture was taken from `https://commons.wikimedia.org/wiki/File:Ayn_Rand_(1943_Talbot_portrait).jpg`.

C.7 Comparison of methods

There are number of ways how to obtain data from Wikipedia. At first sight, the wikitext of some Wikipedia pages looks like it has pretty straightforward structure of the infobox as one can observe in Figure C.2. In reality, this cannot be further from the truth. Wikipedia templates are recursive so one might bump into stuff like `param1 = convert|10|km|mi`; template parameters might contain complex wikitext or HTML markup; some parameters might be missing from the article wikitext and retrieved by the template from a subpage or other data repository. Just finding out where a parameter starts and ends might not be a simple task if it contains other templates which have their own parameters. So this basically excludes web scraping and downloading a snapshot and one is left with using MediaWiki API, available knowledge bases or libraries like Pywikibot.

If the main concern is speed then one should go with knowledge bases as they can process a return a lot of data at once, also Wikidata contains more data about people than Wikipedia itself as discussed in Section 3.1.2. If the main concern is simplicity, then the best shot should be MediaWiki API. Even though its documentation is very incomplete, using another library would make the code just more complicated.

For this project Wikidata and MediaWiki API were chosen as prime sources. In case more sources were needed, Pywikibot was intended as an additional source.

Appendix D

Detailed summary of dataset creation process

This chapter introduces the whole process of dataset creation. Detailed look into the data collection process is discussed in Appendix D.1. Detailed look into the face detection process is discussed in Appendix D.2. Challenges and known issues of implementation are discussed in Appendix D.3.

There are two parts of the entire process:

- Data collection process – this process is responsible for creating the entire dataset, obtaining all of the available data. This process should be run on CPU.
- Face detection process – this process is responsible for face detection part of the dataset creation. Each picture in the dataset is annotated with bounding boxes of faces found. This process should be run on GPU.

D.1 Overview of data collection process

In this section the whole data collection process is described. Every subsection will be dedicated to one subprocess as depicted in Figure D.1. Some of the subprocesses are used multiple times, in that case said subprocesses is introduced only once. One should follow the process diagram and refer to the corresponding subsections.

The process can be initiated by running the `dataCollectionProcess.py`. Code in this module follows the Figure D.1 step by step. As one can observe in the process diagram, there is a loop over the years, which can be set in `constants.py`. More on this in Appendix D.1.1. One iteration of the loop gathers all the data for people born in year `Y`. The reasoning behind using one year for one iteration is explained in Appendix D.1.3.

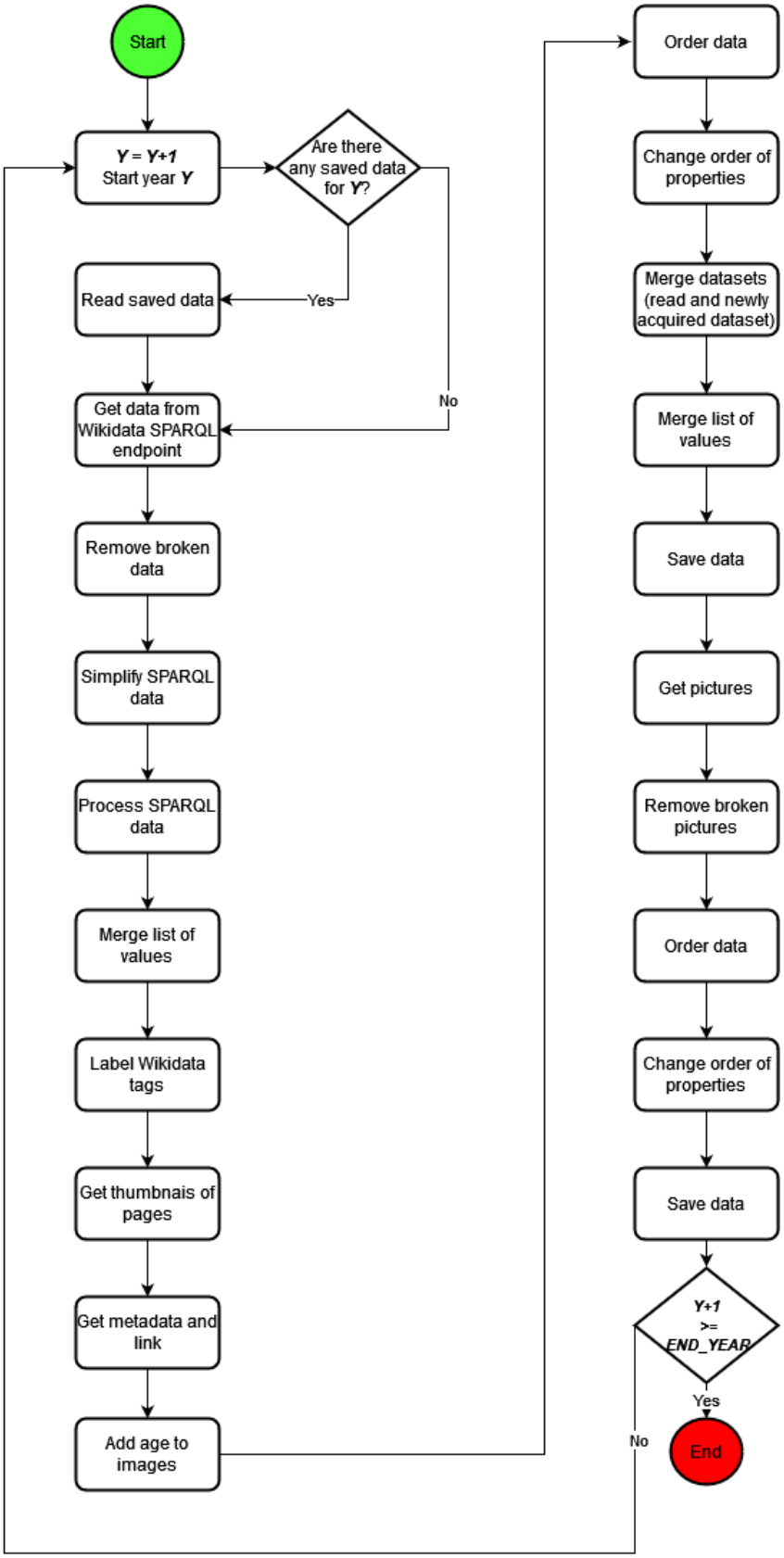


Figure D.1: Process diagram of the data collection part of dataset creation.

D.1.1 Constants

Before starting the data collection process, module `constants.py` should be checked and some of the constants should be set. Those constants can be found under the comment `# CONSTANTS INTENDED TO BE CHANGED`. Those include:

- directories – where collected data will be saved, only the first two are important for the data collection process
  - `IMAGES_DIRECTORY` – directory where all images are saved
  - `DATA_DIRECTORY` – directory where all data by year in JSON format are saved
  - `DATASET_DIRECTORY`, `AGE_DB_IMAGES_DIRECTORY` – these directories were used when creating dataset for experiments, it is not used during the dataset creation
  - `STATS_DIRECTORY` – this directory was used when calculating statistics over the Wikipeople and the Wikifaces datasets, it is not used during the dataset creation
  - `LOGS_DIRECTORY` – directory where log of the creation is saved
  - `RESULTS_DIRECTORY` – this directory was used during the experiments evaluation, it is not used during the dataset creation
- year range – only data about people born between `START_YEAR` and `END_YEAR` are collected
  - `START_YEAR` – start of the range (inclusive), the default was set to 1840, because the reliable camera was invented around 1830s.
  - `END_YEAR` – end of the range (exclusive), the default was set to 2015, because that was the year when enough data are available. This constant should be changed in the future as new data are created every day.
- request headers – `HEADERS` are sent with every request used during the building of the dataset. It is necessary to include a real header in the request otherwise one's request rate might be limited.

D.1.2 Read saved data

The script looks for data for the year `Y` in the `constants.DATA_DIRECTORY` that might have been downloaded in previous runs of the program. If anything is found, those data are read and later merged with the newly acquired data in Appendix D.1.14. If no data are found, the script continues with downloading data from Wikidata SPARQL endpoint. Documentations of used methods can be found in Appendix E.9.4.

D.1.3 Downloading data from Wikidata SPARQL endpoint

The first necessary step is to obtain all the people presented on Wikipedia. This is done via Wikidata SPARQL API. Wikidata SPARQL endpoint is repetitevely called requesting all people born in specific year `Y` together with a wide variety of attributes about them. These attributes are: name, description, image, image caption, image date, birth date, death date, gender, nationality, occupation, wikipedia title, wikidata ID. Each attribute refers to one property of a person and each property has its specific Wikidata tag. For example property P21 refers to a gender of a person. All attributes with their sources and specific Wikidata tags can be found in Table 4.1 and Table 4.2. An example call to SPARQL endpoint requesting all the important information about all the people on Wikidata born between years 1840 and 1841 can be found in Appendix B.3. One can view the call in user interface of Wikidata Query Service. The query used in the script was put together with the help of [18].

Some of the attributes are collected from multiple sources, namely image, image date and image caption are also collected in steps which are discussed in Section 4.2.2 and Section 4.2.3. Data presented on Wikidata API are also presented in human readable form directly on specific Wikidata page of the person. Property P18 (image) sometimes contains property P585 (point in time) or property P2096 (media legend). See Figure 4.2 for more detail.

The reason behind iteration representing exactly one year is that trying to request multiple years at once usually results in time-out. These requests are very heavy as in the most dense years they contain around 60,000 people. One has to take into account the structure of the data returned by the service as mentioned in Appendix D.1.6. Documentations of used methods can be found in Appendix E.3.1.

D.1.4 Remove broken data

This step is necessary, because sometimes Wikidata statements can link to a blank page, usually a page which was deleted due to notability criterion [37]. Filtering out of such records is simple, because the link to a blank page always looks the same. Broken data are defined in `constants.BROKEN_DATA`. Documentations of used methods can be found in Appendix E.8.1.

D.1.5 Simplify SPARQL data

When using the UI version of Wikidata SPARQL endpoint on [38] one can choose between different respond structures. From HTML table, CSV, TSV to two different JSONs. One is verbose and contains a far more complex structure than the other one. When accessing the service from Python, only the complex JSON, which contains a lot of useless data, can be returned. Thus, this step just simplifies the returned data to a less complex structure. Documentations of used methods can be found in Appendix E.8.2.

D.1.6 Process SPARQL data

As aforementioned SPARQL works with RDF triples, these are difficult to present in JSON or basic table format. Trying to do that results in many redundant lines, because one property (predicate) can have multiple values (objects). For an example Ayn Rand is a writer and a philosopher. To represent this one will end up with two records in a JSON or a table returned from the SPARQL endpoint. This step proccesses the JSON data and creates the structure presented in Section 4.1. Documentations of used methods can be found in Appendix E.8.3.

D.1.7 Merge list of values

Most of the attributes are of a list nature, but this does not apply to birth and death date. Those need to have exact values, otherwise age detection would not work. Sometimes people have multiple birth or death dates associated with them. This usually happens, because someone added just a year to the knowledge base and later someone added an exact date. For an example Ayn Rand could have 2 February 1905 and also 1905 in her record. The first one would be listed as 1905-02-02 and the other one as 1905-01-01, because years are always represented as the 1 January.

This step removes the year date if a normal one is present. If one person has multiple dates which are not represented as YEAR-01-01, then a random one is kept. Documentations of used methods can be found in Appendix E.6.1.

D.1.8 Label Wikidata tags

Part of the Wikidata query service is label service. Wikidata operates with entities in a standard IRI format. But IRI such as

$$https://www.wikidata.org/wiki/Q36180$$

is meaningless. The label service can translate IRIs to more human readable format. For example the aforementioned IRI can be translated to 'Writer'. Label service is usually part of the SPARQL query itself, but it is time-consuming especially with heavy calls as those used in this project. As mentioned before, there is also a one minute limit for all the standard queries. Putting the labeling into a separate step can save a lot of time, because all values are labeled only once. Documentations of used methods can be found in Appendix E.5.1.

D.1.9 Get thumbnails for pages

Not all records of people in Wikidata come with a representative facial picture. Therefore, more pictures need to be collected from Wikipedia. As aforementioned not all people presented on Wikidata have a corresponding page on Wikipedia. MediaWiki API is used for getting the thumbnail pictures for people with Wikipedia pages. Result of the query is a name of an image as

presented on Wikimedia Commons, this name can be further use for getting more information about the picture (caption, date, EXIF, etc.). An example call can be seen in Appendix B.4. Documentations of used methods can be found in Appendix E.3.2.

There are usually more pictures on a personal Wikipedia page, but some of the pictures contains multiple people, some of them do not contain a person at all and some of them might have completely different person in them. This is further discussed in Section 6.2.

D.1.10 Get metadata and links

Before this stage begins the dataset contains all the people with their attributes and names of the images as presented on Wikimedia Commons. This stage uses MediaWiki API for gathering additional information about those pictures, such as dates of creation, captions and URL from which to download the picture. Some of those attributes were already collected during the initial Wikidata SPARQL call, but different information can be found in this step and therefore, all of them are stored in a list and later processed in Appendix D.1.11. An example call can be seen in Appendix B.5. Data presented on MediaWiki API are also presented in human readable form directly on Wikimedia Commons. More specifically Wikipedia file page – summary table under the image usually contains a caption of the image as well as the date of the creation. See Figure 4.3 for more detail.

D.1.11 Add age to images

Once all the information is collected, the script can continue with finding the age of a person in an image. For this step only two pieces of information are needed, namely the date of image creation (image year) and the birth date of a person (birth year).

For simplicity, only the years are used to calculate the age of a person in an image. Therefore, the following description uses 'image year' to refer to the date of an image creation and 'birth year' to refer to the birth date of a person.

The image is annotated with the age of the person found in it. This process is based on a very simple heuristic algorithm. Firstly, the script finds the year which is most likely the image year. This is done by gathering all years (four digits in a range from 1800 to 2500) found in date and caption attributes as well as in the image file name. All found years are stored in a list. Then those that do not fit into the possible range are removed. The possible range is defined as a range between the birth and death date of a person the image belongs to and between 0 and 110 exclusively. In the end, the most common value is found and chosen as the image year. Finally, this birth year is subtracted from the image year and found value is saved in the dataset as the age of a person in an image. Documentations of used methods can be found in Appendix E.1.1.

D.1.12 Order data

Everything is alphabetically or numerically ordered in the data in this step. Documentations of used methods can be found in Appendix E.7.1.

D.1.13 Change order of properties

Properties are reordered according to `constants.PERSON_STRUCTURE`, `constants.IMAGE_STRUCTURE` and `constants.FACE_STRUCTURE` if the faces have been already detected. Documentations of used methods can be found in Appendix E.7.2.

D.1.14 Merge datasets

In this step, data that were previously found in the very first step discussed in Appendix D.1.2 are merged with newly acquired data. Only distinct values are kept in the dataset. Documentations of used methods can be found in Appendix E.6.3.

D.1.15 Save data

Data are saved to a `constants.DATA_DIRECTORY`. Every file represents one year and the name of the file is in this format `YEAR.json`. Documentations of used methods can be found in Appendix E.9.3.

D.1.16 Get pictures

URLs of images were collected in the 'get metadata and links' step discussed in Appendix D.1.10. So in this one all pictures in a dataset are downloaded. The local image file name is derived from SHA-256 hash of the image content shortened by the hexdigest() method. The reason behind this decision is that files can be removed from Wikimedia Commons and different file can be uploaded under the same name. This could create some inconsistencies in face detection. Suggest naming should be robust enough to work even if that happens. This is the most time-consuming process of all in the script. Documentations of used methods can be found in Appendix E.3.6.

D.1.17 Remove broken pictures

Some of the downloaded images cannot be opened or used by the face detector. These images are simply removed from the dataset. The number of removed pictures is not too big, which is the reason why there is no mechanism of trying to download them again. This would only make the process more complicated. Documentations of used methods can be found in Appendix E.8.1.

D.2 Face detection process

This process detects faces in all images in the created dataset and saves them back to the dataset. The process diagram of can be found in Figure D.2. It starts with looking for file called `processedImages.json`, which contains faces for all of the images that have already been processed by the face detector. This file contains a dictionary where keys are local file names of images and values are faces detected for said image. If the file is found then it is read and later used in detect faces phase.

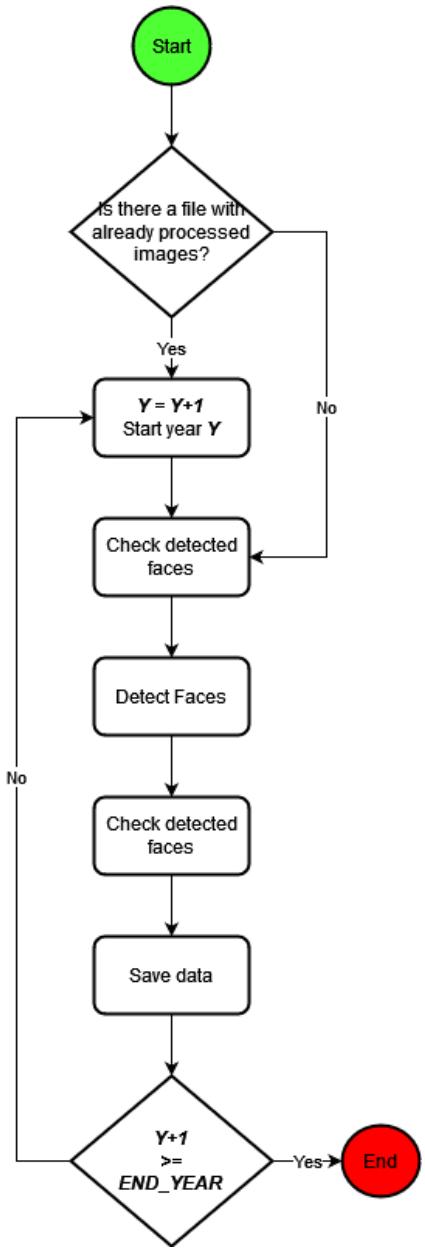


Figure D.2: Process diagram of the face detection part of dataset creation.

D.2.1 Check detected faces

The purpose of this step is to give the user feedback on how many pictures in the dataset have already been put through the face detection. This is done before and after the face detection, so one can immediately see the difference. Debugging on GPU is difficult, so this could be one of the metrics to check the correctness of an algorithm.

D.2.2 Detect faces

As aforementioned face detection is a time-costly process. Therefore, multiple conditions were implemented, so no image is processed more than once. First, if an image is annotated with a face, then the face detection step is skipped altogether. Second, if the image is found in `processedImages.json`, then the saved values are added to the image object in JSON data file. If none of those conditions are true, then the face detection is run with the image and results are saved in both the JSON data file as well as to `processedImages.json`. Documentations of used methods can be found in Appendix E.4.1.

D.2.3 Save data

Data are saved to a `constants.DATA_DIRECTORY` just as previously mentioned in Appendix D.1.15. The file `processedImages.json` is also saved in the same directory. Documentations of used methods can be found in Appendix E.9.3.

D.3 Challenges and possible improvements

Not all design choices turned out to be the best, so in this section challenges and possible improvements are reflected. These improvements only refer to work done in this thesis. Future work proposal is presented in Chapter 6.

D.3.1 Long URLs of images

MediaWiki API allows 'chunking' of requests (ability to ask for multiple titles or files at once). Chunking is done by putting the pipe character behind every title, the limit is 50. Very interesting bug appeared when some of the titles were really long, and therefore, the Wikipedia server refused to process that request. Some file names were 200 and more chars long. Oddly enough a lot of those files with long names belonged to the same year. After the initial confusion, because the script was missing the response code check, limit on URL length was introduced.

D.3.2 Limit on face detectors

Wikipedia rules for image files extensions are very loose, because MediaWiki software can display even quite unusual formats such as .pdf. Those files are,

of course also downloaded, but the face detectors cannot work with them (.gif, .svg, .pdf, etc.). For now those files are simply ignored.

D.3.3 Imperfections of face detectors

Some of the downloaded images contained drawings, statues or busts, which were later annotated with bounding boxes of faces. Those images were left in the dataset, because there is no way to filter them out and some of them could actually be used for training. Example can be seen in Figure D.3

Figure D.3: Example of drawing in dataset, which can confuse the face detector. This picture was taken from `https://commons.wikimedia.org/wiki/File:Portrait_of_Murad_V.jpg`.

D.3.4 Speed

It takes several days to build the dataset. It could be sped up with parallelization, which can be easily implemented as every thread can process a different range of years. Even though there is no specific limit on number of request to both MediaWiki API and Wikidata API that can be done in parallel, one should restrain from flooding the servers with requests as suggested by MediaWiki page on API etiquette: *„There is no hard and fast limit on read requests, but be considerate and try not to take a site down. Most system administrators reserve the right to unceremoniously block you if you do endanger the stability of their site.“* [39].

D.3.5 Inaccuracy in dates of birth

In case a person's record on Wikidata contains multiple dates of birth, and they belong to different years, this person would be in the dataset multiple times. There is no way to detect which date is the correct one. In the end images belonging to such people are annotated with different ages. In case those images end up in the Wikifaces dataset and are used in the attribute prediction task their incorrect ages can even out. The adopted strategy was to include as much data as possible, even though in this case it might be better to simply omit those people and do not put them in the dataset.

D.3.6 Storing results of face detector

The idea of speeding up the process of face detection by storing its results and reusing it in next run, turned out to save a lot of time. The only problem of the first implementation was that it was using a file name as presented on Wikimedia Commons as a key. This can introduce an error in case the file is replaced with a different picture. It should hardly ever happen, but the possibility is there. This issue was solved by using a hash of an image content as a local image name.

D.3.7 Error in getting the page thumbnails

There was an error in getting the page thumbnails phase. This process is described in Appendix D.1.9. It was a typo in one of the JSON's attributes, which resulted in discarding all data gathered in this step. It was discovered after the data were collected, processed and datasets were created. After this error was discovered and solved, the script was run again and approximately 150,000 more images were downloaded. All the statistics and the evaluation were unfortunately completed over the smaller dataset.

D.3.8 Age finder inaccuracy

Some of the images presented on Wikimedia Commons are marked with the entire decade as a date of creation. Because of the simple heuristic used in the age finder, images would be annotated with an inaccurate age. For example, with 1920s the age finder would use the first year – 1920. The more correct approach would be using the middle year of 1925 or using a range instead so no information is lost.

D.3.9 Storage format

The reasoning behind using separate JSON files for every single year is that there is a limit on how large of a file can be read in Python. Creating the dictionary from a JSON file produces a large number objects, which can result in inner Python memory error. There are ways as how to get around it. An example is by using ijson [40]. But there is probably an even better data structure or format that can be used for saving data.

D.3.10 Wikipedia as a live database

Lastly, it is important to remember that Wikipedia is a 'live' site, so its data are added and edited every minute. For example, this can be observed with this live tool [41].

This constantly changing nature can introduce many unexpected errors. They can be prevented, but the complexity would be much complicated, because of the compounded try-catch clauses. To keep the script simple, a solution of these unexpected errors was omitted, and instead, the work was focused on script robustness instead. So, the adopted approach is just rerunning the script from the year that was not successfully processed, which means changing the `constants.START_YEAR` and running the script again.

Appendix E

Documentation

One of the aims of this thesis was to create a script, which will be simple to use and easy to maintain. For the implementation of this aim as well as better understanding of how the whole process of creating the dataset works, the full documentation of the script is presented in this chapter.

The script is written in pseudo-functional style and contains modules which are simply put in a chain. The final script should be easy to parallelize. Modules from package `create` are fully documented, whilst packages `eval` and `model` are used only for statistical evaluation. `Eval` and `model` include code snippets used for:

- Generation of all of the statistics over the Wikipeople and the Wikifaces datasets which are mentioned in Section 4.3.
- Transforming all images into smaller dimension.
- Creating dataset Wikifaces, which is used for attribute prediction from an image discussed in Section 5.2.
- Generating sample for manual evaluation which is mentioned in Section 5.1.
- Checking results of face detection.

Those code snippets can serve as inspiration for future experiments with the dataset created, but with the documentation missing.

Following sections in this chapter refer to single modules from the `create` package.

E.1 AgeFinder

This module contains functions to determine the year the image was created.

E.1.1 Method addAgeToImages

`addAgeToImages(data)` – This method adds age to all people in passed dataset. Keyword arguments are:

- `data` - processed data from SPARQL endpoint

E.1.2 Method addAgeToImage

`addAgeToImage(image, person)` – This method adds age to the image that belongs to a specific person. For simplicity, only the years are used to calculate the age of a person in an image. Therefore, the following description uses 'image year' to refer to the date of an image creation and 'birth year' to refer to the birth date of a person. The image is annotated with the age of the person found in it. This process is based on a very simple heuristic algorithm. Firstly, the script finds the year which is most likely the image year. This is done by gathering all years (four digits in a range from 1800 to 2500) found in date and caption attributes as well as in the image file name. All found years are stored in a list. Then those that do not fit into the possible range are removed. The possible range is defined as a range between the birth and death date of a person the image belongs to and between 0 and 110 exclusively. In the end, the most common value is found and chosen as the image year. Finally, this birth year is subtracted from the image year and the found value is returned. None is returned if there is no value which can be used. Keyword arguments are:

- `image` – specific image to add age to
- `person` – specific person the image belongs to

E.1.3 Method findPotentialYears

`findPotentialYears(text)` – This method finds all potential years in the string. Potential years are defined as four digits in a range from 1800 to 2500. Keyword arguments are:

- `text` – string to search years in

E.1.4 Method isYearInRange

`isYearInRange(year, person)` – This method checks if year is in range of years, when a specified person was alive. Keyword arguments are:

- `year` – year to be checked
- `person` – specific person to check year against

E.2 Corrector

This module contains functions to check and remove broken images in the dataset.

E.2.1 Method removeBrokenImages

`removeBrokenImages(data)` – This method removes broken images in passed dataset. Broken images are images that cannot be opened or worked with packages Pillow or cv2. Keyword arguments are:

- `data` – processed data from SPARQL endpoint

E.2.2 Method isImageBroken

`isImageBroken(location)` – This method checks if image is broken by trying to open it and performing some basic transposition with Pillow package. The image is also verified with package cv2. Keyword arguments are:

- `location` – location of a specific image

E.3 Downloader

This module contains functions to download data that are used to build the dataset. Data come from various APIs (Wikidata SPARQL API, EN MediaWiki API, Wikipedia Commons API).

E.3.1 Method getRawSparqlData

`getRawSparqlData(fromYear=None, toYear=None)` – This method retrieves data about all the people from Wikidata between certain years. Keyword arguments are:

- `fromYear` – beginning of the range (default defined in constants module is set to 1840)
- `toYear` – end of the range (default defined in constants module is set to 2015)

E.3.2 Method getThumbnails

`getThumbnails(data, chunkSize=50)` – This method adds thumbnail images to all people in the passed dataset if they have one on the Wikipedia page, the picture is only added if it is not in the set already (from Wikidata or other source). Keyword arguments are:

- `data` – processed data from SPARQL endpoint
- `chunkSize` – number of people that can be processed at once, limited by mediawiki API to 50 for standard user or 500 for user with additional rights (default: 50)

E.3.3 Method getThumbnailsForChunk

`getThumbnailsForChunk(titles, people)` – This method finds all thumbnail images for Wikipedia titles passed in and saves them in the dataset. Keyword arguments are:

- `titles` – list of Wikipedia titles
- `people` – processed data from SPARQL endpoint

E.3.4 Method getMetadataAndLinks

`getMetadataAndLinks(data, chunkSize=50)` – This method retrieves metadata and links of all images in the passed dataset and saves it back to it. Keyword arguments are:

- `data` – processed data from SPARQL endpoint
- `chunkSize` – number of files that can be processed at once, limited by mediawiki API to 50 for standard user or 500 for user with additional rights (default: 50)

E.3.5 Method getMetadataAndLinksForChunk

`getMetadataAndLinksForChunk(titlesDict, data)` – This method retrieves metadata and links of images in titlesDict and saves it to the dataset. Keyword arguments are:

- `titlesDict` – dictionary with image file names as keys and wikidataID as values, used for storing the retrieved data in the correct place
- `data` – processed data from SPARQL endpoint

E.3.6 Method getPictures

`getPictures(data, startIndex=None, endIndex=None)` – This method downloads all the pictures from the dataset passed in. Keyword arguments are:

- `data` – processed data from SPARQL endpoint
- `startIndex` – starting point in dataset, can be used for dividing dataset or for threading
- `endIndex` – ending point in dataset, can be used for dividing dataset or for threading

E.3.7 Method getPicturesForPerson

`getPicturesForPerson(person)` – This method downloads all of the pictures for the person passed into the method. Name of the local file representing the image is SHA-256 hash of the image content shortened by the hexdigest() method. Keyword arguments are:

- `person` – one person from processed dataset

E.4 FaceDetector

This module contains functions to detect faces in images.

E.4.1 Method detectFaces

`detectFaces(data, processedImages=None)` – This method detects faces in images in the passed dataset. There is a limitation on images extensions. If processedImages dictionary is passed as well, then before detecting faces, the image is searched in the dictionary of images that have been already put through face detection. Keyword arguments are:

- `data` – processed data from SPARQL endpoint
- `processedImages` – dictionary of images that have been already processed with face detection (default: None)

E.4.2 Method detectFacesInImage

`detectFacesInImage(image)` – This method detects faces in one specific image passed into the method. Detected faces are saved directly into the image which is part of the dataset being built. The structure of faces is described in Section 4.1. Keyword arguments are:

- `image` – image from dataset in which to find faces

E.5 Labeler

This module contains functions to label Wikidata tags in dataset.

E.5.1 Method labelTags

`labelTags(data)` – This method labels all Wikidata tags in a dataset passed into the method. Keyword arguments are:

- `data` – processed data from SPARQL endpoint

E.5.2 Method createTagsDictionary

`createTagsDictionary(data, chunkSize=500)` – This method creates a dictionary with tags and their labels from all the data passed into the method. Keyword arguments are:

- `data` – processed data from SPARQL endpoint
- `chunkSize` – number of tags that can be labeled at once, there is a limitation on the endpoint (default: 500)

E.5.3 Method getAllTags

`getAllTags(data, properties)` – This method creates a set of all labels presented in the dataset. Tags are Wikidata identifiers. Usually they can be labeled by the service, but because these SPARQL calls are already quite heavy, they can be labelled afterwards. Keyword arguments are:

- `data` – processed data from SPARQL endpoint
- `properties` – list of properties which contain tags to be labeled

E.6 Merger

This module contains auxiliary functions to merge data into various subsets as well as functions for merging certain attributes.

E.6.1 Method mergeListOfValues

`mergeListOfValues(data)` – This method merges list of values into only one value for all people in data that are passed as an argument. Keyword arguments are:

- `data` – processed data from SPARQL endpoint

E.6.2 Method reduceDate

`reduceDate(date1, date2)` – This method reduces two dates into one. E.g. there might be multiple birth dates for one person. This method returns only one. Usually a person with multiple birth dates has one marked as only the birth year and the other one as the factual birth date. The birth year is entered as YEAR-01-01. The factual birth dates are preferred, if there are two of them then random one is returned. Keyword arguments are:

- `date1` – first date
- `date2` – second date

E.6.3 Method mergeDatasets

`mergeDatasets(datasets)` – This method merges all datasets from list that is passed as an argument. Keyword arguments are:

- `datasets` – list of datasets to be merged

E.6.4 Method mergeAllData

`mergeAllData()` – This method merges data from all years into one dictionary so calculations can be performed on all the data. There are no keyword arguments.

E.6.5 Method splitAllData

`splitAllData(data)` – This method split merged data into data bins defined in constants.YEAR_STEP. Keyword arguments are:

- `data` – processed data from SPARQL endpoint

E.6.6 Method mergeAllDataForTrainingAge

`mergeAllDataForTrainingAge()` – This method merges data from all years into one dictionary and filters out those data not suitable for age estimation model training. There are no keyword arguments.

E.6.7 Method mergeAllDataForTrainingGender

`mergeAllDataForTrainingGender()` – This method merges data from all years into one dictionary and filters out those data not suitable for gender estimation model training. There are no keyword arguments.

E.6.8 Method mergeAllDataForTraining

`mergeAllDataForTraining()` – This method merges data from all years into one dictionary and filters out those data not suitable for model training. There are no keyword arguments.

E.6.9 Method mergeAllDataForEvaluation

`mergeAllDataForEvaluation()` – This method merges data from all years into one dictionary and filters out those data not suitable for evaluation. There are no keyword arguments.

E.7 Sorter

This module contains functions for sorting data.

E.7.1 Method orderData

`orderData(data)` – This method recursively orders data in dataset alphabetically. It is usually called before the method changeOrderOfProperties, because this method is used to sort mostly list of strings anywhere in the dataset. Keyword arguments are:

- `data` – any JSON, dictionary or list

E.7.2 Method changeOrderOfProperties

`changeOrderOfProperties(data)` – This method reorders properties in dataset according to structure defined in constants.PERSON_STRUCTURE, constants.IMAGE_STRUCTURE and constants.FACE_STRUCTURE. Keyword arguments are:

- `data` – processed data from SPARQL endpoint

E.8 Transformer

This module contains functions for transforming data into various forms used for training, evaluating or processing.

E.8.1 Method removeBrokenData

`removeBrokenData(data)` – This method removes unknown data from the dataset, some data available on Wikidata are marked as unknown and look like URLs to empty page. An example looks like: `http://www.wikidata.org/.well-known/genid/eccd559869b138d57`. We can just remove that property and merge it as usual in simplifySparqlData step. Keyword arguments are:

- `data` – raw data from SPARQL endpoint

E.8.2 Method simplifySparqlData

`simplifySparqlData(data)` – This method simplifies raw data from SPARQL endpoint into more usable format, creates all necessary object properties or transform them. This is a custom function, one can change this if one wants to process data differently. Keyword arguments are:

- `data` – raw data from SPARQL endpoint

E.8.3 Method processSparqlData

`processSparqlData(data)` – This method processes simplified data from SPARQL endpoint, checks for duplicities in RDF triplets and only save distinct values. This is a custom function, one can change this if one wants to process data differently. Keyword arguments are:

- `data` – simplified data from SPARQL endpoint

E.8.4 Method toImageData

`toImageData(data)` – This method creates dataset, which is images-oriented. This dataset is better usable for model training. Keyword arguments are:

- `data` – processed data from SPARQL endpoint

E.8.5 Method toTrainingPeople

`toTrainingPeople(data)` – This method creates dataset, which contains only those people used for training. This dataset is then returned. Keyword arguments are:

- `data` – processed data from SPARQL endpoint

E.8.6 Method toPeopleWithGender

`toPeopleWithGender(data)` – This method creates dataset, which contains only people with exactly one gender value. Keyword arguments are:

- `data` – processed data from SPARQL endpoint

E.8.7 Method toImagesBetween17And80

`toImagesBetween17And80(data)` – This method creates a dataset, which contains only images with detected age between 17 and 80 inclusive. Keyword arguments are:

- `data` – processed data from SPARQL endpoint

E.8.8 Method toImagesWithoutTif

`toImagesWithoutTif(data)` – This method creates a dataset, which contains only images without the .tif extension. Keyword arguments are:

- `data` – processed data from SPARQL endpoint

E.8.9 Method toPeopleWithWikipedia

`toPeopleWithWikipedia(data)` – This method creates a dataset, which contains only people that are associated with Wikipedia page. Keyword arguments are:

- `data` – processed data from SPARQL endpoint

E.8.10 Method toPeopleWithAllProps

`toPeopleWithAllProps(data)` – This method creates a dataset, which contains only people with all the props. Keyword arguments are:

- `data` – processed data from SPARQL endpoint

E.8.11 Method toImageDataEvaluation

`toImageDataEvaluation(data)` – This method creates smaller dataset, which contains only images usable for evaluation. Keyword arguments are:

- `data` – processed data from SPARQL endpoint

E.8.12 Method toEvaluationSample

`toEvaluationSample()` – This method creates an evaluation sample. All the training data, annotated images with exactly one face detected in them and age found, are put together. These images are further filtered for a related Wikipedia page. All images without a related Wikipedia page are filtered out. From those 500 are randomly chosen. There are no keyword arguments.

E.9 Utils

This module contains functions that are used in multiple modules.

E.9.1 Method addDistinctValues

`addDistinctValues(property, value, node)` – This method checks for duplicate values in any structure (list, dictionary) and only adds it once. In case there are multiple unique string values, a list is created and values stored in it. Keyword arguments are:

- `property` – name of the property to be added
- `value` – value of the property to be added
- `node` – any object that contains data, object to check the duplicity against

E.9.2 Method getLastPartOfURL

`getLastPartOfURL(url)` – This method returns last part of passed URL, it can be either file, ID, basically anything behind the last slash, be aware of this methods limitation, if there is a special character like ? or ; in the last part of URL, the method would not work properly and it is better to use custom string strip. Keyword arguments are:

- `url` – desired url

E.9.3 Method saveData

`saveData(data, file, indent=2)` – This method saves data to a file with a specific indent. Keyword arguments are:

- `data` – processed data from SPARQL endpoint

- `file` – file and location to save data to
- `indent` – number of spaces for nicer formatting of JSON (default: 2)

E.9.4 Method readData

`readData(file)` – This method reads data from a file passed as an argument. Keyword arguments are:

- `file` – file and location to read data from

E.9.5 Method countProperty

`countProperty(data, properties=, verbose=False)` – This method does a basic statistics on data and counts desired properties. Keyword arguments are:

- `data` – processed data from SPARQL endpoint
- `properties` – dictionary of desired properties to count with assigned boolean value, True for adding all values from lists. False for only counting a number of occurrences of property e.g.: 'images': True will count number of all images in data, adding the length of the list in comparison to: 'deathDate': False will only count number of deathDate presented in data. Mind that all properties with non-list values are percieved as False
- `verbose` – switch between verbose output (True) and nonverbose (False)

E.9.6 Method addToDictionary

`addToDictionary(item, stats)` – This method adds an item to a stats dictionary. This method is used in stats module for calculating dataset attributes. Keyword arguments are:

- `item` – item to be added
- `stats` – dictionary to add item to

E.10 Setup

This module contains functions to set up the environment prior to downloading the dataset.

E.10.1 Method directoriesConfig

`directoriesConfig()` – This method creates all the directories needed for dataset creation. There are no keyword arguments.

E.10.2 Method loggerConfig

`loggerConfig()` – This method sets up the logger. There are no keyword arguments.

E.11 DataCollectionSetup

This module contains function to set up the environment specific to the data collection part of the program.

E.11.1 Method config

`config()` – This method creates a configuration specific to he data collection part of the program. There are no keyword arguments.

E.12 FaceDetectionSetup

This module contains function to set up the environment specific to the face detection part of the program.

E.12.1 Method config

`config()` – This method creates a configuration specific to the face detection part of the program. There are no keyword arguments.

E.13 DataCollectionProcess

This module contains the main function for creating the dataset.

E.13.1 Method fullDataDownload

`fullDataDownload()` – This method create the dataset, download all the data and process it. This method contains only the data collection part. There are no keyword arguments.

E.14 FaceDetectionProcess

This module contains main function for detecting faces in the dataset.

E.14.1 Method detectFacesJob

`detectFacesJob()` – This method detects all the faces in dataset. It looks for the processedImages.json file and uses it, if it is found. There are no keyword arguments.

Appendix F

Repository with code used in this thesis on GitHub

- short link: https://bit.ly/wikip30pl3
- full link: https://github.com/dreamwaffer/wikipeople

Figure F.1: QR code linked to the GitHub repository of this thesis.